



DOCTORAL THESIS

Toward Automated Augmentation for Traffic Classification

Chao WANG

*A thesis submitted in 2025 in fulfillment of the requirements
for the degree of Doctor of Philosophy
in the*

Computer Science, Telecommunications and Electronics Doctoral School
Sorbonne University (ED130) - EDITE de Paris
EURECOM Data Science Department

Committee in charge

Pietro Michiardi	EURECOM	Advisor
Alessandro Finamore	Huawei Research	Co-advisor
Giulio Franzese	EURECOM	Co-advisor
Marco Fiore	IMDEA	Reviewer
Paul Patras	the University of Edinburgh	Reviewer
Francesco Bronzino	École normale supérieure de Lyon	Examiner
Maria A. Zuluaga	EURECOM	Examiner (Jury President)

SORBONNE UNIVERSITY (ED130) - EDITE DE PARIS

Abstract

Doctor of Philosophy

Toward Automated Augmentation for Traffic Classification

by Chao WANG

Network Traffic Classification (TC), which identifies the application responsible for generating a given traffic flow, is essential for managing modern networks by enabling effective resource allocation and service differentiation. However, TC faces growing challenges due to encryption, traffic diversity, inference efficiency, and the scarcity of labeled data. While Machine Learning (ML) offers a promising approach by learning patterns from limited information, it still faces challenges like distribution shifts and a scarcity of publicly available annotations.

This dissertation tackles the challenges mentioned above through three main contributions. First, we benchmark 18 hand-crafted augmentations for supervised TC and integrate them into contrastive learning to handle label-scarce scenarios, demonstrating improved performance and generalization. Second, to deepen our understanding of state-of-the-art generative models, we begin by exploring diffusion models in the ML domain, focusing specifically on conditional text-to-image generation task. We propose novel mutual information estimators using pretrained diffusion and rectified flow models, and apply them in self-supervised fine-tuning to enhance text-to-image alignment without external models or annotations. Third, we advance generative modeling for network traffic by building a standardized benchmarking framework including datasets, preprocessing, baselines, and evaluation, and developing a diffusion model for packet series that outperforms existing generative models in fidelity and downstream utility.

By addressing these areas, this dissertation advances TC under data scarcity, improves alignment in conditional generative models, and provides benchmarks for generative modeling of traffic — laying a foundation for more robust and adaptable ML systems in networking contexts.

Keywords: traffic classification – data augmentation – contrastive learning – diffusion model – flow matching – mutual information

Acknowledgements

First and foremost, I am deeply grateful to my academic advisors, Prof. Pietro Michiardi and Prof. Giulio Franzese, and to my company advisors, Dr. Alessandro Finamore and Dr. Massimo Gallo, for their invaluable support, insightful guidance, and precise remarks. Pietro and Giulio, your remarkable insights into the theoretical aspects of AI, along with your ability to identify key problems and recognize promising directions, have consistently impressed me. They have shaped the way I think about the subject and taught me to approach scientific questions with more clarity and depth. Alessandro and Massimo, your expertise in network traffic and practical implementation has continually impressed me. Thank you for empowering me with knowledge in these areas and for navigating me how to probe and solve real-world problems. These skills were essential to the development and realization of my research. Beyond your dedicated technical support, your encouragement during moments of doubt and your thoughtful guidance throughout my academic journey have been truly instructive. Without your invaluable mentorship, this thesis would not be what it is today. I consider myself incredibly fortunate to have had the opportunity to work with such dedicated and inspiring mentors. Thank you all for everything.

Further, I would like to thank the members at Huawei Paris Research Center for their treasured support and technical insights. The weekly seminars were a source of inspiration, and it was a pleasure to work with such talented and friendly researchers. Thank you also to the friends I met at EURECOM. Although I didn't have many opportunities to spend time on campus, you made me feel part of a warm and welcoming community. I truly enjoyed our coffee chats, which were lighthearted and often sparked fresh perspectives.

Last but not least, I want to thank my family and friends around the globe for their understanding from afar. In particular, I would like to express my profound gratitude to my partner Zeyuan for his immense support, both emotionally and in daily life, which brightened the difficult moments.

Contents

Abstract	iii
Acknowledgements	v
1 Introduction	1
1.1 Overview	1
1.2 Hand-crafted augmentations and contrastive learning for traffic classification	2
1.3 Mutual information for conditional generation	4
1.4 Generative data augmentation for traffic classification	6
I Hand-crafted augmentation and contrastive learning for traffic classification	9
2 Introduction to traffic classification	11
2.1 Input data representations	12
2.1.1 Payload	12
2.1.2 Packet time series	12
2.2 Classifiers	14
2.2.1 Rule-based classifiers	14
2.2.2 Machine learning classifiers	14
2.2.3 Deep learning classifiers	15
3 Benchmark of hand-crafted data augmentation	17
3.1 Introduction	17
3.2 Related work	18
3.2.1 Data augmentation in traditional machine learning tasks	19
3.2.2 Data augmentation in traffic classification	21
3.2.3 Design space	22
3.3 Research goals	24
3.4 Method	24
3.4.1 Benchmarking hand-crafted data augmentation (G1)	24
3.4.2 Training batches composition (G2)	28
3.4.3 Latent space geometry (G3)	29
3.4.4 Combining augmentations (G4)	30

3.5	Experimental settings	30
3.5.1	Datasets	30
3.6	Results	33
3.6.1	Augmentations benchmark (G1)	33
3.6.2	Training batches composition (G2)	37
3.6.3	Latent space geometry (G3)	41
3.6.4	Combining augmentations (G4)	44
3.7	Conclusions	44
4	Contrastive learning for few-shot learning	47
4.1	Introduction	47
4.2	Preliminaries: contrastive learning	49
4.2.1	Implementation of the critic	50
4.2.2	Construction of the subset	52
4.2.3	Related work	54
4.3	Self-supervised contrastive learning on flowpic	55
4.3.1	Experimental protocol	56
4.3.2	Reproducing quantitative results of data augmentation	58
4.3.3	Reproducing qualitative ranking of data augmentation	60
4.3.4	Reproducing constrastive learning results	62
4.3.5	Concluding remarks	64
4.4	Comparing self-supervised and supervised contrastive learning for few shot learning	64
4.4.1	Related work	65
4.4.2	Research questions	69
4.4.3	Experimental protocol	69
4.4.4	Results	71
4.4.5	Concluding remarks	72
II	Mutual information for conditional generation	73
5	Introduction to generative models	75
5.1	Preliminaries – diffusion model	76
5.1.1	Continuous version	77
5.1.2	Discrete version of VP-SDE (i.e. DDPM)	80
5.2	Preliminaries – conditional flow matching	83
5.2.1	Link between the 3 concepts	84
5.2.2	Conditional flow matching: a tractable and valid loss	84
5.2.3	Conditional generation with guidance signal	88
6	Mutual information estimation	89
6.1	Introduction	89
6.2	DDPM-based point-wise MI estimator	91

6.3	RF-based point-wise MI estimator	93
6.3.1	RFMI: estimating MI with RF	93
6.3.2	Experimental evaluation on MI estimation benchmark	100
6.4	Conclusion	101
7	Information theoretic text-to-image alignment	103
7.1	Introduction	103
7.2	Is mutual information meaningful for alignment?	106
7.3	Self-supervised fine-tuning with MITUNE	107
7.4	Experimental protocols	109
7.4.1	Benchmark and metrics	109
7.4.2	MI-TUNE fine-tuning	111
7.4.3	Alternative methods	112
7.5	Benchmark results	113
7.5.1	T2I-CompBench on DM	113
7.5.2	T2I-CompBench on RF	114
7.6	Ablation studies	115
7.7	Conclusion	119
III	Generative modeling of packet series for traffic classification	121
8	Generative modeling of network traffic data	123
8.1	Introduction	123
8.2	Related work	125
8.2.1	Traditional networking traffic synthesis	125
8.2.2	Generative traffic synthesis	126
8.2.3	Generative tabular data synthesis	129
8.2.4	Generative time series synthesis	131
8.3	Research goals	133
8.3.1	Baselines	135
8.4	Our method	138
8.4.1	Diffusion scheduler	138
8.4.2	Neural network architecture	139
8.5	Experiments	141
8.5.1	Datasets and curation	141
8.5.2	Hyperparameters and training details	143
8.6	Replication of NetDiffusion	144
8.7	Classification evaluation	147
8.7.1	Base classifier	148
8.7.2	Class-semantics fidelity (utility test)	151
8.7.3	Additional variety beyond the real training data (augmentation test)	154

8.7.4	Concluding remarks	155
8.8	Similarity evaluation	156
8.8.1	Realism via real/fake discrimination (discriminative score) . .	157
8.8.2	Low-order distribution-level similarity (feature-wise density estimation and pairwise features correlation estimation)	158
8.8.3	High-order sample-level similarity (α -precision and β -recall) . .	161
8.8.4	Concluding remarks	165
9	Discussion and future work	169
A	Appendix for Chapter 4	173
A.1	Layout of DL Network Architectures	173
B	Appendix for Chapter 7	177
B.1	Details on user study	177
B.1.1	Comparing alignment metrics <i>Preliminary analysis to understand if MI is a meaningful alignment signal · Referenced in Section 7.2</i>	177
B.1.2	Comparing alignment methods <i>Actual benchmark of MI-TUNE against alternative methods · Referenced in Section 7.4.1 . .</i>	179
B.2	Experimental protocol details <i>List of parameters used and computation costs considerations. · Referenced in Section 7.4.2</i>	180
B.3	HPS scores range <i>Discussion about the natural small values range provided by HPS · Referenced in Section 7.5</i>	181
B.4	Additional results and ablations	182
B.4.1	Ablation: Fine-tuning set selection strategies <i>Discussing alternative strategies to MI for composing the fine-tuning set · Related to Table 7.4 .</i>	182
B.4.2	Ablation: Fine-tuning model adapters and modalities <i>Investigating LoRA, DoRA and fine-tuning or not CLIP · Referenced in Section 7.4.2</i>	184
B.4.3	Ablation: Combining categories into a single model <i>Investigating policies to create a monolithic model merging multiple categories · Reference in Section 7.4.2</i>	184
B.5	Qualitative examples for Text-to-Image (T2I)-CompBench using STA- BLE DIFFUSION (SD)-2.1-base	186
B.5.1	Color prompts	186
B.5.2	Shape prompts	187
B.5.3	Texture prompts	188
B.5.4	2D-Spatial prompts	189
B.5.5	Non-Spatial prompts	190
B.5.6	Complex prompts	191
B.6	Qualitative examples for T2I-CompBench using SDXL	192
B.7	Fine-tuning with DiffusionDB dataset	193

B.7.1	Selecting images and BLIP-VQA prompts decomposition	
	<i>Discussing BLIP-VQA limitation when handling DiffusionDB prompts · Referenced in Section 7.5</i>	193
B.7.2	Qualitative examples for DiffusionDB	195
B.8	Qualitative analysis of Mutual Information (MI) as an alignment measure	
	<i>Expanded version of Figure 7.1 including all T2I-CompBench categories</i>	196
B.9	BLIP-VQA, HPS and MI score distributions	
	<i>Comparing distributions and MI rank · Related to results in Section 7.2</i>	197
	Bibliography	199

List of Figures

2.1	Example of a packet time series transformed into a flowpic representation for a randomly selected YouTube flow in the UCDAVIS19 dataset. Heatmaps are in a log scale normalized between the max and min value for each flowpic, with higher packets count values having darker shades (images better viewed digitally).	14
3.1	Input sample x shape and related notation.	24
3.2	Training batch creation policies.	29
3.3	Model architecture.	34
3.4	Augmentations rank and critical distance (G1).	36
3.5	Comparing performance improvement and training length.	37
3.6	Comparing <i>Replace</i> , <i>Inject</i> and <i>Pre-augment</i> batch creation policies (G2).	38
3.7	Investigating train, augmented and test samples relationships (G3).	40
3.8	Comparing original and augmented samples in the latent space (G3).	42
4.1	Contrastive learning principles.	53
4.2	Average confusion matrices for the 32×32 resolution across all experiments in Table 4.2.	59
4.3	Average 32×32 flowpic for each class across dataset partitions.	60
4.4	Critical distance plot of the accuracy obtained with each augmentation for the 32×32 and 64×64 resolutions. Augmentations joined by a horizontal line are not statistically different. The lower the ranking (closer to 1, the right side of the plot) the better the performance. Transformations highlighted in bold are selected as the best performing one in the REF-PAPER.	61
4.5	Transfer learning.	66
4.6	Meta-learning.	68
5.1	The discrete-time Markov chain of forward (reverse) diffusion process of generating a sample by slowly adding (removing) noise. (Ho et al., 2020a)	77
5.2	Links between the probability path, the velocity field and the flow.	85

5.3	Links between (tractable) conditional probability path and velocity field and their (intractable) marginal counterparts, for any condition Random Variable (RV) z	86
6.1	Mutual Information (MI) estimation results. Color indicates relative negative bias (red) and positive bias (blue).	101
7.1	Qualitative analysis of MI as an alignment measure (all metrics decrease from left to right). See also Appendix B.8.	106
7.2	Qualitative examples from Table 7.2 (same seed used for a given prompt).	115
7.3	Qualitative examples from Table 7.1 (same seed used for a given prompt). More examples in Appendix B.5.	116
7.4	Qualitative examples from Table 7.5 (same seed used for a given prompt). More examples in Appendix B.6.	117
7.5	Qualitative examples from Table 7.6 (same seed used for a given prompt). More examples in Appendix B.7.	117
7.6	Hyper-params search.	117
8.1	ImagenTime delay embedding.	137
8.2	ImagenTime pipeline.	137
8.3	Example of NetDiffusion: canny edges are extracted from the original Nprint image, serving as additional control signal when using ControlNet. Post-generation heuristics are applied to refine field details for protocol conformance.	138
8.4	DDPMS4 neural network architecture	139
8.5	Class imbalance comparison among Enterprise, CESNET-QUIC22 and CESNET-TLS22.	143
8.6	Example of NetDiffusion set of postprocessing rules applied sequentially (left column first, top to bottom). After selecting a real image from the dataset (top-left picture) and extracting the edges via a canny filter (2nd image in the left column), a sample is extracted (3rd image in the left column) and the rules are applied sequentially.	146
8.7	Example of NetDiffusion sampling (with ControlNet).	147
8.8	Qualitative comparison of packet time series generated from UNSW24.	148
8.9	Scaling in number of generated data - utility test	154
8.10	Scaling in number of generated data - augmentation test	156
8.11	Closest real training record statistic.	166
B.1	Web app screenshot example of the alignment metric comparison survey.	178
B.2	Weights merging: $\lambda \times \text{Color} + (1.0 - \lambda) \times \text{Shape}$	185
B.3	Qualitative examples for Color from Table 7.1 (same seed used for a given prompt).	186

B.4	Qualitative examples of the Shape category from Table 7.1 (same seed used for a given prompt).	187
B.5	Qualitative examples of the Texture category from Table 7.1 (same seed used for a given prompt).	188
B.6	Qualitative examples of the 2D-Spatial category from Table 7.1 (same seed used for a given prompt).	189
B.7	Qualitative examples of the Non-spatial category from Table 7.1 (same seed used for a given prompt).	190
B.8	Qualitative examples of the Complex category from Table 7.1 (same seed used for a given prompt).	191
B.9	Qualitative examples from Table 7.5 (same seed used for a given prompt).	192
B.10	Qualitative examples from Table 7.6 (same seed used for a given prompt).	195
B.11	Qualitative analysis of MI as an alignment measure (all metrics decrease from left to right).	196
B.12	CDF of alignment scores. Color reflect images rank based on MI. . . .	197

List of Tables

3.3	Sequence order augmentations.	25
3.1	Amplitude augmentations.	27
3.2	Masking augmentations	28
3.4	Summary of datasets properties.	31
3.5	Model architecture printout (MIRAGE-19-HALF, 20 classes)	32
3.6	Augmentations benchmark (G1)	35
3.7	Impact of class-weighted sampler on MIRAGE-19-HALF (G2)	39
3.8	Combining augmentations (G4)	43
4.1	Summary of datasets properties.	57
4.2	Comparing data augmentation functions in a supervised training. Values marked as “ours” correspond to the average accuracy across 15 modeling experiments and the related 95-th confidence intervals.	59
4.3	Comparing the fine-tuning performance when using different pairs of augmentations for pretraining (32×32 resolution, fine-tuning on 10 samples only).	63
4.4	Impact of dropout and SimCLR projection layer dimension on fine- tuning (32×32 only, with 10 samples for fine-tuning training).	63
4.5	Accuracy on 32×32 flowpic when enlarging training set (without dropout).	63
4.6	Computer vision literature summary.	66
4.7	Traffic classification literature summary.	66
4.8	Datasets summary.	70
4.9	Comparing transfer, meta- and contrastive learning in N -shots 4-ways classification (i.e. N training samples for each of the 4 target classes) (p.s. Query set is fixed as 15 samples per class)	71
7.1	T2I-CompBench alignment results (%) on images generated by Dif- fusion Models (DM). Gray highlighted style when MI-TUNE outper- forms all competitors; Grayed text for under-performing methods per-family; Green heatmaps show per-category absolute gains w.r.t. the base model.	114
7.2	T2I-CompBench alignment results (%) on images generated by Recti- fied Flow (RF) with CFG=4.5	114
7.3	Comparing image quality/variety scores.	118

7.4	FT set selection.	118
7.5	Alignment (%) using SDXL.	119
7.6	DiffusionDB.	119
8.1	State of the art methods for network traffic.	129
8.2	Diffusion models for tabular data: from TabDDPM to recent advances	130
8.3	Diffusion models for time series data.	132
8.4	Dataset properties.	141
8.5	Hyperparameters	144
8.6	XGBoost classifier Optuna search hyper params.	149
8.7	ResNet classifier hyper params.	149
8.8	Performance of ML and DL base classifiers.	149
8.9	Investigating XGBoost model size -vs- performance.	150
8.10	Utility test of generative models and hand-crafted augmentations. . .	152
8.11	Augmentation test.	154
8.12	Discriminative score (%)	159
8.13	Low-order distribution-level evaluation (%)	160
8.14	High-order sample-level evaluation (%)	163
B.1	User study about comparing alignment metrics.	178
B.2	Users study comparing alignment methods. Bold shows best performance; \diamond shows the best method per-family.	179
B.3	Training hyperparameters on SD-2.1-Base.	180
B.4	Training hyperparameters on SD-3.5-Medium.	181
B.5	HPS benchmark across multiple Stable Diffusion models extracted for HPS-v2 GitHub repo.	182
B.6	FT set selection.	183
B.7	BLIP-VQA alignment results on T2I-CompBench's Color and Shape categories varying size and composition of fine-tuning set. Results obtained using $R=1$	183
B.8	BLIP-VQA alignment results on T2I-CompBench's Color and Shape categories finetuning different portions of the model.	184
B.9	Benchmarking strategies for combining models.	185
B.10	DiffusionDB.	193

List of Abbreviations

AI	Artificial Intelligence
BYOL	Bootstrap Your Own Latent
CD	Critical Distance
CDF	Cumulative Distribution Function
CE	Cross-Entropy
CFG	Classifier-Free Guidance
CFM	Conditional Flow Matching
CNN	Convolutional Neural Network
CT	ConTrastive
CV	Computer Vision
DA	Data Augmentation
DDPM	Denoising Diffusion Probabilistic Model
DL	Deep Learning
DM	Diffusion Models
DPI	Deep Packet Inspection
FM	Flow Matching
FPE	Fokker-Planck Equation
FSL	Few-Shot Learning
GAN	Generative Adversarial Network
GASF	Gramian Angular Summation Field
HTTPS	Hypertext Transfer Protocol Secure
IAT	Inter-Arrival Time
ISDA	Implicit Semantic Data Augmentation
ISPs	Internet Service Providers
KL	Kullback-Leibler
KNN	K-Nearest Neighbors
LightGBM	Light Gradient Boosting Machine
MI	Mutual Information
ML	Machine Learning
MSE	Mean Squared Error
MSS	Maximum Segment Size
NLP	Natural Language Processing
NN	Neural Network
NT-Xent	normalized temperature-scaled cross entropy loss
ODE	Ordinary Differential Equation
OOD	Out-of-Distribution
PDF	Probability Density Function
QoE	Quality of Experience
regex	regular expression
RF	Rectified Flow
ROS	Random Over Sampling
RUS	Random Under Sampling

RV	Random Variable
S4	Structured State Space for Sequence Modeling
SAFA	Sample-Adaptive Feature Augmentation
SDE	Stochastic Differential Equation
SMOTE	Synthetic Minority Over-sampling TEchnique
SOTA	State-Of-The-Art
SVM	Support Vector Machines
T2I	Text-to-Image
TC	Traffic Classification
TCP	Transmission Control Protocol
TLS	Transport Layer Security
VAE	Variational AutoEncoder
XGBoost	Extreme Gradient Boosting

List of Symbols

a	Scalar
\mathbf{a}	Vector
\mathbf{A}	Matrix
\mathcal{D}_N	Dataset of N samples
\mathbf{X}	Dataset inputs (matrix of N rows, one for each instance)
\mathbf{Y}, \mathbf{y}	Dataset outputs (matrix, vector of N rows, one for each instance)
\mathbf{x}_i, x_i	i -th input instance (vector, scalar)
\mathbf{y}_i, y_i	i -th output instance (vector, scalar)
$\hat{\mathbf{y}}_i, \hat{y}_i$	i -th prediction of a model (vector, scalar)
\mathcal{L}	Loss
$f : \mathcal{X} \rightarrow \mathcal{Y}$	Function f with domain \mathcal{X} and range \mathcal{Y}
$f(x)$	Evaluation of function f at location x
T_j	Table j
$T_j[x]$	Bucket from table j at location x
\mathbb{N}	Natural numbers
\mathbb{Z}	Integer numbers
\mathbb{R}	Real numbers
$\text{Card}(S)$	Cardinality of the multiset S
$\mathbb{E}[\cdot]$	Expectation
$\mathcal{O}(\cdot)$	Big-O
$ \cdot $	Absolute value function
$\lfloor \cdot \rfloor$	Floor function
$\lceil \cdot \rceil$	Ceiling function
$\ \mathbf{a}\ _1$	L_1 norm of vector \mathbf{a}

*To all my mentors, with gratitude for your invaluable
guidance...*

Chapter 1

Introduction

1.1 Overview

The Internet forms the foundation of global digital connectivity, linking billions of devices and supporting a vast range of applications, from messaging and media streaming to autonomous systems and industrial automation. Underlying these services is the ongoing transmission of data packets, commonly referred to as network traffic. Understanding the nature of this traffic is essential for managing modern networks effectively, making network TC, which identifies the application that has generated a given traffic flow, a fundamental step in network operation. For instance, TC enables intelligent resource allocation, adaptive congestion control, and low-latency service guarantees for latency-sensitive applications such as online gaming and real-time video conferencing. It also plays a key role in policy enforcement and service differentiation, ensuring that diverse applications receive the quality of service they require.

Classifying network traffic in modern networks presents several key challenges. The increasing use of encryption, which hides payload content, limits the effectiveness of traditional deep packet inspection techniques. Combined with the growing volume and diversity of traffic, this makes it difficult to maintain static rule sets or rely on protocol signatures. In response, ML has gained attention as a data-driven approach that extracts patterns from limited information and adapts to changing conditions, enabling efficient, automated TC. Still, machine learning-based classifiers face their own challenges. Real-time processing requirements in high-speed environments impose strict constraints on latency and memory usage. Moreover, traffic patterns evolve rapidly due to distribution shifts in existing applications over time and the continuous emergence of new applications, requiring classification systems to be both robust and scalable. A further limitation in this research area is the lack of large, open-source datasets. Unlike in computer vision or natural language processing, publicly available labeled traffic data is scarce, due to privacy concerns, legal restrictions, and the complexity of capturing representative network traffic at scale.

The ultimate goal of this dissertation is to study how to improve TC in a data-drive fashion. In particular, recent advances ML literature related to data augmentation, contrastive learning, and generative modeling have been shown to be effective in scenarios suffering from data scarcity and distribution shifts, i.e., issues that are commonly found also in TC. Building on ideas from this body of literature, this study investigates if/how to take advantage of synthetic data for improving TC. Specifically, our work begins with hand-crafted augmentations, then shifts toward understanding diffusion-based generative models in a more established domain of Computer Vision (CV), and ultimately returns to networking with an integrated generative approach. The contributions of this dissertation are threefold:

- Exploration of hand-crafted augmentation and contrastive learning to TC task: we benchmark 18 hand-crafted augmentations in supervised learning, and then apply them within contrastive learning frameworks to address more challenging scenarios with limited labeled data for target classes, demonstrating their effectiveness in enhancing classification performance and generalization.
- Estimation of MI and its application in improving text-to-image alignment: to better understand diffusion models, we study conditional generation in the text-to-image domain. We propose novel MI estimators based on pretrained diffusion and rectified flow models, and apply them in a self-supervised fine-tuning strategy to enhance alignment between generated samples and conditioning inputs, achieving improved alignment for text-to-image synthesis without relying on external models or human annotations.
- Development of generative modeling for network traffic: we return to the original problem of traffic modeling by focusing on generative approaches to enable more automated and scalable solutions. To address the lack of standardized benchmarks, we build a framework that includes datasets, preprocessing, baseline methods, and evaluation protocols covering both fidelity and downstream classification utility. We also develop a diffusion model for packet series that achieves better performance than existing generative models.

By addressing these areas, the dissertation advances TC under data scarcity, improves alignment in conditional generative models, and provides benchmarks for generative modeling of traffic — laying a foundation for more robust and adaptable ML systems in networking contexts.

1.2 Hand-crafted augmentations and contrastive learning for traffic classification

In the first part of this dissertation, we focus on introducing hand crafted augmentations and contrastive learning for TC studies. Specifically, after discussing the background of TC task, we present the use of hand crafted augmentations in traditional

supervised classification, and its combination with contrastive learning in various challenging few-shot scenarios.

Chapter 2 provides background on TC. After framing the importance of this task, the discussion outlines the three main phases of classification: input encoding, decision process, and accuracy evaluation, with a focus on the first two. First, commonly used input representations are reviewed, including payloads, packet time series, and flowpic, emphasizing the need for robustness to encryption and support for early classification. Then, decision methods are reviewed in order of their evolution, from rule-based approaches like Deep Packet Inspection (DPI) to classical ML and modern Deep Learning (DL) models, discussing their respective strengths and limitations. Overall, this chapter lays the groundwork for the more advanced methods explored afterwards.

Chapter 3 builds on the foundational concepts and methods of TC introduced earlier and shifts the focus from model design to the largely unexplored potential of the data itself. Specifically, the chapter focuses on the application of hand-crafted Data Augmentation (DA)s to enhance the classifier's performance in traditional supervised classification task, taking packet time series as input. We conduct a comprehensive empirical study on 18 hand-crafted augmentation techniques grouped into three categories: amplitude, masking, and sequence transformations. We evaluate their effectiveness across multiple datasets, with emphasis on performance improvements, class imbalance mitigation, latent space geometry, and augmentation combination strategies. Results indicate that, while no single augmentation is universally optimal, those that provide moderate but meaningful variation tend to perform best, and injecting augmented samples into the real data mini-batch is the most effective way to apply them. As a whole, the chapter provides a systematic benchmark of hand-crafted DA, presenting both their individual and combined effects on classification performance across several TC datasets, offering valuable insights into the design space of DA and motivates the application of DA in more challenging scenarios in the following chapter.

Chapter 4 addresses Few-Shot Learning (FSL) challenges using DA together with contrastive learning methods, given the success of DA in traditional supervised classification. Starting with clarifying the task definition, FSL aims to build effective classifiers from a small number of labeled training samples for the target classes, which is particularly useful in network environments where data collection and annotation is at high cost. In this context, this chapter examines two specific FSL cases: (i) pretraining on a large collection of unlabeled samples and fine-tuning on a few labeled samples from the target classes, and (ii) pretraining on labeled samples from non-target classes, followed by fine-tuning on the target classes. Regarding the methodologies, contrastive learning that builds on data augmentation plays an important role in both cases by explicitly enforcing geometric properties in the latent

space. Specifically, contrastive learning optimizes the feature representation by maximizing the distance between samples from different classes (namely negative pairs) and minimizing the distance between samples from the same class (namely positive pairs) in the latent space. After introducing the mathematical foundation of contrastive learning, a series of experiments for the first FSL case with flowpic input reveals that Change RTT and Time Shift are the best augmentations and confirms their effectiveness in self-supervised contrastive pretraining. Furthermore, in the second FSL case with packet time series input, supervised contrastive learning also outperforms other monolithic- and meta-learning methods. Overall, these findings highlight how contrastive learning, in conjunction with DA, facilitates robust TC in data-scarce regime.

Taken together, these chapters demonstrate the value of DA and contrastive learning in building data-efficient, generalizable TC classifiers.

Publications

- **Chao Wang**, Alessandro Finamore, Pietro Michiardi, Massimo Gallo, Dario Rossi. “Data Augmentation for Traffic Classification”. In *Passive and Active Measurement Conference (PAM)*, 2024.
- Alessandro Finamore, **Chao Wang**, Jonatan Krolkowski, Jose M Navarro, Fuxing Chen, Dario Rossi. “Replication: Contrastive Learning and Data Augmentation in Traffic Classification Using a Flowpic Input Representation”. In *Proceedings of ACM on Internet Measurement Conference (IMC)*, 2023.
- Idio Guarino, **Chao Wang**, Alessandro Finamore, Antonio Pescape, Dario Rossi. “Many or Few Samples? Comparing Transfer, Contrastive and Meta-Learning in Encrypted Traffic Classification”. In *Network Traffic Measurement and Analysis Conference (TMA)*, 2023.

1.3 Mutual information for conditional generation

In the previous part, we tackled TC using hand-crafted data augmentations. While effective, these methods require manual design and extensive tuning of intensity. This motivates the use of generative models to automatically produce realistic, diverse samples for augmentation. Among them, diffusion models are particularly promising for their stability during training and ability to generate high-quality, diverse samples. Before applying diffusion models to traffic packet series, we first turn to a more mature ML domain: text-to-image generation, where state-of-the-art open-source diffusion models such as Stable Diffusion have been extensively studied and evaluated. This setting provides a well-defined conditioning framework, supported by rich datasets and established tools for evaluating the alignment between inputs and generated samples. In the second part of this dissertation, by studying how to enhance this alignment through MI, we build the foundation for more controllable

generative models. This investigation also provides deeper insight into the underlying mechanisms of diffusion models, which helps guide their application in Part 3.

Chapter 5 provides an introduction to the two prominent types of generative models that we focus on in this part: DM and RF. DM progressively adds noise to data and then learns to reverse this process to generate samples, while RF models, a flow-based approach, minimize kinetic energy through flow matching techniques. By introducing the core mathematical principles (e.g., the processes' Stochastic Differential Equation (SDE)/Ordinary Differential Equation (ODE), training objective, and conditional generation), the chapter provides the foundation for the application of these models in advanced generative tasks in the following chapters.

Chapter 6 delves into the estimation of MI, building upon the concepts introduced in the previous chapter. MI is a measure of the mutual dependence between two RVs, and in this chapter, we derive two innovative estimators of point-wise MI using conditional generative models — one using the discrete time DMs, the other using the RF models. Experimental evaluation on MI estimation benchmark demonstrates the effectiveness of the RF-based MI estimator. The formulas and results from this chapter lay the groundwork for using MI to enhance the alignment between generated data and the conditioning variables, such as images and text prompts in Text-to-Image (T2I) generation, which will be explored in subsequent chapter.

Chapter 7 applies the MI estimator from the previous chapter to address the key challenge of alignment in T2I conditional generation with DMs and RF models. DMs and RF models for T2I conditional generation have recently achieved tremendous success. Yet, aligning these models with user's intentions still involves a laborious trial-and-error process, and this challenging alignment problem has attracted considerable attention from the research community. In this chapter, instead of relying on fine-grained linguistic analyses of prompts, human annotation, or auxiliary vision-language models, we use MI to guide model alignment. Specifically, our method uses self-supervised fine-tuning and relies on a point-wise MI estimation between prompts and images to create a synthetic fine-tuning set for improving model alignment. Our analysis indicates that our method is superior to the State-Of-The-Art (SOTA), yet it only requires the pre-trained denoising network of the T2I model itself to estimate MI, and a simple fine-tuning strategy that improves alignment while maintaining image quality.

Publications

- **Chao Wang**, Giulio Franzese, Alessandro Finamore, Massimo Gallo, Pietro Michiardi. "Information Theoretic Text-to-Image Alignment". In *International Conference on Learning Representations (ICLR)*, 2025.

- **Chao Wang**, Giulio Franzese, Alessandro Finamore, Pietro Michiardi. “RFMI: Estimating Mutual Information on Rectified Flow for Text-to-Image Alignment”. In *International Conference on Learning Representations Workshop on Deep Generative Models: Theory, Principle, and Efficacy (ICLR DeLTa Workshop)*, 2025.

1.4 Generative data augmentation for traffic classification

In Part 1, we explored hand-crafted augmentations for packet time series, which proved empirically effective for TC. However, their performance is highly dataset-dependent and requires extensive hyperparameter tuning. To address these limitations, we turned to insights from Part 2, where conditional diffusion models demonstrated the ability to generate diverse, realistic outputs that are semantically aligned with input conditions, as shown in the text-to-image domain. This motivated us to ask: can class-conditional diffusion models be adapted to model the distribution of packet series such that the generated data maintain fidelity and further introduce meaningful variation for downstream classification? In Part 3, we present an initial investigation into this direction.

Chapter 8 builds a systematic framework for evaluating generative modeling of packet series and proposes a diffusion model, DDPMS4, that surpasses baseline generative models in both fidelity and downstream utility. Motivated by the inconsistency in datasets and input representations across the networking literature, our first goal was to build a benchmark with curated datasets, unified preprocessing, and baseline implementations. From a ML perspective, we observed that most generative models in tabular and time series domains overlook augmentation utility in their evaluations, leading to our second goal: establishing an evaluation protocol encompassing class-agnostic fidelity, class-conditional fidelity, and augmentation utility. We benchmark three generative models and three hand-crafted augmentations using XGBoost for downstream classification. While DDPMS4 performs best among the generative models, it still underperforms compared to training solely on real data, due to limited coverage of low-density regions. We also find that simply generating more samples is not effective for augmentation, as the diffusion model is constrained by the information present in the training data, and the score function approximation is inherently imperfect. Therefore, generative models must be redesigned to go beyond fidelity and explicitly promote extra diversity that benefits downstream tasks. Finally, although hand-crafted augmentations can be effective, simple metrics like L2 distance fail to capture semantic similarity in packet series, highlighting the need for more expressive measures of class-relevant structure.

Chapter 9 concludes our work and outlines future research directions.

Publications

-
- **Chao Wang**, Alessandro Finamore, Pietro Michiardi, Massimo Gallo, Dario Rossi. “Toward Generative Data Augmentation for Traffic Classification”. In *Proceedings of International Conference on emerging Networking EXperiments and Technologies Student Workshop (CoNEXT SW)*, 2023.

Part I

Hand-crafted augmentation and contrastive learning for traffic classification

Chapter 2

Introduction to traffic classification

The classification object of TC is a *flow* – a directional communication in which a host A sends traffic to a specific host B ; formally this means all the packets sharing the tuple $(IP_A, IP_B, L4_{\text{port } A}, L4_{\text{port } B}, L4_{\text{protocol}})$. Given a flow, TC is the process of identifying the application that has generated it.

In the nowadays Internet, TC helps to understand both the users' behavior and the network status, which is valuable for Internet Service Providers (ISPs) to optimize services in multiple use-cases. For instance, companies may rely on firewalls to block some applications (e.g., use of social networks in working environments), or some latency sensitive traffic (e.g., gaming and video meetings) may need to be prioritized to improve the Quality of Experience (QoE) perceived by the users – real-time TC is needed to be able to enforce such desired policies. Furthermore, TC also supports anomaly detection, such as identifying malware or DDoS attacks, and can help spot illegal activities like content piracy. Overall, TC is crucial for understanding and managing networks.

Given its importance, TC is a long investigated topic by both industry and academia with seminal works dating back nearly two decades ago (Moore and Zuev, 2005) which have been instrumental for bringing ML tools into networks operation and management. Since then, the TC field has been flourishing with literature and it is regularly surveyed (Nguyen and Armitage, 2008; Pacheco et al., 2018).

In general, when TC is operated via ML, the classification process involves the three common phases of an ML classifier: *input encoding*, where we transform the flow into a data representation that is usable for classifier as input and characterizes flows belonging to the same application; *decision process*, where a classifier model is fitted on the set of input data at training and is used to predict the application as output at inference; *accuracy evaluation*, where the effectiveness of the classifier model is assessed with metrics like accuracy.

In the following, we first present two popular input data representations and their variants (Sec. 2.1), next we introduce the general mechanism behind three main-stream classification method in the order in which they were proposed (Sec. 2.2).

2.1 Input data representations

There are two popular choices for classifiers' input data representation: raw payloads, and statistical profiling of flows dynamics.

2.1.1 Payload

A network packet consists of a set of stacked headers and an application layer payload, where the headers carry control information commonly standardized in RFCs (such as source and destination addresses and protocol identifiers) and the payload contains the actual application data being transmitted. Since the payload holds content that can reveal the type of application, it is potentially a very informative input representation for TC.

Several recent works, such as ET-BERT (Lin et al., 2022) and YaTC (Zhao et al., 2023), have successfully leveraged payload information by applying Natural Language Processing (NLP)-inspired models to classify traffic based on packet content. These methods demonstrate the potential of deep models to capture fine-grained semantics embedded in payloads.

However, relying on the payload presents notable challenges. First, payloads can span thousands of bytes, making real-time inference computationally expensive on high-speed links. More critically, payload-based methods become ineffective when the traffic is encrypted, as the payload's content cannot be inspected to discern useful information. These limitations have motivated the networking community to explore alternative, payload-agnostic representations. In this part, we adopt this direction and focus on features that remain informative even when the payload is encrypted.

2.1.2 Packet time series

A widely adopted payload-agnostic input data representation is packet time series, which includes the raw properties of the first few packets in a flow. Commonly used packet properties include packet size, direction, and Inter-Arrival Time (IAT). Unlike statistical summaries such as means or quantiles, a packet time series consists of the raw per-packet values themselves, preserving the fine-grained temporal and structural characteristics of the early flow.

This representation offers two key advantages. First, it enables early classification by focusing only on the initial portion of a flow, making it well-suited for real-time applications where decisions must be made before the flow completes. Second, it focuses on the initial segment of the flow, where handshake dynamics and protocol behaviors typically reveal the most discriminative features for identifying the application.

In addition to using the packet series directly as sequential input, some works further transform them into 2D images (namely flowpic) to leverage DL architectures originally designed for CV, such as Convolutional Neural Networks (CNNs). Since we adopt this image-based format in [Chapter 4](#), we provide a brief introduction to it here.

Flowpic. The flowpic representation was originally introduced by Shavitt and Shavitt, [2019](#), and then adopted by Horowicz et al., [2022](#). In [Figure 2.1](#) we show a YouTube flow (extracted randomly from the UCDAVIS19 dataset (Rezaei and Liu, [2019](#))) as well as its related flowpic at different resolutions. The left most plot shows the packet time series. Notice the expected bursty nature typical of video streaming services. To compute a flowpic, Horowicz et al., [2022](#) use only the first 15s of the time series. Specifically, both the 15s and the packets size range (0-1500) are split into bins based on the resolution of the target flowpic.¹ For instance a 32×32 flowpic leads to 469.8ms time bins and 46B packet size bins. Then, the count of the packets occurring in each time window are tallied based on the defined packet size bins. In other words, each time window provides a frequency histogram of the packet sizes, and by vertically stacking all the histograms we obtain a "picture" of the flow dynamics. For instance, at the 32×32 resolution, the vertical stripes match the packet bursts of the original time series. This sort of patterns make the flowpic representation appealing for CNN-based DL architectures as convolutional layers are explicitly designed to extract features to detect such patterns. Yet, the higher the flowpic resolution, the sparser the representation, and the higher their computational process. While flowpic was introduced with a 1500×1500 resolution, in Horowicz et al., [2022](#) this is compared against a 32×32 resolution, i.e., a mini-flowpic. Despite being well suited for CV-related methods, the flowpic representation is not a mainstream choice for TC as it requires to observe multiple seconds of traffic. This can enforce a late/post-mortem classification (i.e., after the flow ends), which, while still useful for monitoring, might not fit network management needs—prioritization, scheduling and shaping benefit from classification after the first few packets, so waiting for multiple seconds to take action can be sub-optimal. Conversely, original packet time series and payload bytes do not face this limitation.

Last, we note that although payload is not interpretable in the encrypted case, in unencrypted case, since both time series and payload input enable early classification, they can also be combined in “multi-modal” architectures (Aceto et al., [2019a](#); Akbari et al., [2021a](#); Luxemburk and Čejka, [2023a](#)) that have become popular in networking and other fields—rather than selecting either one representation or the other, a DL model can be designed to learn from different input formats at the same time.

¹Traffic directionality is not considered when composing the flowpic in (Horowicz et al., [2022](#)) although the representation could be reformulated to take it into account.

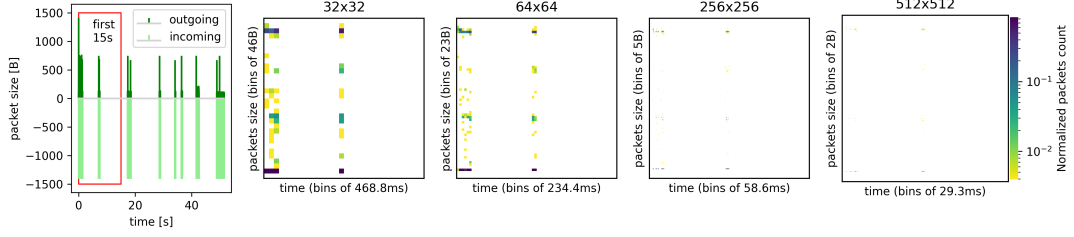


FIGURE 2.1: Example of a packet time series transformed into a flowpic representation for a randomly selected YouTube flow in the UCDAVIS19 dataset. Heatmaps are in a log scale normalized between the max and min value for each flowpic, with higher packets count values having darker shades (images better viewed digitally).

2.2 Classifiers

2.2.1 Rule-based classifiers

Traditional techniques classify traffic based on rules applied to the content of network packets. One of the most used technologies is DPI, which examines the existence of specific keywords within the packet payload using pattern matching and regular expressions – e.g., if a packet’s payload begins with the string "BitTorrent", it is highly likely that the traffic originates from the BitTorrent application. In other words, given the input data representation being the packet’s payload, the decision process involves sequentially checking some pre-defined rules (namely regular expression (regex)) about its patterns, signatures, and fingerprints (Piet et al., 2023), until a match is found. However, despite their popularity, DPI is nowadays more and more challenged by the over increasing adoption of encrypted protocols (e.g. Hypertext Transfer Protocol Secure (HTTPS) and Transport Layer Security (TLS)) as the payload in this case corresponds to a sequence of random number and by the policies imposed by regulatory bodies to protect user privacy (e.g. GDPR). Moreover, DPI is difficult to automate as it requires large domain knowledge and manual effort to construct effective classification rules.

2.2.2 Machine learning classifiers

The search for alternatives to DPI already started two decades ago with early proposals of application of traditional ML classifiers. Depending on the availability of associated ground-truth application labels in the training dataset, these algorithms are commonly divided between supervised and unsupervised methods. Members of the former class are for example Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Light Gradient Boosting Machine (LightGBM) and Extreme Gradient Boosting (XGBoost), while the latter approach includes notably clustering and its variants. Although these ML algorithms are more automated than DPI, their trained models are not flexible for continuous monitoring to catch up with the updates of applications, and have no intermediate feature space for the analysis of Out-of-Distribution

(OOD) detection.

2.2.3 Deep learning classifiers

Thanks to the rapid development of Artificial Intelligence (AI), more recent TC proposals focus on DL with an emphasis on exploring/adapting training techniques, algorithms and Neural Network (NN) architectures proven to be successful in traditional DL fields like CV and NLP, and this thesis falls within such research area.

A NN classifier is a function $\varphi : \mathbf{x} \in \mathcal{X} \rightarrow y \in \mathcal{Y}$ mapping an input \mathbf{x} to its label y . It can be regarded as a composition of two functions $\varphi(\mathbf{x}) = h(f(\mathbf{x})) = y$: a base encoder network $f(\cdot)$ and a simple classifier head $h(\cdot)$ (e.g., a linear layer or KNN). In other words, an input sample \mathbf{x} is first encoded into an intermediate space, namely the latent space, where different classes' latent representations $f(\mathbf{x}) = \mathbf{z}$ are expected to occupy different regions. The better such separation, the easier is for the classifier head $h(\mathbf{z}) = y$ to identify the correct label.

With the NN architecture divided into two parts, its training algorithms can be categorized into two paradigms as well: single-stage end-to-end training of $\varphi = h \circ f$, or two-stages approach consisting of pretraining the encoder f followed by training the classifier head h if needed.

The most common approach in a *supervised* setting is a single-stage training, with a classifier head being a linear layer, and both the encoder and the head are jointly trained in a supervised manner on a labeled dataset. By minimizing the difference between the classifier's predicted outputs and the ground-truth labels quantified typically by Cross-Entropy (CE) loss, the model learns an end-to-end mapping function $h \circ f$ from the input data to the output label.

In contrast, two-stages approaches are common for *transfer learning* settings. First we train an encoder f to yield good latent representations, for which a training loss optimizing the latent representation directly is required. One popular choice is Contrastive (CT) loss, which explicitly regularizes latent space geometry by penalizing cases where samples from the same class are far apart and samples from different classes are too close. Then, after the encoder f being trained to identify similarities and clusters within the input data, a linear classifier h is trained on top of the base encoder with CE loss or a KNN is applied on the latent space, in order to output the predicted class label.

Last, we note that depending on the properties of training set and test set, there are multiple variants of classification tasks that are meaningful in TC. For instance, in the traditional classification task, no data shift is expected between training set and test set. However, in reality, training and managing classifiers for networks face the "infinite loop" of collecting new data and re-train models to keep them up to date.

The key to break this cycle resides in (i) training more generalized models and (ii) adapting them to new scenarios by means of little-to-no extra data. This calls for DL techniques that enable the reuse of knowledge from a source dataset to address a target dataset with limited labeled data and distribution shift — such as sharing class labels but differing in feature patterns due to variations in time or location, or having entirely different class labels.

Specifically, in this thesis, we focus on data generation considering both manually designed augmentations as well as state of the art generative models. In fact, even if many studies investigated data augmentations for CV and time series (e.g., in the medical field), reusing such methods is not trivial for TC as data suffers from two extra undesirable restrictions: input samples are short—traditionally, they are time series of the first N packets of a flow (e.g., the first 10 packet sizes and inter arrival times) and are semantically weak — interpreting packet time series is less obvious than interpreting electrocardiograms.

Chapter 3

Benchmark of hand-crafted data augmentation

3.1 Introduction

As outlined in Sec. 2.2.3, a renewed thrust in addressing TC via data-driven modeling is fueled today by the rise of DL. Despite the existing literature, we argue that *opportunities laying in the data itself are still unexplored* based on three observations. First, CV and NLP methods usually leverage “cheap” DA strategies (e.g., image rotation or synonym replacement) to complement training data by increasing samples variety. Empirical studies show that this leads to improved classification accuracy. Yet to the best of our knowledge, only a handful of TC studies considered DA (Horowicz et al., 2022; Rezaei and Liu, 2019; Xie et al., 2023b) and multiple aspects of DA design space remain unexplored. In addition, network traffic datasets are imbalanced due to the natural skew of app/service popularity and traffic dynamics. In turn, this calls for training strategies focused on improving classification performance for classes with few examples. However, the interplay between imbalance and model performance is typically ignored in TC literature. Finally, the pursuit of better model generalization and robustness necessitates large-scale datasets with high-quality labeling resulting in expensive data collection processes. In this context, the extent to which DA can alleviate this burden remains unexplored.

In this chapter, we fill these gaps by first providing a comprehensive evaluation of “hand-crafted” augmentations — transformations designed based on domain knowledge — applied to packets time series typically used as input in TC. Given the broad design space, we defined research goals across multiple dimensions. First of all, we selected a large pool of 18 augmentations across 3 families (amplitude, masking, and sequence) which we benchmark when used in isolation or combined (e.g., via stacking or ensembling). Augmented time series are combined with original training data via different batching policies (e.g., replacing training data with augmentation, adding augmented data to each training step, or pre-augmenting the dataset before training). We also included scenarios where imbalanced datasets are

re-balanced during training to give more importance to minority classes. Last, we dissected augmentations performance by exploring their geometry in the classifiers latent space to pinpoint root causes driving performance. Our experimental campaigns were carried over 2 mid-sized public datasets, namely MIRAGE-19-HALF and MIRAGE-22 (up to 20 classes, 64k flows), and a larger private dataset Enterprise-100 (100 classes, 2.9M flows).

We summarize our major findings as follows:

- We confirm that augmentations improve performance (up to +4.4% weighted F1) and expanding training batches during training (i.e., the Injection policy) is the most effective policy to introduce augmentations. Yet, improvements are dataset dependent and not necessarily linearly related to dataset size or number of classes to model;
- Sequence ordering and masking are more effective augmentation families for TC tasks. Yet, no single augmentation is found consistently superior across datasets, nor domain knowledge suffice to craft effective augmentations, i.e., the quest for effective augmentations is an intrinsic trial-and-error process;
- Effective augmentations introduce good sample variety, i.e., they synthesize samples that are neither too close nor too far from the original training data.

To the best of our knowledge, a broad and systematic study of hand-crafted DA techniques in TC as the one performed in our study is unprecedented. Ultimately, our analysis confirms that DA is currently suffering from a single pain point—exploring the design space via brute force. However, our results suggest a possible road map to achieve better augmentations via generative models which might render obsolete the use of brute force.

In the remainder, we start by introducing DA basic concepts and reviewing relevant ML and TC literature (Sec. 3.2). We then introduce and discuss our research goals (Sec. 3.3) and the method used to address them (Sec. 3.4). Finally, we present our experimental setting (Sec. 3.5) and results (Sec. 3.6) before closing with final remarks (Sec. 3.7).

3.2 Related work

DA consists in adding synthetic samples (typically derived from real ones) to the training set to increase its variety. DA has been popularized across many ML disciplines (Mumuni and Mumuni, 2022; Shorten and Khoshgoftaar, 2019; Wen et al., 2021) with a large number of variants which we can be broadly grouped into two categories (Mumuni and Mumuni, 2022): hand-crafted DA and data synthesis. Hand-crafted DA (also known as data transformations) involves creating new samples by

applying predefined rules to existing samples. Instead, data synthesis relates to generating new samples via generative models, e.g., Variational AutoEncoder (VAE), Generative Adversarial Network (GAN), DM, etc., trained on existing and typically large datasets.

In this section, we overview the existing DA literature with an emphasis on hand-crafted DA and methods closer to the scope of this chapter. We begin by introducing relevant CV and time series ML literature. Then, we review TC literature using DA and close with a discussion about general design principles/requirements that we used for defining our research goals outlined in Sec. 3.3.

3.2.1 Data augmentation in traditional machine learning tasks

As explained in Sec. 2.2.3, supervised learning of good NN classifiers corresponds to discover a good function $\varphi(\cdot)$ based on a large labeled training set that includes high variety. As these are notoriously difficult to share and correct labeling is costly, enlarging the labeled dataset in supervised learning or reducing dependency from labels by self-supervised learning with the help of DA is particularly appealing.

When performing DA, the training set is expanded by adding new samples $\mathbf{x}' = \text{Aug}(\mathbf{x})$ created by altering original samples \mathbf{x} —these transformations act directly in the input space \mathcal{X} and the additional synthetic samples contribute in defining $\varphi(\cdot)$ as much as the original ones. It follows that having a comprehensive understanding of samples/classes properties and their contribution to models training is beneficial for designing *effective* augmentations, i.e., transformations enabling higher classification performance. Beside operating in the input space, DL models offer also a latent space. It follows that this design enables a second form of augmentations based on altering samples in the latent space rather than in the input space. Last, differently from hand-crafted DA, generative models aims to learn the training set data distribution. In this way generating new synthetic data corresponds to sampling from the learned distribution. In the following we expand on each of these three methodologies.

Input space transformations. In traditional ML, Synthetic Minority Over-sampling TEchnique (SMOTE) (Chawla et al., 2002) is a popular augmentation technique. This approach generates new samples by interpolating the nearest neighbors of a given training sample. To address class imbalance, SMOTE is often employed with a sampling mechanism that prioritizes minority classes (Han et al., 2005; He et al., 2008).

In CV, several image transformations have been proposed to improve samples variety while preserving classes semantics. These transformations operate on colors (e.g., contrast and brightness changes, gray scaling) and geometry (e.g., rotation, flipping, and zooming), or via filters (e.g., blurring with Gaussian kernel) and masks

(e.g., randomly set to zero a patch of pixels). Furthermore, transformations like CutMix (Yun et al., 2019) and Mixup (Zhang et al., 2018) not only increase samples variety but also increase *classes variety* by creating synthetic classes from a linear combination of existing ones. The rationale behind this approach is that by introducing new artificial classes sharing similarities with the true classes the classification task becomes intentionally more complex, thereby pushing the training process to extract better data representations. Empirical validations of DA techniques in CV have consistently demonstrated their effectiveness across a diverse range of datasets, tasks, and training paradigms (Chen et al., 2020; He et al., 2015; Redmon et al., 2016). As a result, DA has become a ubiquitous component in the CV models training pipelines.

Considering time series instead, input transformations can either modify data *amplitude* (e.g., additive Gaussian noise) or manipulate *time* (e.g., composing new time series by combining different segments of existing ones). Similarly to CV, the research community has provided empirical evidence supporting the effectiveness of these transformations in biobehavioral (Yang et al., 2022a) and health (Yu and Sano, 2022) domains. However, contrarily to CV, these transformations are less diverse and have been less widely adopted, possibly due to the stronger reliance on domain knowledge—an amplitude change on an electrocardiogram can be more difficult to properly tune compared to simply rotating an image.

Latent space transformations. Differently from traditional ML, DL models offer the ability to shape the feature extractor to create more “abstract” features. For example, Implicit Semantic Data Augmentation (ISDA) (Wang et al., 2020d) first computes class-conditional covariance matrices based on intra-class feature variety; then, it augments features by translating real features along random directions sampled from a Gaussian distribution defined by the class-conditional covariance matrix. To avoid computational inefficiencies caused by explicitly augmenting each sample many times, ISDA computes an upper bound of the expected cross entropy loss on an enlarged feature set and takes this upper bound as the new loss function. Based on ISDA, and focused on data imbalance, Sample-Adaptive Feature Augmentation (SAFA) (Hong et al., 2022) extracts transferable features from the majority classes and translates features from the minority classes in accordance with the extracted semantic directions for augmentation.

Generative models. In addition to traditional hand-crafted DA techniques, generative models offer an alternative solution to generate samples variety. For instance, (Burg et al., 2023; Trabucco et al., 2023) use a multi-modal diffusion model trained on an Internet-scale dataset composed of (image, text) pairs. Then, the model is used to synthesize new samples—text prompts tailored to specific downstream classification tasks are used as conditioning signal to create task-specific samples—to enlarge the training set for a classification task. While these types of generative models can

provide high-quality samples variety, their design and application still requires a considerable amount of domain knowledge to be effective.

3.2.2 Data augmentation in traffic classification

TC tasks usually rely on either packet time series (e.g., packet size, direction, IAT, etc., of the first 10-30 packets of a flow) or payload bytes (e.g., the first 784 bytes of a flow, possibly gathered by concatenating payload across different packets) arranged as 2D matrices. Recent literature also considers combining both input types into multi-modal architectures (Akbari et al., 2021b; Aceto et al., 2019b; Luxemburk et al., 2023a).

Such input representations and datasets exhibit three notable distinctions when compared to datasets from other ML/DL disciplines. First, TC datasets show *significant class imbalance*—this is a “native” property of network traffic as different applications enjoy different popularity and traffic dynamics while, for instance, many CV datasets are balanced. Second, TC input representation is typically “*small*” to adhere to desirable system design properties—network traffic should be (i) *early classified*, i.e., the application associated to a flow should be identified within the first few packets of a flow, and (ii) computational/memory resources required to represent a flow should be minimal as an in-network TC system needs to cope with hundreds of thousands of flow per second. Last, TC input data has *weak semantics*—the underlying application protocols (which may or not be known a priori) may not be easy to interpret even for domain experts when visually inspecting packet time series.

Hand-crafted DA. The combination of the above observations leads to have only a handful of studies adopting DA in TC literature. Rezaei and Liu, 2019 created synthetic packet time series input by sampling multiple short sequences across the duration of a complete flow based on different policies (e.g., selecting one packet every N from a random starting point); hence, from one flow they obtain multiple “subflows” which semantically correspond to a coarser-grained “view” of the original flow. Horowicz et al., 2022 instead focused on a *flowpic* input representation—a 2D summary of the evolution of packets size throughout the first 15s of a flow, and considered DA techniques applied to either flowpics (e.g., rotation) or to the packets time series (e.g., altering inter-arrival times) from which the flowpics are then computed. While both studies show the benefit of DA, these strategies violate the early classification principle as they both consider multiple seconds of traffic, thus they are better suited for post-mortem analysis only. Conversely, Xie et al. (Xie et al., 2023a; Xie et al., 2023b) recently proposed some packet series hand-crafted DA to tackle data shifts arising when applying a model on network traffic gathered from networks different from the ones used to collect the training dataset. Specifically,

inspired by Transmission Control Protocol (TCP) protocol dynamics, authors proposed five packets time series augmentations (e.g., to mimic a packet lost/retransmission one can replicate a value at a later position in the time series) showing that they help to mitigate data shifts. Yet, differently from (Horowicz et al., 2022; Rezaei and Liu, 2019), the study in (Xie et al., 2023b) lacks from an ablation of each individual augmentation’s performance.

Generative models. Last, (Wang et al., 2020a; Wang et al., 2019; Yin et al., 2018) investigate augmentations based on GAN methods when using payload bytes as input for intrusion detection scenarios, i.e., a very special case of TC where the classification task is binary. More recently, (Jiang et al., 2023) compared GAN and DM for generating raw payload bytes traces while (Sivaroopan et al., 2023a; Sivaroopan et al., 2023b) instead leveraged GAN or DM to generate 2D representations (namely Gramian Angular Summation Field (GASF)) of longer traffic flow signals for downstream traffic fingerprinting, anomaly detection, and TC.

3.2.3 Design space

Search space. Independently from the methodology and application discipline, DA performance can only be assessed via empirical studies, i.e., results are bound to the task definition, the datasets used, and the input representation format. Moreover, to find an efficient strategy one should consider an array of options, each likely subject to a different parametrization. In the case of hand-crafted DA, one can also opt for using *stacking* (i.e., applying a sequence of transformations) or *ensembling* (i.e., applying augmentations by selecting from a pool of candidates according to some sampling logic)—an exhaustive grid search is unfeasible given the large search space. Besides following guidelines to reduce the number of options (Cubuk et al., 2019b), some studies suggest the use of reinforcement learning to guide the search space exploration (Cubuk et al., 2019a). Yet, no standard practice has emerged.

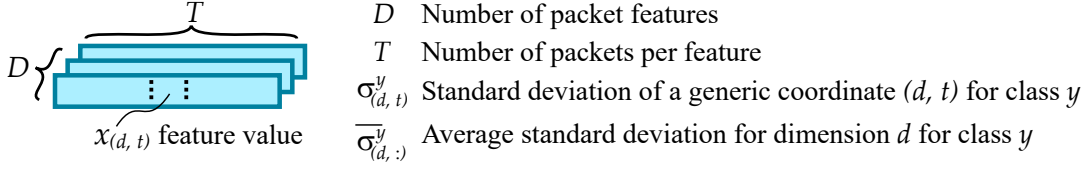
Quantifying good variety. As observed in TC literature (Horowicz et al., 2022; Rezaei and Liu, 2019; Xie et al., 2023b), domain knowledge is key to design efficient augmentations. Yet, ingenuity might not be enough as models are commonly used as “black boxes”, making it extremely challenging to establish a direct link between an augmentation technique and its impact on the final classification performance. For instance, rotation is considered a good image transformation as a result of empirical studies. Likewise, generative models are trained on large image datasets but without an explicit connection to a classification task (Schuhmann et al., 2022)—the design of the augmentation method itself is part of a trial-and-error approach and the definition of metrics quantifying the augmentation quality is still an open question.

One of the aspects to be considered when formulating such metrics is the *variety* introduced by the augmentations. Gontijo-Lopes et al. (Cubuk et al., 2021) propose metrics quantifying the distribution shift and diversity introduced by DA contrasting models performance with and without augmentations. Other literature instead focuses on mechanisms that can help defining desirable properties for augmentations. For instance, from the feature learning literature, (Shen et al., 2022; Zou et al., 2023) find that DA induces models to better learn rare/less popular but good features by altering their importance, thus improving model generalization performance. Samyak et al. (Jain et al., 2023) find that optimization trajectories are different when training on different augmentations and propose to aggregate the weights of models trained on different augmentations to obtain a more uniform distribution of feature patches, encouraging the learning of diverse and robust features.

Training loss. Self-supervision and contrastive learning are DL training strategies that take advantage of augmentations by design. In a nutshell, contrastive learning consists of a 2-steps training process. First, a feature extractor is trained in a *self-supervised* manner (or in a supervised manner) with a contrastive loss function that pulls together different augmented “views” of a given sample while distancing them from views of other samples (or pulls together different augmented “views” of samples from the same class while distancing them from views of samples from different classes). Then, a classifier head is trained on top of the learned representation in a *supervised* manner using a few labeled samples—the better the feature representation, the lower the number of labeled samples required for training the head. Empirical studies have demonstrated the robustness of the feature representations learned with contrastive learning (Chen et al., 2020; Eldele et al., 2021; Pöppelbaum et al., 2022; Yue et al., 2022) and a few recent studies investigated contrastive learning also in TC (Guarino et al., 2023; Horowicz et al., 2022; Towhid and Shahriar, 2022; Xie et al., 2023b).

Linking generative models to classifiers. When we consider the specific case of using generative models to augment training data, we face a major challenge—generative models are not designed to target a specific downstream task (Sivaroopan et al., 2023a; Sivaroopan et al., 2023b; Wang et al., 2019). While studies like (Odena et al., 2017) integrated a classifier in GAN training in the pursuit of improving the reliability of the model, how to properly link and train a generative model to be sensitive to a downstream classification task is still an open question even in CV literature.

Overall, while we believe that a broad and systematic (i.e., across multiple datasets and inputs) comparison of DA techniques in the TC field should be of community-wide interest.

FIGURE 3.1: Input sample \mathbf{x} shape and related notation.

3.3 Research goals

Drawing insights from the literature reviewed in Sec. 3.2, we undertake a set of empirical campaigns to better understand hand-crafted DA when applied in the input space of NN classifier for TC task.

The hand-crafted DA in the input space is based on a clear definition of the input format. In this chapter, the input data representation of a flow is a multivariate packet time series \mathbf{x} with D dimensions (one for each packet feature) each having T values (one for each packet) while $x_{(d,t)}$ is the value of \mathbf{x} at coordinates (d, t) where $d \in \{0..D-1\}$ and $t \in \{0..T-1\}$. In particular, in this chapter, we consider $D = 3$ packet features, namely packet size, direction, and IAT, and the first $T = 20$ packets of a flow, as sketched in Figure 3.1.

With input representation being this multi-variate time series of D properties of the first T packets of a flow, we will address the following research goals under the setting of traditional supervised learning:

- G1.** What is the performance of different individual augmentations? This includes investigating augmentations sensitivity to their hyper-parametrization and different dataset properties (e.g., number of samples and classes).
- G2.** How augmented samples should be added to the training set and how many samples should be added? Is augmenting minority classes beneficial to mitigate class imbalance?
- G3.** Why some augmentations are more effective than others?
- G4.** Does combining multiple augmentations provide extra performance improvement?

In Sec. 3.4, we motivate each goal and introduce the methodology we adopted to address them.

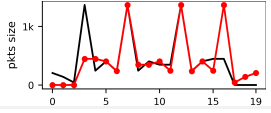
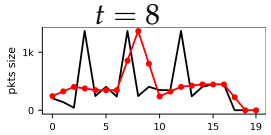
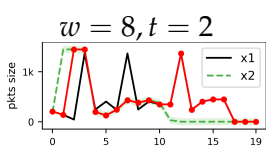
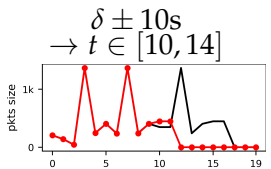
3.4 Method

3.4.1 Benchmarking hand-crafted data augmentation (G1)

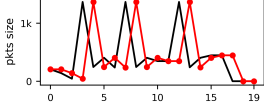
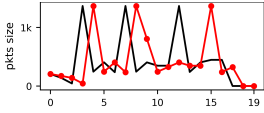
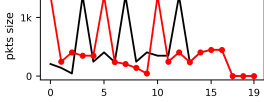
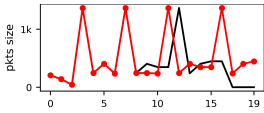
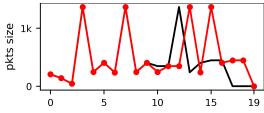
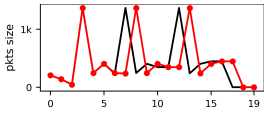
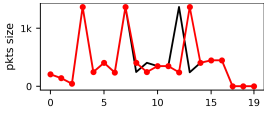
Given a real multi-variate packet time series \mathbf{x} , we define $\mathbf{x}' = \text{Aug}(\mathbf{x}, \alpha)$ as an augmentation, i.e., the transformation \mathbf{x}' of sample \mathbf{x} is subject to a *magnitude* $\alpha \in]0, 1[$ controlling the intensity of the transformation ($1 = \text{maximum modification}$).

Augmentations pool. In this study, we considered a set \mathcal{A} of 18 augmentation functions. These functions can be categorized into 3 families: 5 *amplitude* transformations, which introduce different type of jittering to the feature values (Table 3.1); 2 *masking* transformations, which force certain feature values to zero (Table 3.2); and 11 *sequence* transformations, which modify the order of feature values (Table 3.3). It is important to note that, given a sample \mathbf{x} , amplitude augmentations are solely applied to either packet size or IAT while packet direction is never altered since the latter is a binary feature and does not have amplitude (i.e., it can be -1 or 1). On the contrary, masking and sequence augmentations are applied to all features in parallel (e.g., if a transformation requires to swap $t = 1$ with $t = 6$, all features are swapped accordingly $x_{(i,1)} \leftrightarrow x_{(i,6)}$ for $\forall i \in \{0 \dots D - 1\}$). For each augmentation, Tables 3.1-3.3 report a reference example annotating its parametrization (if any).

TABLE 3.3: Sequence order augmentations.

Name	Description	Example magnitude $\alpha = 0.5$
Horizontal Flip	Swap values left to right (no magnitude needed)	
Interpolation	Densify time series by injecting average values and then sample a new sequence of length T <i>Details:</i> Expand each feature by inserting the average $0.5(x_{(d,t)} + x_{(d,t+1)})$ in-between each pair of values. Then randomly select a starting point $t = U[0, T - 1]$ and extract the following T values for all features $x_{(:,t:t+T)}$ (no magnitude needed).	
CutMix	Swap segments of two different samples <i>Details:</i> Given a training mini-batch, define pairs of samples ($\mathbf{x1}, \mathbf{x2}$) by sampling without replacement. Then sample a <i>segment</i> of length $w \sim U[0, T - 1]$ starting at $t \sim U[0, T - 1 - w]$ and swap the segment of each feature between $\mathbf{x1}$ and $\mathbf{x2}$ (no magnitude needed).	
Packet Loss	Remove values in a random time range (as if packets were not received) <i>Details:</i> Defining Δ as time to observe the first T packets, sample $\delta \sim U[0, \Delta]$ and remove values across all features in the interval $\delta \pm (10\alpha + 5)$. Then recompute the IAT and pad with zeroes at the end (if needed).	

Continued on next page

Translation	<p>Move a segment to left (\approxpkt drop) or the right (\approx pkt dup/retran)</p> <p><i>Details:</i> Define $N = 1 + \arg \max_i \{a_i \leq \alpha\}$ where $a_i \in \{0.15, 0.3, 0.5, 0.8\}$ and sample $n \sim U[1, N]$. Then, sample a direction $b \in \{left, right\}$ and a starting point $t \sim U[0, T]$: If $b = left$, left shift each feature values n times starting from t and replace shifted values with zero; if $b = right$, right shift each feature values n times starting from t and replace shifted values with the single value $x_{(d,t)}$</p>	<p>$t = 0, n = 1, b = right$</p> 
Wrap	<p>Mixing interpolation, drop and no change</p> <p><i>Details:</i> Compose a new sample x' by manipulating each $x_{(:,t)}$ based on three options with probabilities $P_{interpolate} = P_{discard} = 0.5\alpha$ and $P_{nochange} = 1 - \alpha$. If "nochange" then keep $x_{(:,t)}$; if "interpolate" then keep $x_{(:,t)}$ and $x_{(:,t)} = 0.5(x_{(:,t)} + x_{(:,t+1)})$; if "discard" then do nothing. Stop when $x' = T$ or apply tail padding (if needed).</p>	
Permutation	<p>Segment the time series and reorder the segments</p> <p><i>Details:</i> Define $N = 2 + \arg \max_i \{a_i \leq \alpha\}$ where $a_i \in \{0.15, 0.45, 0.75, 0.9\}$, a sample $n \sim U[2, N]$ and split the range $[0:T-1]$ into n segments of random length. Compose a new sample x' by concatenating $x_{(:,t)}$ from a random order of segments</p>	<p>$n = 3,$ $x' = [7:13] \cup [0:6] \cup [14:19]$</p> 
Dup-RTO	<p>Mimic TCP pkt retrains due to timeout by duplicating values</p> <p><i>Details:</i> Duplicating a range of packets according to a Bernoulli($p = 0.1\alpha$) (see Algo. 1 in (Xie et al., 2023a))</p>	
Dup-FastRetr	<p>Mimic TCP fast retrains by duplicating values</p> <p><i>Details:</i> Duplicating one packet according to a Bernoulli($p = 0.1\alpha$) (see Algo. 2 in (Xie et al., 2023a))</p>	
Perm-RTO	<p>Mimic TCP pkt retrains due to timeout by permuting values</p> <p><i>Details:</i> Delaying a range of packets according to a Bernoulli($p = 0.1\alpha$) (see Algo. 3 in (Xie et al., 2023a))</p>	
Perm-FastRetr	<p>Mimic TCP fast retrains by permuting values</p> <p><i>Details:</i> Delaying one packet according to a Bernoulli($p = 0.1\alpha$) (see Algo. 4 in (Xie et al., 2023a))</p>	

In the figures, black solid lines — for original samples, red ● lines for augmented samples.

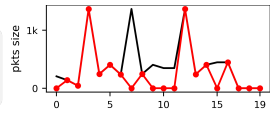
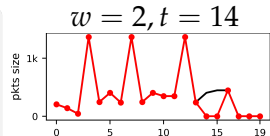
Name	Pkts	Feat.	Size	Dir	IAT	Description	Example magnitude $\alpha = 0.5$
Gaussian Noise	●	○	●			<p>Add independently sampled Gaussian noise to Size or IAT</p> <p><i>Details:</i> Sample a feature $d \in \{\text{Size, IAT}\}$ and add Gaussian noise to its values $x_{(d,t)} + \varepsilon_t$ where $\varepsilon_t \sim \mathcal{N}(0, \alpha \{\sigma_{(d,t)}^y\}^2)$</p>	
Spike Noise	●	○	●			<p>Add independently sampled Gaussian noise to Size or IAT</p> <p><i>Details:</i> Sample a feature $d \in \{\text{Size, IAT}\}$ and add Gaussian noise to up to 3 of its non-zero values $x_{(d,t)} + \varepsilon_t$ where $\varepsilon_t \sim \mathcal{N}(0, \alpha \{\sigma_{(d,t)}^y\}^2)$</p>	
Gaussian WrapUp	●	○	●			<p>Scale Size or IAT by independently sampled Gaussian values</p> <p><i>Details:</i> Sample a feature $d \in \{\text{Size, IAT}\}$ and multiply Gaussian noise to its values $x_{(d,t)} \cdot \varepsilon_t$ with $\varepsilon_t \sim \mathcal{N}(1 + 0.01\alpha, 0.02\alpha \{\sigma_{(d,t)}^y\}^2)$</p>	
Sine WrapUp	●	○	●			<p>Scale Size or IAT by sinusoidal noise</p> <p><i>Details:</i> Sample a feature $d \in \{\text{Size, IAT}\}$ and multiply its values by a sine-like noise $x_{(d,t)} \cdot \varepsilon_i$ with $\varepsilon_i = [1 + 0.02\alpha \cdot \overline{\sigma_{(d,:)}^y} \cdot \sin(\frac{4\pi i}{T} + \theta)]$ and $\theta \sim U[0, 2\pi[$</p>	
Constant WrapUp	○	○	●			<p>Scale IAT by a single randomly sampled value</p> <p><i>Details:</i> Sample a single uniformly sampled value $\epsilon \sim U[a, b]$ and perform $x_i \cdot \epsilon$ to all x_i of IAT with $a = 1 + \overline{\sigma_{(d,:)}^y} \cdot (0.06 - 0.02\alpha);$ $b = 1 + \overline{\sigma_{(d,:)}^y} \cdot (0.14 + 0.02\alpha)$</p>	

○ feature never used; ● feature selected randomly; ● feature always used.

In the figures, black solid lines — for original samples, red ● lines for augmented samples; x-axis for time series index and y-axis the feature value (either packet size or IAT).

TABLE 3.1: Amplitude augmentations.

By adopting such a large pool of augmentations, our empirical campaign offers several advantages. First, we are able to investigate a broader range of design possibilities compared to previous studies. Second, it enables us to contrast different families and assess if any of them is more prone to disrupt class semantics. Considering the latter, TC literature (Horowicz et al., 2022; Rezaei and Liu, 2019; Xie et al., 2023b) predominantly investigate sequence transformations (typically acting only on packet timestamp) with only (Xie et al., 2023b) experimenting with masking and amplitude variation, yet targeting scenarios where models are exposed to data shifts due to Maximum Segment Size (MSS) changes, i.e., the network properties related

Name	Description	Example magnitude $\alpha = 0.5$
Bernoulli Mask	Random masking values	
	Details: Independently set to zero feature values by sampling a Bernoulli($p = 0.6\alpha$)	
Window Mask	Masking the same sequences across all features	
	Details: Given a configured maximum size $W = \lfloor 1 + 2.5\alpha \rfloor$, sample a window length $w \sim U[1, W]$ and a random starting point $t = U[0, T - w]$ and set to zero all $x_{(:,t)}$ falling in the sampled window	

All three features (Size, DIR and IAT) are affected by all transformations. In the figures, black solid lines — for original samples, red ● lines for augmented samples.

TABLE 3.2: Masking augmentations

to the training set are different from the ones of the test set.

Augmentations magnitude. As described in Tables 3.1-3.3, each augmentation has some predefined static parameters¹ while the magnitude α is the single hyperparameter controlling random sampling mechanisms contributing to defining the final transformed samples. To quantify augmentations sensitivity to α , we contrast two scenarios following CV literature practice: a static value of $\alpha = 0.5$ and a uniformly sampled value $\alpha \sim U[0, 1]$ extracted for each augmented sample.

Datasets size and task complexity. Supervised tasks, especially when modeled via DL, benefit from large datasets. For instance, as previously mentioned, some CV literature pretrains generative models on large datasets and use those models to obtain auxiliary training data for classification tasks. While data availability clearly plays a role, at the same time the task complexity is equivalently important—a task with just a few classes but a lot of data does not necessarily yield higher accuracy than a task with more classes and less data. To understand how augmentations interplay with these dynamics, it is relevant to evaluate augmentations across datasets of different sizes and number of classes.

3.4.2 Training batches composition (G2)

In order to mitigate any undesirable shifts introduced by artificial samples, it is necessary to balance original and augmented samples. Yet, the way original and augmented samples are combined to form the augmented training set is a design choice. For instance, in TC literature, (Horowicz et al., 2022; Rezaei and Liu, 2019) augment the data before starting the training, while (Xie et al., 2023b) augments mini-batches

¹These parameters are tuned via preliminary investigations.

during the training process. In this work, we apply augmentation samples to a training mini-batch of size B , with the two policies sketched in Fig. 3.2. *Replace* substitutes an original sample \mathbf{x}_i with its augmentation by sampling from a Bernoulli($P=P_{\text{replace}}$) random variable—during one training epoch, approximately a P_{replace} fraction of the original data is “hidden”. Instead, *Inject* increases the batch size by augmenting each sample N_{inject} times (e.g., in Fig. 3.2 the original batch size is doubled by setting $N_{\text{inject}} = 1$).

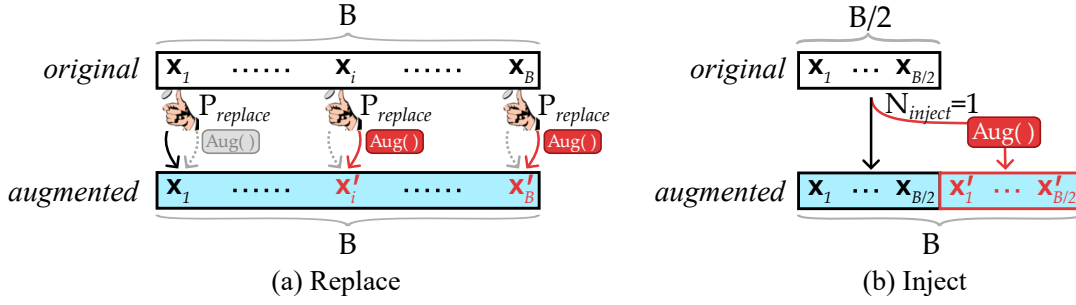


FIGURE 3.2: Training batch creation policies.

3.4.3 Latent space geometry (G3)

Augmented samples play a crucial role in model training, just like the original training samples from which they are derived. To understand the impact of augmentations on the improvement or detriment of classification performance, we propose to examine the latent space of the classifier. In order to conduct a comprehensive analysis, we need to consider two aspects applicable to any supervised classification task.

Augmentation-vs-Test. ML methods operate on the assumption that training data serves as a “proxy” for test samples, i.e., the patterns learned on training data “generalize” to testing data as the two sets of data resemble each other properties. In this context, augmentations can be considered as a means for fostering data generalization by incorporating samples that resemble even more testing data compared to what is available in training data. However, it is important to empirically quantify this effect by measuring, for instance, the distance between augmented and test samples. In other words, we aim to quantify up to which extent augmented samples are better at mimicking test samples compared to the original data.

Augmentation-vs-Train. The performance of a feature extractor greatly depends on how well the feature extractor separates different classes in the latent space. Data augmentations play a role in shaping intra/inter-class relationships created by the feature extractor in the latent space. For instance, an augmentation that generates samples far away from the region of a class can *disrupt class semantics*—the augmentation is introducing a new behavior/mode making it hard for the classifier to be

effective. At the same time, however, expanding the region of a class can be a beneficial design choice—augmentations that enable a better definition of class boundaries simplify the task of the classifier. Understanding such dynamics requires empirical observations, for instance, by comparing the distance between the original training data and augmented data. In other words, we aim to verify if augmentations yielding good performance are in a “sweet spot”: they create samples that are neither too close (i.e., introduce too little variety) nor too far (i.e., disrupt class semantics) from original samples.

3.4.4 Combining augmentations (G4)

To address **G1**, each trained model is associated to an individual augmentation. However, in CV it is very common to combine multiple augmentations (Chen et al., 2020). Hence, we aim to complement **G1** by measuring the performance of three different policies augmenting mini-batches based on a set $\mathcal{A}' \subset \mathcal{A}$ composed of top-performing augmentations based on the **G1** benchmark: the *Ensemble* policy uniformly samples one of the augmentations in \mathcal{A}' independently for each mini-batch sample; the *RandomStack* policy randomly shuffles \mathcal{A}' independently for each mini-batch sample before applying all augmentations; finally, the *MaskedStack* policy uses a predefined order for \mathcal{A}' but each augmentation is associated with a masking probability, i.e., each sample in the mini-batch independently selects a subset of augmentations of the predefined order.

3.5 Experimental settings

3.5.1 Datasets

To address our research goals we considered the datasets summarized in Table 3.4.

MIRAGE-19-HALF (Aceto et al., 2019c) is a *public* dataset gathering traffic logs from 20 popular Android apps collected at the ARCLAB laboratory of the University of Napoli Federico II. Multiple measurement campaigns were operated by instrumenting 3 Android devices handed off to ≈ 300 volunteers (students and researchers) for interacting with the selected apps for short sessions. Each session resulted in a pcap file and an strace log mapping each socket to the corresponding Android application name. Pcaps were then post-processed to obtain bidirectional flow logs by grouping all packets belonging to the same 5-tuple (srcIP, srcPort, dstIP, dstPort, L4proto) and extracting both aggregate metrics (e.g., total bytes, packets, etc.), per-packet time series (packet size, direction, TCP flags, etc.), raw packets payload bytes (encoded as a list of integer values) and mapping a ground-truth label by means of the strace logs.

Name	Classes	Curation	Flows per-class			ρ	Pkts mean
			<i>all</i>	<i>min</i>	<i>max</i>		
MIRAGE-19-HALF *	20	none	122 k	1,986	11,737	5.9	23
		>10pkts	64 k	1,013	7,505	7.4	17
MIRAGE-22	9	none	59 k	2,252	18,882	8.4	3,068
		>10pkts	26 k	970	4,437	4.6	6,598
Enterprise-100	100	none	2.9 M	501,221	5,715	87.7	2,312

ρ : ratio between max and min number of flows per-class—the larger the value, the higher the imbalance;

*: Despite being advertised of having traffic from 40 apps, the *public* version of the dataset only contains 20 apps.

TABLE 3.4: Summary of datasets properties.

MIRAGE-22 (Guarino et al., 2021a) is another *public* dataset collected by the same research team and with the same instrumentation as MIRAGE-19-HALF which targets 9 video meeting applications used to perform webinars (i.e., meetings with multiple attendees and a single broadcaster), audio calls (i.e., meetings with two participants using audio-only), video calls (i.e., meetings with two participants using both audio and video), and video conferences, (i.e., meetings involving more than two participants broadcasting audio and video).

Enterprise-100 is instead a *private* dataset collected by monitoring network flows from vantage points deployed in residential access and enterprise campus networks. For each flow, the logs report multiple aggregate metrics (number of bytes, packets, TCP flags counters, round trip time statistics, etc.), and the packet time series of packet size, direction and IAT for the first 50 packets of each flow. Moreover, each flow record is also enriched with an application label provided by a commercial DPI software directly integrated into the monitoring solution and supporting hundreds of applications and services.

Data curation. Table 3.4 compares different dataset properties. For instance, the datasets MIRAGE-19-HALF and MIRAGE-22 are quite different from each other despite being obtained via the same platform. Specifically, MIRAGE-19-HALF gathers around $2\times$ more flows than MIRAGE-22 but those are $100\times$ shorter. As expected, all datasets are subject to class imbalance measured by ρ , i.e., the ratio between maximum and minimum number of samples per class. However, Enterprise-100 exhibits a larger class imbalance with respect to the other two datasets. Last, while Enterprise-100 did not require specific pre-processing, both MIRAGE-19-HALF and MIRAGE-22 required a curation to remove *background traffic*—flows created by netd daemon, SSDP, Android google management services and other services unrelated to the target Android apps—and flows having less than 10 packets.

Data folds and normalization. As described in Sec. 3.4.1, each flow is modeled via a multivariate time series \mathbf{x} consisting of $D = 3$ features (packets size, direction, and IAT) related to the first $T = 20$ packets (applying zero padding in the tail where needed). From the curated datasets we created 80 random 70/15/15 train/-validation/test folds. We then processed each train+val split to extract statistics that we used for normalizing the data and to drive the augmentation process. Specifically, we computed both per-coordinate (d, t) and global (i.e., flattening all flows time series into a single array) mean and standard deviation for each class—these statistics provided us the $\sigma_{(d,t)}^y$ and $\sigma_{(:,t)}^y$ needed for the augmentations (see Fig. 3.1 and Tables 3.1-3.3). For IAT, we also computed the global 99th percentile across all classes q_{iat}^{99} . Given a multi-variate input \mathbf{x} , we first clip packet size values in the range $[0, 1460]$ and IAT values in the range $[1e-7, q_{iat}^{99}]$. Due to high skew of IAT distributions, we also log10-scaled the IAT feature values.² Last, all features are standardized to provide values $x_{(d,t)} \in [0, 1]$.

Model architecture and training. We rely on a 1d-CNN based neural network architecture with a backbone including 2 ResNet blocks followed by a linear head resulting in a compact architecture of $\approx 100k$ parameters. (see Fig. 3.3 and Table 3.5 for details). Models are trained for a maximum of 500 epochs with a batch size $B=1,024$ via an AdamW optimizer with a weight decay of 0.0001 and a cosine annealing learning rate scheduler initialized at 0.001. Training is subject to early stopping by monitoring if the validation accuracy does not improve by 0.02 within 20 epochs. We coded our modeling framework using PyTorch and PyTorch Lightning and ran our modeling campaigns on Linux servers equipped with multiple NVIDIA Tesla V100 GPUs. We measured the classification performance via the weighted F1 score considering a reference baseline where training is not subject to augmentations.

TABLE 3.5: Model architecture printout (MIRAGE-19-HALF, 20 classes)

Layer (type)	Output Shape	Param #
Conv1d-1	[-1, 64, 20]	576
BatchNorm1d-2	[-1, 64, 20]	128
Conv1d-3	[-1, 64, 10]	12,288
BatchNorm1d-4	[-1, 64, 10]	128
Conv1d-5	[-1, 64, 10]	12,288
BatchNorm1d-6	[-1, 64, 10]	128
Conv1d-7	[-1, 64, 10]	4,096
BatchNorm1d-8	[-1, 64, 10]	128
Conv1d-9	[-1, 128, 5]	24,576
BatchNorm1d-10	[-1, 128, 5]	256
Conv1d-11	[-1, 128, 5]	49,152

²We did not log-scale packet sizes values as we found this can reduce accuracy based on preliminary empirical assessments.

BatchNorm1d-12	[-1, 128, 5]	256
Conv1d-13	[-1, 128, 5]	8,192
BatchNorm1d-14	[-1, 128, 5]	256
AdaptiveAvgPool1d-15	[-1, 128, 1]	0
Linear-16	[-1, 20]	2,580
Total params:		115,028
Trainable params:		115,028
Non-trainable params:		0
Input size (MB):		0.00
Forward/backward pass size (MB):		0.09
Params size (MB):		0.44
Estimated Total Size (MB):		0.53

3.6 Results

In this section, we discuss the results of our modeling campaigns closely following the research goals introduced in Sec. 3.3.

3.6.1 Augmentations benchmark (G1)

We start by presenting the overall performance of the selected augmentations. Specifically, Table 3.6 collects results obtained by applying augmentations via Inject with $N_{inject} = 1$ (i.e., each original sample is augmented once)³ and sampling uniformly the magnitude $\alpha \sim U[0, 1]$. Table 3.6 shows the average weighted F1 score across 80 runs and related 95th-percentile confidence intervals.

Reference baseline. We highlight that our reference baseline performance on the datasets MIRAGE-19-HALF and MIRAGE-22 are *qualitatively* aligned with previous literature that used those datasets. For instance, Table 1 in (Guarino et al., 2021a) reports a weighted F1 of 97.89 for a 1d-CNN model when using the first 2,048 payload bytes as input for MIRAGE-22; Figure 1 in (Bovenzi et al., 2021) instead shows a weighted F1 of $\approx 75\%$ for 100 packets time series input for MIRAGE-19-HALF. Notice however that since these studies use training configurations not exactly identical to ours, a direct comparison with our results should be taken with caution. Yet, despite these differences, we confirm MIRAGE-19-HALF to be a more challenging classification task compared to MIRAGE-22. However, we argue that such a difference is unlikely depending only on the different number of classes (MIRAGE-19-HALF has 20 classes while MIRAGE-22 only 9). This is evident by observing that Enterprise-100 yields very

³Since we train the reference baseline with a batch size $B=1024$, when adding augmentations we instead adopt $B=512$ (which doubles via injection).

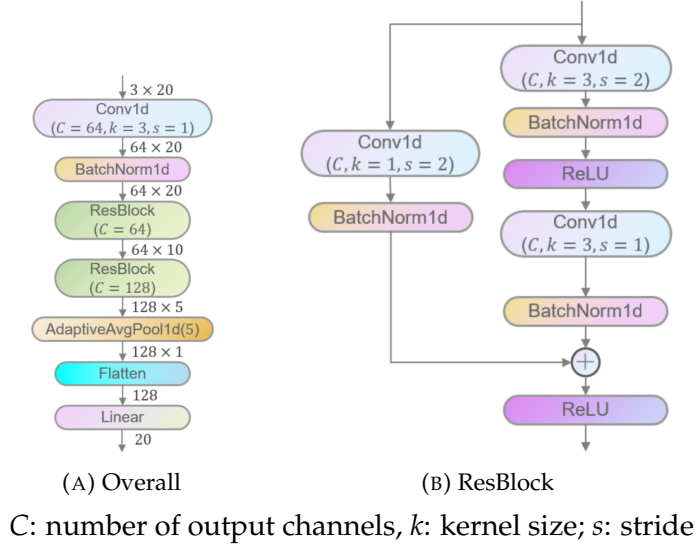


FIGURE 3.3: Model architecture.

high performance despite having $10\times$ more classes than the other two datasets. We conjecture instead the presence of “cross-app traffic” such as flows generated by libraries/services common across multiple apps from the same provider (e.g., apps or services provided by Google or Facebook) and/or the presence of ads traffic,⁴ but the datasets raw data is not sufficiently detailed to investigate our hypothesis.

Takeaways. While the classification tasks complexity is well captured by models performance, it does not necessarily relate to the number of classes or dataset size. These effects are visible only when studying multiple datasets at once, but unfortunately a lot of TC studies focus on individual datasets.

Augmentations rank. Overall, all augmentations are beneficial except for Horizontal Flip which, as we shall see in Sec. 3.6.3, breaks class semantics. As expected, not all augmentations provide the same gain and their effectiveness may vary across datasets. Specifically, *sequence* and *masking* better suit our TC tasks.

For a more fine-grained performance comparison, we complement Table 3.6 results by analyzing augmentations rank via a critical distance by following the procedure described in (Demšar, 2006). Specifically, for each of the 80 modeling runs we first ranked the augmentations from best to worst (e.g., if augmentations A, B, and C yield a weighted F1 of 0.9, 0.7, and 0.8, their associated rankings would be 1, 3, and 2) splitting ties using the average ranking of the group (e.g., if augmentations A, B, and C yield a weighted F1 of 0.9, 0.9 and 0.8, their associated rankings would be 1.5, 1.5, and 3). This process is then repeated across the 80 runs and a global rank is obtained by computing the mean rank for each augmentation. Last, these averages are compared pairwise using a post-hoc Nemenyi test to identify which groups

⁴MIRAGE-22 focuses on video meeting apps which are all from different providers and ads free by design.

	Augmentation	MIRAGE-19-HALF	MIRAGE-22	Enterprise-100
Baseline	None	75.43 \pm .10	94.92 \pm .07	92.43 \pm .33
Amplitude	Constant WrapUp	0.61 \pm .12	0.36 \pm .09	-0.02 \pm .15
	Gaussian Noise	0.89 \pm .11	0.24 \pm .09	0.15 \pm .14
	Gaussian WrapUp	1.01 \pm .13	0.74 \pm .09	0.24 \pm .12
	Spike Noise	1.66 \pm .12	0.91 \pm .09	0.93 \pm .13
	Sine WrapUp	0.63 \pm .11	0.25 \pm .09	-0.06 \pm .16
Masking	Bernoulli Mask	2.55 \pm .12	1.29 \pm .09	1.25 \pm .16
	Window Mask	2.37 \pm .13	1.08 \pm .09	1.18 \pm .16
Sequence	CutMix	2.65 \pm .13	1.40 \pm .10	-0.21 \pm .10
	Dup-FastRetr	3.23 \pm .13	1.56 \pm .09	0.83 \pm .15
	Dup-RTO	2.89 \pm .13	1.33 \pm .09	0.91 \pm .15
	Horizontal Flip	-0.71 \pm .11	-0.52 \pm .09	-0.88 \pm .15
	Interpolation	0.44 \pm .12	0.53 \pm .10	-0.61 \pm .14
	Packet Loss	0.88 \pm .12	0.66 \pm .09	0.60 \pm .22
	Permutation	3.67 \pm .13	1.97 \pm .09	0.89 \pm .08
	Perm-RTO	3.15 \pm .12	1.54 \pm .09	0.88 \pm .12
	Perm-FastRetr	2.11 \pm .12	1.00 \pm .09	0.74 \pm .26
	Translation	4.40 \pm .13	2.02 \pm .09	0.95 \pm .15
	Wrap	4.11 \pm .13	2.09 \pm .08	0.57 \pm .12

The top-3 best and worst augmentations are color-coded.

TABLE 3.6: Augmentations benchmark (G1).

of augmentations are statistically equivalent. This decision is made using a Critical Distance (CD) $CD = q_\alpha \sqrt{k(k+1)/6N}$, where q_α is based on the Studentized range statistic divided by $\sqrt{2}$, k is equal to the number of augmentations compared and N is equal to the number of samples used. Results are then collected in Fig. 3.4 where each augmentation is highlighted with its average rank (the lower the better) and horizontal bars connect augmentations that are statistically equivalent. For instance, while Table 3.6 shows that Translate is the best on average, Fig. 3.4 shows that {Translate, Wrap, Permutation, Dup-FastRetr} are statistically equivalent. We remark that Fig. 3.4 refers to MIRAGE-19-HALF and MIRAGE-22 but similar considerations hold for Enterprise-100 as well.

Recall that our training process is subject to an early stop mechanism. Interestingly, we observed that augmentations yielding better performance also present a longer number of training epochs (see Fig. 3.5). This hints that effective augmentations foster better data representations extraction, although some CV studies also show that early stopping might not necessarily be the best option to achieve high accuracy in some scenarios. An in-depth investigation of these training mechanisms is however out of scope for this chapter.

Takeaways. *Augmentations bring benefits that, in absolute scale, are comparable to what is observed in CV literature (Müller and Hutter, 2021). Our benchmark shows that TC sequencing and masking augmentations are better options than amplitude augmentations.*

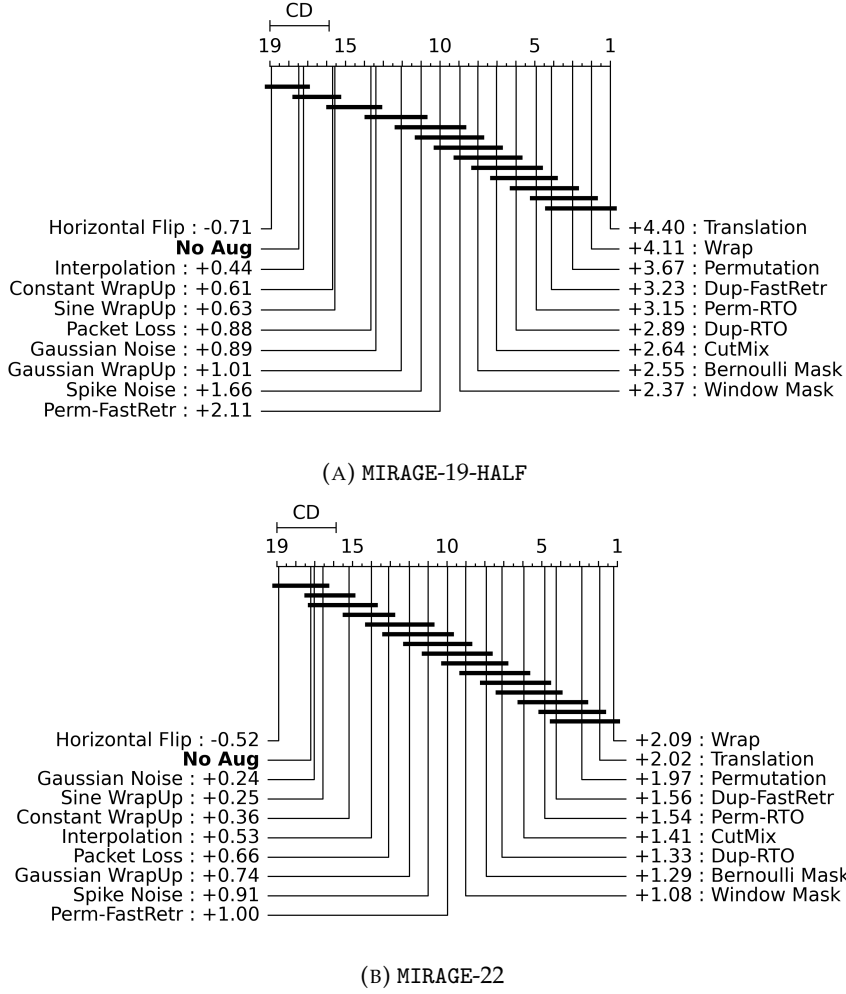


FIGURE 3.4: Augmentations rank and critical distance (G1).

This is in line with previous literature that implicitly discarded amplitude augmentations. Finally, despite performance ranks may suggest more efficient augmentations (e.g., Translation or Bernoulli mask), agreement between datasets seems more qualitative than punctual (e.g., masking is preferred to sequencing for Enterprise-100, but the reverse is true for the other two datasets).

Sensitivity to magnitude. Most of the augmentations we analyzed are subject to a magnitude α hyper-parameter (see Tables 3.1-3.3) that is randomly selected for the results in Table 3.6. To investigate the relationship between classification performance and augmentation magnitude we selected 3 augmentations among the top performing ones {Translation, Wrap, Permutation} and three among the worst performing {Gaussian Noise, Sine WrapUp, Constant WrapUp}.⁵ For each augmentation, we performed 10 modeling runs using magnitude $\alpha = 0.5$ and we contrasted these results with the related runs from the previous modeling campaign. Specifically, by grouping all results we obtained a binary random-vs-static performance

⁵We excluded HorizontalFlip as it hurts performance and Interpolation since it does not depend from a magnitude.

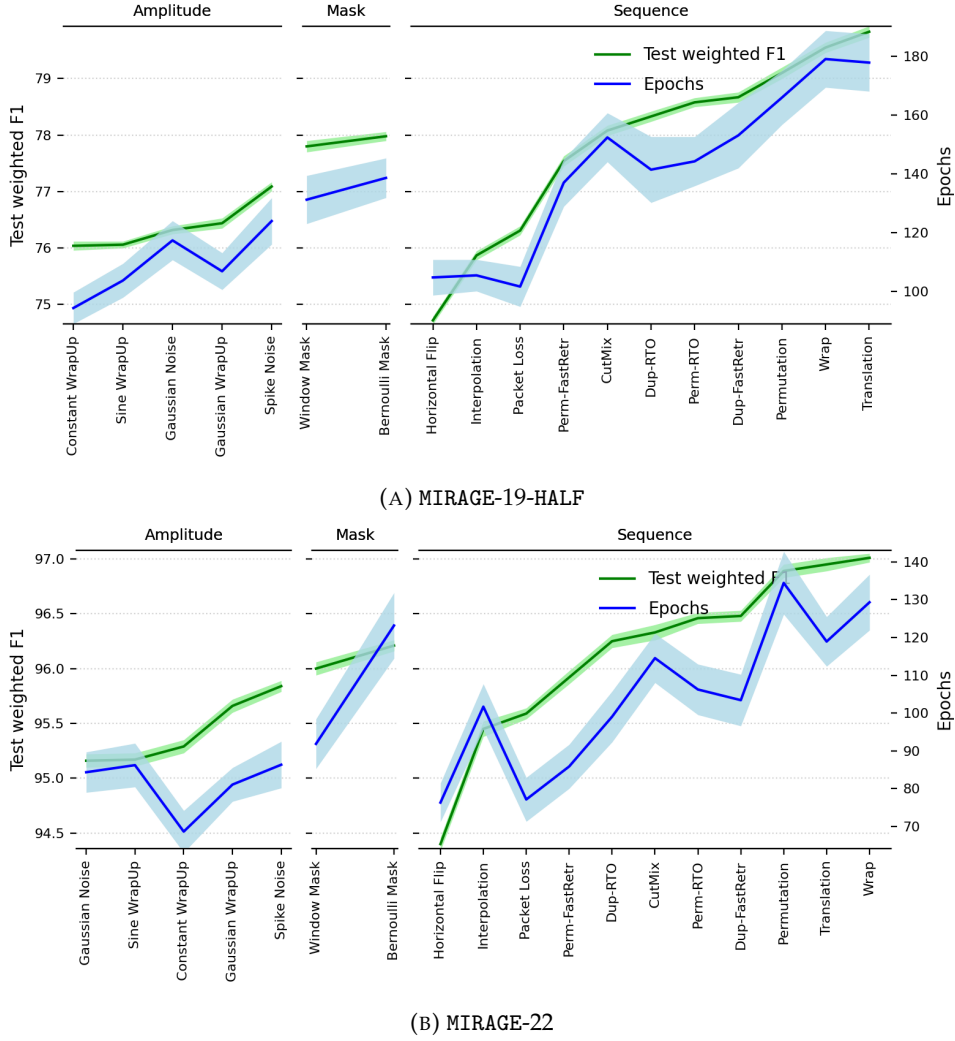


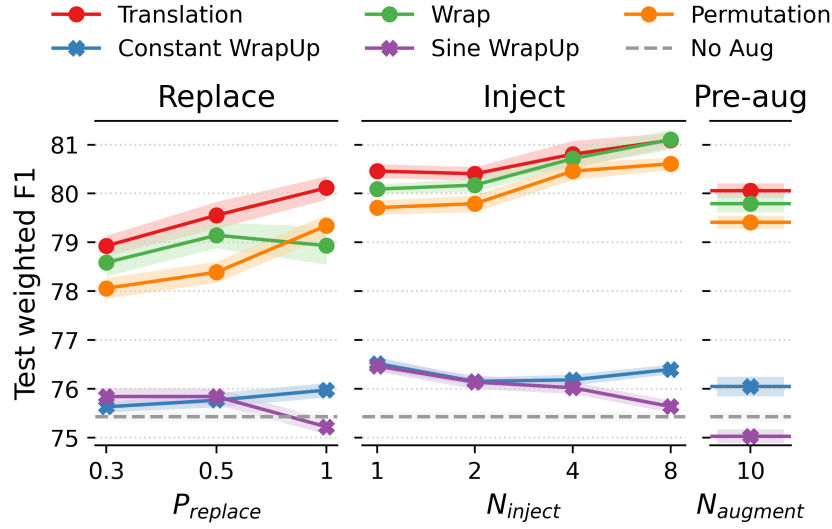
FIGURE 3.5: Comparing performance improvement and training length.

comparison which we investigated through a Wilcoxon signed rank sum test that indicated *no statistical difference*, i.e., the selection of magnitude is not a distinctive factor to drive the augmentation performance. The same conclusion holds true when repeating the analysis for each individual augmentation rather than grouping them together.

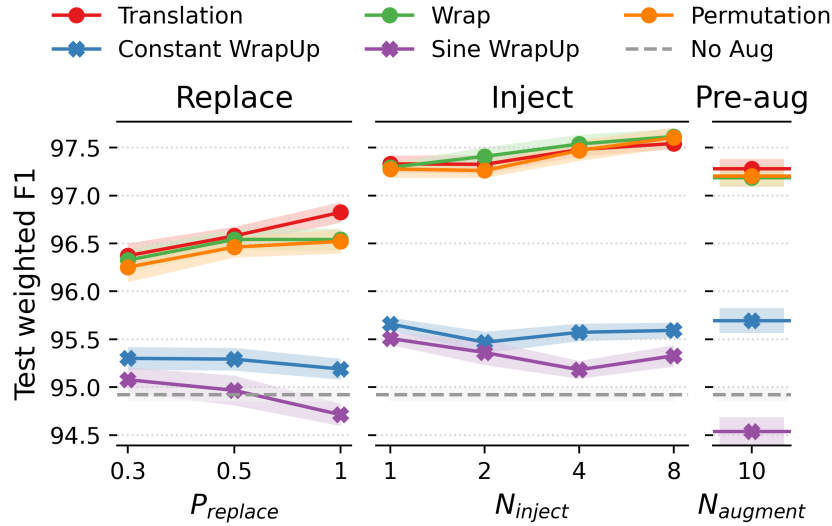
Takeaways. Although we do not observe any dependency on the augmentation magnitude α , augmentations performance can still be affected by their tuning (as will be discussed further in Sec. 3.6.3). Unfortunately, this tuning process often relies on a trial-and-error process, making it challenging to operate manually.

3.6.2 Training batches composition (G2)

Correctly mixing original with augmented data is an important design choice.



(A) MIRAGE-19-HALF



(B) MIRAGE-22

FIGURE 3.6: Comparing *Replace*, *Inject* and *Pre-augment* batch creation policies (G2).

Batching policies. To show this, we considered the three policies introduced in Sec. 3.4.2: Replace (which randomly substitutes training samples with augmented ones), Inject (which expands batches by adding augmented samples), and Pre-augment (which expands the whole training set before the training start).⁶ Batching policies are compared against training without augmentations making sure that each training step has the same batch size $B=1,204$.⁷ Based on Sec. 3.6.1 results, we limited our comparison to {Translation, Wrap, Permutation} against {Sine WrapUp, Constant WrapUp} as representative of good and poor augmentations across the

⁶Based on our experience of using code-bases related to publications, we were unable to pinpoint if any of those techniques is preferred in CV literature.

⁷For instance, when $N_{inject} = 1$, a training run needs to be configured with $B=512$ as the mini-batches size doubles via augmentation.

	Cls samp.	Majority classes			Minority classes		
		Pre	Rec	weight F1	Pre	Rec	weight F1
No Aug	with	83.90 \pm .21	81.01 \pm .21	82.36 \pm .14	56.63 \pm .38	60.78 \pm .26	58.18 \pm .21
	without	81.60 \pm .23	82.93 \pm .19	82.16 \pm .12	62.29 \pm .48	58.02 \pm .38	59.78 \pm .27
	diff	2.30 \pm .32	-1.92 \pm .28	0.20 \pm .20	-5.66 \pm .60	2.76 \pm .46	-1.60 \pm .35
Translation	with	89.12 \pm .09	84.26 \pm .11	86.43 \pm .08	60.71 \pm .24	68.64 \pm .17	63.65 \pm .19
	without	85.36 \pm .14	86.73 \pm .10	85.86 \pm .09	69.69 \pm .25	64.14 \pm .25	66.20 \pm .22
	diff	3.77 \pm .04	-2.48 \pm .02	0.57 \pm .02	-8.98 \pm .04	4.50 \pm .09	-2.55 \pm .05

TABLE 3.7: Impact of class-weighted sampler on MIRAGE-19-HALF (G2).

three datasets under study. We configured Replace with $P_{replace} \in \{0.3, 0.5, 1\}$, Inject with $N_{inject} \in \{1, 2, 4, 8\}$ and augmented each training sample 10 times for Pre-augment. Fig. 3.6 collects the results with lines showing the average performance while shaded areas correspond to 95th percentile confidence intervals. Overall, top-performing augmentations (● marker) show a positive trend—the higher the volume of augmentations the better the performance—while poor-performing augmentations (× marker) have small deviations from the baseline (dashed line). Based on performance, we can order $Replace < Pre\text{-}augment < Inject$, i.e., the computationally cheaper Pre-augment is on par with the more expensive Replace when $P_{replace} = 1$ but Inject is superior to both alternatives.

Takeaways. On the one hand, Inject shows a positive trend that perhaps continues beyond $N_{inject} > 8$.⁸ On the other hand, the performance gain may be too little compared to the computational cost when using many augmentations. For instance, $N_{inject} = 8$ requires $3\times$ longer training compared to $N_{inject} = 1$.

Class-weighted sampling. TC datasets are typically imbalanced (see Table 3.4). It is then natural to wonder if/how augmentations can help improve performance for classes with fewer samples, namely *minority classes*. Although the batching policies discussed do not alter the natural distribution of the number of samples per class, alternative techniques like Random Over Sampling (ROS) and Random Under Sampling (RUS) allow to replicate/drop samples for minority/majority classes (Johnson and Khoshgoftaar, 2019). A *class-weighted sampler* embodies a more refined version of those mechanisms and composes training mini-batches by selecting samples with a probability inversely proportional to the classes size—each training epoch results in a balanced dataset. When combined with augmentations, this further enhance minority classes variety.

The adoption of a class-weighted sampler seems a good idea in principle. Yet, the enforced balancing in our experience leads to conflicting results. We showcase this

⁸The limit of our experimental campaigns were just bounded by training time and servers availability so it is feasible to go beyond the considered scenarios.

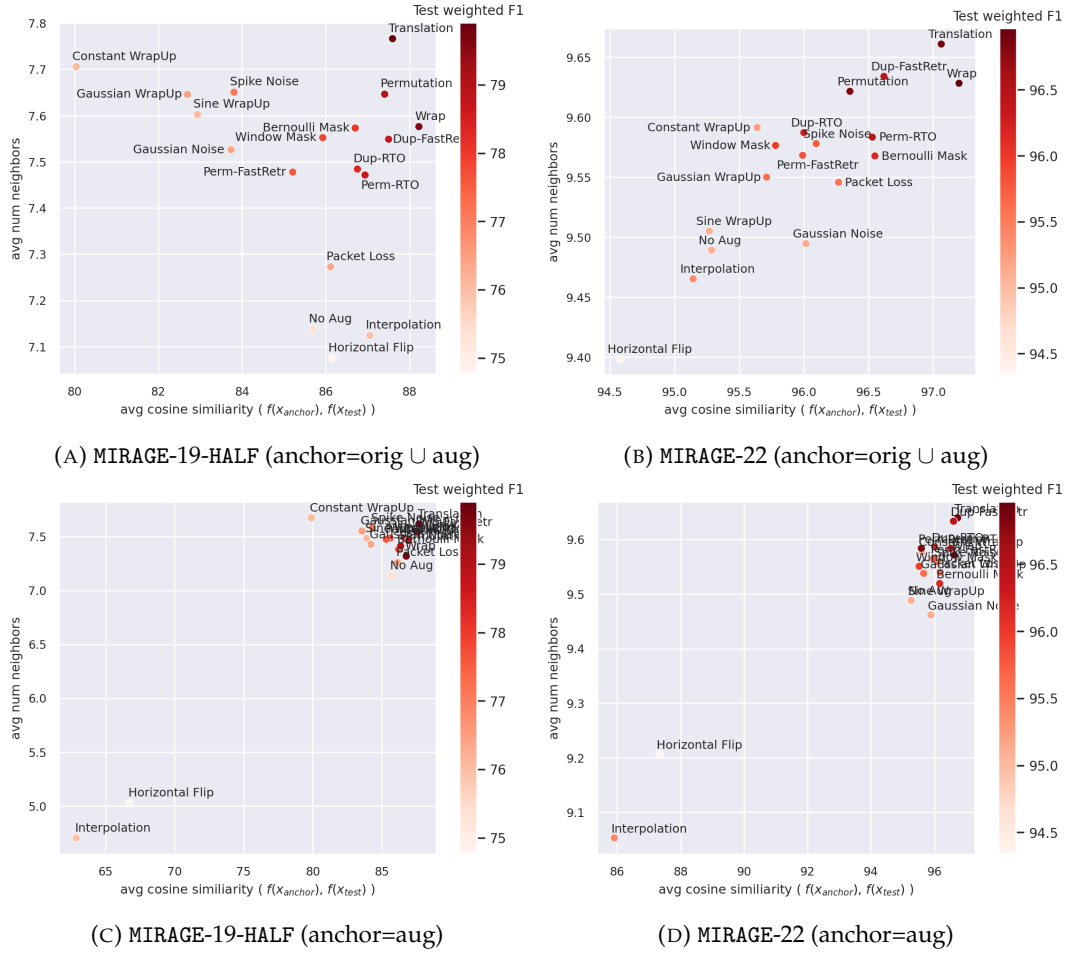


FIGURE 3.7: Investigating train, augmented and test samples relationships (G3).

in Table 3.7 where we show Precision, Recall, and weighted F1 for 20 runs trained with/without a weighted sampler and with/without Translation (selected as representative of a good augmentation across datasets). We break down the performance between majority and minority classes and report per-metric differences when using or not the weighted sampler. The table refers to MIRAGE-19-HALF but similar results can be obtained for the other datasets. Ideally, one would hope to observe only positive differences with larger benefits for minority classes. In practice, only the Recall for minority classes improves and overall we observe a poorer weighted F1 (-0.26 across all classes). By investigating misclassifications, we found that majority classes are more confused with minority classes and when introducing augmentations those effects are further magnified.

Takeaways. *Paying too much attention to minority classes can perturb the overall classifier balance, so we discourage the use of class-weighted samplers.*

3.6.3 Latent space geometry (G3)

Table 3.6 allows to identify effective augmentations bringing significant benefits in terms of model performance. However, to understand the causes behind the performance gaps we need to investigate how original, augmented, and test samples relate to each other.

Augmented-vs-test samples. We start our analysis by taking the point of view of the test samples. Specifically, we investigated which type of points are found in the “neighborhood” of a test sample. To do so, we started creating “true anchors” by projecting both the original training data and 5 augmentations of each training sample—these anchors are “proxy” of what is presented to the model during training. Then we projected the test samples and looked for the closest 10 anchors (based on cosine similarity) of each test sample. Finally, we counted how many of the 10 anchors share the same label as the test samples. Results for each augmentation are reported in Fig. 3.7 for MIRAGE-19-HALF and MIRAGE-22 (similar results holds for Enterprise-100) as a scatter plot where the coordinates of each point correspond to the average number of anchors with the correct label found and their average cosine similarity with respect to the test sample. Each augmentation is color-coded with respect to its weighted F1 score.

Despite both metrics vary in a subtle range, such variations suffice to capture multiple effects. First of all, considering the layout of the scatter plot, we expected good transformations to be placed in the top-right corner. This is indeed the case as presented in Fig. 3.7 (a-b) where darker colors (higher weighted F1) concentrate in the top-right corner. However, while MIRAGE-22 (Fig. 3.7(a)) shows a linear correlation between the two metrics, MIRAGE-19-HALF (Fig. 3.7(a)) shows outliers, most notably Horizontal Flip, Interpolation, and Constant Wrapup.

Fig. 3.7 (c-d) complement the analysis by showing results when considering only augmented samples as anchors. Differently from before, now Horizontal Flip and Interpolation are found to be the most dissimilar to the test samples—this is signaling that augmentations are possibly disrupting class semantics, i.e., they are introducing unnecessary high variety.

Last, for each test sample we looked at the closest augmented anchor and the closest original sample anchor with the same label. The average ratio of those pairwise distances is centered around 1—augmented samples “mimic” test samples as much as the original samples do.

***Takeaways.** Top-performing augmentations do not better mimic test samples compared to original samples. Rather, they help training the feature extractor $f(\cdot)$ so that projected test samples are found in neighborhood of points likely to have the expected label.*

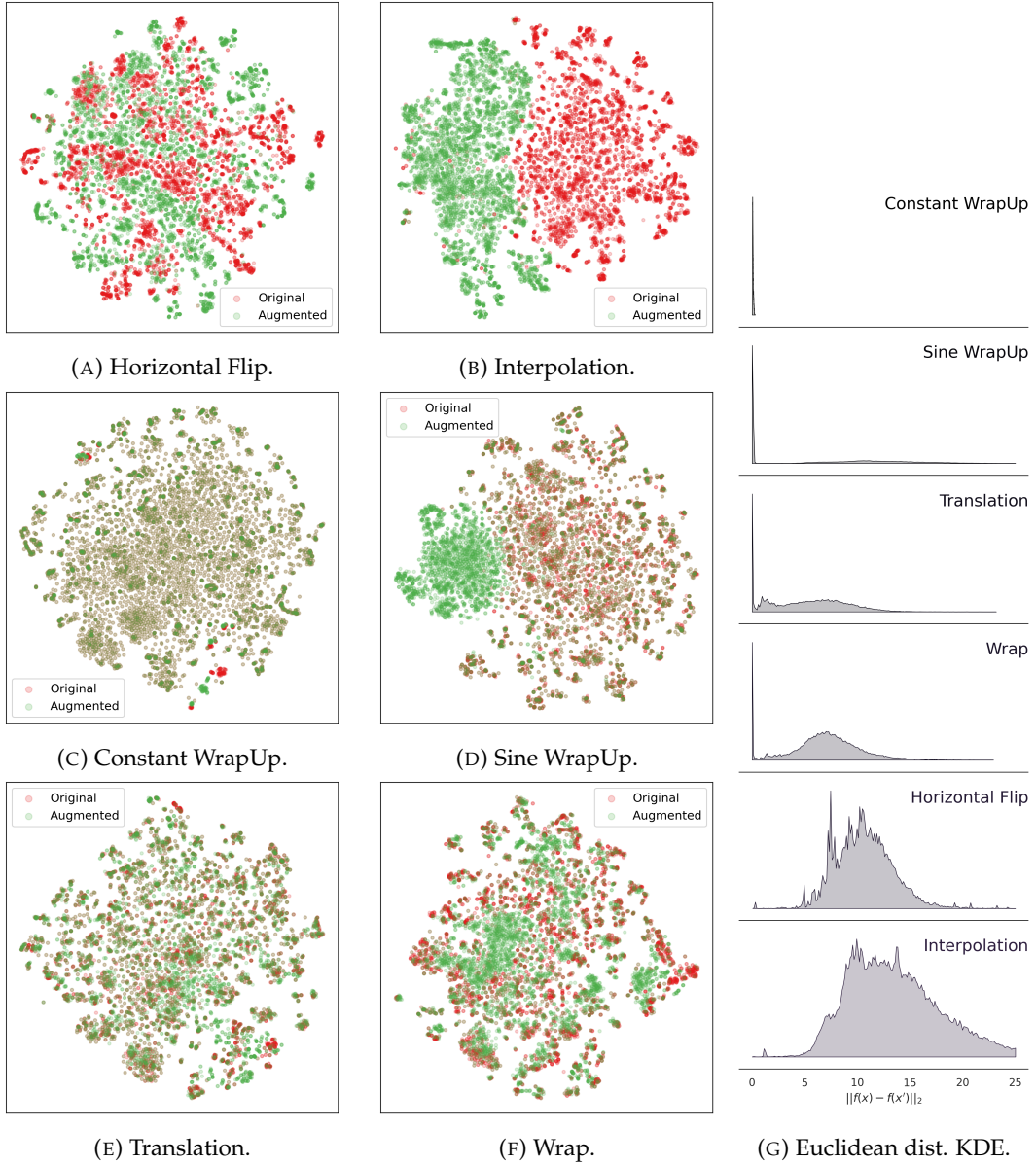


FIGURE 3.8: Comparing original and augmented samples in the latent space (G3).

Augmented-vs-original samples. We complement the previous analysis by investigating original \mathbf{x} and augmented \mathbf{x}' samples relationships. Differently from before, for this analysis original samples are augmented once. Then all points are projected in the latent space $f(\mathbf{x})$ and $f(\mathbf{x}')$ and visualized by means of a 2d t-SNE projection.⁹ We also compute the Kernel Density Estimation (KDE) of the Euclidean distance across all pairs. Figure 3.8 presents the results for 2 top-performing (Translate, Wrap) and 4 poor-performing (Constant Wrapup, Interpolation, Sine Wrapup, Horizontal Flip) augmentations for MIRAGE-19-HALF. Points in the t-SNE charts are plotted with alpha transparency, hence color saturation highlights prevalence of either

⁹Our model architecture uses a latent space of 256 dimensions (see Table 3.5) which the t-SNE representation compresses into a 2d space.

	Augmentation	MIRAGE-19-HALF	MIRAGE-22
Baseline	No Aug	75.43 \pm .10	94.92 \pm .07
Single	Translation	4.40 \pm .13	2.02 \pm .09
	Wrap	4.11 \pm .13	2.09 \pm .08
	Permutation	3.67 \pm .13	1.97 \pm .09
Combined	Ensemble	4.44 \pm .12	2.18 \pm .09
	RandomStack	4.17 \pm .12	2.18 \pm .09
	MaskedStack ($p = 0.3$)	4.45 \pm .13	2.26 \pm .09
	MaskedStack ($p = 0.5$)	4.60 \pm .15	2.24 \pm .09
	MaskedStack ($p = 0.7$)	4.63 \pm .14	2.18 \pm .10

TABLE 3.8: Combining augmentations (G4).

augmented or original samples.

Linking back to the previous observations about Horizontal Flip and Interpolation, results now show the more “aggressive” nature of Interpolation—the t-SNE chart is split vertically with the left (right) side occupied by augmented (original) samples only and the Euclidean distance KDEs show heavier tails. By recalling their definition, while it might be easy to realize why Horizontal Flip is a poor choice—a client will never observe the end of the flow before seeing the beginning, hence they are too artificial—it is difficult to assess a priori the effect of Interpolation. Overall, both augmentations break class semantics.

At the opposite side of the performance range we find augmentations like Sine WrapUp and Constant WrapUp. From Fig. 3.8 we can see that both introduce little-to-no variety—the Euclidean distance distributions are centered around zero. That said, comparing their t-SNE charts we can still observe a major difference between the two transformations which relates to their design. Specifically, Constant WrapUp is applied only to IAT and introduces negligible modifications to the original samples. Conversely, Sine WrapUp is applied on either packet size or IAT. As for Constant WrapUp, the changes to IAT are subtle, while variations of packet size lead to generating an extra “mode” (notice the saturated cluster of points on the left side of the t-SNE plot). In other words, besides the design of the augmentation itself, identifying a good parametrization is very challenging and in this case is also feature-dependent.

Compared to the previous, Translate and Wrap have an in-between behavior—the body of the KDEs show distances neither too far nor too close and the t-SNE charts show a non-perfect overlap with respect to the original samples. Overall, both these augmentations show positive signs of good sample variety.

Takeaways. *Effective transformations operate in a “sweet spot”: they neither introduce too little variety—traditional policies like Random Over Sampling (ROS) and Random Under*

Sampling (RUS) (Johnson and Khoshgoftaar, 2019) are ineffective—nor they break classes semantic by introducing artificial “modes”.

3.6.4 Combining augmentations (G4)

We conclude our analysis by analyzing the impact of combining different augmentations. For this analysis, we selected 3 top-performing augmentations and compared their performance when used in isolation against relying on *Ensemble*, *RandomStack* and *MaskedStack* (see Sec. 3.4.4). Table 3.8 collects results obtained from 80 modeling runs for each configuration. Overall, mixing multiple augmentations is beneficial but gains are small, i.e., $<1\%$.

Takeaways. While one would expect that mixing good augmentations can only improve performance, we note that also CV literature is split on the subject. If on the one hand combining augmentations is commonly done in training pipelines, recent literature shows that such combinations bring marginal benefits (Müller and Hutter, 2021).

3.7 Conclusions

In this chapter we presented a benchmark of hand-crafted DA for TC covering multiple dimensions: a total of 18 augmentations across 3 families, with 3 policies for introducing augmentations during training, investigating the classification performance sensitivity with respect to augmentations magnitude and class-weighted sampling across 3 datasets with different sizes and number of classes. Overall, our results confirm what previously observed in CV literature—*augmentations are beneficial even for large datasets, but in absolute terms the gains are dataset-dependent*. While from a qualitative standpoint, sequence and mask augmentations are better suited for TC tasks than amplitude augmentations, no single augmentation is found superior to alternatives and combining them (via stacking or ensembling), even when selecting top-performing ones, marginally improves performance compared to using augmentations in isolation. Last, by investigating the models latent space geometry, we confirm that *effective augmentations provide good sample variety* by creating samples that are neither too similar nor too different from the original ones which fosters better data representations extraction (as suggested by the longer training time).

Despite the multiple dimensions covered, our work suffers from some limitations. Most notably, it would be desirable to include the larger and more recent datasets like CESNET-TLS22 (Luxemburk and Čejka, 2023a) and CESNET-QUIC22 (Luxemburk et al., 2023a), but such expansion requires large computational power.¹⁰ Still related to using large datasets, we can also envision more experiments tailored to investigate the relationship between datasets size and augmentations. For instance, one

¹⁰For reference, models trained on Enterprise-100 can take up to 6 hours. Since CESNET datasets contains $100\times$ the number of samples of Enterprise-100, performing a thorough exploration of the DA design space is extremely resource demanding.

could sample down a large dataset (e.g., by randomly selecting 1% or 10% of the available samples) and investigate if augmentations result more effective with the reduced datasets. In particular, since Inject shows a positive trend with respect to its intensity N_{inject} we hypothesize that by augmenting a small dataset one can achieve the same performance as using larger datasets—showing these effects are clearly relevant for TC as collecting and releasing large datasets is currently a pain point. Last, our campaigns rely only a CNN-based architecture while assessing DA with other architectures (e.g., Transformer-based for time series (Wen et al., 2023)) is also relevant.

Ultimately, DA modeling campaigns as the one we performed require operating with a grid of configurations and parameters—it is daunting to explore the design space by means of brute forcing all possible scenarios. While domain knowledge can help in pruning the search space, it can also prevent from considering valuable alternatives. For instance, recall that Xie et al., 2023b suggest to use augmentations inspired by TCP protocol dynamics. According to our benchmark, these augmentations are indeed among the top performing ones, yet not necessarily the best ones—navigating the search space results in a balancing act between aiming for qualitative and quantitative results.

We identify two viable options to simplify the design space exploration. On the one hand, re-engineering the augmentations so that their parametrization is discovered during training might resolve issues similar to what observed for Sine Wrap (see Sec. 3.6.3). On the other hand, a more efficient solution would be to rely on generative models avoiding the burden of designing hand-crafted augmentations. More specifically, we envision a first exploration based on conditioning the generative models on the latent space properties learned via hand-crafted DA (e.g., the distance between original and augmented samples should be in the “sweet spot”). Then, we could target the more challenging scenario of training unconditionally and verify if effective representations are automatically learned.

Overall, while our experimental results demonstrate the effectiveness of hand-crafted DA, they likely represent a lower bound on achievable performance. Further improvements could be realized through more advanced techniques, such as contrastive learning to enhance feature representations (chapter 4) and generative models to create more diverse and realistic augmentations (chapter 8). This motivates the exploration of data-driven augmentation strategies in the following chapters.

Chapter 4

Contrastive learning for few-shot learning

4.1 Introduction

Following our systematic evaluation of hand-crafted DA in the traditional supervised setting (Chapter 3), this chapter investigates how such augmentations can be integrated into contrastive learning frameworks to tackle a challenging setting: FSL. While FSL is not strictly required for TC, it is highly desirable in practice, due to the high cost of large-scale data collection and annotation. Therefore in real-world scenarios, being able to train effective classifiers from just a few labeled examples becomes especially valuable. To make learning feasible under such constraints, contrastive learning with suitable augmentations can help — provided the augmentations are well designed to reflect meaningful variations in the data.

FSL involves a training set in which only *a small number of labeled samples for the target classes* are available. In the remainder of this chapter, we refer to the small labeled dataset for the target task as D . Since training a model directly on a small dataset typically leads to poor performance, the FSL setting often relies on a two-stage training strategy. First, a base model is pre-trained on a large-scale dataset D' to learn generalizable representations. Then, this model is fine-tuned on the smaller task-specific dataset D . Essentially, the effectiveness of learning from few samples largely depends on the quality of the representations learned during pre-training.

There are two typical ways to construct the pre-training dataset D' for FSL, under the constraint that labeled samples from the target classes must be excluded, as their inclusion would violate the FSL assumption:

- (i) a large collection of unlabeled samples from the target classes

In this case, although there is no distribution shift between D and D' , since no label is available in D' , the pre-training stage must rely on unsupervised learning.

- (ii) a large collection of labeled samples from non-target classes

In this case, the pre-training stage is able to leverage supervised learning thanks to the class labels, but faces a distribution shift during fine-tuning on the target classes that are unseen in the pre-training dataset D' . This requires the learned model to generalize well to novel classes introduced only in the second stage.

In this chapter, we address these two challenging cases sequentially, using contrastive learning. We begin by introducing the foundation of contrastive learning, including the implementation of the critic and the construction of positive and negative sample pairs (Section 4.2). The former includes the mathematical loss formula, the link between the loss and MI estimation, and the NN design and training paradigm in practice; while the latter details how to use DA functions and class labels to build views from a real batch for computing the loss. Then, to investigate the effectiveness of contrastive learning and augmentation in FSL, we conduct two targeted experimental studies, each addressing one of the two previously described challenging cases:

- (i) Self-supervised contrastive learning on flowpic (Section 4.3)

We replicate and extend an existing study (Horowicz et al., 2022) on self-supervised contrastive learning using the 2D flowpic representation. Given that contrastive loss heavily relies on DA in the self-supervised case (Section 4.2), this experiment serves as a testbed to assess the effectiveness of different hand-crafted augmentations in the self-supervised pretraining. The results on few-shot fine-tuning confirm the value of this approach. Moreover, our statistical analysis reveals that Change RTT and Time Shift significantly outperform other augmentations, underscoring the importance of selecting semantically meaningful transformations.

- (ii) Comparing self-supervised and supervised contrastive learning for adaptation to novel classes (Section 4.4)

For FSL on new target classes, we shift to the time series representation and investigate contrastive learning in a classic N -way- K -shot scenario, where N refers to the number of target classes and K refers to the number of samples per target class, i.e. the fine-tuning task specific dataset D contains $N \times K$ samples in total. Since the pre-training dataset D' is labeled, we have the opportunity to compare self-supervised and supervised contrastive loss for pre-training, together with other existing FSL approaches reviewed in Section 4.4.1. We find that supervised contrastive pre-training on D' followed with fine-tuning the linear classifier head with a cosine distance-based loss on the target dataset D yields the best performance, by effectively leveraging both data and labels for improved generalization to unseen target classes.

Through these two complementary studies, this chapter demonstrates how augmentation-aware contrastive learning can substantially enhance traffic classification performance in data-scarce regimes. By investigating both self-supervised pretraining on flowpic and supervised transfer learning on time series, we show that carefully designed augmentations enable contrastive methods to generalize effectively in few-shot learning scenarios.

4.2 Preliminaries: contrastive learning

In this section, we will review the principles behind contrastive learning, followed by the literature of contrastive learning in traditional classification tasks within both CV and TC.

As presented in Section 2.2.3, a NN classifier is typically a composition of a feature extractor and a classifier head. Especially in the contrastive learning literature (Oord et al., 2019a; Khosla et al., 2020), the classifier head is usually the last linear layer that outputs classification logits, and the feature extractor refers to all layers before the classifier head. In this context, the feature extractor trained with contrastive loss learns how to project the input data into a latent space to group samples of the same class and distance them from other classes samples, and the linear classifier head serves to probe and evaluate what the representations learned by the feature extractor contain. Relying on such geometrical separations, a simple linear classifier suffices to identify classes. Unlike CE loss that jointly trains feature extractor and linear classifier with loss computed at the output of the final linear layer and therefore such geometrical properties in the latent space are *implicitly learned*, contrastive learning aims to *explicitly* enforce geometrical properties in the latent space by means of DAs.

Mathematically speaking, the idea behind contrastive learning is to train only the feature extractor such that it can yield an embedding that separates (contrasts) samples from two different distributions – the joint distribution and the product of marginals. Given a dataset consisting of N pairs $(v_1^i, v_2^i)_{i=1}^N$, where each pair contains two different views of the same class (either created through DA or grouped by class labels), the goal is to learn to contrast "positives" pairs from the joint distribution $x = (v_1^i, v_2^i) \sim p(v_1, v_2)$, versus "negative" pairs from the product of marginals $y = (v_1^i, v_2^j) \sim p(v_1) p(v_2)$.

To realize this contrasting goal, a subset of data $S = \{x, y_1, y_2, \dots, y_k\}$ is selected, containing 1 positive pair and k negative pairs. A "critic" function (a discriminating function) $h_\theta(\cdot)$ is then trained to assign a high value to the positive pair and low values to all the negative pairs, by optimizing the following loss:

$$\mathcal{L}_{\text{contrast}} = -\mathbb{E}_S \left[\log \frac{h_\theta(x)}{h_\theta(x) + \sum_{j=1}^k h_\theta(y_j)} \right]. \quad (4.1)$$

Without loss of generality, the first pair $x = (v_1^0, v_2^0)$ in S is assumed to be positive and all others $y_j = (v_1^j, v_2^j), j = 1, 2, \dots, k$ are negative, then equation 4.1 becomes

$$\mathcal{L}_{\text{contrast}} = -\mathbb{E}_S \left[\log \frac{h_\theta(v_1^0, v_2^0)}{\sum_{j=0}^k h_\theta(v_1^j, v_2^j)} \right]. \quad (4.2)$$

Consequently, this involves with 2 subtasks: implementing the critic $h_\theta(\cdot)$ and constructing the subset S .

4.2.1 Implementation of the critic

The critic $h_\theta(\cdot)$ is implemented with a NN, namely an encoder $f_\theta(\cdot)$ parametrized by θ . To extract compact latent representations of v_1 and v_2 , these two views are first passed through the encoder $f_\theta(\cdot)$ to extract the normalized latent representations $z_1 = f_\theta(v_1) / \|f_\theta(v_1)\|$, $z_2 = f_\theta(v_2) / \|f_\theta(v_2)\|$, so that all vectors lie on the same unit hyper-sphere. Then, their cosine similarity are computed as score, whose range is scaled by a hyper-parameter temperature τ : $h_\theta(v_1, v_2) = \exp(z_1 \cdot z_2 / \tau)$.

Regarding to the hyper-parameter temperature τ , in practice τ is usually lower than 1. The reason of this choice is that, with $x \in [-1, 1]$ being the cosine similarity, the gradient of $f : x \mapsto f(x) = e^{\frac{x}{\tau}}$ is $f'(x) = e^{\frac{x}{\tau}} \frac{1}{\tau}$. For $x \approx 0$ (i.e. hard positives and hard negatives that have $z_1 \cdot z_2 \approx 0$), this gradient becomes $f'(x) \approx \frac{1}{\tau}$, which is larger when τ is smaller. In other words, smaller τ (cold temperature) can help the model learn from hard pairs more efficiently.

Estimator of MI. On the one hand, note that with (i) softmax defined as a function that turns a list of raw scores s into a probability distribution p in which $p_i = \text{softmax}_i(s) = \frac{e^{s_i}}{\sum_{j=0}^k e^{s_j}}$, and (ii) CE loss defined as a function that measure the difference between the ground-truth distribution y (typically a one-hot vector with value of 1 for the correct class and 0 for all other classes) and the predicted distribution p — $\text{CE}(y, p) = -\sum y_i \log(p_i) = -\log(p_{\text{correct class}})$, $\mathcal{L}_{\text{contrast}}$ can also be considered as a softmax CE loss of classifying the correct positive pair out from the given set S , where $p_{\text{correct class}} = p(\text{pos} = 0 \mid S)$ is predicted as $\text{softmax}_0(s) = \frac{h_\theta(v_1^0, v_2^0)}{\sum_{j=0}^k h_\theta(v_1^j, v_2^j)}$. Following this mindset, Chen et al., 2020 termed $\mathcal{L}_{\text{contrast}}$ as normalized temperature-scaled cross entropy loss (NT-Xent).

On the other hand, since the ground truth probability $p_{\text{correct class}} = p(\text{pos} = 0 \mid S)$ should depict the fact that the positive pair coming from the joint distribution $p(v_1, v_2)$ is the first pair (v_1^0, v_2^0) while the k negative pairs coming from the product of marginals $p(v_1) p(v_2)$ are the latter k pairs $(v_1^j, v_2^j), j = 1, 2, \dots, k$ in S , this ground truth probability can be written as

$$\frac{p(v_1^0, v_2^0) \prod_{i \neq 0} p(v_1^i) p(v_2^i)}{\sum_{j=0}^k (p(v_1^j, v_2^j) \prod_{i \neq j} p(v_1^i) p(v_2^i))} = \frac{\frac{p(v_1^0, v_2^0)}{p(v_1^0) p(v_2^0)}}{\sum_{j=0}^k \frac{p(v_1^j, v_2^j)}{p(v_1^j) p(v_2^j)}}$$

By comparing this ground-truth with the prediction, Oord et al., 2019b shows that the optimal value for the critic $h_\theta(v_1, v_2)$ is proportional to (i.e. up to a multiplicative constant) the ratio $\frac{p(v_1, v_2)}{p(v_1)p(v_2)}$. This ratio is known as point-wise MI, where the definition of MI between two RV V_1 and V_2 is $I(V_1; V_2) = \mathbb{E} \left[\log \frac{p(v_1, v_2)}{p(v_1)p(v_2)} \right]$.

Furthermore, by substituting the critic in equation 4.2 with its optimal value $h_\theta^*(v_1, v_2) \propto \frac{p(v_1, v_2)}{p(v_1)p(v_2)}$, it could be proved that the expectation in equation 4.2 is an lower bound of -MI:

$$\begin{aligned} \mathcal{L}_{\text{contrast}}^{\text{opt}} &= -\mathbb{E}_S \log \left[\frac{h^*(v_1^0, v_2^0)}{\sum_{j=0}^k h^*(v_1^j, v_2^j)} \right] \\ &= -\mathbb{E}_S \log \left[\frac{\frac{p(v_1^0, v_2^0)}{p(v_1^0)p(v_2^0)}}{\sum_{j=0}^k \frac{p(v_1^j, v_2^j)}{p(v_1^j)p(v_2^j)}} \right] \\ &= \mathbb{E}_S \log \left[\frac{\frac{p(v_1^0, v_2^0)}{p(v_1^0)p(v_2^0)} + \sum_{j=1}^k \frac{p(v_1^j, v_2^j)}{p(v_1^j)p(v_2^j)}}{\frac{p(v_1^0, v_2^0)}{p(v_1^0)p(v_2^0)}} \right] \\ &= \mathbb{E}_S \log \left[1 + \frac{p(v_1^0)p(v_2^0)}{p(v_1^0)p(v_2^0)} \sum_{j=1}^k \frac{p(v_1^j, v_2^j)}{p(v_1^j)p(v_2^j)} \right] \\ &\approx \mathbb{E}_S \log \left[1 + \frac{p(v_1^0)p(v_2^0)}{p(v_1^0)p(v_2^0)} k \mathbb{E}_{v_1} \left[\frac{p(v_1|v_2)}{p(v_1)} \right] \right] \\ &= \mathbb{E}_S \log \left[1 + \frac{p(v_1^0)p(v_2^0)}{p(v_1^0)p(v_2^0)} k \cdot 1 \right] \\ &\geq -\mathbb{E}_S \log \left[\frac{p(v_1^0, v_2^0)}{p(v_1^0)p(v_2^0)} \right] + \log(k) \\ &= -I(v_1; v_2) + \log(k). \end{aligned} \tag{4.3}$$

We now present several remarks on equation 4.3. First, the inequality in equation 4.3 also holds for critics h that are worse the optimal critic h^* , since h has a worse (higher) $\mathcal{L}_{\text{contrast}} > \mathcal{L}_{\text{contrast}}^{\text{opt}}$, which makes $I(v_1; v_2) \geq \log(k) - \mathcal{L}_{\text{contrast}}^{\text{opt}} \geq \log(k) - \mathcal{L}_{\text{contrast}}$. Second, equation 4.3 signifies that minimization of contrastive loss is actually a maximization of MI between different views of the data. Third, a larger number of negative pairs k is beneficial, since it not only makes the approximation of the expectation with its empirical mean in equation 4.3 more accurate, but also makes $\log(k)$ larger and therefore $\log(k) - \mathcal{L}_{\text{contrast}}$ serves as a tighter lower bound of $I(v_1; v_2)$. Finally,

this lower bound on $MI \log(k) - \mathcal{L}_{\text{contrast}}$ cannot be larger than $\log(k)$, meaning that this bound will be loose when $I(v_1; v_2) > \log(k)$ — a limitation that reflects the high-bias nature of this MI estimator. As a preview, unlike $\mathcal{L}_{\text{contrast}}$, the generative MI estimators introduced in the next chapter do not have this limitation.

NN Design and training. Regarding to the NN implementation in practice, the encoder $f_\theta(\cdot)$ is actually the classifier’s feature extractor $f(\cdot)$ followed by a small non-linear projection head $g(\cdot)$ (typically a MLP with one hidden layer and ReLU non-linear activation). During training, the contrastive loss $\mathcal{L}_{\text{contrast}}$ is computed on the outputs of the projection head, since (Chen et al., 2020) empirically found that the hidden layer before the nonlinear projection head provides more informative representations than the outputs of the projection head, and empirically attributed it to the fact that contrastive loss encourages the projection head’s outputs invariant to views of the same real sample, where views are obtained by applying different DA transformations (see details in Sec. 4.2.2). In this way, transformation-sensitive features like color and orientation in CV are removed in the minimization of contrastive loss, while these information may be useful for downstream classification task. However, with a nonlinear projection head $g(\cdot)$, these information are extracted and preserved in the layer before it.

After the contrastive learning phase, this projector head $g(\cdot)$ is discarded, and only the well-trained feature extractor $f(\cdot)$ is kept. To obtain the final end-to-end classifier, a simple classifier head (e.g. a linear layer) $h(\cdot)$ is trained on top of the frozen feature extractor $f(\cdot)$ with labeled dataset (where this labeled dataset can be small thanks to the powerful representation extraction capacity of the feature extractor). Overall, the more powerful the representation, the lower the number of samples required for training the classifier head. At inference, test accuracy is used as a proxy for representation quality.

4.2.2 Construction of the subset

To construct a subset of data S containing 1 positive pair and k negative pairs, where k should be large to make $\mathcal{L}_{\text{contrast}}$ as a more accurate estimation of MI (details in Sec. 4.2.1) but also in a reasonable size to allow for tractable computation, the approximation of S is typically achieved by augmenting a batch of N real data points. Specifically, for each real data point, we augment it twice using two hand-crafted DA sampled from a pool of DA functions. This ends up with turning the real batch into a multiviewed batch $\{v_1^1, v_2^1, v_1^2, v_2^2, \dots, v_1^N, v_2^N\} = \{v_i^n\}_{n \in [1, N], i \in [1, 2]}$ containing all the $2N$ augmented data points, where $n \in [1, N]$ is the index of the real data point from which v_i^n was created, and $i \in [1, 2]$ is the index of one of the two augmented views of this real data point. In the following, we will index the samples in this multiviewed batch uniformly with $i \in I = \{1 \dots 2N\}$.

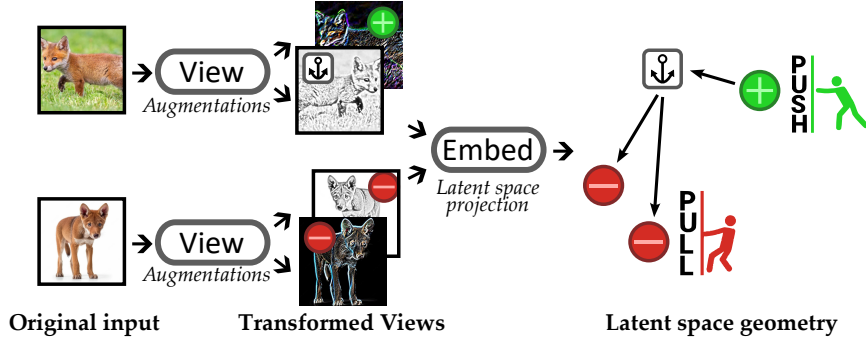


FIGURE 4.1: Contrastive learning principles.

Self-supervised contrastive loss. In unsupervised setting where no ground-truth class label is available, given an anchor v_a indexed with $a \in I$, its positive counterpart is the other augmented view v_p originating from the same real data point as the anchor v_a , and its negatives are all the other $2(N - 1) = 2N - 2$ samples in the multiviewed batch, indexed with $\{i \in I \setminus \{a, p\}\}$. So, each anchor v_a has 1 positive counterpart and $2N - 2$ negative counterparts, and therefore the denominator has $2N - 1$ terms (the positive and negatives) in total. In other words, the contrastive loss aims at distinguishing (contrasting) augmented pair from the same real sample from augmented pairs from different real samples. In this case, equation 4.2 can be written as

$$\mathcal{L}^{\text{self}} = \frac{1}{|I|} \sum_{a \in I} \mathcal{L}_a^{\text{self}} = \frac{-1}{|I|} \sum_{a \in I} \log \frac{\exp(z_a \cdot z_p / \tau)}{\sum_{i \in I \setminus \{a\}} \exp(z_a \cdot z_i / \tau)} \quad (4.4)$$

which is named as SimCLR (Chen et al., 2020).

Figure 4.1 sketches the principles behind the technique. Input samples are transformed into “views” using augmentation functions. Then views are compared using a “contrastive game”: a given view (an anchor) is compared in the normalized latent space against other views of the same real data (positive samples) and all the other views in the batch (negative samples). The contrastive loss aims at pushing an anchor closer to its positives and distancing it from negatives. Figure 4.1 depicts one specific configuration of (anchor, positive, negatives) but as illustrated in equation 4.4 all possible permutations are computed during training.

In $\mathcal{L}^{\text{self}}$, views of the same image form “their own class”. With artificial labels derived automatically by leveraging DA, $\mathcal{L}^{\text{self}}$ is called *self-supervised* contrastive loss. Although it creates a “supervised-style” task from the unsupervised setting, this task is harder than the true supervised task because in $\mathcal{L}^{\text{self}}$ negative samples can be of the same underlying class of the anchor. However, this task complexity is expected to foster the extraction of better representations.

Supervised contrastive loss. In supervised learning, the availability of class labels provides the possibility to modify the loss such that it distinguishes (contrasts) augmented pairs from the same class against those from different classes. To realize this goal, contrastive loss is generalized to include an arbitrary number of positive pairs belonging to the same class. In practice, Khosla et al., 2020 found that the following supervised contrastive loss (namely SupCon) performs well:

$$\mathcal{L}_{\text{out}}^{\text{sup}} = \frac{1}{|I|} \sum_{a \in I} \mathcal{L}_{\text{out},a}^{\text{sup}} = \frac{-1}{|I|} \sum_{a \in I} \frac{1}{|P(a)|} \sum_{p \in P(a)} \log \frac{\exp(z_a \cdot z_p / \tau)}{\sum_{i \in I \setminus \{a\}} \exp(z_a \cdot z_i / \tau)} \quad (4.5)$$

where $P(a) \equiv \{p \in I \setminus \{a\} : \tilde{\mathbf{y}}_p = \tilde{\mathbf{y}}_a\}$ is the set of indices of all positives in the multiviewed batch belonging to the same class as the anchor v_a . Compared to self-supervised contrastive loss (equation 4.4), this supervised variant encourages the encoder to produce tightly clustered representations for all samples belonging to the same class, leading to a more robust clustering in the latent space.

4.2.3 Related work

CV literature. Contrastive learning is a very active research area in ML. As explained in Section 4.2.2, across variants, the self-supervised contrastive loss SimCLR (Chen et al., 2020) is the most popular method. SupCon (Khosla et al., 2020) extends SimCLR for a supervised setting by simply considering as positive samples also all other augmented version of samples belonging to the same class of the selected anchor—it moves from self-similarity to class-similarity. Furthermore, we note that contrastive learning has many variants besides these most popular forms. For example, triplet loss only involves with one positive and one negative, and Bootstrap Your Own Latent (BYOL) (Grill et al., 2020) does not use negative samples but applies two encoders and encourages one encoder’s output of one view to match the other encoder’s output of the other view.

TC literature. The work using contrastive learning for TC is more limited. In (Horowicz et al., 2022), authors applied contrastive learning in a “few-shot learning settings” without episodic training. More specifically, authors trained a self-supervised model using SimCLR and with a flowpic input representation—packet time-series are transformed into images representing the evolution of traffic over a time window; hence transformation is possible by either manipulating the time series (e.g., time shift) or the related image (color jittering, occlusion, etc.)—and transferred it to the supervised classification task using just a few labeled samples. In (Zhao et al., 2022b) instead, authors adopt BYOL (Grill et al., 2020) to a similar problem setting as in (Horowicz et al., 2022) – pre-training on augmented data with contrastive loss and fine-tuning on a few samples. The authors relied on the same dataset as in (Horowicz et al., 2022) but adopted packet time series as their input

(rather than flowpics), leveraging the data transformations proposed by (Rezaei and Liu, 2019) and a ResNet18 architecture. Overall, (Towhid and Shahriar, 2022) shows comparable performance with respect to (Horowicz et al., 2022). Worth also of mention is (Rezaei and Liu, 2019) where authors used augmentations similar to the one used in (Horowicz et al., 2022), yet no contrastive learning was applied. Last, a few more recent studies investigated contrastive learning on raw packet bytes as input (Meng et al., 2022; Zhao et al., 2022b) and compared it against transfer learning and meta-learning Guarino et al., 2023, highlighting its recent relevance.

4.3 Self-supervised contrastive learning on flowpic

Supervised learning requires large labeled datasets. As they are notoriously difficult to share and labeling is costly, the ability to learn from as few labeled samples as possible is particularly appealing. In this direction, to analyze contrastive learning and DAs' effect in the unsupervised setting, in this section we aim at replicating Horowicz et al., 2022, which in the remainder of this section we will also refer to as REF-PAPER or Horowicz et al.. In REF-PAPER, Horowicz et al. quantify the performance of classification tasks when using only up to 100 training samples yet increasing the training dataset with synthetic samples created with DA functions. Horowicz et al. consider both a supervised and an unsupervised setting, the former with CE serve for finding the most effective augmentations, which are then applied on the latter using the popular self-supervised *contrastive learning* approach SimCLR (Chen et al., 2020) that starts from pre-training a model in an unsupervised setting and later fine-tunes it to address a target task using a small number of labeled samples. Rather than using packet time series, Horowicz et al. use a *flowpic input representation*, i.e., a 2d summary of a network flow dynamics (visualized in Sec. 2.1).

Overall, we identify two DA and CT related contributions from the REF-PAPER: (i) a benchmark of the effect on the performance of CNN models across 7 DAs (including DA techniques applied to either flowpics (e.g., rotation) or to the packets time series (e.g., altering inter-arrival times) from which the flowpics are then computed), tested on two mid-sized datasets—UCDAVIS19 (Rezaei and Liu, 2019), which contains 5 classes with up to $\approx 1,000$ samples per class, and ISCX (New Brunswick, 2016), which were processed and combined to obtain 10 classes with a few flows each. The REF-PAPER shows that simple DAs can indeed be beneficial even when using a few samples (with authors preferring time series transformations over image-related ones). (ii) an evaluation of SimCLR and its sensitivity to the number of samples used during fine-tuning. These are interesting findings worth reproducing.

More specifically, during the replication of the REF-PAPER by employing the same

methodologies and data, we aim to achieve the following research goals: (G1) reproducing the benchmarking of the proposed DA techniques for flowpics as a preliminary selection step for their use in contrastive learning, in which we aim for a quantitative reproduction (G1.1) and a qualitative reproduction (G1.2) of the augmentations ranks; (G2) reproducing the SimCLR-related results with special attention to their internal details (e.g., use of dropout, projection layer size, training set size and impact of the combination of augmentations used). We note that, since the REF-PAPER does not provide confidence intervals nor perform any kind of statistical analysis of their results, G1 and G2 not only reproduce the original results but also add a layer of statistical significance to them.

Given the replicability and reproducibility goals clarified, in Sec. 4.3.1, we present our experimental protocol. Then in Sec. 4.3.2 - Sec. 4.3.4, we detail the analysis and results for the research goals sequentially. Finally, we conclude this section with final remarks in Sec. 4.3.5. Details that we believe to be needed to make the section self contained are deferred to App. A.

4.3.1 Experimental protocol

We closely followed the configurations and scenarios from the REF-PAPER which we complemented with ablation studies and modeling campaigns. In this section we provide a summary of the main aspects of REF-PAPER.

DL architectures. We adopted the same CNN-based networks of the REF-PAPER, namely a LeNet5 (Lecun et al., 1998) for *mini*-flowpic and a larger version of it for *full*-flowpic.¹ The printout of the networks is reported in App. A.

DA. Next to applying no augmentation, we adopted the 6 augmentations used in the REF-PAPER—3 packet time series transformations (Change RTT, Time Shift and Packet Loss) and 3 image transformations (Rotation, Horizontal Flip, and Color jitter)—with the same hyper-parameters (see (Horowicz et al., 2022) for details).

Training steps. As in the REF-PAPER, we compared two DL modeling techniques: *fully supervised* training and SimCLR + *fewshot fine-tune* training. For the former, samples are augmented before starting the training. For the latter, given a labeled dataset and a selected augmentation function, each sample is processed to create 2 views of it using the Change RTT and Time shift transformations. Both views are created when forming the mini batches used during training. First, a representation of the dataset is learned by *pre-training* a model via SimCLR, contrasting pairs of augmented “views” of a sample. Then, a new model is formed by freezing the pre-trained representation and combining it with a classifier layer which is fine-tuned based on a few labeled samples. As in the REF-PAPER, we use Change RTT and Time Shift as

¹The terminology in the REF-PAPER overloaded as mini- and full-flowpic also refer to the resolution of the flowpic created.

Name	Partition	Filter	Classes	Flows				Pkts mean
				all	min	max	ρ	
(Rezaei and Liu, 2019) UCDAVIS19	pretraining	none	5	6,439	592	1,915	3.2	6,653
	human			83	15	20	1.3	7,666
	script			150	30	30	1.0	7,131
(Aceto et al., 2019c) MIRAGE-19-HALF	n.a.	none	20 ^(*)	122,007	1,986	11,737	5.9	23
		>10pkts		64,172	1,013	7,505	7.4	17
(Guarino et al., 2021b) MIRAGE-22	n.a.	none	9	59,071	2,252	18,882	8.4	3,068
		>10pkts		26,773	970	4,437	4.6	6,598
		>1,000pkts		4,569	190	2,220	11.7	38,321
(Heng et al., 2021a) UTMOBILENET21	4-into-1	none	17	34,378	159	5,591	35.2	664
		>10pkts	10	9,460	130	2,496	19.2	2,366

ρ : ratio between max and min number of flows—the larger the value, the higher the class imbalance;

(*) Despite being advertised of having traffic from 40 apps, the *public* version of the dataset only contains 20 apps.

TABLE 4.1: Summary of datasets properties.

DA functions, yet we complement the analysis testing other augmentation pairs too. Augmentations are used only during pre-training.

Datasets and Data curation. To address our goals we used the four datasets summarized in Table 4.1. UCDAVIS19 is used in the REF-PAPER while we selected the others because of their interesting and complementary properties with respect to UCDAVIS19: (i) they are collected in similar setups — research projects related to mobile traffic monitoring — (ii) they cover a larger number of classes and users behavior — MIRAGE-19-HALF and UTMOBILENET21 are gathered from volunteering students interacting with instrumented phones while MIRAGE-22 focuses on video meeting services — and (iii) they are imbalanced — the ρ values in the table reflect the ratio between the number of samples of the largest and smallest class in a dataset; notice the larger imbalance of the three datasets compared to UCDAVIS19, which is an expected property of network traffic. More importantly, all these datasets provide per-packet time series for the whole flows duration, which is a key requirement for composing flowpic representations. For instance, we cannot use the larger AppClassNet (Wang et al., 2022a) and CESNET-TLS22 (Luxemburk and Čejka, 2023a) datasets because they only provide the packet time series for the first 20-30 packets of each flow.

As detailed in the table, UCDAVIS19 is pre-partitioned (and pre-filtered) by the authors of the dataset to create a large set of samples for self-supervised training (namely pretraining) and two smaller testing set partitions, namely script and human.² Conversely, for the other three datasets we filtered out flows with less than 10 packets and removed classes with less than 100 samples. To replicate the setting provided in UCDAVIS19, for MIRAGE-19-HALF and MIRAGE-22 we also first removed TCP ACK packets from time series and then discarded flows related to background

²According to (Rezaei and Liu, 2019), both pretraining and script correspond to automated collection of data, while human is gathered monitoring traffic when real users were interacting with the selected 5 services.

traffic.³ The right-most column of the table details the average number of packets in a flow. Notice how UCDAVIS19 has very long flows while MIRAGE-19-HALF is the dataset with shortest ones. To further focus on very long flows, we also created a version of MIRAGE-22 with flows having more than 1,000 packets. Lastly, through our curation we also created reference train/test splits for the datasets. Specifically, since in the REF-PAPER the training dataset needs to have 100 samples, for UCDAVIS19 we create 5 folds (the smallest class in the dataset has 592 flows) of 100 samples per-class each. However, for the other datasets we opted for having 5 random splits each having a random selection of 80% of samples for training (and the rest for testing).

4.3.2 Reproducing quantitative results of data augmentation

We start by reproducing results related to Tables 1–2 of REF-PAPER, which contrast different augmentations applied in a supervised setting (G1.1).

Approach. As from Sec. 4.3.1, we created k splits from UCDAVIS19 dataset by sampling without replacement groups of 100 samples for each class from the pretraining partition. Then, a given set of 100 samples is split randomly s times, with each split corresponding to a 80/20 train/validation split for training. Using these data, we performed a campaign to test the 7 augmentations across $k=5$ splits each having $s=3$ train/validation splits for a total of 105 experiments. This is repeated for the three flowpic resolutions with the same training settings as in the REF-PAPER: static learning rate at 0.001, early stopping on validation loss after 5 steps in which the loss does not improve by more than 0.001, batch size of 32, performance measured via accuracy, flowpic created from the first 15s of a flow.

Results. Table 4.2 summarizes our results reporting the mean accuracy and related 95% CI for each scenario. We complement the evaluation of the REF-PAPER by reporting a new test set corresponding to all pretraining samples not belonging to a selected 100 samples split (i.e., what would be called a test set in a traditional evaluation). As, to the best of our understanding, these samples have been discarded in the REF-PAPER, we refer to this test set as `leftover`.

Overall, we obtained lower performance than what was reported in REF-PAPER. While differences are modest on `script`, we observe a reduction of over 20% on `human`. Notice that no gap appears when comparing `script` with `leftover`. The gap is (slightly) reduced when using a higher resolution flowpic but the lower performance on `human` (and the larger confidence intervals with respect to `script` and `leftover`) suggests the presence of a hidden problem with this predefined test set. To understand the reason of this gap, we conjectured that its root case might lie in the data itself.

³Traffic is collected on mobile phones with labeling ground-truth provided by `netstat`. One measurement experiment generates traffic logs for a specific target app. We processed such logs so that traffic of apps and services different from the target app (e.g., `netd` daemon, `SSDP`, `Android gms`) is removed as it represents “background” traffic.

<i>flowpic res</i>	Test on script			Test on human			Test on leftover [†]		
	32	64	1500	32	64	1500	32	64	1500
No augmentation	95.64 \pm 0.37	95.87 \pm 0.29	94.93 \pm 0.72	68.84 \pm 1.45	69.08 \pm 1.35	69.32 \pm 1.63	95.78 \pm 0.29	96.09 \pm 0.38	95.79 \pm 0.51
Rotate	96.31 \pm 0.44	96.93 \pm 0.46	95.69 \pm 0.39	71.65 \pm 1.98	71.08 \pm 1.51	68.19 \pm 0.97	96.74 \pm 0.35	97.00 \pm 0.38	95.79 \pm 0.31
Horizontal flip	95.47 \pm 0.45	96.00 \pm 0.59	94.89 \pm 0.79	69.40 \pm 1.63	70.52 \pm 2.03	73.90 \pm 1.06	95.68 \pm 0.40	96.32 \pm 0.59	95.97 \pm 0.80
Color jitter	97.56 \pm 0.55	97.16 \pm 0.62	94.93 \pm 0.68	68.43 \pm 2.82	70.20 \pm 1.99	69.08 \pm 1.72	96.93 \pm 0.56	96.46 \pm 0.46	95.47 \pm 0.49
Packet loss	96.89 \pm 0.52	96.84 \pm 0.63	95.96 \pm 0.51	70.68 \pm 1.35	71.33 \pm 1.45	71.08 \pm 1.13	96.99 \pm 0.39	97.25 \pm 0.39	96.84 \pm 0.49
Time shift	96.71 \pm 0.60	97.16 \pm 0.49	96.89 \pm 0.27	70.36 \pm 1.63	71.89 \pm 1.59	71.08 \pm 1.33	97.02 \pm 0.50	97.51 \pm 0.46	97.67 \pm 0.29
Change RTT	97.29 \pm 0.35	97.02 \pm 0.46	96.93 \pm 0.31	70.76 \pm 1.99	71.49 \pm 1.59	71.97 \pm 1.08	98.38 \pm 0.18	97.97 \pm 0.39	98.19 \pm 0.22

Each of *our* result is an aggregation of 15 experiments (5 splits \times 3 seeds).

[†] We named “leftover” the samples from the *pretraining* partition not belonging to the 100 samples of a given split. Traditionally this would correspond to the test set.

TABLE 4.2: Comparing data augmentation functions in a supervised training. Values marked as “ours” correspond to the average accuracy across 15 modeling experiments and the related 95-th confidence intervals.

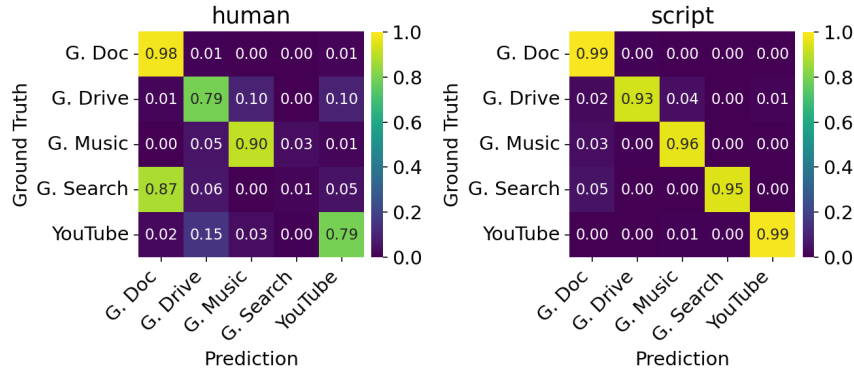


FIGURE 4.2: Average confusion matrices for the 32×32 resolution across all experiments in Table 4.2.

To start verifying this assumption, the heatmaps in Fig. 4.2 break down the results in Table 4.2 by showing the average per-class accuracy across the 105 runs for the 32×32 flowpic resolution. Specifically, we summed all the confusion matrices for script and human and we normalized them by row. For human we observe multiple sources of confusion with *Google doc* and *Google search* having the most evident clash. Conversely, no specific issues can be detected for script.

We highlight that, while 32×32 and 64×64 experiments run in about 1 min, it takes about 30min to run one experiment on 1500×1500 . Given this computational cost, motivated by the marginal performance gap across resolutions and as done by Horowicz et al., in the remainder of this chapter we focus only on the 32×32 resolution.

To drill down, Fig. 4.3 collects an average flowpic per class across the original dataset partitions and one training split. Recall that the horizontal axis of a flowpic corresponds to time (time zero on the left) while the vertical axis corresponds to packet sizes (zero length on the top). The first row in Fig. 4.3 corresponds to all flows available in the pretraining partition, while the second one corresponds to a training split, i.e., an aggregation of 100 samples per class. We can clearly see that the reduction of samples has a visual impact, but overall the first two rows are visually very similar.

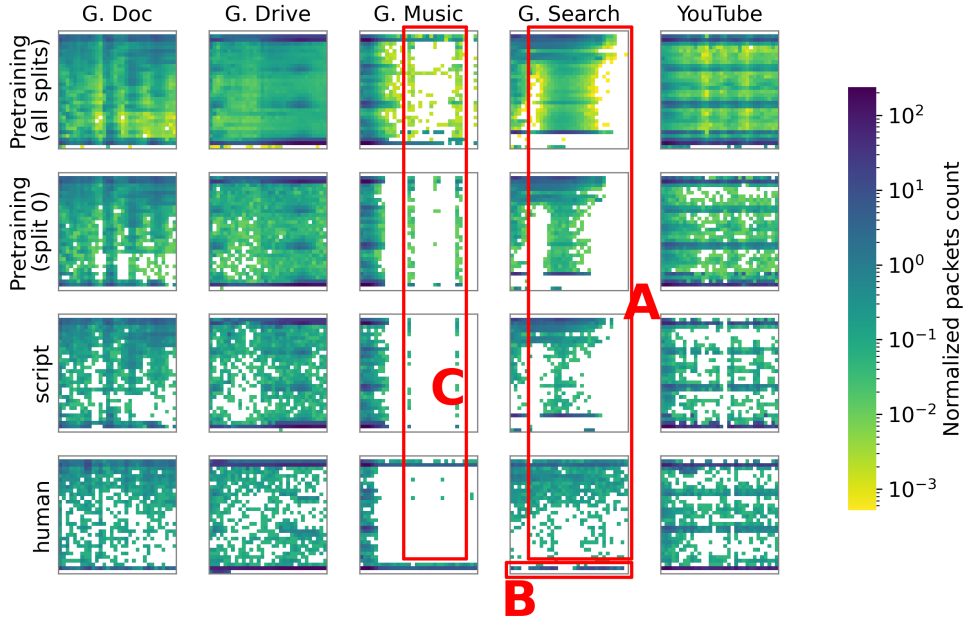


FIGURE 4.3: Average 32×32 flowpic for each class across dataset partitions.

The third and the fourth rows correspond to the *script* and *human* partition respectively, i.e., they have 30 and ≈ 15 samples per class. When comparing the last two rows with the first two, we can clearly see differences which we further annotate with rectangles. Notice how *Google search* is expected to have two vertical groups of pixels around the left-axis and the center of the picture. Surprisingly, for *human* these groups are “shifted” to the right (rectangle A). Moreover, notice how all splits but *human* saturate the maximum packet size for *Google search*—there is a distinctive horizontal line (around pixels on row 28) for *human* (rectangle B) while in the other cases there are distinct dark lines at row 32. Interestingly, Fig. 4.3 also highlights macroscopic differences for *Google music*—vertical “stripes” of pixels are visible in all splits but *human* (rectangle C). Yet, according to Fig. 4.2, this seems less of a problem. We conjecture that this might be due to the stark difference between *Google music* and the other services. In other words, despite the different behavior between the partitions, *Google music* is still very different from the other 4 classes (thus it might be easier to classify).

Takeaway. Given the strong evidence provided by our analysis, we concluded that the *human* test split is affected by a data-shift. Yet, we cannot comment on the reason why this was not detected in the REF-PAPER.⁴

4.3.3 Reproducing qualitative ranking of data augmentation

The original key question behind benchmarking the different augmentations was to understand if, and by how much, they were beneficial with respect to not performing

⁴Authors did not provide us comments about this aspect.

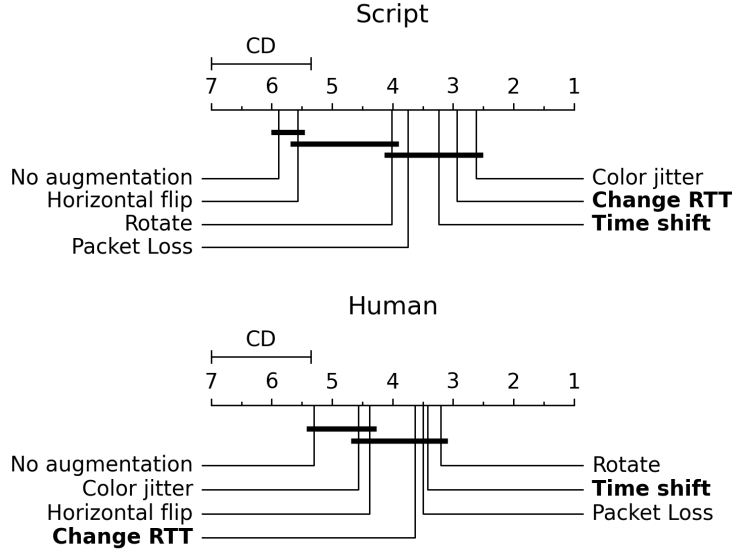


FIGURE 4.4: Critical distance plot of the accuracy obtained with each augmentation for the 32×32 and 64×64 resolutions. Augmentations joined by a horizontal line are not statistically different. The lower the ranking (closer to 1, the right side of the plot) the better the performance. Transformations highlighted in bold are selected as the best performing one in the REF-PAPER.

any augmentation, for which we opted for performing a statistical analysis (G1.2) of our modeling campaign to understand if Change RTT and Time shift were the best performing augmentations as reported in the REF-PAPER.

Approach. The CI values in Table 4.2 show clear overlaps between different augmentations. To investigate our results, we compare each augmentation according to the procedures presented in (Demšar, 2006). First, accuracy results are turned into rankings (e.g., if augmentations A, B and C yield an accuracy of 0.9, 0.7 and 0.8, their associated rankings would be 1, 3, and 2) with ties being assigned with the average ranking of the group (e.g., if augmentations A, B and C yield 0.9, 0.9 and 0.8, their associated rankings would be 1.5, 1.5 and 3). This process is repeated across all tested datasets and splits. Then, an average ranking value is extracted per augmentation. These values are compared pairwise using a post-hoc Nemenyi test, which compares these average rankings to decide if the performance difference between augmentations is significant. This decision is made using a Critical Distance (CD) in ranking equal to $CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$, where q_α is based on the Studentized range statistic divided by $\sqrt{2}$, k is equal to the number of augmentations compared and N is equal to the number of samples used.

Results. Figure 4.4 displays the results of these comparisons. We combined the 32×32 and 64×64 resolutions as we did not find statistically significant differences between them. In our case, with $\alpha = 0.05$, $k = 7$ and $N = 30$ and $q_{0.05} = 2.949$ the critical distance is $CD = 1.644$. The closer an augmentation is to the right side of

the plot (a higher average rank), the better the performance. From our analysis for the script partition, we cannot conclude significant differences within three groups, which we sort by increasing performance: {No augmentation and Horizontal flip}; {Horizontal flip and Rotate}; {Rotate, Packet loss, Time shift, Change RTT and Color jitter}. Similar groups exist also for the human partition. As annotated in Fig. 4.4, Horowicz et al. selected **Change RTT** and **Time shift** as the best augmentations: whereas these augmentations are in the best performing group both for script and human, it is easy to gather that other transformations consistently appear in the same (statistically relevant) group.

Takeaway. On the one hand, the Time shift and Change RTT transformations are in the best performing group, a finding aligned with the ones in the REF-PAPER. On the other hand, from a statistical viewpoint, they are not distinguishable from other options, like Color jitter (for script) or Rotate (for human) or Packet Loss (for both).

4.3.4 Reproducing constrastive learning results

The second goal of our reproducibility study concerns the use of contrastive learning and fine-tuning (G2). A few observations are needed to contextualize the modeling campaign to perform.

Approach. First of all, we need to select augmentations for SimCLR, where two augmented views are obtained from each sample in a training mini-batch. Following the REF-PAPER, we opted for *applying the two transformations Time shift and Change RTT in random order for every image in a mini-batch*. Yet, given our ranking analysis showed equivalence among multiple top performing transformations, we also perform a small-scale ablation study considering three other pairs beside the pair selected in the REF-PAPER. Second, with respect to the projection head used after the feature extractor in the training of SimCLR, our ablation on the usage of dropout and the dimension of the projected representation (results in Table 4.4) shows that we can rely on a network *without* dropout (differently from the REF-PAPER) but we confirm the original choice of a projection layer of 30 units.

Results. First, we investigated to which extent alternative pairs of augmentations affect the fine-tuning performance. Namely, we considered *Time shift* and *Change RTT* next to *Rotate* and *Color jitter*, selected because they achieved good positions in our ranking analysis. Then we formed groups by either pairing time series with image transformations or pairing the image transformations. Results collected in Table 4.3 show that, despite the punctual differences between pairs, our observation on Table 4.2 and the ranking analysis (Sec 4.3.3) still holds—all pairs are *qualitatively* equivalent.

Second, we expand the methodology used so far by quantifying the effect of using a (pre)training set larger than 100 samples. Specifically, we created 5 random

1st augment.	Change RTT*	Packet loss		Change RTT		Color Jitter
2nd augment.	Time shift*	Color jitter	Rotate	Color Jitter	Rotate	Rotate
test on script	92.18 \pm 0.31	90.17 \pm 0.41	91.94 \pm 0.30	91.72 \pm 0.36	92.38 \pm 0.32	91.79 \pm 0.34
test on human	74.69 \pm 1.13	73.67 \pm 1.24	71.22 \pm 1.20	75.56 \pm 1.23	74.33 \pm 1.26	71.64 \pm 1.23

Each value is an aggreg. of 125 exp. (5 splits \times 5 SimCLR seeds \times 5 fine-tune seeds).

(*) pair of augmentations used in (Horowicz et al., 2022).

TABLE 4.3: Comparing the fine-tuning performance when using different pairs of augmentations for pretraining (32 \times 32 resolution, fine-tuning on 10 samples only).

Projection dimension	test on script		test on human	
	w dropout	w/o dropout	w/ dropout	w/o dropout
30	91.81 \pm 0.38 [†]	92.18 \pm 0.31	72.12 \pm 1.37 [‡]	74.69 \pm 1.13
84	92.02 \pm 0.36	92.54 \pm 0.33	73.31 \pm 1.04	74.35 \pm 1.38

Each value is an aggr. of 125 exp. (5 splits \times 5 SimCLR seeds \times 5 fine-tune seeds).

The reference value for [†] from Horowicz et al., 2022 reports in the text (94.5% for 10 samples); for [‡] no specific values are reported but should be \approx 80% based on Fig. 4 of Horowicz et al., 2022.

TABLE 4.4: Impact of dropout and SimCLR projection layer dimension on fine-tuning (32 \times 32 only, with 10 samples for fine-tuning training).

80/20 train/validation split using the full pretraining partition, i.e., the dataset result imbalanced with up to 1,532 training samples for the largest class and 473 for the smallest. Table 4.5 reports the results of the modeling campaign in both a supervised and contrastive learning settings. As expected, compared to Table 4.2 and Table 4.4, enlarging the dataset is effective in improving performance in both settings. In particular, for contrastive learning the gain is smaller for script (+1.72% on average) than for human (+5.76% on average)—the latent space created via contrastive learning is better at mitigating the data shift.

Supervised		script	human
	No augmentation	98.37 \pm 0.19	72.95 \pm 0.96
	Rotate	98.47 \pm 0.25	73.73 \pm 1.09
	Horizontal flip	98.20 \pm 0.15	74.58 \pm 1.16
	Color jitter	98.63 \pm 0.21	72.47 \pm 1.02
	Packet loss	98.63 \pm 0.19	73.43 \pm 1.25
	Time shift	98.60 \pm 0.22	73.25 \pm 1.17
	Change RTT	98.33 \pm 0.16	72.47 \pm 1.04
	SimCLR + fine-tuning	93.90 \pm 0.74	80.45 \pm 2.37

^a

TABLE 4.5: Accuracy on 32 \times 32 flowpic when enlarging training set (without dropout).

^aEach value is an aggregation of 20 experiments (20 different seeds)

Takeaway. The transformations selected in the REF-PAPER constitute a good enough choice, although image transformations cannot be fully ruled out based on our assessment. This confirms that identifying the most suitable transformations is tied to the input representation and datasets used, which remains an open problem. Moreover, while a very limited number of samples can be enough for training models, the same scenarios can benefit from more data—the selected augmentations alone are not a final replacement for real input samples.

4.3.5 Concluding remarks

In this section, the task we focused on is the FSL case (i), where the encoder is first trained on abundant unlabeled samples S of target classes, then finetuned on a few labeled samples S' of target classes. To tackle this FSL setting in TC, we reproduced and replicated the methodology of (Horowicz et al., 2022) which investigated self-supervision via contrastive learning and DA. These methods are particularly appealing as they allow for learning from a few labeled samples instead of requiring large amounts of labeled training data.

Summarizing our analysis, we have been able to *qualitatively* reproduce most of the original results, so we confirm the interest in self-supervised contrastive learning and DA. At the same time, our modeling campaigns found unexpected *quantitative* discrepancies that we rooted in data shifts in the UCDAVIS19 dataset (undetected in the REF-PAPER).

Another remarkable consideration can be gathered by contrasting our reproducibility vs replicability results. Indeed, the reproducibility results on UCDAVIS19 show little statistical significance in the differences among the proposed DA techniques—just by reproducing the study on UCDAVIS19 alone would therefore have not allowed us to validate Horowicz et al.’s choices. Conversely, by replicating the methodology on three additional datasets, we gathered evidence that finally validated Change RTT and Time Shift as more beneficial than other augmentations for the flowpic input representation.

4.4 Comparing self-supervised and supervised contrastive learning for few shot learning

Building on Sec. 4.3, which confirmed the effectiveness of self-supervised contrastive learning for the FSL case (i) — pretraining on unlabeled data, this section extends the evaluation to both self-supervised and supervised contrastive learning in the case (ii) — pretraining on labeled data of non-target classes. We note that instead of sticking onto the flowpic representation adopted in Sec. 4.3 where the motivation was to replicate the REF-PAPER, in this section we return back to packet time series

as input data representation which enables early classification and is more widely adopted.

Recall that for this specific FSL case, despite all training samples are associated with their own class labels (i.e., supervised), the data split is different from that in traditional supervised classification. To clarify these differences, we begin with a brief introduction to commonly used notations for defining the training splits.

In FSL case (ii), the goal is to classify samples of *new categories* efficiently from few samples only. Each few-shot task (also called target tasks) consists of a support set $\mathcal{S} = \{(x_i^s, y_i^s)\}_{i=1}^{|\mathcal{S}|}$ containing $|\mathcal{S}|$ labeled samples and a query set $\mathcal{Q} = \{(x_j^q, y_j^q)\}_{j=1}^{|\mathcal{Q}|}$ containing $|\mathcal{Q}|$ samples to be classified. Note that \mathcal{S} and \mathcal{Q} share the same set of N classes (i.e. $\mathcal{C}_{\mathcal{S}} = \mathcal{C}_{\mathcal{Q}}$), plus \mathcal{S} contains K samples per class (also denoted as N -way K -shot), and these two sets' samples are mutually exclusive (i.e., $\mathcal{S} \cap \mathcal{Q} = \emptyset$). The goal is to train a classifier on \mathcal{S} that could accurately predict the labels of \mathcal{Q} at inference. However, as K is much smaller in FSL compared to the training set size in traditional supervised learning, it would be difficult to achieve good test performance on \mathcal{Q} if the classifier is only trained on the small \mathcal{S} . To address that, in the setting of FSL, during training we also have access to a large labeled base dataset \mathcal{D}_b , but there is no intersection between the base classes in \mathcal{D}_b and the novel classes in \mathcal{S} (or \mathcal{Q}) (i.e. $\mathcal{C}_{\mathcal{D}_b} \cap \mathcal{C}_{\mathcal{S}} = \emptyset$). This leads to the challenge of FSL: what training algorithm could effectively transfer knowledge from a large dataset of source classes (i.e. source task \mathcal{D}_b) to the sparsely annotated target new categories (i.e. target task $(\mathcal{S}, \mathcal{Q})$)?

In this section, we start with a discussion of the mainstream training algorithms in this FSL context (Sec. 4.4.1). Then, to benchmark the FSL methods on TC datasets and investigate the impact of incorporating label information (as in supervised contrastive learning) on the model's ability to generalize from limited examples, we state our research questions (Sec. 4.4.2) used when designing our experiments (Sec. 4.4.3). We conclude by discussing our results (Sec. 4.4.4) and outlining the final conclusions (Sec. 4.4.5).

4.4.1 Related work

Next, we will introduce the two main branches of FSL training paradigms – *transfer learning* and *meta-learning*, each complimented with a review of CV and TC literature as summarized in Table 4.6 and Table 4.7.

Transfer learning approaches. As sketched in Fig. 4.5, transfer learning solve FSL problem in two steps, involving (i) first train a conventional NN classifier (i.e. a feature extractor f_θ followed by a classifier head f_ϕ) $f_\phi^{(source)} \circ f_\theta^{(source)}$ on the source data \mathcal{D}_b , (ii) then adapt it to the few-shot target data \mathcal{S} by training only a simple linear or centroid classifier head $f_\phi^{(target)}$ on the fixed representation yielded by the

	Approach	Classifier	Distance-based?
Transfer Learning	(Chen et al., 2019b) Baseline	Linear	○
	(Chen et al., 2019b) Baseline++	Cosine Sim.	●
Meta Learning	(Snell et al., 2017) ProtoNet	Euclidean distance	●
	(Sung et al., 2018) RelationNet	MSE	○
	(Finn et al., 2017) MAML	Linear	○
Contrastive Learning	(Chen et al., 2020) SimCLR	Linear	●
	(Khosla et al., 2020) SupCon	Linear	●

TABLE 4.6: Computer vision literature summary.

Reference	Approach	Data	Classes		Shots	Inp.Type	Inters?	
		Year	All	Target				
Transf. Learn.	(Rezaei and Liu, 2020) <i>Razaei20</i>	Multi-task learn.	18	5	5	n.a.	3 PS	●
	(Sun et al., 2018) <i>Sun18</i>	TrAdaBoost	05	12	12	n.a.	– FF	●
Meta Learn.	(Ouyang et al., 2021) FS-IDS	ProtoNet	14	8	3	1:10	26 FF	●
	(Rong et al., 2021) UMVD-FSL	ProtoNet	17,20	≈76	5/20	1/5	784 B	○
	(Yu and Bian, 2020) <i>Yu20</i>	ProtoNet	09,15	≈8	2/5	50	44 FF	●
	(Liang et al., 2022) OICS-VFSL	ProtoNet	09,15	≈12	2/4/7	1:50	—FF	●
	(Zheng et al., 2020) RBRN	RelationNet	12,16	≈15	—	—	—B	●
	(Xu et al., 2020) FCNet	RelationNet	12,17	≈10	2	5/10	200 B	●
	(Zhao et al., 2022a) Festic	RelationNet	18,19	≈25	14	5:15	256 B	●
	(Feng et al., 2021) FCAD	MAML	17	43	13	5:20	33FF+8PS	○

InpType: FF = flow features; B = payload bytes; PS = univariate packet time series; — = unspecified
Inters? target and base classes are ○disjoint or either ●completely or ●partially overlapped.

TABLE 4.7: Traffic classification literature summary.

pretrained frozen feature extractor $f_{\theta}^{(source)}$. Alternatively, in step (ii) one can fine-tune all parameters at once, or use a hybrid policy unfreezing feature extractor after a certain number of epochs. No matter the selected option, dataset partitions are the same: (i) pretraining entirely scanned the source dataset \mathcal{D}_b and (ii) finetuning enlists the target support set \mathcal{S} whose categories are disjoint with respect to \mathcal{D}_b .

- CV literature: Transfer learning is the most adopted methodology across literature. In particular, while early meta-learning literature proves it is possible to learn even from single samples (Snell et al., 2017), more recent literature (Chen et al., 2019b; Tian et al., 2020) suggests transferring from models created on large dataset (with many samples and many classes) yields better performance, especially when combined with fine-tuning with episodic training for the final target task, i.e., they mix transfer learning with meta-learning.

A prominent example is Baseline (Chen et al., 2019b) which trains a M_{source}

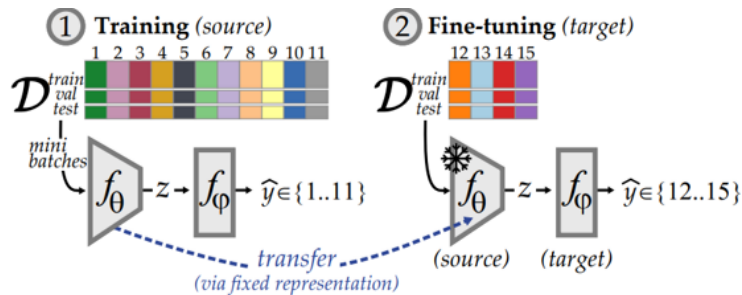


FIGURE 4.5: Transfer learning.

model using a large dataset (many classes and many samples) and then fine-tune a target task (from a disjoint set of classes) via episodic training. Baseline++ (Chen et al., 2019b) is a variant of the same approach where the parameters of the classifier are treated as *class embedding*. More in details, both methods rely on a fully connected layer $\mathbf{W} \in \mathbb{R}^{d \times c}$ where d and c represent the dimension of the latent space vectors and the number of target classes respectively. Unlike Baseline that simply uses a standard CE classification loss, Baseline++ relies on a cosine distance-based loss between $\mathbf{W} = [\mathbf{w}_1 \cdots \mathbf{w}_c]$ columns and latent space projections \mathbf{z} of input samples—each column \mathbf{w}_i "embeds" a class.

- TC literature: Only (Rezaei and Liu, 2020; Sun et al., 2018) used transfer learning but none of the methods mentioned above. Namely, (Rezaei and Liu, 2020) pre-trained a self-supervised multi-task model targeting flows duration and bandwidth which was then transferred to a 5 classes task. Results show that this transfer was under-performing with respect to training directly a single-task model. Instead, (Sun et al., 2018) used TrAdaBoost (Dai et al., 2007), an ensemble ML method using reversed boosting, considering a very old dataset.

Meta-learning approaches. While transfer learning relies on *implicit* tasks affinity, meta-learning is designed to *explicitly* push cross-tasks representation extraction. To do so, during an epoch, rather than completing a scan of \mathcal{D}_b by means of random mini-batches, synthetic tasks (also called episodes) \mathcal{T}_i are created by randomly sampling a subset of N training classes $\mathcal{C}_i^{(train)}$ and randomly sampling support samples \mathcal{S}_i and query \mathcal{Q}_i samples for these selected classes, in which \mathcal{S}_i contains K samples per class. Notice that while episodes are small batches of samples, by design they differ from monolithic training mini-batches as they guarantee class balance – an episode has $(|\mathcal{S}| + |\mathcal{Q}|)$ samples for the (randomly selected) N classes (ways). Since episodic training mimics the target task scenario on the source set, it is also known as *(N-way, S-shot) training*, where support \mathcal{S}_i and query \mathcal{Q}_i acts as “micro-training” and “micro-validation” set. Through dual sampling of task and data space, meta-learning is enabled to construct a large number of auxiliary tasks related to unseen tasks.

In practice, given a dataset, we first split it into source dataset \mathcal{D}_b and meta-test set \mathcal{D}_{test} with disjoint set of classes (i.e. $\mathcal{D}_b \cap \mathcal{D}_{test} = \emptyset$), then \mathcal{D}_b is further divided into meta-training set \mathcal{D}_{train} and meta-validation set \mathcal{D}_{val} with disjoint set of classes (i.e. $\mathcal{D}_{train} \cap \mathcal{D}_{val} = \emptyset$). Training episodes are sampled from \mathcal{D}_{train} , while the evaluation (being validation or testing) creates multiple models (one per episode, where episode is sampled from \mathcal{D}_{val} or \mathcal{D}_{test}), so overall performance is an aggregation of per-episode models performance. As in traditional monolithic training, meta-validation facilitates both hyper-parameters tuning and over-fitting assessment, while meta-testing yields the final performance. Note that meta-validation

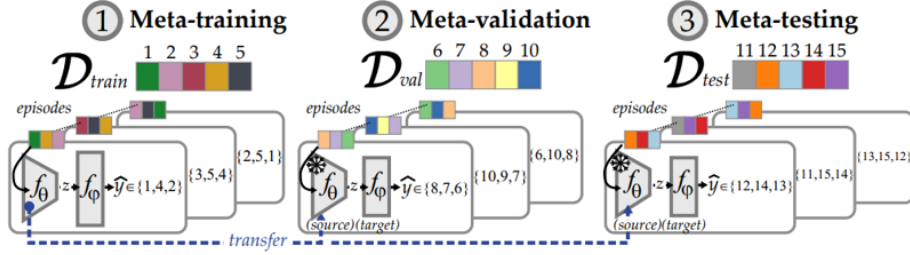


FIGURE 4.6: Meta-learning.

and meta-testing still includes fine-tuning the models for the target tasks, by transferring the meta-learned source model feature extractor $f_{\theta}^{(source)}$ and fine-tune a classifier head specific for an episode, as pictured in Fig. 4.6. In a nutshell, meta-training is designed to push representation learning through tasks variety.

- CV literature: CV literature is ripe with meta-learning and FSL methods (Wang et al., 2020c). In this study we focus on a small selection of methods based on their extreme popularity.

ProtoNet (Snell et al., 2017) is the most well known metric-based meta-learning approach. ProtoNet learns class prototype which geometrically corresponds to the mean centroid of a class in the latent space. Query samples are then classified based on their euclidean distance with respect to class prototypes. The idea of class prototypes inspired different meta-learning methods, including Baseline(ClassEmb) (i.e., class embedding are semantically equivalent to class prototypes).

RelationNet (Sung et al., 2018) is another popular metric-based meta-learning method. While ProtoNet uses a closed form distance metric (i.e., euclidean distance), RelationNet introduces the idea of “meta-learning” such distance. Specifically, the classifier head $f_{\phi}(\cdot)$ embodies a “relation” module trained to provide a similarity score between support and query samples. Curiously, the classification loss is based on Mean Squared Error (MSE) rather than softmax.

MAML (Finn et al., 2017) is the most popular optimization-based meta-learning approach. Differently from ProtoNet and RelationNet which optimize model parameters considering episodes in isolation, MAML uses a two-nested loops process: the inner loop fine-tunes based on each individual episode; the outer loop “re-weights” inner loop contributions across episodes via a second order gradient of the classification loss.

- TC literature: As from Table 4.7, several studies successfully applied these methods on network traffic, mostly targeting normal-vs-attack classification tasks in intrusion detection scenarios. While we consider those classifiers as specific forms of TC, we find *most of these studies violate the meta-learning principle of dis-joining train/val/test partitions*. Only (Rong et al., 2021; Feng et al.,

2021) followed the expected protocol, while in all the other studies the partitions overlapped either perfectly (e.g., meta-train a binary classifier normal-vs-attack and meta-test on the same classes) or partially (e.g., normal and the same attack classes traffic belongs to both meta-train and meta-test).

4.4.2 Research questions

TC literature favors meta-learning with respect to transfer learning (Table 4.7). Yet, CV literature suggests to pay attention to the latter. Moreover, all previous TC literature on meta-learning provides positive results, but most of these studies violate the meta-learning principle of dis-joining the classes in \mathcal{D}_{train} , \mathcal{D}_{val} and \mathcal{D}_{test} . Thus, we claim the need to re-assess these methodologies under different settings.

Q1: Do the benefits, as observed in CV, of transferring from source models trained on many samples and many classes apply to TC use-cases as well? Among the FSL methods, is transfer learning yielding better performance compared to meta-learning?

For transfer learning’s pre-training stage specifically, since contrastive learning is designed to facilitate better representation learning and has been shown to be successful for time series (Yang et al., 2022c) outside the TC domain, we ask

Q2: Does using contrastive learning in transfer learning’s pre-training assist in creating more general models also for TC? Is supervised contrastive learning superior to its more traditional self-supervised version?

4.4.3 Experimental protocol

To address our research questions we relied on two publicly available datasets and designed multiple modeling campaigns.

Datasets and dataset partitioning. The two considered datasets are MIRAGE-19-ALL and AppClassNet. MIRAGE-19-ALL (Aceto et al., 2019d) encompasses per-biflow traffic logs of 40 Android apps (20 in public and 20 acquired by asking the authors) collected at the ARCLAB laboratories of the University of Napoli Federico II. It was collected by instrumenting 3 Android devices used by ≈ 300 volunteers (students and researchers) interacting with the selected apps for short sessions. AppClassNet (Wang et al., 2022b) is an anonymized commercial-grade dataset released by Huawei Technologies in 2022 and gathered from real residential and enterprise networks. The dataset encompasses the traffic of 500 applications for a total of 10M biflows each represented by a time series of packet-size and direction of the first 20 packets and, most important, labeled by means of a commercial and proprietary DPI tool.

Table 4.8 summarizes datasets properties. Defining the popularity of a class as its number of samples, the table reports ρ measuring the datasets imbalance as the ratio

Data Partition	MIRAGE-19-ALL					AppClassNet				
	Num. Classes	Samples			ρ	Num. Classes	Samples			ρ
\mathcal{D}_{train}	24	82k	8.2k	1.3k	6.3	320	9.8M	1M	958	1,044
\mathcal{D}_{val}	8	9.4k	1.3k	1.1k	1.2	80	60.5k	956	579	1.6
\mathcal{D}_{test}	8	5.1k	904	361	2.5	100	47.4k	578	383	1.5
\mathcal{D}_{all}	40	97k	8.2k	361	22.7	500	9.9M	$\approx 1M$	383	2,611

ρ = ratio *Max/Min* samples per class

TABLE 4.8: Datasets summary.

between the most popular and least popular class. Unlike CV datasets, in TC the imbalance is severe. Adhering to meta-learning protocols, we partition the datasets by dis-joining train/val/test. To do so, we use class popularity resulting in \mathcal{D}_{train} containing the largest classes pool—imbalance here still reflects network traffic scenarios and the large availability of data allows to address (Q1)—while \mathcal{D}_{val} and \mathcal{D}_{test} focus on unpopular classes—imbalance here is reduced, better reflecting the typical FSL settings. We argue that such partitioning is preferable to both (i) artificially random under-sampling to enforce a “few-shot” setting and (ii) randomly splitting classes, obtaining scenarios where a target class has many samples. Conversely, we aim at target tasks with a naturally reduced number of samples.

Input type. All models are created using packet time series as input. Specifically, for MIRAGE-19-ALL we consider 4 features (packet size, direction, IAT, and TCP window-size⁵) for the first 10 packets; for AppClassNet we consider 2 features (packet size and direction) for the first 20 packets.⁶

NN Architectures. About the feature extractor, we used a CNN architectures containing 4 CNN blocks (where each block is a convolution layer followed by a batch normalization layer and a ReLU activation): and a fully connected layer of 500 units (1.3M parameters).

FSL Approaches. We considered a total of 10 methods (Q1): 3 for transfer learning in the literature, 3 for meta-learning, and 4 for transfer learning with contrastive learning. The transfer learning and meta-learning methods we used are reported in Table 4.6, however, for transfer learning, we modify original names to better express their relationship and semantic: Baseline++ \rightarrow Baseline(ClassEmb), RFS-simple(NN) \rightarrow Baseline(NN).⁷

For transfer learning with contrastive learning, we considered both self-supervised (SimCLR) and supervised (SupCon) variants (Q2), for models with and without

⁵For UDP traffic the time series is padded with 0.

⁶The reason for the different time series length resides in properties of the datasets: MIRAGE-19-ALL contains many short leaved flows, thus using 10 packets reduces padding.

⁷We acknowledge that Baseline is a sub-optimal naming convention, but we kept it to preserve the relationship with (Chen et al., 2019b).

		MIRAGE-19-ALL			AppClassNet		
Approach		5-shots	50-shots	200-shots	5-shots	50-shots	200-shots
Transf. Learn.	Baseline	60.24 \pm 0.57	82.26 \pm 0.41	88.72 \pm 0.32	77.30 \pm 0.65	90.11 \pm 0.38	93.03 \pm 0.32
	Baseline(ClassEmb)	59.98 \pm 0.54	79.03 \pm 0.42	84.48 \pm 0.38	76.54 \pm 0.61	89.16 \pm 0.41	92.30 \pm 0.32
	Baseline(NN)	65.48 \pm 0.56	86.41 \pm 0.36	92.84 \pm 0.24	76.93 \pm 0.61	90.18 \pm 0.39	93.77 \pm 0.29
Meta. Learn.	MAML	57.10 \pm 0.58	70.68 \pm 0.44	73.97 \pm 0.44	61.93 \pm 0.71	75.57 \pm 0.60	78.27 \pm 0.55
	ProtoNet	62.62 \pm 0.56	69.93 \pm 0.41	72.09 \pm 0.40	69.93 \pm 0.74	80.31 \pm 0.51	81.94 \pm 0.50
	RelationNet	54.28 \pm 0.58	57.73 \pm 0.51	61.60 \pm 0.47	68.65 \pm 0.74	77.24 \pm 0.59	75.09 \pm 0.57
Transf. with Contr. Learn.	SimCLR	63.97 \pm 1.01	79.26 \pm 0.76	81.52 \pm 0.63	77.88 \pm 2.05	91.10 \pm 1.16	91.65 \pm 1.11
	SupCon	64.72 \pm 0.83	86.55 \pm 0.50	91.00 \pm 0.39	81.74 \pm 1.07	91.70 \pm 0.64	93.33 \pm 0.51
	SimCLR(ClassEmb)	62.82 \pm 0.98	86.91 \pm 0.57	91.01 \pm 0.43	78.27 \pm 2.17	92.05 \pm 1.08	93.35 \pm 0.92
	SupCon(ClassEmb)	66.42 \pm 0.84	87.01 \pm 0.48	91.87 \pm 0.37	81.05 \pm 1.09	93.75 \pm 0.53	95.94 \pm 0.44

TABLE 4.9: Comparing transfer, meta- and contrastive learning in N -shots 4-ways classification (i.e. N training samples for each of the 4 target classes) (p.s. Query set is fixed as 15 samples per class)

Baseline’s class embedding (4 variants). We also randomly apply 4 transformations: *horizontal flip* reverses the order of packets (1st become last, etc.); *shuffle* randomly re-orders packets; *tail-occlusion* masks the second half of an input time series with zeros; *Gaussian noise* adds noise sampled from a normal distribution $\epsilon \sim \mathcal{N}(0, 1)$ to each time series value.

4.4.4 Results

Comparison between transfer learning and meta-learning. Transfer learning methods are better performing than meta-learning (Q1). In particular, on AppClassNet all methods are within a $\pm 1\%$ gap, while for MIRAGE-19-ALL we observe $\pm 8.36\%$ between the best and worst performing methods with 200 shots. Recall that most of these methods differ mostly for the classifier $f_\varphi(\cdot)$. In particular, on AppClassNet we can rank their performance as *class embedding* < *linear layer* < *nearest neighbor*. However, on MIRAGE-19-ALL class embedding is the worst while nearest neighbor has +4% gap compared to the 2nd performing classifier (logistic regression). This hints that the latent space representation learned on MIRAGE-19-ALL is worse than the one learned on AppClassNet despite the very different task complexity (32-vs-400 classes)—a nearest neighbor classifier is more flexible than a linear one, which possibly justifies the better performance on MIRAGE-19-ALL if classes are not well separated; yet, a nearest neighbour classifier needs to carry training data with the model in order to have labeled “anchors” to use for the classification.

Contrastive learning. As suspected, by leveraging label information, supervised contrastive learning outperforms the traditional version—e.g., up to +10.35% and 3.98% on MIRAGE-19-ALL and AppClassNet respectively (Q2). Moreover, while class embedding where ineffective for transfer learning, they are beneficial for contrastive learning.

4.4.5 Concluding remarks

In this section, we focused on the challenging FSL classification on new classes with few labeled training samples, we evaluated transfer learning, meta-learning and the application of contrastive learning in transfer learning, using two publicly available datasets with larger classes variety than in previous TC literature. Differently from previous TC literature but aligned with previous CV literature, our results show that meta-learning methods are the worst performing methods (Q1), while transfer learning based on episodic fine-tuning is a better option—training with many samples (and many classes) is the best option to create M_{source} models; these can then be used to fine-tune M_{target} models using 100s samples. Moreover, our findings highlight that contrastive learning with DAs, especially supervised contrastive learning (SupCon), outperforms its self-supervised counterpart SimCLR and other methods by effectively leveraging class label information to enhance representation learning (Q2). These insights underscore the potential of contrastive learning techniques in improving the accuracy and efficiency of encrypted TC.

Building on our investigation of contrastive learning for encrypted traffic’s few-shot classification and considering that contrastive learning relies implicitly on maximizing MI between views or classes to learn effective representations (Sec. 4.2), in the next part we will turn our attention to explicit MI estimation, using generative approaches rather than discriminative ones. Furthermore, we will explore the application of MI on improving generation performance.

Part II

Mutual information for conditional generation

Chapter 5

Introduction to generative models

Generative models are a class of machine learning models designed to learn the underlying distribution of the data in order to generate new synthetic samples that resemble the original data. These models differ from discriminative models described in [Part I](#) which focus on finding a mapping between an input with the related label. In other words, discriminative models learn to model the decision boundary between different classes rather than the data distribution itself.

Some well-known examples of generative models include VAE (Kingma and Welling, [2022](#)) and GAN (Goodfellow et al., [2014](#)). While these methods have proven effective in generation tasks, in this part we focus on two specific types of generative models: Diffusion Models (DMs) and Rectified Flow (RF) models. DM are generative models that progressively add noise to data and then learn to reverse this noising process to generate new samples, with recent models like SD2 (Rombach et al., [2022](#)) showcasing their ability to generate high-quality data. RF is a flow-based generative model that minimizes the kinetic energy of the flow, and has been adopted in several state-of-the-art (SOTA) models such as SD3 (Esser et al., [2024](#)) and Flux (FLUX, [2023](#)) to enhance their performance. These models utilize stochastic processes and flow matching techniques to generate high-quality samples, making them increasingly popular in the generative modeling landscape.

This chapter provides the necessary background to understand the core principles of DM and RF models, laying the foundation for their application in advanced generative tasks in the following chapters.

In [Section 5.1](#), we first introduce the fundamentals of DM, starting from continuous time and covering both its forward and reverse processes. Then, we move on to the discrete version, namely Denoising Diffusion Probabilistic Model (DDPM), introducing the related forward and reverse Markov chains, along with the training objective and guided generation technique Classifier-Free Guidance (CFG) to enhance sample quality during sampling.

In [Section 5.2](#), we move to the RF models starting from core concepts related to Conditional Flow Matching (CFM), introducing the definition of flow, velocity field,

and probability path, and discussing their relationship. We then discuss how to learn a parametric velocity field that matches the ground truth using CFM, with a detailed explanation of the associated loss function and condition signal. Finally, we explore generation with a guidance signal, where labeled data allows for the generation of conditioned samples, and introduce CFG in the context of CFM.

Through these sections, we will establish a deep understanding of diffusion models and rectified flow models, enabling their application in complex generative tasks and providing a solid theoretical foundation for the next chapter.

5.1 Preliminaries – diffusion model

A DM is a probabilistic generative model that pre-defines a forward process to gradually add random noise to a set of data transforming its unknown distribution into a simple known prior distribution (e.g., standard Gaussian), and then learns to denoise this process so that it is possible to generate synthetic samples starting from a random point selected from the known prior. Specifically, given a data distribution $p(x, 0)$ at time $t = 0$, a prior distribution $p(x, T)$ at time $t = T$, and a hand-designed forward process that progressively transforms $p(x, 0)$ into $p(x, T)$, to model the denoising reverse process in order to generate samples, a NN is optimized to approximate the score function $\nabla_x \log p(x, t)$, which is defined as the gradient of the log probability density. Both the hand-designed forward process and the modeled reverse process can be implemented in two forms: as a continuous-time SDE or as a discrete-time Markov chain (as shown in Fig. 5.1), where the latter's limit, when the step size is infinitely small, becomes the former. In this section, we will provide a brief introduction to the basic principles of DM, clarifying the following points sequentially:

1. Starting with the generalized continuous-time SDE formulation, we consider a forward SDE with predefined drift and diffusion functions, along with its corresponding Fokker-Planck Equation (FPE). From this, we derive the drift and diffusion terms of the reverse SDE.
 - (a) Crucially, we show that if the score function at each time step t is known, the reverse SDE becomes fully specified, making it possible to generate samples. This motivates the need to parametrize the score function.
 - (b) Additionally, we show that by introducing a specific hyperparameter, the level of stochasticity in the reverse process can be controlled — allowing it to operate either as a SDE or as a probability flow ODE, both of which share the same marginal distributions.
2. Next, we consider the discretized version of a continuous-time forward SDE with specified drift and diffusion — namely, the VP-SDE. We show that a pre-defined discrete-time forward Markov chain converges to the continuous-time

VP-SDE in the infinite-step limit, and we build a link between the added noise and the score function.

From the discrete-time forward Markov chain, we then derive the transition kernel of the corresponding reverse Markov chain, $q(\mathbf{x}_{t-1} | \mathbf{x}_t)$, which becomes fully determined once the added noise is known. This motivates modeling the added noise using a NN trained with an Mean Squared Error (MSE) loss, thereby linking noise prediction to score-based modeling. Furthermore, in the conditional generation setting, the predicted noise can be shifted to better emphasize the conditioning signal.

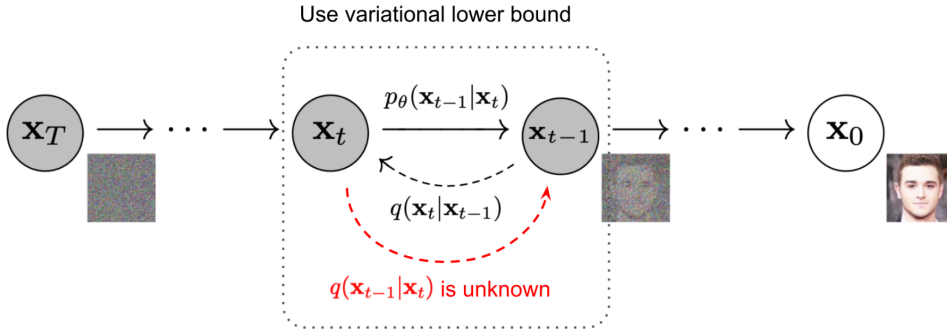


FIGURE 5.1: The discrete-time Markov chain of forward (reverse) diffusion process of generating a sample by slowly adding (removing) noise. (Ho et al., 2020a)

5.1.1 Continuous version

Forward process. Consider an SDE of the form

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)dt + \mathbf{g}(\mathbf{x}, t) \cdot d\mathbf{W}_t \quad (5.1)$$

with corresponding FPE

$$\partial_t p(\mathbf{x}, t) = -\nabla_{\mathbf{x}} \cdot (\mathbf{f}(\mathbf{x}, t)p(\mathbf{x}, t)) + \frac{1}{2} \nabla_{\mathbf{x}}^2 : (\mathbf{g}(\mathbf{x}, t)\mathbf{g}^T(\mathbf{x}, t)p(\mathbf{x}, t)), \quad (5.2)$$

where p denotes the Probability Density Function (PDF) of the random process, and $:$ denotes the Frobenius inner product between matrices. For simplicity, we assume \mathbf{g} is position-independent and diagonal (i.e., $\mathbf{g}(\mathbf{x}, t) = \mathbf{g}(t)$, $\mathbf{g}^T(t)\mathbf{g}(t) = \mathbf{g}^2(t)$), then the SDE Equation (5.1) becomes

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)dt + \mathbf{g}(t) \cdot d\mathbf{W}_t, \quad (5.3)$$

and the corresponding FPE Equation (5.2) simplifies to

$$\partial_t p(\mathbf{x}, t) = -\nabla_{\mathbf{x}} \cdot (\mathbf{f}(\mathbf{x}, t)p(\mathbf{x}, t)) + \frac{1}{2} \mathbf{g}^2(t) \nabla_{\mathbf{x}}^2 : p(\mathbf{x}, t) \quad (5.4)$$

Now consider a probability flow ODE

$$dx = \tilde{f}(x, t) dt, \quad \text{where} \quad \tilde{f}(x, t) := f(x, t) - \frac{1}{2} g^2(t) \nabla_x \log p(x, t). \quad (5.5)$$

with the corresponding FPE

$$\partial_t p(x, t) = -\nabla_x \cdot \left[\left[f(x, t) - \frac{1}{2} g^2(t) \nabla_x \log p(x, t) \right] p(x, t) \right]. \quad (5.6)$$

Since the SDE and the ODE have the same marginal PDF evolution according to their respective FPE, we aim to rewrite their FPE in a unified form, by introducing a parameter $\eta \in [0, 1]$:

$$\partial_t p(x, t) = -\nabla_x \cdot \left[\underbrace{\left[f(x, t) - \frac{\sqrt{1-\eta^2}}{2} g^2(t) \nabla_x \log p(x, t) \right]}_{\text{denote as } f_\eta(x, t)} p(x, t) \right] + \frac{1}{2} \underbrace{\eta^2 g^2(t)}_{\text{denote as } g_\eta^2(t)} \nabla_x^2 : p(x, t) \quad (5.7)$$

of which the original SDE's FPE Equation (5.4) is a special case with $\eta = 1$, and the ODE's FPE Equation (5.6) is a special case with $\eta = 0$. The corresponding SDE for general η is:

$$dx = f_\eta(x, t) dt + g_\eta(t) \cdot dW_t, \quad (5.8)$$

which forms a family of equivalent SDE.

Reverse process. We start with rewritting the right-hand side of the forward process' unified FPE Equation (5.7) as

$$\begin{aligned} \partial_t p(x, t) &= -\nabla_x \cdot (f_\eta(x, t) p(x, t)) + \frac{1}{2} g_\eta^2(t) \nabla_x^2 : p(x, t) \\ &= -\nabla_x \cdot \left[\left[f_\eta(x, t) - g_\eta^2(t) \nabla_x \log p(x, t) \right] p(x, t) \right] + \frac{1}{2} g_\eta^2(t) \nabla_x^2 : p(x, t) \\ &= -\nabla_x \cdot \left[\underbrace{\left[f_\eta(x, t) - g_\eta^2(t) \nabla_x \log p(x, t) \right] p(x, t)}_{= -g_\eta^2(t) \nabla_x \cdot (p(x, t) \nabla_x \log p(x, t))} \right] + \frac{1}{2} g_\eta^2(t) \nabla_x^2 : p(x, t) \\ &= -g_\eta^2(t) \nabla_x \cdot \left(p(x, t) \frac{\nabla_x p(x, t)}{p(x, t)} \right) \\ &= -g_\eta^2(t) \nabla_x^2 : p(x, t) \\ &= -\nabla_x \cdot \left[\underbrace{\left[f_\eta(x, t) - g_\eta^2(t) \nabla_x \log p(x, t) \right] p(x, t)}_{\text{denote as } -\mu_\eta(x, T-t)} \right] - \frac{1}{2} \underbrace{g_\eta^2(t)}_{\text{denote as } \sigma_\eta^2(T-t)} \nabla_x^2 : p(x, t) \\ &= +\nabla_x \cdot (\mu_\eta(x, T-t) p(x, t)) - \frac{1}{2} \sigma_\eta^2(T-t) \nabla_x^2 : p(x, t) \end{aligned} \quad (5.9)$$

Since the relation between the forward PDF $p(x, t)$ and the reverse PDF $q(x, T-t)$ is

$$\begin{cases} p(x, t) = q(x, T-t) \\ \partial_t p(x, t) = -\partial_{T-t} q(x, T-t), \end{cases} \quad (5.10)$$

Injecting Equation (5.10) into Equation (5.9) leads to

$$\partial_{T-t}q(\mathbf{x}, T-t) = -\nabla_{\mathbf{x}} \cdot (\mu_{\eta}(\mathbf{x}, T-t)q(\mathbf{x}, T-t)) + \frac{1}{2}\sigma_{\eta}^2(T-t)\nabla_{\mathbf{x}}^2 : q(\mathbf{x}, T-t) \quad (5.11)$$

Finally, denoting $T-t$ as s turns Equation (5.11) to

$$\partial_s q(\mathbf{x}, s) = -\nabla_{\mathbf{x}} \cdot (\mu_{\eta}(\mathbf{x}, s)q(\mathbf{x}, s)) + \frac{1}{2}\sigma_{\eta}^2(s)\nabla_{\mathbf{x}}^2 : q(\mathbf{x}, s) \quad (5.12)$$

whose corresponding SDE is

$$d\mathbf{x} = \mu_{\eta}(\mathbf{x}, s)ds + \sigma_{\eta}(s)dW_s. \quad (5.13)$$

This SDE describes the reverse process: as s increases, t decreases. By leveraging $ds = d(T-t) = \frac{d(T-t)}{dt}dt = -dt$, the reverse SDE Equation (5.13) can be rewritten as

$$d\mathbf{x} = -\mu_{\eta}(\mathbf{x}, s)dt + \sigma_{\eta}(s)dW_s. \quad (5.14)$$

Since the original forward SDE Equation (5.3)'s drift $\mathbf{f}(\mathbf{x}, t)$ and diffusion $\mathbf{g}(t)$ are pre-defined, we aim to express the reverse SDE's drift $\mu_{\eta}(\mathbf{x}, s)$ and diffusion $\sigma_{\eta}(s)$ with them. By leveraging Equation (5.9) and Equation (5.8), and denoting $\nabla_{\mathbf{x}} \log p(\mathbf{x}, t)$ as the score function \mathbf{s} of the forward process, we have

$$\left\{ \begin{array}{l} \mu_{\eta}(\mathbf{x}, s) = \mathbf{g}_{\eta}^2(t)\nabla_{\mathbf{x}} \log p(\mathbf{x}, t) - \mathbf{f}_{\eta}(\mathbf{x}, t) \\ \quad = \eta^2 \mathbf{g}^2(t)\nabla_{\mathbf{x}} \log p(\mathbf{x}, t) - \left[\mathbf{f}(\mathbf{x}, t) - \frac{1-\eta^2}{2} \mathbf{g}^2(t)\nabla_{\mathbf{x}} \log p(\mathbf{x}, t) \right] \\ \quad = \eta^2 \mathbf{g}^2(t)\mathbf{s} - \mathbf{f}(\mathbf{x}, t) + \frac{1-\eta^2}{2} \mathbf{g}^2(t)\mathbf{s} \\ \quad = \frac{1+\eta^2}{2} \mathbf{g}^2(t)\mathbf{s} - \mathbf{f}(\mathbf{x}, t) \\ \sigma_{\eta}(s) = \mathbf{g}_{\eta}(t) = \eta \mathbf{g}(t). \end{array} \right. \quad (5.15)$$

From which we have two observations:

Remark 1. The introduced parameter η controls the amount of stochasticity during sampling: in the case $\eta = 1$, the reverse process is SDE, which results in the same diffusion process $\{\mathbf{x}_s\}_{0 \leq s \leq T}$ as the forward-time SDE Equation (5.3) if the initial condition is chosen as $\mathbf{x}_{s=0} \sim p_T$; in the case $\eta = 0$, the reverse process is ODE, which is the time-reversal of the forward-time ODE Equation (5.5) and therefore yields a deterministic process with the same marginal densities.

Remark 2. If one can learn the score function $\mathbf{s} = \nabla_{\mathbf{x}} \log p(\mathbf{x}, t)$, then one can sample from the distribution p_0 by simulating the process \mathbf{x}_s forward in time s (i.e. \mathbf{x}_t backwards in time t) with the reverse SDE (Equation (5.13)). In practice, \mathbf{s} is approximated with a NN $\mathbf{s}_{\theta}(\mathbf{x}, t)$ (namely a score-based model). This is at the core of score-based diffusion generative models.

Furthermore, if we assume $\mathbf{f}(\mathbf{x}, t) = \mathbf{f}(t)\mathbf{x}$, then

$$\begin{cases} \mu_\lambda(\mathbf{x}, s) = \frac{1+\eta^2}{2}\mathbf{g}^2(t)\mathbf{s} - \mathbf{f}(t)\mathbf{x} \\ \sigma_\lambda(s) = \eta\mathbf{g}(t) \end{cases} \quad (5.16)$$

In the Variance-Preserving (VP) forward SDE, we assume $\mathbf{f}(t) = -\frac{1}{2}\beta_t$, $\mathbf{g}(t) = \sqrt{\beta_t}$, where $\beta_t = \beta_{\min} + \frac{t}{T}(\beta_{\max} - \beta_{\min})$. Then Equation (5.16) becomes

$$\begin{cases} \mu_\lambda(\mathbf{x}, s) = \frac{1+\eta^2}{2}\beta_t\mathbf{s} + \frac{1}{2}\beta_t\mathbf{x} \\ \sigma_\lambda(s) = \eta\sqrt{\beta_t} \end{cases} \quad (5.17)$$

5.1.2 Discrete version of VP-SDE (i.e. DDPM)

Building on the continuous VP-SDE, we now turn our focus to DDPM, a discrete counterpart in the family of diffusion-based generative models. While VPSDE provides a continuous framework for modeling the forward and reverse dynamics of the diffusion process, DDPM discretizes these dynamics to create a practical algorithm for generative tasks. Specifically, DDPM(Ho et al., 2020a; Sohl-Dickstein et al., 2015) is built upon two Markov chains. The forward chain is typically hand-designed, which transforms the data distribution p_{data} into a (simple) reference distribution p_{ref} (e.g., standard Gaussian), by gradually adding Gaussian noise over T steps according to a predefined variance schedule β_t . The reverse chain invert this noising procedure using a denoising network ϵ_θ to iteratively remove noise, enabling data generation via ancestral sampling from an initial noise vector drawn from the reference distribution. This discrete approach allows DDPM to effectively model complex distributions through iterative sampling and optimization. This section explores the key components of DDPM, including the discrete forward and reverse process, training objective, and conditional generation with guidance signal.

Forward process. To introduce the discrete forward process, we begin by considering its definition as a Markov chain and building link with the continuous forward process:

Proposition 1. *Given a discrete forward process defined as Markov chain*

$$\mathbf{x}_i = \sqrt{1 - \beta_i}\mathbf{x}_{i-1} + \sqrt{\beta_i}\epsilon_{i-1}, \quad \text{where } \epsilon_{i-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), i = 1, \dots, N, \quad (5.18)$$

its limit at $N \rightarrow \infty$ is the forward VP-SDE: $d\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}dt + \sqrt{\beta(t)}d\mathbf{W}_t$

Proof. To obtain the limit of this Markov chain at $N \rightarrow \infty$, we define an auxiliary continuous function $\beta(t)$ whose $t \in [0, 1]$, s.t. $\frac{\beta(\frac{i}{N})}{N} = \beta_i$. In other words, when $t = \frac{i-1}{N}$, with $\Delta t = \frac{1}{N}$, $\beta_i = \frac{\beta(t+\Delta t)}{N} = \beta(t + \Delta t)\Delta t$. So we can rewrite the Markov chain as following and then use Taylor expansion:

$$\begin{aligned}
\mathbf{x}(t + \Delta t) &= \sqrt{1 - \beta(t + \Delta t)\Delta t}\mathbf{x}(t) + \sqrt{\beta(t + \Delta t)\Delta t}\boldsymbol{\epsilon}(t) \\
&\approx \mathbf{x}(t) - \frac{1}{2}\beta(t + \Delta t)\Delta t\mathbf{x}(t) + \sqrt{\beta(t + \Delta t)\Delta t}\boldsymbol{\epsilon}(t) \\
&\approx \mathbf{x}(t) - \frac{1}{2}\beta(t)\Delta t\mathbf{x}(t) + \sqrt{\beta(t)\Delta t}\boldsymbol{\epsilon}(t)
\end{aligned} \tag{5.19}$$

where the approximate equality holds when $\Delta t \ll 1$. Therefore, at the limit of $\Delta t \rightarrow 0$, it converges to the forward VP-SDE: $d\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}dt + \sqrt{\beta(t)}dW_t$ \square

A nice property of the above process is that we can sample at any arbitrary time step in a closed form of perturbation kernel:

Proposition 2. *The perturbation kernel of the discrete forward process is*

$$p(\mathbf{x}_i | \mathbf{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_i}\mathbf{x}_0, (1 - \bar{\alpha}_i)\mathbf{I}) \tag{5.20}$$

Proof. By leveraging the fact that the addition of two Gaussians $\mathcal{N}(\mathbf{0}, \sigma_1^2\mathbf{I})$ and $\mathcal{N}(\mathbf{0}, \sigma_2^2\mathbf{I})$ is still Gaussian $\mathcal{N}(\mathbf{0}, (\sigma_1^2 + \sigma_2^2)\mathbf{I})$, and by denoting $\alpha_t = 1 - \beta_t$, $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$, we could rewrite \mathbf{x}_i as:

$$\begin{aligned}
\mathbf{x}_i &= \sqrt{\alpha_i}\mathbf{x}_{i-1} + \sqrt{1 - \alpha_i}\boldsymbol{\epsilon}_{i-1} \\
&= \sqrt{\alpha_i}(\sqrt{\alpha_{i-1}}\mathbf{x}_{i-2} + \sqrt{1 - \alpha_{i-1}}\boldsymbol{\epsilon}_{i-2}) + \sqrt{1 - \alpha_i}\boldsymbol{\epsilon}_{i-1} \\
&= \sqrt{\alpha_i}\sqrt{\alpha_{i-1}}\mathbf{x}_{i-2} + \sqrt{\alpha_i}\sqrt{1 - \alpha_{i-1}}\boldsymbol{\epsilon}_{i-2} + \sqrt{1 - \alpha_i}\boldsymbol{\epsilon}_{i-1} \\
&= \sqrt{\alpha_i\alpha_{i-1}}\mathbf{x}_{i-2} + \sqrt{\alpha_i(1 - \alpha_{i-1}) + (1 - \alpha_i)}\bar{\boldsymbol{\epsilon}}_{i-2} \\
&= \sqrt{\alpha_i\alpha_{i-1}}\mathbf{x}_{i-2} + \sqrt{1 - \alpha_i\alpha_{i-1}}\bar{\boldsymbol{\epsilon}}_{i-2} \\
&= \dots \\
&= \sqrt{\bar{\alpha}_i}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_i}\boldsymbol{\epsilon}
\end{aligned} \tag{5.21}$$

from which we could derive the transition density as a Gaussian distribution: $p(\mathbf{x}_i | \mathbf{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_i}\mathbf{x}_0, (1 - \bar{\alpha}_i)\mathbf{I})$. \square

Remark 3. *Since the perturbation kernel of the discrete forward process is Gaussian, its score function can be expressed as:*

$$\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t | \mathbf{x}_0) = -\frac{1}{1 - \bar{\alpha}_t}(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0) = -\frac{1}{1 - \bar{\alpha}_t}\sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon} = -\frac{\boldsymbol{\epsilon}}{\sqrt{1 - \bar{\alpha}_t}} \tag{5.22}$$

Reverse process. Building on the discrete forward process, we now define its corresponding discrete reverse process.

Proposition 3. *Given the defined discrete forward process, its corresponding discrete reverse process is a Markov chain*

$$q(\mathbf{x}_{t-1} | \mathbf{x}_t) = \mathcal{N}(\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t), \tilde{\sigma}_t^2 \mathbf{I}), \text{ where } \begin{cases} \tilde{\boldsymbol{\mu}}_t = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\alpha_t}} \boldsymbol{\epsilon} \right) \\ \tilde{\sigma}_t^2 = \frac{1-\alpha_{t-1}}{1-\alpha_t} \cdot \beta_t \end{cases} \quad (5.23)$$

Proof. Using Bayes' Rule and the discrete version of forward process, we have

$$q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) = q(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{x}_0) \frac{q(\mathbf{x}_{t-1} | \mathbf{x}_0)}{q(\mathbf{x}_t | \mathbf{x}_0)} \quad (5.24)$$

$$\propto \exp \left(-\frac{1}{2} \left(\frac{(\mathbf{x}_t - \sqrt{\alpha_t} \mathbf{x}_{t-1})^2}{\beta_t} + \frac{(\mathbf{x}_{t-1} - \sqrt{\alpha_{t-1}} \mathbf{x}_0)^2}{1 - \alpha_{t-1}} - \frac{(\mathbf{x}_t - \sqrt{\alpha_t} \mathbf{x}_0)^2}{1 - \alpha_t} \right) \right) \quad (5.25)$$

$$= \exp \left(-\frac{1}{2} \left(\frac{\mathbf{x}_t^2 - 2\sqrt{\alpha_t} \mathbf{x}_t \mathbf{x}_{t-1} + \alpha_t \mathbf{x}_{t-1}^2}{\beta_t} + \frac{\mathbf{x}_{t-1}^2 - 2\sqrt{\alpha_{t-1}} \mathbf{x}_0 \mathbf{x}_{t-1} + \alpha_{t-1} \mathbf{x}_0^2}{1 - \alpha_{t-1}} - \frac{(\mathbf{x}_t - \sqrt{\alpha_t} \mathbf{x}_0)^2}{1 - \alpha_t} \right) \right) \quad (5.26)$$

$$= \exp \left(-\frac{1}{2} \left(\underbrace{\left(\frac{\alpha_t}{\beta_t} + \frac{1}{1 - \alpha_{t-1}} \right) \mathbf{x}_{t-1}^2}_{\text{denote as } \frac{1}{\tilde{\sigma}_t^2}} - \underbrace{\left(\frac{2\sqrt{\alpha_t}}{\beta_t} \mathbf{x}_t + \frac{2\sqrt{\alpha_{t-1}}}{1 - \alpha_{t-1}} \mathbf{x}_0 \right) \mathbf{x}_{t-1}}_{\text{denote as } \frac{2\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0)}{\tilde{\sigma}_t^2}} + \underbrace{C(\mathbf{x}_t, \mathbf{x}_0)}_{\text{not involving } \mathbf{x}_{t-1}} \right) \right) \quad (5.27)$$

$$\propto \exp \left(-\frac{1}{2} \left(\frac{(\mathbf{x}_{t-1} - \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0))^2}{\tilde{\sigma}_t^2} \right) \right) \quad (5.28)$$

$$= \mathcal{N}(\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\sigma}_t^2 \mathbf{I}) \quad (5.29)$$

where by using the discrete forward process' perturbation kernel **Proposition 2** $\mathbf{x}_t = \sqrt{\alpha_t} \mathbf{x}_0 + \sqrt{1 - \alpha_t} \boldsymbol{\epsilon}$ to replace \mathbf{x}_0 with \mathbf{x}_t , one can make $\tilde{\boldsymbol{\mu}}_t$ \mathbf{x}_0 -independent. Then, the mean and variance can be expressed as **Equation (5.23)**. \square

Remark 4. *At sampling, in practice the variance in **Equation (5.23)** is approximated as $\tilde{\sigma}_t^2 = \beta_t$.*

Remark 5. *To enable sampling and generate samples, one needs to learn the probability **Equation (5.23)**, in which the noise $\boldsymbol{\epsilon}$ needs to be modeled.*

In practice $\boldsymbol{\epsilon}$ is approximated with a denoising network $\boldsymbol{\epsilon}_\theta$, which is optimized using a re-weighted variational lower bound of the marginal likelihood:

$$\mathcal{L}_{\text{simple}}(\boldsymbol{\theta}) = \mathbb{E}_{t \sim U(0, T), \mathbf{x}_0 \sim p_{\text{data}}, \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\alpha_t} \mathbf{x}_0 + \sqrt{1 - \alpha_t} \boldsymbol{\epsilon}, t)\|^2 \right] \quad (5.30)$$

*Furthermore, since the score function is a function of the noise $\boldsymbol{\epsilon}$ as shown in **Equation (5.22)**, modeling the noise is fundamentally equivalent to modeling the score function. This also*

links the discrete-time setting back to the continuous-time setting, as a similar variational objective can be obtained in the perspective of continuous time (Song et al., 2021), whereby the denoising network approximates the score function of the data distribution (Remark 2).

Remark 6. Conditional generation with guidance signal: the simple DDPM formulation has been extended to guided generation (Ho and Salimans, 2021), whereby a guidance signal p injects “external information” in the iterative denoising process. This requires a simple extension to the denoising network such that it can accept the guidance signal: $\epsilon_\theta(x_t, p, t)$. Then, during training, a randomized approach allows to learn both the guided and unconditional variants of the denoising network, for example by assigning a null value to the guidance signal and mask a guidance signal with null value with certain probability. At sampling time, a weighted linear combination of the guided and unconditional networks, such as $\tilde{\epsilon}_\theta(x_t, p, t) = \epsilon_\theta(x_t, \emptyset, t) + \gamma(\epsilon_\theta(x_t, p, t) - \epsilon_\theta(x_t, \emptyset, t))$ can be used in modeling the reverse Markov chain Equation (5.23) to further emphasize the guidance, which is a technique known as CFG.

5.2 Preliminaries – conditional flow matching

Although the diffusion/SDE-based generative models have achieved huge success on real-world tasks like image generation, it has a key drawback: the high computational cost at inference. As an alternative, a simpler model namely RF has gained popularity recently, which learns a probability flow ODE and aims at transporting distribution p to q by following straight line paths as much as possible. An advantage of sampling along nearly straight flows is that it can produce good results even with a coarse time discretization. Mathematically, RF’s training loss is built upon the conditional flow matching framework, which will be presented in this section.

Let $x \in \mathbb{R}^d$ denote a data point in the d -dimensional Euclidean space associated with the standard Euclidean inner product, and $X \in \mathbb{R}^d$ a RV with continuous PDF $p_X : \mathbb{R}^d \rightarrow \mathbb{R}_{\geq 0}$, where $\int_{\mathbb{R}^d} p_X(x) dx = 1$. We use the notation $X \sim p_X$ to indicate that X is distributed according to p_X .

The key concepts we consider within the framework of flow matching are:

1. flow : a time-dependent C^r $([0, 1] \times \mathbb{R}^d, \mathbb{R}^d)$ mapping $\psi : (t, x) \mapsto \psi_t(x)$
2. velocity field : a time-dependent C^r $([0, 1] \times \mathbb{R}^d, \mathbb{R}^d)$ mapping $u : (t, x) \mapsto u_t(x)$
3. probability path : a time-dependent PDF $(p_t)_{0 \leq t \leq 1}$

Given a source distribution p – e.g., standard Gaussian distribution $\mathcal{N}(0, I)$, and the data target distribution q , the goal of generative flow modeling is to build a flow ψ that transforms $X_0 \sim p$ into $X_1 := \psi_1(X_0)$ such that $X_1 \sim q$.

In the following, we first clarify the link between these 3 concepts, then present methods to realize this goal.

5.2.1 Link between the 3 concepts

(a) Flow and Velocity Field:

First, it is essential to note that a flow is uniquely determined by a velocity field. This relation is derived from the ODE:

$$\begin{cases} \frac{d}{dt}\psi_t(x) = u_t(\psi_t(x)) & \text{(flow ODE)} \\ \psi_0(x) = x & \text{(flow initial conditions).} \end{cases} \quad (5.31)$$

This ODE, referred to as an initial value problem, describes how the flow evolves under the influence of the velocity field with initial condition $\psi_0(x) = x$. The solution to this ODE at any time t gives the flow ψ_t .

(b) Velocity Field generating a Probability Path:

Next, we consider a probability path p_t that evolves over time. The velocity field u_t is said to *generate* p_t if its flow ψ_t , which solves the initial value problem driven by u_t , satisfies the condition

$$X_t := \psi_t(X_0) \sim p_t, \text{ for } t \in [0, 1), X_0 \sim p_0. \quad (5.32)$$

This means that the distribution of $\psi_t(X_0)$, where $X_0 \sim p_0$, matches the distribution p_t . Recall that, given the distribution of X_0 as p_0 (i.e., $X_0 \sim p_0$), and applying the flow mapping ψ_t to X_0 , the resulting distribution of the RV $\psi_t(X_0)$ is referred to as the *pushforward* of p_0 by ψ_t , denoted $\psi_t(\cdot) \# p_0$ (i.e., $\psi_t(X_0) \sim \psi_t(\cdot) \# p_0$). Using this notation, the condition Equation (5.32) can be equivalently written as $p_t = \psi_t(\cdot) \# p_0$.

(c) Continuity Equation:

Finally, a practical method for verifying whether a velocity field u_t generates a probability path p_t is to check if the pair (u_t, p_t) satisfies the Continuity Equation:

$$\frac{d}{dt}p_t(x) + \text{div}(p_t u_t)(x) = 0. \quad (5.33)$$

This equation expresses the conservation of probability and ensures that the evolution of p_t under the flow generated by u_t is physically consistent. It is important to note that the correspondence between u_t and p_t is unique only up to divergence-free fields.

By synthesizing the points (a), (b), and (c), we can visualize the links between the probability path, the velocity field, and the flow, as illustrated in Figure 5.2.

5.2.2 Conditional flow matching: a tractable and valid loss

Building upon these links, given a prescribed¹ probability path p_t that satisfies the boundary conditions $p_0 = p$ and $p_1 = q$, the goal of Flow Matching (FM) is to learn a parametric velocity field u_t^θ that matches the ground truth velocity field u_t known

¹prescribing makes the training supervised for all t and therefore easier.

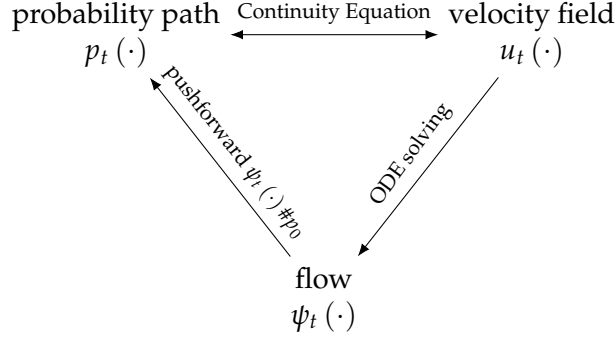


FIGURE 5.2: Links between the probability path, the velocity field and the flow.

to generate the desired probability path p_t (i.e., link (c)). Sampling is then achieved by solving the initial value problem, with x_0 sampled from the source distribution p_0 (i.e., link (a)). In other words, based on the ODE, the original goal of modeling a flow is now transformed into modeling a velocity field.

To model the ground-truth velocity field u_t , in principle the NN just needs to be optimized by minimizing the regression loss:

$$\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{t, x \sim p(\cdot|t)} \|u_t^\theta(x) - u_t(x)\|^2. \quad (5.34)$$

However in practice, this ground truth marginal velocity field u_t is not tractable, as it requires marginalizing over the entire training set — i.e., $u_t(x) = \int u_t(x | x_1) p_{1|t}(x_1 | x) dx_1$. To bypass this problem, Lipman et al., 2023 develop CFM.

In CFM, rather than prescribing the marginal probability path p_t that satisfies the boundary conditions, we prescribe the conditional probability path $p_t(\cdot | z)$ instead. This involves (1) the choice of the conditional RV z that is independent of t , and (2) the choice of conditional PDF $p_t(\cdot | z)$ such that it not only satisfies the boundary condition (i.e., its marginalized counterpart $\mathbb{E}_z[p_t(\cdot | z)]$ equals to p_0 at $t = 0$ and p_{target} at $t = 1$), but also has a tractable $u_t(x | z)$ (obtained by solving Continuity Equation), which serves as the ground truth in the training loss.

One popular choice for defining a conditional probability path is linear interpolation. In this case, the two choices are: (1) $z = (x_0, x_1) \sim p_0 \otimes p_{\text{target}}$, where z represents the independent coupling of $x_0 \sim p_0$ and $x_1 \sim p_{\text{target}}$, and (2) $p_t(\cdot | z) = \delta_{(1-t)x_0 + tx_1}(x)$, where δ denotes the Dirac delta function. This choice satisfies the boundary conditions, as shown by the following integrals:

$$\begin{cases} \int_z p_{t=0}(x | z) p(z) dz = \int_z \delta_{x_0}(x) p(z) dz = p_0(x), \\ \int_z p_{t=1}(x | z) p(z) dz = \int_z \delta_{x_1}(x) p(z) dz = p_{\text{target}}(x). \end{cases} \quad (5.35)$$

The corresponding conditional velocity field for this interpolation is $u_t(x | z) = x_1 - x_0$. Another popular choice is the Gaussian path, where the two choices are: (1) $z = x_1 \sim p_{\text{target}}$, and (2) $p_t(\cdot | z) = \mathcal{N}(tx_1, (1-t)^2 I)$. This choice also satisfies the boundary conditions, as shown by:

$$\begin{cases} \int_z p(x | z, t=0) p(z) dz = \int_z p_0(x) p(z) dz = p_0(x), \\ \int_z p(x | z, t=1) p(z) dz = \int_z \delta_z(x) p(z) dz = p_{\text{target}}(x). \end{cases} \quad (5.36)$$

The corresponding conditional velocity field for this Gaussian path is $u_t(x | z) = \frac{x_1 - x}{1-t}$.

Given a prescribed conditional probability path $p_t(x | z)$ with its marginalized counterpart being $p_t(x)$, their associated velocity fields $u_t(x | z)$ and $u_t(x)$ can be found by solving their corresponding Continuity Equations. Additionally, the relationship between $u_t(x | z)$ and $u_t(x)$ is demonstrated as follows:

$$\forall (x, t), u(x, t) = \mathbb{E}_{z|(x,t)} [u(x, t, z)] \quad (5.37)$$

Figure 5.3 summarizes the connections between the (tractable) conditional velocity field, the corresponding conditional probability path, and their (intractable) marginal counterparts, for any given conditional random variable z .

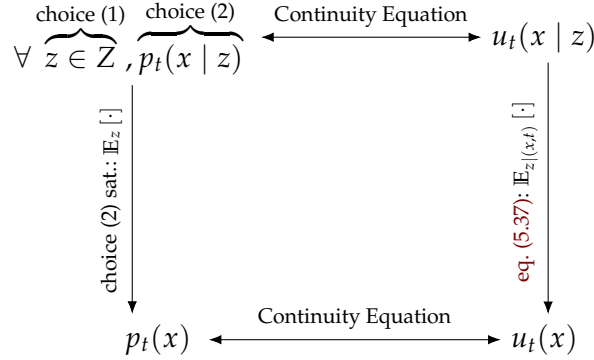


FIGURE 5.3: Links between (tractable) conditional probability path and velocity field and their (intractable) marginal counterparts, for any condition RV z .

With the link between conditional and marginal components clarified, now we come to a revision of the training loss. CFM regresses against the tractable conditional velocity field, with loss being

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t,z \sim p(z), x \sim p(\cdot | t, z)} \left\| u_t^\theta(x) - u_t(x | z) \right\|^2. \quad (5.38)$$

Here, the ground truth conditional velocity field $u_t(x | z)$ is tractable, as it only depends on a single sample of z . Leveraging eq. (5.37), Lipman et al., 2023 proves that the FM loss Equation (5.34) and the CFM loss Equation (5.38) are equivalent for

learning purposes: since their gradients coincide, the minimizer of CFM loss is the desired marginal velocity $u_t(x)$ useful at sampling.

On the one hand, we note that for a Gaussian source and independent coupling of (x_0, x_1) , the two choices for defining a conditional probability path — linear interpolation and Gaussian path share the same CFM loss thanks to the expectation in [Equation \(5.38\)](#). In linear interpolation, the expectation over $x \sim p(\cdot | t, z)$ has no stochasticity as it is a Dirac, therefore x can be directly replaced with $x = tx_1 + (1 - t)x_0$; the expectation over $z = (x_0, x_1)$ is the average with respect to $p(z) = p_0(x_0)p_{\text{target}}(x_1)$, so the CFM loss is

$$\mathcal{L}_{\text{CFM}}(\theta) = \iiint p_T(t) p_{X_0}(x_0) p_{X_1}(x_1) \left[u^\theta(tx_1 + (1 - t)x_0) - (x_1 - x_0) \right]^2 dx_0 dx_1 dt. \quad (5.39)$$

In Gaussian path, given $z = x_1$, the conditional distribution $x \sim p(\cdot | t, z) = \mathcal{N}(tx_1, (1 - t)^2 I)$ is equivalent to $X = tx_1 + (1 - t)X_0$, where $X_0 \sim \mathcal{N}(0, I)$. This equivalence leads to the ground-truth conditional velocity field: $u_t(x | z) = \frac{x_1 - x}{1 - t} = \frac{x_1 - [tx_1 + (1 - t)X_0]}{1 - t} = x_1 - X_0$. As a result, the expectation over x is equivalent to the expectation over X_0 : $\mathbb{E}_{x \sim p(\cdot | x_1)} \left[(u^\theta(x) - u_t(x | x_1))^2 \right] = \mathbb{E}_{X_0 \sim \mathcal{N}(0, I)} \left[(u^\theta(tx_1 + (1 - t)X_0) - (x_1 - X_0))^2 \right]$. Injecting them back into the CFM loss, we get

$$\mathcal{L}_{\text{CFM}}(\theta) = \int p_T(t) \int p_{X_1}(x_1) \left[\int p_{X_0}(x_0) \left[u^\theta(tx_1 + (1 - t)x_0) - (x_1 - x_0) \right]^2 dx_0 \right] dx_1 dt. \quad (5.40)$$

[Equation \(5.39\)](#) and [Equation \(5.40\)](#) are equivalent, given that we assume X_0 and X_1 as independent. Given their equivalence, we will focus solely on the Gaussian path in the following sections.

On the other hand, in the linear interpolation choice (1), where the condition RV $z = (x_0, x_1)$ is chosen, the joint probability distribution of z , denoted $\pi_{0,1}(x_0, x_1)$, can be any coupling that preserves the marginal distributions p_0 and p_{target} . What we previously considered is the independent coupling $\pi_{0,1}(x_0, x_1) = p(x_0)p_{\text{target}}(x_1)$. An alternative, non-independent coupling approach involves selecting a source point, sampling from it, and generating a new target sample using a pretrained model. This defines a new "line" between the source points and target samples, and this process can be repeated multiple times to build a training set. A key property of linear interpolation is that the kinetic energy of the marginal velocity $u_t(x)$ is not greater than the kinetic energy of the original coupling $\pi_{0,1}$ used to train the model. This process is referred to as *rectifying*, and the trained velocity field u_t^θ is called the *RF*, as described by Liu et al., [2022b](#).

Finally, an essential result known as the Instantaneous Change of Variables (Chen et al., [2019a](#)) is required for the following section. Given the Continuity Equation,

the differential equation governing the evolution of the log-probability density is:

$$\frac{d}{dt} \log p_t(\psi_t(x)) = -\operatorname{div}(u_t)(\psi_t(x)). \quad (5.41)$$

5.2.3 Conditional generation with guidance signal

As we are interested in conditional generation, we define the guidance RV as $Y \sim p_Y$, with data samples $y \in \mathcal{Y} \subset \mathbb{R}^k$. Given access to labeled target samples (x_1, y) , the goal of FM is to train the parameters θ of a single velocity field $u_t^\theta : \mathbb{R}^d \times \mathbb{R}^k \rightarrow \mathbb{R}^d$ to match the ground truth guided velocity field $u_t(\cdot | y)$ known to generate the desired guided probability path $p_{t|Y}(\cdot | y)$ satisfying the boundary conditions $p_{t=0|Y}(\cdot | y) = p(\cdot)$ and $p_{t=1|Y}(\cdot | y) = q(\cdot | y)$, for all values of y . The guided version of tractable CFM loss is $\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t, (x_0, x_1, y) \sim \pi_{0,1,y}} \|u_t^\theta(x_t | Y) - (x_1 - x_0)\|^2$. Furthermore, if the model is trained with Gaussian paths, the CFG technique can be applied at sampling to enhance the sample quality, for which during training y will be masked as null-condition \emptyset with probability p_{uncond} , in order to train $u_t^\theta(\cdot | \emptyset)$ to approximate the unconditional velocity field u_t generating the unconditional probability path p_t .

Chapter 6

Mutual information estimation

6.1 Introduction

As explained in [Chapter 5](#), both DDPM and RF can be extended to conditional generation, where the reverse sampling process is guided by a guidance signal. In practical applications, one of its popular use-cases is generating images conditioned on textual prompts. T2I generative models have reached an incredible popularity thanks to their high-quality image synthesis, ease of use, and integration across a variety of end-users services (e.g., image editing software, chat bots, smartphones apps, websites). T2I models are trained on large-scale datasets (LAION project, [2024](#)) to generate images *semantically aligned* with a user text input. Yet, recent benchmarks (Huang et al., [2023](#); Wu et al., [2024](#)) show that even SOTA DM models (e.g., Stable Diffusion XL (Rombach et al., [2022](#))) and RF models (e.g., Stable Diffusion 3 (Esser et al., [2024](#)) and FLUX (FLUX, [2023](#))), despite achieving a new SOTA, still suffer from a variety of alignment issues (subjects in the images might be missing, or have the wrong attributes, such as numeracy, positioning, etc). To address this issue, we aim to use a pre-trained generative model as a neural estimator for the point-wise MI between the generated image and the prompt, and further use the estimated point-wise MI to score and improve T2I alignment. In turn, this raises three research questions: (1) How to estimate MI using a generative model? (2) Is MI meaningful for T2I alignment? (3) How to use the MI estimates to improve T2I alignment? In this chapter, we address the first research section by deriving MI estimation formulas using DM or RF, and leave the latter two for [Chapter 7](#).

In information theory, MI is a key statistical metric that captures the non-linear dependencies between RVs by quantifying their mutual dependence. Unlike simpler measures like the correlation coefficient that only capture linear relationships between real-valued RVs, MI is more general, as it assesses how the joint distribution of two RVs, $X \sim P_X$ and $Y \sim P_Y$, diverges from the product of their individual

marginal distributions. Mathematically, MI is defined as

$$\begin{aligned} I(X; Y) &= D_{\text{KL}}(P_{XY} \| P_X P_Y) \\ &= \mathbb{E}_{(x,y) \sim P_{XY}} \left[\log \left(\frac{P_{XY}(x, y)}{P_X(x) P_Y(y)} \right) \right] \end{aligned} \quad (6.1)$$

where D_{KL} denotes the Kullback-Leibler (KL) divergence. Intuitively, $I(X; Y)$ measures the dependence between X and Y , or, the information about X (resp. Y) provided by Y (resp. X). Furthermore, as hinted in the expectation, while MI measures the average information shared between two RVs, point-wise MI quantifies the dependence in a specific pair (x, y) .

MI has been widely applied in ML, particularly in representation learning and evaluation of discriminative models and generative models, where it helps to effectively capture complex dependencies and enhance feature extraction. However, for many problems of interest, precise computation of MI remains a challenging task, leading to the development of various estimation techniques. Since applying traditional parametric and non-parametric methods (Nemenman et al., 2001; Gao et al., 2015) to high-dimensional, real-world data is often impractical or infeasible, recent research has shifted toward variational approaches and neural estimators for more scalable MI estimation. In particular, the work by Song and Ermon, 2019a classifies recent MI estimation methods into discriminative and generative approaches — discriminative approaches (McAllester and Stratos, 2020) focus on directly estimating the ratio between joint distribution and product of marginals, but these estimators present tradeoffs between bias and variance in estimating variational bounds on mutual information (e.g., the contrastive loss $\mathcal{L}_{\text{contrast}}$ detailed in Section 4.2.1 has low variance but high bias); generative approaches estimate the two marginal densities separately using generative models like VAE (Kingma and Welling, 2022), despite being more scalable, they still face estimation accuracy challenges on high-dimensional or complex data according to benchmark testing on synthetic distributions (Czyż et al., 2023). Furthermore, to the best of our knowledge, while multiple MI neural estimators have been proposed, no previous work considers estimating MI using the SOTA generative models – discrete version of DMs (i.e., DDPM) or RF models.

In this chapter, we explore the problem of estimating *point-wise* MI using the pre-trained generative model itself. Specifically, the generative models we considered are DDPM and RF:

- DDPM-based point-wise MI estimator (Section 6.2)

Building upon MINDE (Franzese et al., 2024) that uses score-based continuous-time DMs to estimate the KL divergence between two densities as a difference between their score functions, we extend the work to a point-wise MI estimator suitable for discrete-time setting, which have more interest for real-world applications (e.g. text-to-image generation).

- RF-based point-wise MI estimator (Section 6.3)

We introduce RFMI (Section 6.3.1), a novel RF-based point-wise MI estimator leveraging the relation between the score function and the velocity field. Furthermore, we demonstrate the validity of RFMI empirically, considering a MI estimation benchmark involving various challenging data distributions (Section 6.3.2) where the true MI is known.

6.2 DDPM-based point-wise MI estimator

In this section, we aim to build a point-wise MI estimator that fits the recent advanced T2I diffusion models.

Regarding to the T2I diffusion model, following the common design for realistic and high-dimensional image synthesis, we use pre-trained latent diffusion models operating on a learned projection of the input image data \mathbf{x}_0 into a corresponding latent variable \mathbf{z}_0 which is lower-dimensional compared to the original data. Moreover, the conditioning signal \mathbf{y} is obtained from a pre-trained text encoder such as CLIP (Radford et al., 2021).

Regarding to the MI estimator, we capitalize on a recent method (Franzese et al., 2024), that relies on the theory behind continuous-time diffusion processes (Song et al., 2021) and uses the Girsanov Theorem (Øksendal, 2003) to show that score functions can be used to compute the KL divergence between two distributions. In what follows, we use a simplified notation and gloss over several mathematical details to favor intuition over rigor. Here we consider discrete-time diffusion models, which are equivalent to the continuous-time counterpart under the variational formulation, up to constants and discretization errors (Song et al., 2021).

We begin by considering the two arbitrary random variables \mathbf{z} and \mathbf{y} which are sampled from the joint distribution $p_{\text{latent}, \text{prompt}}$, where the former corresponds to the distribution of the projections in a latent space of the image distribution, and the latter to the distribution of prompts used for conditional generation. Then, following the approach in (Franzese et al., 2024), with the necessary adaptation to the discrete domain, the point-wise MI estimation can be obtained as follows:

$$I(\mathbf{z}, \mathbf{y}) = \mathbb{E}_{t, \epsilon \sim \mathcal{N}(\mathbf{0}, I)} [\kappa_t ||\epsilon_\theta(\mathbf{z}_t, \mathbf{y}, t) - \epsilon_\theta(\mathbf{z}_t, \emptyset, t)||^2], \quad \kappa_t = \frac{\beta_t T}{2\alpha_t(1 - \bar{\alpha}_t)}. \quad (6.2)$$

Proof. Recall that in the definition of a discrete-time diffusion model, for the forward process, we use the following Markov chain

$$q(\mathbf{z}_{0:T}, \mathbf{y}) = q(\mathbf{z}_0, \mathbf{y}) \prod_{t=1}^T q(\mathbf{z}_t | \mathbf{z}_{t-1}), \quad q(\mathbf{z}_t | \mathbf{z}_{t-1}) = \mathcal{N}(\mathbf{z}_t; \sqrt{1 - \beta_t} \mathbf{z}_{t-1}, \beta_t I) \quad (6.3)$$

The backward process (with or without a conditioning signal \mathbf{y}) evolves according to

$$p_{\theta}(\mathbf{z}_{0:T}) = p(\mathbf{z}_T) \prod_{t=1}^T p_{\theta}(\mathbf{z}_{t-1} | \mathbf{z}_t), \quad p_{\theta}(\mathbf{z}_{0:T} | \mathbf{y}) = p(\mathbf{z}_T) \prod_{t=1}^T p_{\theta}(\mathbf{z}_{t-1} | \mathbf{z}_t, \mathbf{y}) \quad (6.4)$$

where $p_{\theta}(\mathbf{z}_{t-1} | \mathbf{z}_t) = \mathcal{N}(\mathbf{z}_{t-1}; \mu_{\theta}(\mathbf{z}_t), \beta_t I)$, with $\mu_{\theta}(\mathbf{z}_t) = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_{\theta}(\mathbf{z}_t, t) \right)$. Similar expressions can be obtained for the conditional version.

Our goal here is to show that the following equality holds

$$E[\text{KL}[q(\mathbf{z}_0 | \mathbf{y}) \parallel q(\mathbf{z}_0)]] = \mathbb{E}_{\mathbf{z}, \mathbf{y}}[\text{I}(\mathbf{z}, \mathbf{y})], \quad (6.5)$$

which is the condition that $\text{I}(\mathbf{z}, \mathbf{y})$ of Equation (6.2) should satisfy to be a valid point-wise MI estimator.

In particular, we will show that

$$E[\text{KL}[q(\mathbf{z}_0 | \mathbf{y}) \parallel q(\mathbf{z}_0)]] = \mathbb{E}_{t, \mathbf{y}, \mathbf{z}, \epsilon} [\kappa_t \|\boldsymbol{\epsilon}_{\theta}(\mathbf{z}_t, \mathbf{y}, t) - \boldsymbol{\epsilon}_{\theta}(\mathbf{z}_t, \emptyset, t)\|^2], \quad \kappa_t = \frac{\beta_t T}{2\alpha_t(1-\bar{\alpha}_t)}. \quad (6.6)$$

To simplify our proof strategy, we consider the ideal case of perfect training, i.e., $p_{\theta}(\mathbf{z}_{0:T}, \mathbf{y}) = q(\mathbf{z}_{0:T}, \mathbf{y})$. Moreover, since $q(\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{y}) = q(\mathbf{z}_t | \mathbf{z}_{t-1})$, we can rewrite the $\text{KL}[q(\mathbf{z}_0 | \mathbf{y}) \parallel q(\mathbf{z}_0)]$ term as follows

$$\text{KL}[q(\mathbf{z}_0 | \mathbf{y}) \parallel q(\mathbf{z}_0)] \quad (6.7)$$

$$= \text{KL}[q(\mathbf{z}_{0:T} | \mathbf{y}) \parallel q(\mathbf{z}_{0:T})] \quad (6.8)$$

$$= \text{KL}[p_{\theta}(\mathbf{z}_{0:T} | \mathbf{y}) \parallel p_{\theta}(\mathbf{z}_{0:T})] \quad (6.9)$$

$$= \int p_{\theta}(\mathbf{z}_{0:T} | \mathbf{y}) \log \frac{p_{\theta}(\mathbf{z}_{0:T} | \mathbf{y})}{p_{\theta}(\mathbf{z}_{0:T})} d\mathbf{z}_{0:T} \quad (6.10)$$

$$= \int p_{\theta}(\mathbf{z}_{0:T} | \mathbf{y}) \sum_{t=1}^T \log \frac{p_{\theta}(\mathbf{z}_{t-1} | \mathbf{z}_t, \mathbf{y})}{p_{\theta}(\mathbf{z}_{t-1} | \mathbf{z}_t)} d\mathbf{z}_{0:T} \quad (6.11)$$

$$= \sum_{t=1}^T \int p_{\theta}(\mathbf{z}_{0:t-2,t:T} | \mathbf{y}) \left(\int p_{\theta}(\mathbf{z}_{t-1} | \mathbf{z}_t, \mathbf{y}) \log \frac{p_{\theta}(\mathbf{z}_{t-1} | \mathbf{z}_t, \mathbf{y})}{p_{\theta}(\mathbf{z}_{t-1} | \mathbf{z}_t)} d\mathbf{z}_{t-1} \right) d\mathbf{z}_{0:t-2,t:T} \quad (6.12)$$

$$= \sum_{t=1}^T \int p_{\theta}(\mathbf{z}_t | \mathbf{y}) \text{KL}[p_{\theta}(\mathbf{z}_{t-1} | \mathbf{z}_t, \mathbf{y}) \parallel p_{\theta}(\mathbf{z}_{t-1} | \mathbf{z}_t)] d\mathbf{z}_t \quad (6.13)$$

$$= \sum_{t=1}^T \frac{1}{2\beta_t} \int p_{\theta}(\mathbf{z}_t | \mathbf{y}) \|\mu_{\theta}(\mathbf{z}_t) - \mu_{\theta}(\mathbf{z}_t, \mathbf{y})\|^2 d\mathbf{z}_t \quad (6.14)$$

$$= \sum_{t=1}^T \frac{1}{2\beta_t} \frac{\beta_t^2}{\alpha_t(1-\bar{\alpha}_t)} \int p_{\theta}(\mathbf{z}_t | \mathbf{y}) \|\boldsymbol{\epsilon}_{\theta}(\mathbf{z}_t) - \boldsymbol{\epsilon}_{\theta}(\mathbf{z}_t, \mathbf{y})\|^2 d\mathbf{z}_t \quad (6.15)$$

$$= \mathbb{E}_{t, \mathbf{z}_t} [\kappa_t \|\boldsymbol{\epsilon}_{\theta}(\mathbf{z}_t, \emptyset, t) - \boldsymbol{\epsilon}_{\theta}(\mathbf{z}_t, \mathbf{y}, t)\|^2] \quad (6.16)$$

$$= \mathbb{E}_{t,z,\epsilon} \left[\kappa_t \|\epsilon_\theta(z_t, \emptyset, t) - \epsilon_\theta(z_t, y, t)\|^2 \right], \quad \kappa_t = \frac{\beta_t T}{2\alpha_t(1 - \bar{\alpha}_t)} \quad (6.17)$$

which allows to prove that the quantity in Equation (6.2) is indeed a valid point-wise MI estimator. \square

Given a pre-trained diffusion model, we compute an expectation (over diffusion times t) of the scaled squared norm of the difference between the conditional $\epsilon_\theta(z_t, y, t)$ and unconditional networks $\epsilon_\theta(z_t, \emptyset, t)$, which corresponds to an estimate of the point-wise MI between an image and a prompt. Intuitively, the difference between these scores quantifies how much extra knowledge of the prompt helps in denoising the perturbed images. This is both a key ingredient and a competitive advantage of our method, as it enables a self-contained approach to alignment based on the T2I model alone without auxiliary models or human feedback.

Experimental validation of a continuous-time version of Equation (6.2) on synthetic MI estimation benchmarks and self-consistency tests are provided in (Franzese et al., 2024).

6.3 RF-based point-wise MI estimator

6.3.1 RFMI: estimating MI with RF

Recall that the MI between two random variables $X \sim p_X$ and $Y \sim p_Y$ can be defined as the KL divergence between the joint distribution and the product of marginals: $I(X; Y) = D_{\text{KL}}(p_{(X,Y)} \| p_X p_Y)$. Furthermore, let $p_{X|Y}(\cdot | y)$ be the conditional distribution of X given $Y = y$. Using the identity $p_{(X,Y)}(x, y) = p_{X|Y}(x | y) p_Y(y)$, it clearly holds that $I(X; Y) = \mathbb{E}_Y [D_{\text{KL}}(p_{X|Y} \| p_X)]$, which indicates that the more the distributions $p_{X|Y}$ and p_X differ on average, the greater the information gain.

In this section we introduce RFMI, a new method to estimate the MI between X and the guidance signal Y leveraging conditional RF models (Esser et al., 2024; FLUX, 2023). To keep the notation concise and aligned with the notions in Section 5.2, we will refer to p_X as q , and $p_{X|Y}(\cdot | y)$ as $q(\cdot | y)$.

Consider the case of linear conditional flow $\psi_t(x | x_1) = tx_1 + (1 - t)x$ with x being samples of Gaussian prior $X_0 \sim p = \mathcal{N}(0, I)$. This flow's conditional velocity field $u_t(\cdot | x_1)$ generates a Gaussian path $p_{t|1}(\cdot | x_1)$ satisfying $p_{0|1}(\cdot | x_1) = p$ and $p_{1|1}(\cdot | x_1) = \delta_{x_1}(\cdot)$. The conditional pair $(u_t(\cdot | x_1), p_{t|1}(\cdot | x_1))$ does not depend from the guidance variable Y , while its marginal counterpart does, as $(u_t(\cdot | y) = \int u_t(\cdot | x_1) p_{1|t,Y}(x_1 | \cdot, y) dx_1, p_{t|Y}(\cdot | y) = \int p_{t|1}(\cdot | x_1) q(x_1 | y) dx_1)$. As a consequence, by applying the Marginalization trick, the guided velocity field $u_t(\cdot | y)$ generates the guided probability path $p_{t|Y}(\cdot | y)$, $p_{t|Y}(\cdot | y)$ satisfies $p_{0|Y}(\cdot | y) = p(\cdot)$ and $p_{1|Y}(\cdot | y) = q(\cdot | y)$. Note that when $y = \emptyset \in \{\emptyset\}$ the marginal case is reduced to unconditional generation: $u_t(\cdot | \emptyset) = u_t$, $p_{t|\{\emptyset\}}(\cdot | \emptyset) = p_t$, and $q(\cdot | \emptyset) = q$.

Overall, we express MI using the guided and the unconditional marginal probability paths at the endpoint $t = 1$ as

$$\begin{aligned} I(X; Y) &= \mathbb{E}_Y [D_{\text{KL}}(p_{X|Y} \| p_X)] \\ &= \mathbb{E}_Y \left[\int q(x | Y) \log \left(\frac{q(x | Y)}{q(x)} \right) dx \right] \\ &= \mathbb{E}_Y \left[\int p_{1|Y}(x_1 | Y) \log \left(\frac{p_{1|Y}(x_1 | Y)}{p_1(x_1)} \right) dx_1 \right]. \end{aligned} \quad (6.18)$$

In practice, we train a single conditional RF neural network $u_t^\theta(x | y)$, using the CFM loss, for all values $y \in \{\mathcal{Y}, \emptyset\}$. Since the minimizer of CFM loss is $u_t(\cdot | y)$, $u_t^\theta(x | y)$ is a valid approximation of $u_t(\cdot | y)$.

Next, we develop an expression of MI using $u_t(\cdot | y)$ and u_t , and use the conditional RF model to estimate the MI between X and Y . To do so, we first express the score functions associated to the marginal probability paths using the marginal velocity fields. These two terms are related according to the following

Proposition 4 (Relation between velocity field and score function). *For Gaussian paths $p_{t|1}(\cdot | x_1) = \mathcal{N}(\cdot | b_t x_1, a_t^2 I)$, the relation between the conditional velocity field $u_t(\cdot | x_1)$ and the **score function** $\nabla \log p_{t|1}(\cdot | x_1)$ of the conditional probability path $p_{t|1}(\cdot | x_1)$ is derived as :*

$$u_t(x | x_1) = \frac{\dot{b}_t}{b_t} x + (\dot{b}_t a_t - b_t \dot{a}_t) \frac{a_t}{b_t} \nabla \log p_{t|1}(x | x_1). \quad (6.19)$$

This relation also holds for their marginal counterpart, both in the guided case and in the unconditional case:

$$\begin{cases} u_t(x | y) = \frac{\dot{b}_t}{b_t} x + (\dot{b}_t a_t - b_t \dot{a}_t) \frac{a_t}{b_t} \nabla \log p_{t|Y}(x | y) \\ u_t(x) = \frac{\dot{b}_t}{b_t} x + (\dot{b}_t a_t - b_t \dot{a}_t) \frac{a_t}{b_t} \nabla \log p_t(x). \end{cases} \quad (6.20)$$

See proof in Eq. (F.4) (Albergo and Vanden-Eijnden, 2023), Eq. (7) (Zheng et al., 2023).

Remark 7. In particular, for linear conditional flow with Gaussian prior, i.e. $b_t = t$, $a_t = 1 - t$, we have $\dot{b}_t = 1$, $\dot{a}_t = -1$, and Equation (6.20) becomes

$$\begin{cases} \nabla \log p_{t|Y}(x | y) = \frac{t u_t(x | y) - x}{1 - t} \\ \nabla \log p_t(x) = \frac{t u_t(x) - x}{1 - t}. \end{cases} \quad (6.21)$$

We note that Equation (6.21) is only defined for $t \in [0, 1)$. As $t \rightarrow 1$, by taking the limit of Equation (6.21) using l'Hopital's rule, the limit of score function is:

$$\begin{cases} \lim_{t \rightarrow 1} \nabla \log p_{t|Y}(x | y) = \lim_{t \rightarrow 1} -\partial_t u_t(x | y) \\ \lim_{t \rightarrow 1} \nabla \log p_t(x) = \lim_{t \rightarrow 1} -\partial_t u_t(x). \end{cases} \quad (6.22)$$

Proof. Here our goal is to prove the limit case Equation (6.22). To simplify notation, we present proof in the *unconditional* setting.

Consider the case of conditional flow at the form $\psi_t(x | x_1) = a_t x + b_t x_1$, where a_t and b_t are chosen to satisfy $\psi_t(x | x_1) = \begin{cases} x & t = 0 \\ x_1 & t = 1 \end{cases}$, x is sample x_0 of RV $X_0 \sim p$, and x_1 is sampled from the target distribution q . The marginalization trick shows that ψ_t generates a p_t satisfying $p_0 = p$ and $p_1 = q$. Using the expression of marginal probability flux (Eq. (14) in Albergo and Vanden-Eijnden, 2023), at $t = 1$, we have:

$$\begin{aligned} j_{t=1}(x) &= \int_{\mathbb{R}^d \times \mathbb{R}^d} [\partial_t \psi_t(x_0 | x_1)]|_{t=1} \delta(x - \psi_{t=1}(x_0 | x_1)) p_0(x_0) p_1(x_1) dx_0 dx_1 \\ &= \int_{\mathbb{R}^d \times \mathbb{R}^d} (\dot{a}_t|_{t=1} x_0 + \dot{b}_t|_{t=1} x_1) \delta(x - x_1) p_0(x_0) p_1(x_1) dx_0 dx_1 \\ &= \int_{x_0 \in \mathbb{R}^d} x_0 p_0(x_0) dx_0 \int_{x_1 \in \mathbb{R}^d} \dot{a}_t|_{t=1} p_1(x_1) \delta(x - x_1) dx_1 + \int_{x_0 \in \mathbb{R}^d} p_0(x_0) dx_0 \int_{x_1 \in \mathbb{R}^d} \dot{b}_t|_{t=1} x_1 p_1(x_1) \delta(x - x_1) dx_1 \\ &= \mathbb{E}[X_0] \dot{a}_1 p_1(x_1) + \dot{b}_1 x_1 p_1(x_1) \end{aligned} \quad (6.23)$$

It follows that the marginal velocity field at $t = 1$ is

$$\begin{aligned} u_1(x) &= j_1(x) / p_1(x) \\ &= \mathbb{E}[X_0] \dot{a}_1 + \dot{b}_1 x_1 \end{aligned} \quad (6.24)$$

If the conditional flow is linear, we have $a_t = (1 - t)$ and $b_t = t$, and therefore $\dot{a}_t = -1$, $\dot{b}_t = 1$. Furthermore, if X_0 is the standard Gaussian, we have $\mathbb{E}[X_0] = 0$. This means that Equation (6.24) becomes $u_1(x) = 0 \times (-1) + 1 \times x_1 = x_1$. Inserting this equality into the numerator in Equation (6.21), both the denominator and the numerator converge to 0 when $t \rightarrow 1$:

$$\begin{cases} t u_t(x) - x \xrightarrow{t \rightarrow 1} 1 \times x_1 - x_1 = 0 \\ 1 - t \xrightarrow{t \rightarrow 1} 1 - 1 = 0 \end{cases} \quad (6.25)$$

By applying l'Hôpital's rule,

$$\begin{aligned}
& \nabla \log p_1(x) \\
&= \lim_{t \rightarrow 1} \frac{\partial_t (tu_t(x) - x) = [\partial_t t]u_t(x) + t[\partial_t u_t(x)] - [\partial_t x] = u_t(x) + t[\partial_t u_t(x)] - u_t(x)}{\partial_t(1-t) = -1} \\
&= \lim_{t \rightarrow 1} [-\partial_t u_t(x)]
\end{aligned} \tag{6.26}$$

It is easy to show that the considerations above also hold in the *guided* case, whereby the marginal items (i.e., the probability flux j_t , flow ψ_t , velocity field u_t , probability path p_t and target distribution q) are expressed in their guided form. \square

Given the guided and unconditional ground truth marginal velocity fields $u_t(\cdot|y)$ and u_t , it is possible to show that MI can be computed exactly, as done in the following

Proposition 5 (MI computation). *Given a linear conditional flow with Gaussian prior, the MI between the target data X and the guidance signal Y is given by*

$$\begin{aligned}
I(X; Y) &= \mathbb{E}_Y \left[\int p_{1|Y}(x_1 | Y) \log \left(\frac{p_{1|Y}(x_1 | Y)}{p_1(x_1)} \right) dx_1 \right] \\
&= \mathbb{E}_Y \left[\int_0^1 \mathbb{E}_{X_t|Y} \left[\frac{t}{1-t} u_t(X_t|Y) \cdot (u_t(X_t|Y) - u_t(X_t)) \right] dt \right].
\end{aligned} \tag{6.27}$$

This can be proven leveraging Equation (5.33), Equation (5.41), and Equation (6.21).

Proof. What needs to be proved in Equation (6.27) is the equivalence of the terms inside the expectation. To keep notation concise, in the following we will rename the guided pair $(p_{t|Y}(\cdot | y), u_t(\cdot | y))$ as simply $(p_t^A(\cdot), u_t^A(\cdot))$, and the marginal pair $(p_t(\cdot), u_t(\cdot))$ as $(p_t^B(\cdot), u_t^B(\cdot))$, so what needs to be proved becomes:

$$\int_{\mathbb{R}^d} p_1^A(x_1) \log \left(\frac{p_1^A(x_1)}{p_1^B(x_1)} \right) dx_1 = \int_0^1 \mathbb{E}_{p_t^A} \left[\frac{t}{1-t} u_t^A(x) \cdot (u_t^A(x) - u_t^B(x)) \right] dt \tag{6.28}$$

To prove Equation (6.28), we start with expanding its LHS:

$$\int_{\mathbb{R}^d} p_1^A(x) \log \left(\frac{p_1^A(x)}{p_1^B(x)} \right) dx \tag{6.29}$$

$$\stackrel{(i)}{=} \int_{\mathbb{R}^d} p_0^A(x) \log \left(\frac{p_0^A(x)}{p_0^B(x)} \right) dx + \int_0^1 \partial_t \int_{\mathbb{R}^d} p_t^A(x) \log \left(\frac{p_t^A(x)}{p_t^B(x)} \right) dx dt \tag{6.30}$$

$$\stackrel{(ii)}{=} 0 + \int_0^1 \partial_t \int_{\mathbb{R}^d} p_t^A(x) \log \left(\frac{p_t^A(x)}{p_t^B(x)} \right) dx dt \tag{6.31}$$

$$\stackrel{(iii)}{=} \int_0^1 \int_{\mathbb{R}^d} \partial_t \left[p_t^A(x) \log \left(\frac{p_t^A(x)}{p_t^B(x)} \right) \right] dx dt \tag{6.32}$$

$$\stackrel{(iv)}{=} \int_0^1 \left[\underbrace{\int_{\mathbb{R}^d} [\partial_t p_t^A(x)] \left[\log \left(\frac{p_t^A(x)}{p_t^B(x)} \right) \right] dx}_{\textcircled{1}} + \underbrace{\int_{\mathbb{R}^d} [p_t^A(x)] \left[\partial_t \log \left(\frac{p_t^A(x)}{p_t^B(x)} \right) \right] dx}_{\textcircled{2}} \right] dt \quad (6.33)$$

where (i) follows from the fundamental theorem of calculus; (ii) follows from the fact that both $p_0^A(x)$ and $p_0^B(x)$ coincide with source distribution p at $t = 0$; (iii) follows from switching differentiation (∂_t) and integration ($\int_{\mathbb{R}^d}$) as justified by Leibniz's rule; (iv) follows from using the Product Rule (i.e. $(u \cdot v)' = u' \cdot v + u \cdot v'$) on ∂_t .

About the term $\textcircled{1}$ in Equation (6.29), using in sequential order (i) the Continuity Equation on $[\partial_t p_t^A(x)]$, (ii) the Product Rule, (iii) the Divergence Theorem, and (iv) the assumption that p_t^A vanishes at infinity gives

$$\begin{aligned} & \int_{\mathbb{R}^d} [\partial_t p_t^A(x)] \left[\log \left(\frac{p_t^A(x)}{p_t^B(x)} \right) \right] dx \\ & \stackrel{(i)}{=} \int_{\mathbb{R}^d} [-\nabla_x \cdot (p_t^A(x) u_t^A(x))] \left[\log \left(\frac{p_t^A(x)}{p_t^B(x)} \right) \right] dx \\ & \stackrel{(ii)}{=} \int_{\mathbb{R}^d} (p_t^A(x) u_t^A(x)) \cdot \left[\nabla_x \log \left(\frac{p_t^A(x)}{p_t^B(x)} \right) \right] dx - \int_{\mathbb{R}^d} \nabla_x \cdot \left(p_t^A(x) u_t^A(x) \log \left(\frac{p_t^A(x)}{p_t^B(x)} \right) \right) dx \\ & \stackrel{(iii)}{=} \int_{\mathbb{R}^d} (p_t^A(x) u_t^A(x)) \cdot \left[\nabla_x \log \left(\frac{p_t^A(x)}{p_t^B(x)} \right) \right] dx - \oint_{\partial \mathbb{R}^d} \left(p_t^A(x) u_t^A(x) \log \left(\frac{p_t^A(x)}{p_t^B(x)} \right) \right) \cdot \mathbf{n} dS \\ & \stackrel{(iv)}{=} \int_{\mathbb{R}^d} (p_t^A(x) u_t^A(x)) \cdot (\nabla_x \log p_t^A(x) - \nabla_x \log p_t^B(x)) dx - 0 \\ & = \mathbb{E}_{p_t^A} \left[u_t^A(x) \cdot (\nabla_x \log p_t^A(x) - \nabla_x \log p_t^B(x)) \right] \end{aligned} \quad (6.34)$$

About the term $\textcircled{2}$ in Equation (6.29), by using in sequence (i) Leibniz's rule and Continuity Equation on $[\partial_t p_t^B(x)]$, and (ii) the equality $\nabla \cdot (\rho \mathbf{v}) = \rho(\nabla \cdot \mathbf{v}) + (\nabla \rho) \cdot \mathbf{v}$ if \mathbf{v} is a vector field and ρ is a scalar function, we obtain

$$\int_{\mathbb{R}^d} [p_t^A(x)] [\partial_t \log \left(\frac{p_t^A(x)}{p_t^B(x)} \right)] dx \quad (6.35)$$

$$= \int_{\mathbb{R}^d} [p_t^A(x)] [\partial_t \log(p_t^A(x)) - \partial_t \log p_t^B(x)] dx \quad (6.36)$$

$$= \int_{\mathbb{R}^d} [p_t^A(x)] \left[\frac{\partial_t p_t^A(x)}{p_t^A(x)} - \frac{\partial_t p_t^B(x)}{p_t^B(x)} \right] dx \quad (6.37)$$

$$= \int_{\mathbb{R}^d} [\partial_t p_t^A(x) - p_t^A(x) \frac{\partial_t p_t^B(x)}{p_t^B(x)}] dx \quad (6.38)$$

$$= \int_{\mathbb{R}^d} \partial_t p_t^A(x) dx - \int_{\mathbb{R}^d} \frac{p_t^A(x)}{p_t^B(x)} \partial_t p_t^B(x) dx \quad (6.39)$$

$$\stackrel{(i)}{=} \partial_t \int_{\mathbb{R}^d} p_t^A(x) dx - \int_{\mathbb{R}^d} \frac{p_t^A(x)}{p_t^B(x)} (-\nabla_x \cdot (p_t^B(x) u_t^B(x))) dx \quad (6.40)$$

$$\stackrel{(ii)}{=} \partial_t 1 + \int_{\mathbb{R}^d} \frac{p_t^A(x)}{p_t^B(x)} \left(u_t^B(x) \cdot \nabla_x p_t^B(x) + p_t^B(x) \nabla_x \cdot u_t^B(x) \right) dx \quad (6.41)$$

$$= 0 + \int_{\mathbb{R}^d} \frac{p_t^A(x)}{p_t^B(x)} u_t^B(x) \cdot \nabla_x p_t^B(x) + \frac{p_t^A(x)}{p_t^B(x)} p_t^B(x) \nabla_x \cdot u_t^B(x) dx \quad (6.42)$$

$$= \int_{\mathbb{R}^d} p_t^A(x) \left(\frac{\nabla_x p_t^B(x)}{p_t^B(x)} \cdot u_t^B(x) + \nabla_x \cdot u_t^B(x) \right) dx \quad (6.43)$$

$$= \int_{\mathbb{R}^d} p_t^A(x) \left((\nabla_x \log p_t^B(x)) \cdot u_t^B(x) + \nabla_x \cdot u_t^B(x) \right) dx \quad (6.44)$$

$$\stackrel{(iii)}{=} 0 \quad (6.45)$$

in which the last equality (iii) follows from Instantaneous Change of Variables: recall that in Instantaneous Change of Variables (Equation (5.41)) $\frac{d}{dt} \log p_t(x) = -\operatorname{div}(u_t)(x)$, its RHS can be rewritten as $-\operatorname{div}(u_t)(x) = -\nabla_x \cdot u_t(x)$; using the chain rule, its LHS is equivalent to $\frac{d}{dt} \log p_t(x) = \frac{d \log p_t(x)}{dx} \cdot \frac{dx}{dt} = (\nabla_x \log p_t(x)) \cdot u_t(x)$. It follows that the Instantaneous Change of Variables yields $(\nabla_x \log p_t^B(x)) \cdot u_t^B(x) + \nabla_x \cdot u_t^B(x) = 0$.

Finally, (i) injecting Equation (6.34) and Equation (6.35) back into Equation (6.29), and (ii) expressing the score functions with velocity fields (Equation (6.21)) give:

$$\begin{aligned} & \int_{\mathbb{R}^d} p_1^A(x) \log \left(\frac{p_1^A(x)}{p_1^B(x)} \right) dx \\ & \stackrel{(i)}{=} \int_0^1 \left[\mathbb{E}_{p_t^A} \left[u_t^A(x) \cdot (\nabla_x \log p_t^A(x) - \nabla_x \log p_t^B(x)) \right] \right] dt \\ & \stackrel{(ii)}{=} \int_0^1 \mathbb{E}_{p_t^A} \left[u_t^A(x) \cdot \left[\frac{t u_t^A(x) - x}{1-t} - \frac{t u_t^B(x) - x}{1-t} \right] \right] dt \\ & = \int_0^1 \mathbb{E}_{p_t^A} \left[\frac{t}{1-t} u_t^A(x) \cdot (u_t^A(x) - u_t^B(x)) \right] dt \end{aligned} \quad (6.46)$$

i.e. Equation (6.28) is proven. \square

Similarly, it is easy to show that, given an individual guidance sample $Y = y$, it is possible to use Equation (6.1) to compute the **point-wise MI** as

$$I(X; y) = \int_0^1 \mathbb{E}_{X_t|Y=y} \left[\frac{t}{1-t} u_t(X_t|Y=y) \cdot (u_t(X_t|Y=y) - u_t(X_t)) \right] dt. \quad (6.47)$$

The integral in Equation (6.1) can be estimated by uniform sampling $t \sim \mathcal{U}(0, 1)$. However, in practice, since the denominator $(1-t) \rightarrow 0$ as $t \rightarrow 1$, this estimator has unbounded variance. To reduce variance, since the argument of the integral has constant magnitude on average, it would be tempting to use importance sampling where $t \sim f(t) \propto \frac{t}{1-t}$. This ratio, however, is hard to normalize (as it integrates to ∞). As an alternative, we consider the following un-normalized density \tilde{f}_ϵ proportional

to such ratio for most of its support, and then truncated to a constant for large t :

$$\tilde{f}_\epsilon(t) = \begin{cases} \frac{t}{1-t} & t \in [0, t_\epsilon) \\ \frac{t_\epsilon}{1-t_\epsilon} & t \in [t_\epsilon, 1] \end{cases} \quad (6.48)$$

To implement such non-uniform sampling in practice, we use the inverse transform sampling method, with the inverse Cumulative Distribution Function (CDF) described in

Proposition 6 (Non-uniform sampling for importance sampling). *The inverse CDF of a PDF proportional to truncated $\frac{t}{1-t}$ is*

$$F_\epsilon^{-1}(u) = \begin{cases} 1 + W(-e^{-Zu-1}) & u \in \left[0, \frac{-\ln(1-t_\epsilon)-t_\epsilon}{Z}\right), \\ 1 + \frac{1-t_\epsilon}{t_\epsilon} [\ln(1-t_\epsilon) + Zu] & u \in \left[\frac{-\ln(1-t_\epsilon)-t_\epsilon}{Z}, 1\right], \end{cases} \quad (6.49)$$

in which W is the Lambert's W -function¹, and the normalizing constant is $Z = -\ln(1-t_\epsilon)$.

Proof. Our goal is to sample from a PDF proportional to $\frac{t}{1-t}$ for most of its support. For some large $t_\epsilon \in [0, 1]$, we define the following un-normalized density (Equation (6.48))

$$\tilde{f}_\epsilon(t) = \begin{cases} \frac{t}{1-t} & t \in [0, t_\epsilon) \\ \frac{t_\epsilon}{1-t_\epsilon} & t \in [t_\epsilon, 1] \end{cases} \quad (6.50)$$

By integrating Equation (6.50) w.r.t. t , we get the cumulative function of the unnormalized density

$$\tilde{F}_\epsilon(t) = \begin{cases} -\ln(1-t) - t & t \in [0, t_\epsilon) \\ -\ln(1-t_\epsilon) + \frac{t_\epsilon}{1-t_\epsilon}(t-1) & t \in [t_\epsilon, 1] \end{cases} \quad (6.51)$$

Evaluating it at $t = 1$ gives us the normalizing constant $Z = \tilde{F}_\epsilon(t = 1) = -\ln(1-t_\epsilon)$, from which we obtain the CDF $F_\epsilon(t) = \frac{\tilde{F}_\epsilon(t)}{Z}$, and the inverse CDF that we need for sampling (using the inverse CDF transform)

$$F_\epsilon^{-1}(u) = \begin{cases} 1 + W(-e^{-Zu-1}) & u \in \left[0, \frac{-\ln(1-t_\epsilon)-t_\epsilon}{Z}\right) \\ 1 + \frac{1-t_\epsilon}{t_\epsilon} [\ln(1-t_\epsilon) + Zu] & u \in \left[\frac{-\ln(1-t_\epsilon)-t_\epsilon}{Z}, 1\right] \end{cases} \quad (6.52)$$

in which W is the Lambert's W -function.

Specifically, to solve the equation

$$\frac{-\ln(1-t) - t}{Z} = u \quad (6.53)$$

¹<https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.lambertw.html>

we denote $w := 1 - t$ and $b := -Zu - 1$ for simplicity, then Equation (6.53) becomes $\ln w = b + w$, which could be rewritten as $-we^{-w} = -e^b$, therefore $-w$ is the Lambert W function $W(-e^b)$. We note that the Lambert W function can be solved because $Z > 0$ and $u \geq 0$ give $-e^b \geq -e^{-1}$. Furthermore, since $-e^b < 0$, both the W_0 and W_{-1} branches of the Lambert W function are defined; but since we are interested in the solution that remains in the range $-1 \leq W(x)$ to make Equation (6.52) well defined, W_0 is the branch of interest. \square

Finally, given the parametric approximations of marginal velocity fields through minimization of CFM loss, and the result in Proposition 5, now we are able to propose an MI estimator defined as

$$I(X; Y) \approx \mathbb{E}_Y \left[\int_0^1 \mathbb{E}_{X_t|Y} \left[\frac{t}{1-t} u_t^\theta(X_t|Y) \cdot (u_t^\theta(X_t|Y) - u_t^\theta(X_t)) \right] dt \right] \quad (6.54)$$

For the estimation of point-wise MI between generated image and guidance prompt, in practice we found that using the velocity field calibrated by CFG as $u_t^\theta(X_t|Y)$ in Equation (6.54) leads to better performance than using the vanilla guided output given by the model. Furthermore, for generating images at high resolution, to reduce training’s computational cost and to speed up inference, it is common to apply RF on a lower dimensional manifold (e.g. the compressed latent space of a pretrained Variational Autoencoder). It is easy to show that the MI between images X and prompts Y equals to that between images’ latents Z and prompts Y , i.e. $I(X; Y) = I(Z; Y)$.

An important property of our estimator is that it is neither an upper nor a lower bound of the true MI, since the difference between the ground truth velocity fields and their parametric approximation can be positive or negative. This property frees our estimation method from the pessimistic results of McAllester and Stratos, 2020.

6.3.2 Experimental evaluation on MI estimation benchmark

In this subsection, we assess the quality of RFMI using a known benchmark (Czyż et al., 2023) composed of 40 tasks with synthetic data generated from a variety of known distributions where the true MI is known, and venturing beyond the typical Gaussian distributions to include harder cases (e.g., distributions with high MI or long tails).

We consider four alternative neural estimators as baselines, namely MINE (Belghazi et al., 2021), InfoNCE (Oord et al., 2019a), NWJ (Nguyen et al., 2010) and DOE (McAllester and Stratos, 2020). All methods are trained/tested using 100k/10k samples, where each sample is composed of two data points x and y concatenated as input for the neural network.

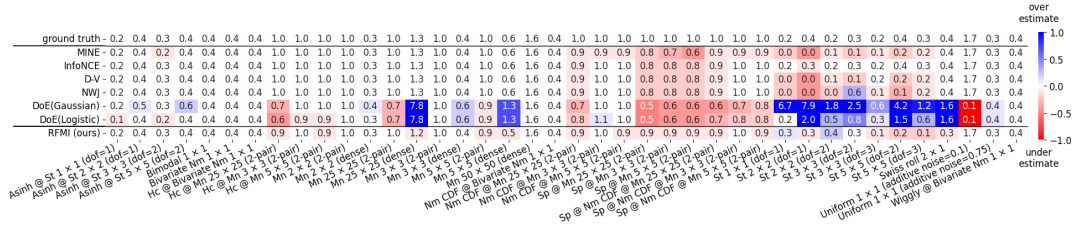


FIGURE 6.1: MI estimation results. Color indicates relative negative bias (red) and positive bias (blue).

Figure 6.1 shows the ground truth MI and each method estimates, with colors reflecting the difference between the MI estimate and the true value – the lighter the shade, the smaller the estimation error. Overall, RFMI is on par or better than alternative methods.

6.4 Conclusion

In this chapter, we focused on MI estimation by leveraging the theory of diffusion-based and flow matching-based generative models.

On DMs, building upon the work of MINDE (Franzese et al., 2024), we extend its computation of KL divergence and entropy of RVs using the score of data distributions to enable the computation of point-wise MI in the discrete-time setting, providing a more fine-grained analysis of MI at the individual data point level.

On RF models, we introduced RFMI, a novel RF-based MI estimator which provides a unique perspective on MI estimation by leveraging the theory of FM-based generative models. To illustrate the effectiveness of RFMI, we considered a synthetic benchmark where the true MI is known and our empirical evaluation showed that RFMI is on par or better than alternative neural estimators.

Given the point-wise MI estimator formulas derived, in Chapter 7 we will use them to evaluate the amount of information “flowing” between natural text and images, and further improve generative models’ T2I alignment by using them to create a synthetic fine-tuning set with a high degree of alignment.

Chapter 7

Information theoretic text-to-image alignment

7.1 Introduction

Generative models used for T2I conditional generation (Rombach et al., 2022; Ramesh et al., 2022; Saharia et al., 2022; Balaji et al., 2022a; Gafni et al., 2022; Podell et al., 2024) have reached impressive performance. In particular, DMs (Song and Ermon, 2019b; Ho et al., 2020b; Kingma et al., 2021; Song and Ermon, 2020; Song et al., 2021; Dhariwal and Nichol, 2021) and RF models (Esser et al., 2024; FLUX, 2023) generate extremely high-quality images by specifying a natural text prompt that acts as a guiding signal (Ho and Salimans, 2022; Nichol et al., 2022; Rombach et al., 2022). Yet, despite achieving a new SOTA, accurately translating prompts into images with the intended semantics is still complex (Conwell and Ullman, 2022; Feng et al., 2023a; Wang et al., 2023a). Issues include catastrophic neglecting (i.e., prompt elements are not generated), incorrect attribute binding (i.e., elements attributes such as color, shape, and texture are missing or wrongly assigned), incorrect spatial layout (i.e., elements are not correctly positioned), and a general difficulty in handling complex prompts (Wu et al., 2024).

On the one hand, quantifying *model alignment* is not trivial. Various works (Hu et al., 2023; Gordon et al., 2023; Grimal et al., 2024) propose different metrics, most of which use complementary Visual Question Answering (VQA) models or Large Language Models (LLMs) to create scores measuring and explaining alignment. Moreover, a recent work (Huang et al., 2023) introduces a comprehensive benchmark suite to ease comparison among different metrics and modeling techniques via “categories”, i.e., a pre-defined set of attribute binding, spatial-related, and other tasks.

On the other hand, addressing T2I model alignment is even more challenging than measuring it. Broadly, we can group the related literature into two main families: *inference-time* and *fine-tuning* methods. For inference-time methods, the key intuition is that the generative process can be optimized by modifying the reverse path of the latent variables. Some works (Chefer et al., 2023a; Li et al., 2023b; Rassin et al.,

2023) mitigate failures by refining the cross-attention units (Tang et al., 2023) of the denoising network of SD (Rombach et al., 2022) on-the-fly, ensuring they attend to all subject tokens in the prompt (typically directly specified as a complementary prompt-specific input for the alignment process) and strengthen their activations. Other inference-time methods (Agarwal et al., 2023; Liu et al., 2022a; Kang et al., 2023; Dahary et al., 2024; Meral et al., 2024; Feng et al., 2023b; Kim et al., 2023; Wu et al., 2023a; Zhang et al., 2024c; Zhang et al., 2024d), focus on individual failure cases. These approaches (i) require a linguistic analysis of prompts, leading to specialized solutions that rely on auxiliary models for prompt understanding, and (ii) result in considerably longer image generation time due to extra optimization costs during sampling.

Considering fine-tuning methods, some works (Wu et al., 2023b; Lee et al., 2023b) require human annotations to prepare a fine-tuning set, while others (Fan et al., 2023; Wallace et al., 2023; Clark et al., 2024) rely on Reinforcement Learning (RL), Direct Preference Optimization (DPO), or a differentiable reward function to steer model behavior. Recent methods use self-playing (Yuan et al., 2024; Xu et al., 2023a; Sun et al., 2023; Wang et al., 2023b; Ma et al., 2023), auxiliary models such as VQA (Li et al., 2023a; Jiang et al., 2024a) or segmentation maps (Kirillov et al., 2023) in a semi-supervised fine-tuning setting. While these methods do not introduce extra inference time costs, they still require human annotation (which is subjective, costly, and does not scale well) and/or auxiliary models to guide the fine-tuning.

Complementary to both families are *heuristic*-based methods that rely on a variety of “tricks”, such as prompt engineering (Witteveen and Andrews, 2022; Liu and Chilton, 2022; Wang et al., 2023a), negative prompting (Negative Prompts 2022; Mahajan et al., 2023; Ogezi and Shi, 2024), prompt rewriting (Mañas et al., 2024) or brute force an appropriate seed selection (Samuel et al., 2024; Karthik et al., 2023). While these methods can be beneficial in specific cases, they fundamentally shift the alignment problem to users.

More important, the literature on T2I alignment primarily focuses on DM such as SD2, while RF-related literature only tangentially considers the problem. For instance, (Li et al., 2024a) extends SD3 with more modalities, (Dalva et al., 2024) enhances FLUX with a linear and fine-grained editing scheme of models’ attention output, (Liu et al., 2024b) proposes a novel text-conditioned pipeline to turn SD into an ultra-fast one-step model – while all these works present CLIP score evaluations, they neither focus nor they are designed to address T2I alignment. At the same time, the intrinsic different nature of the RF models architecture (e.g., SD3 replaces the U-Net of SD2 with a DiT architecture and also add an extra text transformer) calls for redesigning some of the mechanics of DM-based T2I alignment methods.

Overall, current approaches require extra information (human input, auxiliary models, and additional data) and/or are constrained to specific NN architecture. To the

best of our knowledge, no previous work investigates *self-supervised* approaches for T2I alignment, i.e., the use of a pre-trained model to generate images given a specific set of prompts, and select the most aligned ones to prepare a fine-tuning set, without using auxiliary models but instead using the pre-trained model itself to compute a score signaling if the generated images align with the text prompt. In this work, we investigate this strategy from an information theoretic perspective, by using MI to quantify the non-linear prompt-image relationship. In particular, building upon the *point-wise* MI estimation formulas using pre-trained DM or RF as neural estimators that we developed in [Chapter 6](#), by applying them on the generated image and the prompt, this chapter focuses on the two remaining research questions: (2) Is MI meaningful for T2I alignment? (3) How to use the MI estimates to improve T2I alignment?

To tackle these two research questions, our method unfolds as follows. First, taking the DDPM-based MI estimator as an example, we study if and how MI can be used as a meaningful signal to improve T2I alignment, without relying on linguistic analysis of prompts, nor auxiliary models or heuristics. We then proceed with a self-supervised fine-tuning approach, whereby we use point-wise MI to construct a fine-tuning set using synthetic data generated by the T2I model itself. On this constructed dataset, we use the recent adapter presented in (Liu et al., [2024a](#)) to fine-tune a small fraction of weights injected in the T2I model denoising network.

In summary, this chapter presents the following contributions:

1. We empirically study the point-wise MI estimator defined in [Chapter 6](#) between natural prompts and corresponding images considering both qualitative and quantitative approaches. Specifically, taking the DDPM-based MI estimator as an example, we show that MI provides a meaningful indication of alignment with respect to both alignment metrics (BLIP-VQA and HPS) as well as a users study ([Section 7.2](#)).
2. We design a self-supervised fine-tuning approach, called MI-TUNE, that uses a small number of fine-tuning samples to improve the pre-trained T2I model alignment with no inference-time overhead, nor auxiliary models other than the generative model itself ([Section 7.3](#)).
3. We perform an extensive experimental campaign using a recent T2I benchmark suite (Huang et al., [2023](#)) and SD-2.1-base as base DM obtaining sizable improvement compared to six alternative methods. On the same benchmark, we also demonstrate the validity of our MI estimator, taking SD-3.5-Medium as base RF ([Section 7.5](#)).

Those benefits hold also when considering more complex tasks (based on DiffusionDB (Wang et al., [2022c](#))) and alternative base DM (namely, SDXL (Podell et al., [2024](#))). Moreover we study the trade-off between T2I alignment and image quality that has been overlooked in the literature. Specifically, while the

well-known FID, FD-DINO and CMMD metrics suggest a modest image quality/variety deterioration as a consequence of alignment objectives, optimizing the Classifier Free Guidance (CFG) hyper-parameter of the fine-tuned model at generation time, enables finding a “sweet spot” between T2I alignment and image quality. (Section 7.6)

7.2 Is mutual information meaningful for alignment?

To the best of our knowledge, MI has never been evaluated as a *meaningful signal* for T2I alignment. As such, in this section we perform both qualitative and quantitative analyses to investigate this aspect, taking DDPM-based MI estimator as an example.

Qualitative analysis. Starting with a qualitative analysis, we select a set of simple prompts to probe color, texture, and shape attribute binding from T2I-CompBench (Huang et al., 2023) using SD (Rombach et al., 2022) (specifically SD-2.1-base) to generate the corresponding images. We then measure the well-known BLIP-VQA (Huang et al., 2023) and Human Preference Score (HPS) (Wu et al., 2023d) alignment metrics as well as point-wise MI estimates. BLIP-VQA uses a large vision-language model to compute an alignment score, by casting questions against an image to verify that the prompt used to generate it is well represented. HPS is an elaborate metric that uses an auxiliary pre-trained model, blending alignment with aesthetics according to human perception, which are factors that can sometimes be in conflict. Figure 7.1 collects some examples and related metric scores revealing a substantial agreement among all measures: all metrics decrease from left to right in the figure, as prompt-image alignment deteriorates.

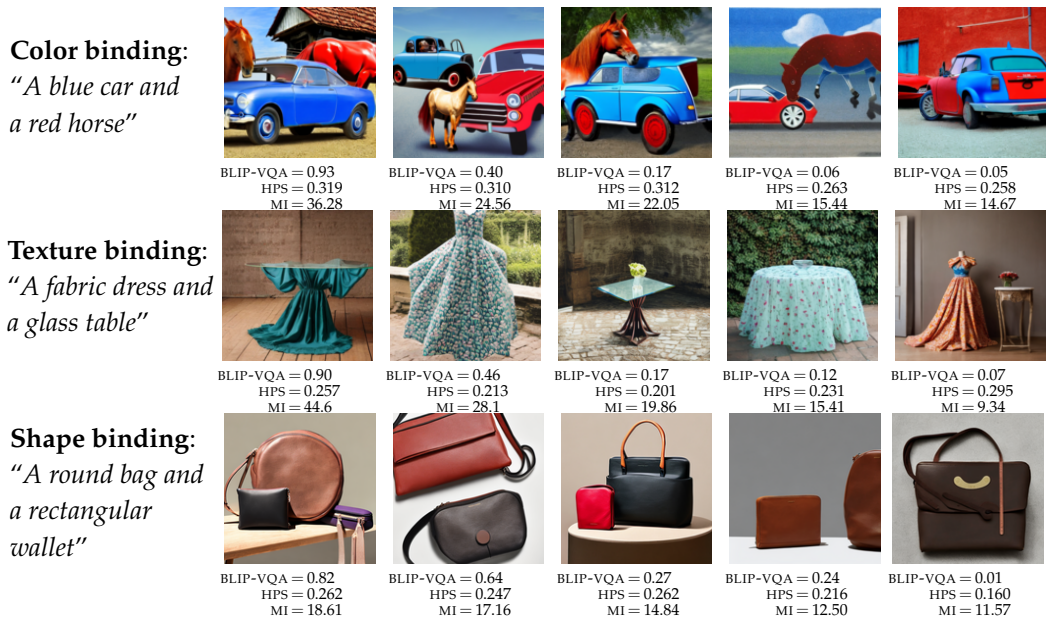


FIGURE 7.1: Qualitative analysis of MI as an alignment measure (all metrics decrease from left to right). See also Appendix B.8.

Quantitative analysis. To quantitatively measure the agreement between MI and well-established alignment metrics, we use all 700 prompts from T2I-CompBench and use SD (again, SD-2.1-base) to generate 50 images per prompt. We use point-wise MI to rank such images and select the 1st, 25th, and 50th. For these three representative images, we compute BLIP-VQA and HPS scores and re-rank them according to both metrics. Last, we measure agreement between the three rankings using Kendall’s τ method (Kendall, 1938), and average results across all prompts. Results indicate good agreement between MI and BLIP-VQA ($\tau = 0.4$), and a strong agreement between MI and HPS ($\tau = 0.68$).

To strengthen our analysis, we also perform a users study eliciting human preference (see [Appendix B.1.1](#) for details). Given a randomly selected prompt from T2I-CompBench that users can read, we present the top-ranked generated image (among the 50) according to MI, BLIP-VQA and HPS, in a randomized order. Users can select one or more images to indicate their preference regarding alignment and aesthetics, for a total of 10 random prompts per user. From the 102 surveys from 46 users, we find that human preference for prompt-image pairs goes to MI for 69.1%, BLIP-VQA for 73.5% and HPS for 52.2% of the cases, respectively.

Relevant literature. Overall, our analyses support our intuition by which *MI is a meaningful signal for alignment* (and possibly aesthetics too), setting the stage for our T2I alignment method. Our intuition is also supported by recent studies investigating the information flow in the generative process of diffusion models. Specifically, Kong et al., 2024 estimates pixel-wise mutual information between natural prompts and the images generated at each time-step of a backward diffusion process. They compare such “information maps” to cross-attention maps (Tang et al., 2023) in an experiment involving prompt manipulation – modifications of the initial prompt during reverse diffusion – and conclude that MI is much more sensitive to information flow from prompt to images. In a similar vein, Franzese et al., 2024 compute MI between prompt and images at different stages of the reverse process of image generation. Experimental evidence indicates that MI can be used to analyze various reverse diffusion phases: noise, semantic, and denoising stages (Balaji et al., 2022b). While previous studies do not explicitly focus on alignment, they *indirectly* support our intuition that MI estimated using a diffusion model gauges the amount of information a text prompt conveys about an image (and vice-versa) which is key for T2I alignment.

7.3 Self-supervised fine-tuning with MITUNE

The T2I alignment problem arises when user’s intentions, as expressed through natural text prompts, fail to materialize in the generated image. Our novel approach

aims to address alignment using a theoretically grounded MI estimation, that applies across various contexts. To improve model alignment, we introduce a self-supervised fine-tuning method. Leveraging the T2I model itself, we estimate MI and generate an information-theoretic enhanced fine-tuning dataset. While our focus in this chapter is on T2I alignment, our framework remains extensible to other modalities.

In summary, given a pre-trained diffusion model (e.g., SD-2.1-base (Rombach et al., 2022) or SDXL (Podell et al., 2024)) or rectified flow model (e.g., SD3 (Esser et al., 2024) or Flux.1 (FLUX, 2023)), we leverage our point-wise MI estimation method to select a small fine-tuning dataset set of information-theoretic aligned examples.

Our self-supervised alignment method **relies on the pre-trained model only** to produce a given amount of fine-tuning data, which is then filtered to retain prompt-image pairs with a high degree of alignment, according to pair-wise MI **estimates obtained using only the pre-trained model**. We begin with a set of fine-tuning prompts \mathcal{Y} , which can be either manually crafted, or borrowed from available prompt collections (Wang et al., 2023a; Huang et al., 2023). Ideally, fine-tuning prompts should be conceived to stress the pre-trained model with challenging attribute and spatial bindings, or complex rendering tasks.

As described in Algorithm 1, for each prompt $y^{(i)}$ in the fine-tuning set \mathcal{Y} , we use the pre-trained model to generate a fixed number M of synthetic images. Given prompt-image pairs $(y^{(i)}, z^{(j)})$, $j \in [1, M]$, we estimate pair-wise MI and select the top k pairs, which will be part of the model fine-tuning dataset \mathcal{S} . Finally, we augment the pre-trained model with adapters (Hu et al., 2021; Liu et al., 2024a), and proceed with fine-tuning.

We study the impact of the adapter choice, and whether only the denoising network or both the denoising and text encoder networks should be fine-tuned (Appendix B.4.2). Moreover, we measure the impact of the number of fine-tuning rounds R to the pre-trained model, i.e., we renew the fine-tuning dataset \mathcal{S} using the fine-tuned model, and re-fine-tune it using Algorithm 1 (Section 7.6).

Our efficient implementation combines latent generation and point-wise MI computation, as shown in Algorithm 2 and Algorithm 3 for DM and RF respectively. Since MI estimation involves computing an expectation over denoising times t , it is easy to integrate the MI estimation into the same generation loop. Moreover, the function is easy to parallelize to significantly speed up the fine-tuning set \mathcal{S} composition.

Algorithm 1: MI-TUNE

Input : Pre-trained model: $f_\theta = \epsilon_\theta$ (resp. u_θ) for DM (resp. RF), Prompt set: \mathcal{Y}

Hyper par : Image pool size: M ; Top MI-aligned samples: k

Output : Fine-tuned $f_{\theta^*} = \epsilon_{\theta^*}$ (resp. u_{θ^*}) for DM (resp. RF)

// Fine-tuning set

```

1  $\mathcal{S} \leftarrow []$ 
2 for  $y^{(i)}$  in  $\mathcal{Y}$  do
3   for  $j \in \{1, \dots, M\}$  do
4     // Generate and compute MI
4      $z^{(j)}, I(z^{(j)}, y^{(i)}) = \text{PointWiseMI}(f_\theta, y^{(i)})$ 
5     // Append samples and MI
5      $\mathcal{S}[y^{(i)}].\text{append}(z^{(j)}, I(z^{(j)}, y^{(i)}))$ 
6   end
7   // Retain only Top- $k$  elements
7    $\mathcal{S}[y^{(i)}] = \text{Top-}k(\mathcal{S}[y^{(i)}])$ 
8 end
9 return  $f_{\theta^*} = \text{FineTune}(f_\theta, \mathcal{S})$ 

```

Algorithm 2: Point-wise MI Estimation - DM

Input : Pre-trained model: ϵ_θ ; Prompt: y

Output : Generated latent: z ; Point-wise MI: $I(z, y)$

```

1 Function PointWiseMI( $\epsilon_\theta, y$ ):
2   // Initial latent sample
2    $z_T \sim \mathcal{N}(\mathbf{0}, I)$  for  $t$  in  $T, \dots, 0$  do
3     // MI estimation (eq. (6.1))
3      $I(z_t, y) += [\kappa_t ||\epsilon_\theta(z_t, y, t) - \epsilon_\theta(z_t, \emptyset, t)||^2]$ 
4     // Noise sample
4      $w \sim \mathcal{N}(\mathbf{0}, I)$  if  $t > 1$ , else  $w = \mathbf{0}$ 
5     // Sampling step
5      $z_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( z_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(z_t, y, t) \right) + \sigma_t w$ 
6   end
7   return  $z, I(z, y)$ 

```

7.4 Experimental protocols

7.4.1 Benchmark and metrics

Benchmark. On DM, we compare all techniques using T2I-CompBench (Huang et al., 2023), a benchmark composed of 700/300 (train/test) prompts across 6 *categories* including attribute binding (color, shape, and texture categories), object relationships (2D-spatial and non-spatial associations), and complex composition tasks. These prompts were generated with predefined rules or ChatGPT (OpenAI, 2024). We also assess MI-TUNE performance on more realistic prompts by sampling 5,000/1,250 (train/test) prompt-image pairs from DiffusionDB (Wang et al., 2022c),

Algorithm 3: Point-wise MI Estimation - RF

Input : Pre-trained model: u_θ ; Prompt: y
Hyperparam: Step size: Δt
Output : Generated clean latent: z ; Point-wise MI: $I(z, y)$

```

1 Function PointWiseMI( $u_\theta, y$ ):
    // Initial latent sample
2    $z_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
3    $I(z, y) = 0$ 
4   for  $t$  in  $0, \dots, 1$  do
        // MI estimation (eq. (6.47))
5        $I(z, y) += \frac{t}{1-t} u_\theta(z_t, y, t) \cdot (u_\theta(z_t, y, t) - u_\theta(z_t, \emptyset, t))$ 
        // Denoising step
6        $z_{t+} = u_\theta(z_t, y, t) \Delta t$ 
7   end
8   return  $z, I(z, y)$ 

```

a large-scale dataset composed of complex human-crafted prompts paired with the corresponding images generated from a SD model.

On RF, we evaluate our method on T2I-CompBench++ (Huang et al., 2023), an improved version of T2I-CompBench, composed of 700/300 (train/test) prompts across 8 categories including attribute binding (color, shape, and texture categories), object relationships (2D-spatial, 3D-spatial, and non-spatial associations), numeracy and complex composition tasks. As Huang et al. (2023) recently found, Stable Diffusion 3 already “saturates” performance on certain categories (see Table XIII in Huang et al., 2023): then, we focus our evaluation only on categories having an alignment score lower than 0.7, namely the 4 categories shape, 2D-spatial, 3D-spatial, and numeracy.

Alignment Metrics. Evaluating T2I alignment is difficult as it requires a detailed understanding of prompt-images pairs, and many metrics have been proposed, e.g., CLIP (Hessel et al., 2021; Radford et al., 2021), MINIGPT-4 (Zhu et al., 2024), and human evaluation. In this chapter we use BLIP-VQA (Huang et al., 2023), HPS (Wu et al., 2023c) and UniDet (Zhou et al., 2022a). While BLIP-VQA computes a score with a questions-answers approach – a given prompt is decomposed and each part is transformed into a question for an auxiliary VQA model; then, answers are aggregated into a single score – only based on alignment, HPS includes both alignment and aesthetics – this is enabled by an auxiliary model pre-trained using human-annotated data. As in (Huang et al., 2023), the 2D-spatial category is evaluated using the UniDet object detection model.

For the experiment on DM, we complement these metrics with a user study. We randomly select 100 prompts per category, and generate 10 pictures per prompt for each method we consider in our evaluation. Then, we run surveys composed of 12

rounds (2 for each category), each showing to the user a randomly selected prompt and a randomly selected image for each method, randomly arranged in a grid. At each round, users need to select zero or more images they consider aligned with the prompt. Overall, we collected 42 surveys from 5 users, from which we computed the total percentage of times each method was selected for each category ([Appendix B.1.2](#)).

Image quality metrics. Assessing performance only considering alignment metrics can hide undesired effects. Intuitively, a strong adherence to a given prompt reduces the generative process “degrees of freedom” and this trade-off might not be visible even to a trained eye. To investigate these dynamics we compute FID (Heusel et al., 2017), FD-DINO (Oquab et al., 2024) and CMMD (Jayasumana et al., 2024) scores – FID favors natural colors and textures but struggles to detect objects/shapes distortion, while FD-DINO and CMMD favor image content. Following (Imagen-Team et al., 2024), rather than using the T2I-CompBench test set, we compute the metrics using 30k samples of the MS-COCO-2014 (Lin et al., 2015) validation set.

7.4.2 MI-TUNE fine-tuning

Base models. We mainly run our benchmark using SD-2.1-base and SD-3.5-M as base model for DM and RF respectively, but we also report results of the application of MI-TUNE on a variant of DM at higher resolution (namely SDXL) to demonstrate its flexibility.

Fine-tuning sets. Both T2I-CompBench and its enlarged version T2I-CompBench++ contain 700 training prompts for each category. When using MI-TUNE, we generate $M = 50$ images for each prompt using the pre-trained model, compute their point-wise MI, and select the top $k = 1$ ¹ (sensitivity to M and k in [Appendix B.4.1](#)). For the 2D-Spatial category, we also compose fine-tuning sets generating images from SPRIGHT (Chatterjee et al., 2024) – a DM optimized for this (more challenging) category and fine-tuned from SD-2.1 (a higher resolution version of SD-2.1-base). Last, we also contrast MI-TUNE fine-tuning set composition against (i) using HPS rather than MI for image selection,² (ii) using both MI-selected and real-pictures and (iii) images from DiffusionDB.

Fine-tuning weights. In this chapter, fine-tuning corresponds to injecting DoRA (Liu et al., 2024a) (resp. LoRA (Hu et al., 2021)) adapters for DM (resp. RF)

¹We remark that, albeit in a different context, this selection resembles an image retrieval task (Krojer et al., 2023)

²We exclude BLIP-VQA for the fine-tuning set composition to avoid biasing the evaluation (Huang et al., 2023).

(rank and scaling factor α are set to 32) only into the attention layers and fully connected layers of the denoising network, whereas other layers are frozen.³

Other hyperparams search. On DM, we consider up to $R \in [1, 3]$ rounds of fine-tuning i.e., using as base model the one obtained from previous round and apply Algorithm 1, and Classifier Free Guidance (CFG) $\in [2.5, 7.5]$. For each fine-tuned model we then compute all alignment and image quality metrics. On RF, due to the computational cost, we only finetune $R = 1$ round using the pre-trained base model, and apply Classifier Free Guidance (CFG) = 4.5. More fine-grained hyperparams details and computational costs considerations in Appendix B.2.

7.4.3 Alternative methods

As introduced in Section 7.1, the literature on T2I alignment primarily focuses on DM methods like SD2, while RF-related works neither focus on nor are designed to address T2I alignment. Therefore, in the benchmark evaluation, we only consider the alternative methods for DM.

Inference-time methods. Pre-trained model alignment can be improved at inference by optimizing the latent variables z_t throughout the numerical integration used to generate the (latent) image. This process steers model alignment with an auxiliary loss based on attention maps and fine-grained linguistic analysis of the prompt (e.g., additional input is used to explicitly indicate which words to focus on). In this family, we consider 3 methods: Attend and Excite (A&E) (Chefer et al., 2023b), Structured Diffusion Guidance (SDG) (Feng et al., 2023b) and Semantic-aware Classifier-Free Guidance (SCG) (Shen et al., 2024).

Fine-tuning methods. Alternatively, a pre-trained model can be fine-tuned with adapters (Hu et al., 2021) optimized via a variety of RL or supervision methods. Specifically, we consider 3 approaches: Diffusion Policy Optimization with KL regularization (DPOK) (Fan et al., 2023), Generative mOdel finetuning with Reward-driven Sample selection (GORS) (Huang et al., 2023) and Hard-Negatives Image-Text-Matching (HN-ITM) (Krojer et al., 2023). Notice that since results in the literature for both families do not necessarily refer to same base models, to guarantee a fair comparison, we adapted and evaluated all methods on SD-2.1-base.

³LoRA adapters (Hu et al., 2021) and fine-tuning also the CLIP-based text encoder do not provide performance improvements (Appendix B.4.2). Likewise, creating a multi-category model by “merging” different per-category models or using a fine-tuning set composed with images from all categories do not provide performance gains (Appendix B.4.3).

7.5 Benchmark results

7.5.1 T2I-CompBench on DM

Table 7.1 reports the alignment results on T2I-CompBench. To simplify its reading, the bottom part of the table summarizes (i) the absolute gain with respect to the SD-2.1-base model for each of the best methods in each family and (ii) the percentage gains of MI-TUNE with respect to the alternative method for each category. We also summarize performance as averages across categories for each metric.

Despite performance varies, MI-TUNE achieves a new state of the art across all categories/metrics, often by a sizable margin. While this is more evident for BLIP-VQA and Human, the literature shows that HPS has natural small variations (see [Appendix B.3](#)), hence MI-TUNE gains are significant also for this metric.

Table 7.1 results are obtained generating fine-tuning sets from SD-2.1-base for all tasks but 2D-Spatial. For this category, we were able to obtain (at best) BLIP-VQA=15.93 and HPS=28.13. Conversely, generating the fine-tuning images from SPRIGHT resulted beneficial. We can link this result to the self-supervision nature of MI-TUNE. On the one hand, our methodology is not bounded to a specific model. On the other hand, the filtering operated via point-wise MI estimation can benefit from “pre-alignment” – MI-TUNE can strengthen existing alignment but might not be sufficient to “induce” it. Notice that all competitors suffer from this trade-off too as no single winner emerges. In particular, despite A&E and GORS are the most frequent best method in their family (winning in 10-out-of-18 scenarios), all competitors show less consistent performance across categories and metrics than MI-TUNE. For instance, for attribute binding (color, shape and texture), fine-tuning methods under-perform according to BLIP-VQA and Human, but the performance gaps are very close considering HPS. Yet, MI-TUNE achieves consistently higher performance across all categories, outperforming alternative fine-tuning methods by a large margin.

Raw alignment performance apart, it is important to highlight MI-TUNE key differences compared to the alternative fine-tuning methods. DPOK uses RL with a reward model (pre-trained with human-labeled real images) to define a prompt-image alignment score to guide the fine-tuning, HN-ITM uses a contrastive learning approach based on an ad-hoc dataset with real positive (good alignment) and negative (poor alignment) prompt-image pairs, and GORS composes a fine-tuning set generating images from the diffusion model and selecting them based on BLIP-VQA. While GORS is very close in spirit to MI-TUNE, its performance is “biased” – the filtering criteria overlaps with the final evaluation strategy – as explicitly acknowledged by its authors (Huang et al., 2023). Overall, while both DPOK and GORS still require external assistance, MI-TUNE generates images *and* selects them using the target model itself, i.e., it is the first fully self-supervised model for T2I alignment to the best of our knowledge.

Method	BLIP-VQA							HPS							Human (user study)						
	Color	Shape	Texture	2D-Sp.	Non-Sp.	Compl.	(avg)	Color	Shape	Texture	2D-Sp.	Non-Sp.	Compl.	(avg)	Color	Shape	Texture	2D-Sp.	Non-Sp.	Compl.	(avg)
SD-2.1-base	49.65	42.71	49.99	15.77	66.23	50.53	(45.81)	27.64	24.56	24.99	27.50	26.66	25.70	(26.17)	29.76	11.90	40.48	35.71	66.67	29.76	(35.71)
Infer.	A&E	61.43	47.39	64.10	16.18	66.21	51.69 (51.17)	28.44	24.43	25.88	28.42	26.60	25.60	(26.56)	31.95	15.48	52.38	32.14	65.48	30.95	(38.06)
	SDG	47.15	45.24	47.13	15.25	66.17	47.41 (44.72)	27.25	24.40	24.71	27.10	26.12	25.83	(25.90)	26.19	15.48	38.10	38.10	61.90	29.76	(34.92)
	SCG	49.82	43.28	50.16	16.31	66.60	51.07 (46.21)	27.86	24.85	25.57	27.76	26.98	26.03	(26.51)	20.24	11.90	33.33	40.48	69.05	39.29	(35.71)
FT	DPOK	53.28	45.63	52.84	17.19	66.95	51.97 (47.98)	28.20	24.99	25.44	28.12	26.80	25.88	(26.57)	23.81	16.67	47.62	34.52	70.24	38.10	(38.49)
	GORS	53.59	43.82	54.47	15.66	67.47	52.28 (47.88)	28.15	24.79	25.56	27.90	26.88	26.07	(26.56)	34.52	14.29	48.81	36.90	65.48	30.95	(38.49)
	HN-ITM	46.51	39.99	48.78	15.24	65.31	49.84 (44.28)	26.90	24.33	24.63	27.15	25.40	25.22	(25.60)	23.81	19.05	30.95	20.24	47.62	23.81	(27.58)
MI-TUNE	65.04	50.08	65.82	18.51	67.77	54.17	(53.56)	29.13	25.57	26.20	28.50	27.15	26.70	(27.21)	46.43	25.01	53.19	45.24	73.81	46.43	(48.35)
best Infer. \square base	11.78	4.68	14.11	0.54	0.37	1.16	(5.44)	0.80	0.29	0.89	0.92	0.32	0.33	(0.59)	2.19	3.58	11.90	4.77	2.38	9.53	(5.72)
best FT \square base	3.94	2.92	4.48	1.42	1.24	1.75	(2.62)	0.56	0.43	0.57	0.62	0.22	0.37	(0.46)	4.76	7.15	8.33	1.19	3.57	8.34	(5.56)
MI-TUNE \square base	15.39	7.37	15.83	2.74	1.54	3.64	(7.75)	1.49	1.01	1.21	1.00	0.49	1.00	(1.03)	16.67	13.11	12.71	9.53	7.14	16.67	(12.64)
MI-TUNE \square best*	3.61	2.69	1.72	1.32	0.30	1.89	(1.92)	0.69	0.58	0.32	0.08	0.17	0.63	(0.41)	11.91	5.96	0.81	4.76	3.57	7.14	(5.69)
MI-TUNE % best*	5.88	5.68	2.68	7.68	0.44	3.62	(4.33)	2.43	2.32	1.24	0.28	0.63	2.42	(1.55)	34.50	31.29	1.55	11.76	5.08	18.17	(17.06)

A \square B indicates the absolute difference between A and B; A % B corresponds to the percentage difference $(A - B) / B$; †: Fine-tuning set obtained from SPRIGHT rather than SD-2.1-base; Human scores do not sum to 100 in each category as users can select multiple methods for each question.

TABLE 7.1: T2I-CompBench alignment results (%) on images generated by DM. Gray highlighted style when MI-TUNE outperforms all competitors; Grayed text for under-performing methods per-family; Green heatmaps show per-category absolute gains w.r.t. the base model.

Category	Shape (BLIP-VQA)	2D-spatial (UniDet)	3D-spatial (UniDet) ²	Numeracy (UniDet)
SD3.5-M	57.96	31.31	38.81	61.15
MI-TUNE	61.78	33.92	42.28	64.00
<i>abs. difference</i>	3.82	2.61	3.47	2.85
<i>relative gain</i>	6.59	8.34	8.94	4.66

TABLE 7.2: T2I-CompBench alignment results (%) on images generated by RF with CFG=4.5

7.5.2 T2I-CompBench on RF

We applied MI-TUNE (Algorithm 1) on Stable Diffusion 3.5-Medium Esser et al., 2024 (SD3.5-M)³ and extended Huang et al. benchmark to include this latest RF-based T2I model. Table 7.2 collects the results, with absolute difference and percentage gain between SD3.5-M and MI-TUNE summarized at the bottom. As expected, MI-TUNE improves the T2I alignment of SD3.5-M by a sizable margin across all the 4 challenging categories. Qualitative visualization examples are shown in Figure 7.2.

We highlight that our approach MI-TUNE is not sensitive to the NN architecture or the type of data, so it could be integrated beyond T2I task and into other disciplines where DM or RF is adopted for conditional generation.

³From a preliminary investigation we observed a $\times 4$ computational costs for FLUX (FLUX, 2023) so we could not collect results in time for the submission.

²For prompts depicting 3D-spatial relation, T2I-CompBench++ leverages UniDet (Zhou et al., 2022b) for object detection and Dense Vision Transformer (Ranftl et al., 2021) for depth estimation.



FIGURE 7.2: Qualitative examples from Table 7.2 (same seed used for a given prompt).

7.6 Ablation studies

In this section, we ablate our method MI-TUNE on DM, considering the trade-off between alignment and image quality-variety, and alternative prompt sets and base models.

Alignment/image quality-variety trade-offs. MI-TUNE results in Table 7.1 are obtained from a grid search across multiple fine-tuning rounds R and CFG values. In fact, we observe different trade-offs between alignment and image quality across different configurations. We exemplify this in Figure 7.6, for the Color category. The figure highlights two opposite dynamics: T2I alignment benefits from multiple fine-tuning rounds (higher BLIP-VQA) but can introduce image artifacts and reduce measured diversity (higher FID). While this trade-off is neither mentioned nor quantified in the literature of the considered methods, it is to be expected – strictly abiding to a prompt impacts the “generative pathways” at sampling time. Interestingly, lowering CFG (typically set to 7.5) counterbalances these dynamics and enables a “sweet spot” – as the model better aligns to a category thanks to fine-tuning, one can alleviate the guidance scale dependency at generation. Table 7.3 complements this analysis by showing FID, FD-DINO and CMMD scores for all categories, as well for SD-2.1-base and three state of the art models – while all metrics indeed suggest

a possible reduction in image variety considering SD-2.1-base, MI-TUNE scores are comparable with other state-of-the-art models (see [Figure 7.3](#) for example images).

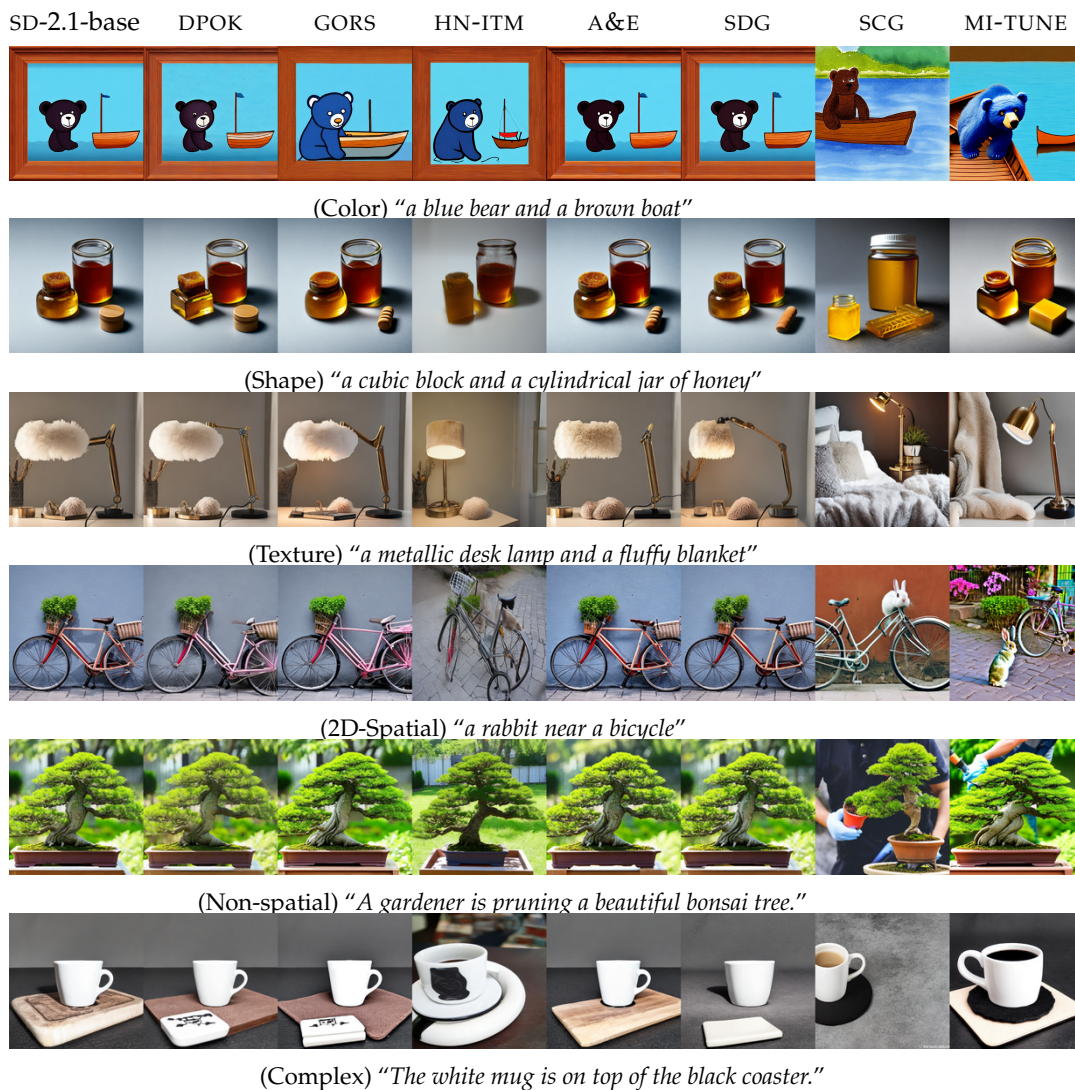


FIGURE 7.3: Qualitative examples from [Table 7.1](#) (same seed used for a given prompt). More examples in [Appendix B.5](#).

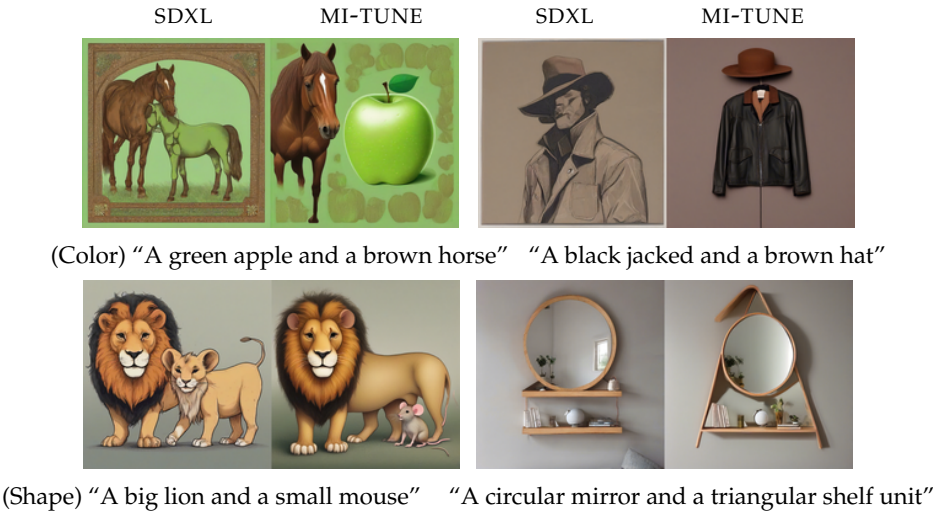


FIGURE 7.4: Qualitative examples from Table 7.5 (same seed used for a given prompt). More examples in Appendix B.6.

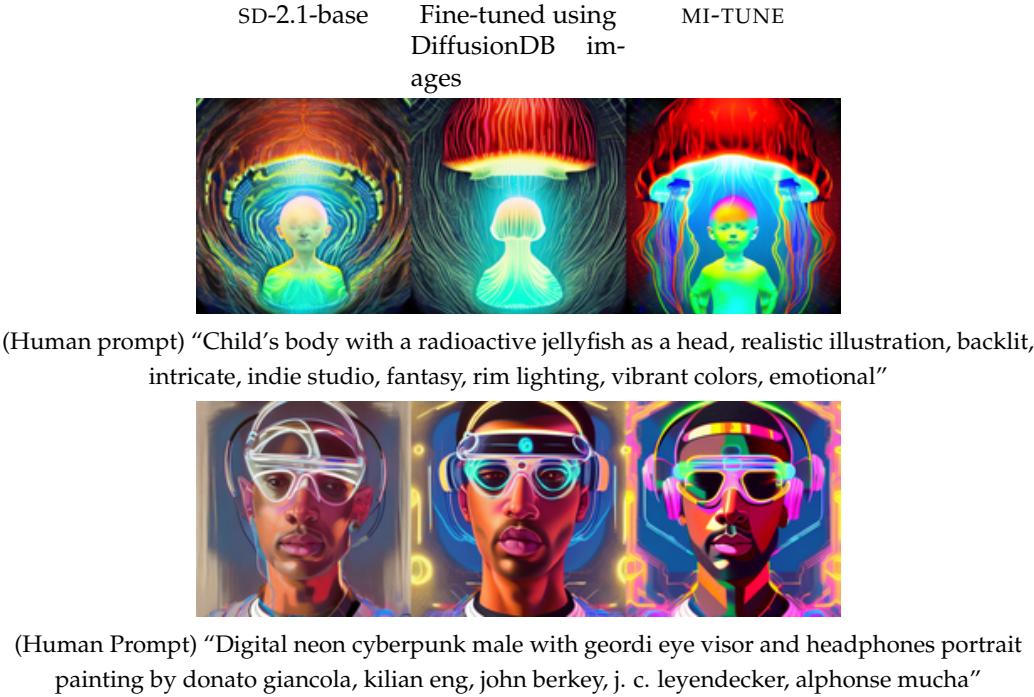


FIGURE 7.5: Qualitative examples from Table 7.6 (same seed used for a given prompt). More examples in Appendix B.7.

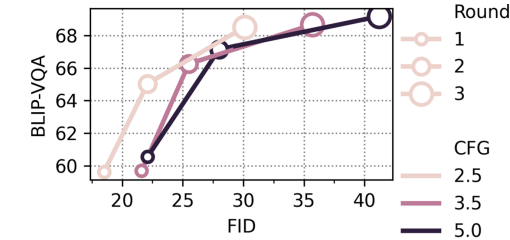


FIGURE 7.6: Hyper-params search.

TABLE 7.3: Comparing image quality/variety scores.

Metric	SD-2.1-base	MI-TUNE ($R=2, CFG=2.5$)							DALLE-3 [‡] IMAGEN-3 [‡] SDXL [‡]		
		Color	Shape	Texture	Spatial	Non-sp.	Comp.	(avg)			
FID(↓)	17.1	22.1	16.8	17.3	18.8	16.8	20.6	(18.7)	20.1	17.2	13.2
FD-DINO(↓)	229.1	279.0	236.9	250.4	251.7	231.9	255.6	(250.9)	284.4	213.9	185.6
CMMD(↓)	0.641	0.681	0.634	0.694	0.669	0.709	0.671	(0.680)	0.894	0.854	0.898

Results from 30k samples of MS-COCO-2014 validation set; [‡] results from (Imagen-Team et al., 2024)

Fine-tuning set composition. The strategy to select prompt-image pairs for the fine-tuning set has a large design space beyond the use of MI. In Table 7.4, we report (for two categories for brevity) alignment performance using two alternative strategies. Specifically, using HPS rather than MI degrades performance.⁴ Results when composing the fine-tuning set by mixing MI-selected and real images selected from the CC2M dataset (Changpinyo et al., 2021) are instead inconsistent (BLIP-VQA steadily degrades but HPS signals an improvement in some scenarios).

TABLE 7.4: FT set selection.

Strategy	BLIP-VQA		HPS	
	Color	Shape	Color	Shape
MI only	65.04	50.08	29.13	25.57
HPS only	59.43	46.87	n.a.	n.a.
MI+Real(0.25)	61.34	48.47	29.16	25.87
MI+Real(0.5)	61.63	49.50	29.38	25.92
MI+Real(0.9)	59.83	48.92	28.60	25.60

SDXL and DiffusionDB. We complete our evaluation by presenting results obtained applying MI-TUNE on SDXL in Table 7.5, and considering an alternative scenario closer to real user application using DiffusionDB in Table 7.6 to complement the synthetic nature of T2I-CompBench. As expected, “vanilla” SDXL significantly outperforms SD-2.1-base, yet MI-TUNE enables sizable improvements on SDXL alignment (see Figure 7.4). For the realistic alignment use case in Table 7.6, we select prompt-images pairs from DiffusionDB and we contrast alignment when fine-tuning using the images already paired with prompts against MI-selected ones. We use SD-2.1-base as base model and report only HPS scores⁵ in Table 7.6. Overall, fine-tuning with DiffusionDB images improves the base model, yet MI-TUNE enables superior performance (see Figure 7.5).

⁴We compute only BLIP-VQA to avoid evaluation bias (Huang et al., 2023).

⁵The higher prompt complexity does not well suit BLIP-VQA text decomposition (see Appendix B.7.1).

TABLE 7.5: Alignment (%) using SDXL.

Method	BLIP-VQA						HPS					
	Color	Shape	Texture	2D-Sp.	Non-Sp.	Comp.	Color	Shape	Texture	2D-Sp.	Non-Sp.	Comp.
(ref) SDXL	60.78	49.70	55.78	21.02	68.16	52.68	28.47	24.99	25.85	28.50	26.64	25.90
SD-2.1-base	49.65	42.71	49.99	15.77	66.23	50.53	27.64	24.56	24.99	27.50	26.66	25.70
MI-TUNE	69.66	55.86	66.74	22.18	72.17	57.74	29.03	25.90	27.15	29.57	27.56	26.70
MI-TUNE \ominus (ref)	8.88	6.16	10.96	1.16	4.01	5.06	0.56	0.91	1.30	1.07	0.92	0.80
MI-TUNE % (ref)	14.61	12.39	19.65	5.52	5.88	9.61	1.97	3.64	5.03	3.75	3.45	3.09

TABLE 7.6: DiffusionDB.

Model	HPS
SD-2.1-base	23.99
DiffusionDB	24.35
MI-TUNE	25.32
MI-TUNE \ominus base	1.33
MI-TUNE \ominus DiffusionDB	0.97

7.7 Conclusion

T2I alignment emerged as an important endeavor to steer image generation to follow the semantics and user intent expressed through a natural text prompt, as it can save considerable manual effort. In this chapter, we presented a novel approach to improve model alignment, that uses point-wise MI between prompt-image pairs as a meaningful signal to evaluate the amount of information “flowing” between natural text and images. We demonstrated, both qualitatively and quantitatively, that point-wise MI is coherent with existing alignment measures that either use auxiliary VQA models or elicit human intervention.

We presented MI-TUNE, a lightweight, self-supervised fine-tuning method that uses a pre-trained T2I model such as diffusion-based SD2 or rectified flow-based SD3 to estimate MI, and to generate a synthetic set of aligned prompt-image pairs, which is then used in a parameter-efficient fine-tuning stage, to align the T2I model. Our approach does not require human annotation, auxiliary VQA models, nor costly inference-time techniques, and achieves a new state-of-the-art across all categories/-metrics explored in the literature, often by a sizable margin. These results carry on in more complex tasks, and for various base models, illustrating the flexibility of MI-TUNE.

Part III

Generative modeling of packet series for traffic classification

Chapter 8

Generative modeling of network traffic data

8.1 Introduction

In **Part I**, our investigation began with hand-crafted augmentations for packet series (**Part I**). While these augmentations showed empirical effectiveness for traffic classification, they suffer from limited robustness: no single transformation consistently performs the best across datasets, and the hyperparameter searching space — including the intensity of individual augmentations and their combinations — is large and hard to tune. In short, it is challenging to manually design hand-crafted augmentations for packet series that both introduce substantial variability and preserve class-discriminative features. In **Part II**, we turned to conditional diffusion models for text-to-image synthesis, where the model was able to generate diverse, realistic images that are also semantically aligned with the given text prompt. Motivated by these findings in the two parts, it is natural to ask whether similar diffusion-based approaches can be effective in networking domain, which leads to a natural question: can class-conditional diffusion models capture the distribution of packet series in a way that preserves both fidelity and class semantics? Furthermore, can such models generate meaningful diversity beyond the training set, making it effective for downstream classification augmentation?

In this chapter, we investigate the generative modeling of packet series. We begin by reviewing relevant literature from both networking and ML communities. Our survey of networking literature reveals its sparsity in terms of experimental datasets and input representations. This motivates our first research objective: to build a benchmark that includes curated datasets, standardized processing pipelines, and implementations of existing methods. On the ML side, since the input data to the ML model is composed by the sizes of the first 30 packets multiplied by their directions (+1 for outgoing and -1 for incoming), we examine works on diffusion-based synthesis for both tabular and time series data. These studies guide us to our evaluation strategy: while assessing the overall fidelity is important, the evaluation should

also reflect our ultimately goal of generating synthetic samples that are useful for downstream classification. This leads to our second research objective: to establish a comprehensive evaluation protocol comprising (i) class-agnostic fidelity, (ii) class-conditional fidelity, and (iii) augmentation utility.

With the research goals clarified, we consider three generative baselines originally developed for networking, tabular, and time series data, respectively, along with three hand-crafted augmentations introduced in [Part I](#). In addition, we propose DDPMS4, a diffusion-based method with an advanced NN architecture for capturing the sequence patterns in packet series. We use XGBoost as the downstream classifier to evaluate the quality of the synthetic data, as it performs comparably to ResNet but trains significantly faster. Based on a multifaceted evaluation, we summarize our main findings as follows:

- Our method, DDPMS4, achieves the best performance among all evaluated generative models in both downstream classification utility and class-agnostic fidelity. This demonstrates that the synthetic data produced by DDPMS4 not only preserves class-relevant semantics but also aligns well with the real training data distribution. Although a performance gap remains compared to training on real data because the generated data does not sufficiently cover rare or low-density patterns present in the original distribution, we believe this gap can be further reduced in future work by modifying the scheduler to capture the characteristics of packet series data.
- Beyond optimizing in-distribution fidelity, we find that naïve sampling from generative models is ineffective for data augmentation in downstream classification, whereas hand-crafted augmentations perform well. This suggests that accurately modeling the training data distribution alone is insufficient, generative models must be redesigned to actively promote diversity useful for downstream classification beyond what is included in the real training data.
- To develop generative models that introduce additional variety while preserving class semantics, it is essential to first understand what constitutes class-relevant semantics. In our short packet series, features are not as visually interpretable as in CV and do not exhibit clear trends or seasonal patterns. Our analysis of hand-crafted augmentations suggests that simple metrics like L2 distance in the input space do not reliably capture semantic similarity. Therefore, more sophisticated approaches are needed to capture class-relevant patterns, which is a necessary foundation for improving downstream utility.

In the remainder, we begin by reviewing relevant literature from both TC and ML communities ([Section 8.2](#)). We then outline our research goals and introduce the baseline methods included in our study ([Section 8.3](#)). Following this, we present our proposed method DDPMS4, a diffusion model based on DDPM scheduler and

a transformer-like architecture where the attention mechanism is replaced by Structured State Space for Sequence Modeling (S4) — a recent alternative to attention that scales linearly and excels at modeling long-range dependencies (Section 8.4). We then describe our experimental setup (Section 8.5) and report the evaluation results. We first discuss the replication of NetDiffusion — a diffusion-based baseline that fine-tunes a pre-trained text-to-image model specifically for synthetic network traffic generation, and analyze potential reasons for its poor generative performance (Section 8.6). Given its specialized design and comparatively lower performance, we analyze it separately from the main benchmark. Subsequently, we evaluate the remaining methods in terms of their utility for downstream classification (Section 8.7) and their class-agnostic fidelity (Section 8.8).

8.2 Related work

Networking traffic synthesis has evolved from traditional statistical models to deep generative approaches. We begin by reviewing works within the networking community, covering both traditional approaches (Section 8.2.1) and more recent generative models (Section 8.2.2), in which the latter facilitates data sharing, support diverse data representations, and bring attention to evaluate the trade-offs among fidelity, privacy, and diversity.

We then turn to relevant ML literature. Among the various branches of generative modeling, our task — modeling 1-D packet sequences of continuous values — is most related to *tabular synthesis* (Section 8.2.3) and *time series synthesis* (Section 8.2.4), so we review these two. Tabular data is 1-D (a single row), where the main challenge lies in jointly handling heterogeneous numerical and categorical features, typically for tasks like generation from scratch and imputation. In contrast, time series data is generally 2-D (features \times timesteps, which reduces to 1-D in the univariate case), and the focus is on modeling temporal dependencies for tasks such as generation, imputation, and forecasting.

8.2.1 Traditional networking traffic synthesis

Traditional methods for generating synthetic network traffic can be grouped into two families: **network simulators**, which create virtual network environment to study synthetic traffic flows within, and **packet generators**, which create/replay real packets and inject them into real networks.

Network simulators are at the core of computer network design, as testified by the many tools developed by industry and academia over the past decade. (Patil et al., 2016; Adeleke et al., 2022). Traditional popular ones are *discrete event* driven simulators (e.g., NS-3 (Riley and Henderson, 2010), OSTINATO (*OSTINATO - Traffic Generator for Network Engineers* n.d.), SEAGULL (*SEAGULL - Multi-protocol traffic generator* n.d.), OMNET++ (Varga, 2010), DONS (Gao et al., 2023)) enabling detailed

packet-level control through component a catalog of network components and APIs, ideal for custom scenarios but requiring complex setup. More recently, *ML-assisted* network simulators (e.g., MimicNet (Zhang et al., 2021), RouteNet-Fermi (Ferriol-Galmés et al., 2023), DeepQueueNet (Yang et al., 2022b)) have been proposed, using *discriminative models* to map raw packets input to aggregate network performance metrics (e.g., latency, flow completion time, throughput). These tools offer more scalable simulation by abstracting the monitoring of (sub)network dynamics without custom event-driven code (Zhang et al., 2021), at the cost of reduced fine-grained control.

Packet generators (e.g., MoonGen (Emmerich et al., 2015), DPDK-pktgen (Turull et al., 2016) or commercial tools from companies like IXIA (*IXIA - High-Volume Traffic Generator Products Catalog* n.d.) and NEOX (*NEOX - Valkyrie Stateless Ethernet Traffic Generation and Analysis up to 800Gbps* n.d.)) offer a practical alternative to simulations by replaying pcap traces or generating synthetic packets on the fly based on some input configurations. These tools complement network simulators and are mainly used to “stress test” real environments and validate their load-handling capabilities.

8.2.2 Generative traffic synthesis

Traditional traffic generation tools usually tailor very specific needs, but they are rigid solutions not generative in nature as they require domain knowledge, and can require replaying previously collected real traces. With the success in computer vision of Generative Adversarial Networks (Goodfellow et al., 2020) first and Diffusion Models (Ho et al., 2020c) more recently, the networking research community started to investigate how to take advantage of generative models for network traffic-related tasks. In Table 8.1 we categorize multiple properties of the most prominent literature.

Easing data sharing. Access to (good) data is paramount in any research field but, differently from the machine learning research, networking research suffers from scarcity and up to date *public* datasets (Claffy et al., 2021). This issue, together with the desire of fine-grained controlling of traffic dynamics, motivates the abundance of the network traffic simulation methodologies mentioned above. At the same time, this data scarcity is linked to costly data gathering — collecting and labeling fine-grained data (e.g., pcap traces) is an engineering challenge — and privacy — network operators are reluctant to collect and share data (even when anonymized) to safeguard both end-users privacy and (in)direct business value related to the data. To mitigate these problems, a wave of recent works (Yin et al., 2022; Lin et al., 2020b; Jiang et al., 2024b; Sivaroopan et al., 2023c; Xu et al., 2021; Sivaroopan et al., 2024) propose the adoption of generative ML arguing that [...] *sharing the model reveals more information than a finite number of synthetic data samples. Thus, we posit that in practice stakeholders will be more likely to share synthetic data rather than the models [...]* (Yin et al., 2022). In other words, the generative model should have “high fidelity”

by creating synthetic datasets (i) encoding the statistical properties of private data, (ii) enabling comparable performance to the private data when used on a downstream task, but (iii) being privacy preserving by avoid synthesis of training sample “replicas”. This line of research includes multiple proposal across different families of generation modeling techniques including GAN (Lin et al., 2020b; Yin et al., 2022; Sivaroopan et al., 2023c), DMs (Jiang et al., 2024b; Zhang et al., 2024b; Sivaroopan et al., 2024), VAE (Aceto et al., 2024) and also traditional statistical approaches likes gaussian mixture of models (Xu et al., 2021).

Data representation. We argue that most of the innovation in this literature come from the *input representation* rather than the mechanics/dynamics/architecture of the training process. This is also due to the varied set of network-data format available. Indeed, despite network protocols and communications are well standardized, the way traffic is “logged” vary and can be broadly split into four major formats: *tabular per-flow records*, *raw pcaps files*, *packet series* and *events sequence*.

Per-flow logs are a very common abstraction level for summarizing network traffic by gathering by aggregating all packets related to a “flow”, e.g., packets sharing the same 5-tuple (ipSrc, srcPort, ipDst, dstPort, l4proto, and computing aggregate stats (number of packets, sum of bytes, timestamps, duration, etc.) resulting then in structured tables. Over the last years, this type of input flourished into “tabular learning” in ML research, focusing on how to handle numerical and categorical features. For instance, NetShare (Yin et al., 2022) models a group of records at the same time with a mechanism similar to PacGAN packing (Lin et al., 2020a); STAN (Xu et al., 2021) models spatial-temporal dependencies via a CNN feature extractor relating a given record with a “context” of previous records; NetDiffus (Zhang et al., 2024b) chains two DMs to first model a sequence of (application, timestamp) pairs which is then used as conditioning for the second DM focusing on generating individual record.

Raw pcaps are at the opposite side of spectrum with respect to tabular data as they gather raw bytes transmitted for each packet. Synthesizing raw pcaps is a significantly more challenging task than modeling tabular per-flow logs, as it requires capturing low-level, packet-wise behavior rather than aggregated flow-level statistics. In fact, beside capturing spatial (packet header fields) and temporal (semantic of fields across packets), attention to syntax is key as a single bit error can break the entire semantic of the packet. As packets are well standardized data structures, the modeling can still relies on tabular-like methods, e.g., both NetShare (Yin et al., 2022) and NetDPSyn (Sun et al., 2024) combines numerical and categorical features encoding similarly to tabular modeling. Alternatively, packets can viewed as images. For instance, NetDiffusion (Jiang et al., 2024b) fine-tunes stable diffusion on images composed by 1-hot encoding the raw packet fields (namely, an nPrint (Holland et al., 2021) input representation), while PacketCGAN (Wang et al., 2020b) “folds” the raw bytes series to form squared black and white image.

Packet series are an intermediate form of representation with respect the previous two as they retain per-flow information, but those are represented as a time series of individual packet properties (e.g., the size, direction, inter arrival time of the first N packets of the flow). Packet series are the most common raw format used in TC, yet they are less explored than the previous two representation. To the best of our knowledge, only CVAE (Aceto et al., 2024) and NetDiffus (Sivaroopan et al., 2024) adopted them combined with two very different postprocessing methods. Specifically, NIDS-CVAE introduces a binning process on the series values to increase privacy guarantees – the packet feature distribution are first segmented it into B uniform bins to obtaining B non-uniform feature value ranges; then real feature values are replaced to the corresponding bin index (1-hot encoded); last, as the trained conditional VAE synthesizes bin index sequences, the authors sample a uniform value from the feature range of values associated to the bin. In contrast, NetDiffus transforms series into GASF (Wang and Oates, 2015) images, i.e., an invertible transformation that encodes space and time relationships between every pair of values in the series which are used to train a different diffusion model for each class.

Last, we define *event sequences* logs as collections or records without necessarily a strict tabular format, nor aggregate statistics. An example of this are control plane logs which CPT-GPT (Jonny Kong et al., 2024) models using a Transformer-based architecture with an ad-hoc feature embedding to capture the specific vocabulary and time dependencies.

Fidelity-vs-Privacy. While high fidelity to real network data is desired in synthetic data generation, the real data often contains sensitive information related to end-users or business operations, raising privacy concerns. To mitigate these risks, methods such as NetDPSyn (Sun et al., 2024) and NetShare (Yin et al., 2022) adopt formal privacy-preserving techniques like Differential Privacy (DP), introducing a trade-off between fidelity and privacy.

Fidelity-vs-Variety. A more subtle tradeoff not well addressed by the literature relates to an alternative goal for the application of generative models, namely *data augmentation*. In this case, the generative models is used to create extra data to complement available training data and improve the performance of a downstream task. The most common scenario is boosting the number of samples for minority class in a classification task.

For instance, ODDS (Jan et al., 2020) addresses a benign-vs-malicious classification task in an intrusion detection system combining an autoencoder, clustering and GANs models. Intuitively, as benign samples are more “stable” compared to malicious ones, an autoencoder is trained with benign data only so to create a latent space where benign data would well cluster while malicious data would result in outliers. To identify these cluster structures, DBSCAN and empirically chosen distance threshold are applied in the latent space to separate high-density clustered

Name	Dataset	†Sample type				Model input	Generate for		Generator		Downstream task		
		public	*samples	bytes	series		tabular	other	fidelity	augment	family	condit.	type
NIDS-CVAE(Aceto et al., 2024)	✓	15k	-	✓	-	-	ad-hoc	✓	-	VAE	-	class	ML
CPT-GPT (Jonny Kong et al., 2024)	-	73M	-	-	-	✓	ad-hoc	✓	-	Transformer	-	n.a.	n.a.
NetDPSyn (Sun et al., 2024)	✓	3M	✓	-	✓	-	ad-hoc	✓	-	iterative algo.	-	class, regr	ML
NetDiff (Zhang et al., 2024b)	✓	5M	-	-	✓	-	series	✓	-	DM	✓	class	ML/DL
NetDiffusion (Jiang et al., 2024b)	-	20k	✓	-	-	-	image/nPrint	✓	✓	DM	✓	class	ML
NetDiffus (Sivaroopan et al., 2024)	-	10k	-	✓	-	-	image/GASF	✓	✓	DM	-	class	ML,DL
SyNIG (Sivaroopan et al., 2023c)	-	10k	-	✓	-	-	image/GASF	✓	✓	GAN	-	class	ML/DL
NetShare (Yin et al., 2022)	✓	n.a.	✓	-	✓	-	series	✓	-	GAN	-	n.a.	n.a.
STAN (Xu et al., 2021)	✓	3M	-	-	✓	-	series	✓	-	auto regressive	✓	n.a.	n.a.
DoppelGANger (Lin et al., 2020b)	✓	100k	-	✓	-	-	series	✓	-	GAN+RNN	✓	n.a.	n.a.
PacketCGAN (Wang et al., 2020b)	✓	10k	✓	-	-	-	raw bytes	-	✓	GAN	✓	class	ML/DL
ODDS (Jan et al., 2020)	-	23M	-	✓	-	-	values frequency	-	✓	GAN	-	class	DL
WGAN-TTUR (Ring et al., 2019)	✓	20M	-	-	✓	-	ip2vec+norm	✓	-	GAN	-	n.a.	n.a.

^{*}approximate values; [†]bytes: a sample corresponds to raw packet bytes; tabular: a sample is a netflow-like record collecting aggregate statistics, timestamps, labels, etc. for a individual flow; series: a sample is a uni/multi-variate series; others: a sample is another form of network event.

TABLE 8.1: State of the art methods for network traffic.

regions from low-density regions. Then, two GANs are trained separately: one on the clustered benign samples and the other on the identified outliers. Once trained, the second GAN generates samples exclusively from these outlier regions to enhance coverage of rare malicious behavior. Other studies (Jiang et al., 2024b; Sivaroopan et al., 2024; Sivaroopan et al., 2023c; Wang et al., 2020b) investigated the performance when boosting minority classes as well but we argue that this happen in a more “opportunistic” fashion rather than being cored in the method design as for ODDS.

8.2.3 Generative tabular data synthesis

As previously mentioned, network data is often structured in tabular format and present traits observed in generic tabular modeling such as the presence of *heterogeneous columns* – numerical counters (e.g., number of bytes and packets in a flow) and categorical fields with predefined alphabet (e.g., the protocol type of a flow is usually only TCP or UDP) – *multi-modality* – column values distributions are usually not Gaussian but rather present multiple modes and long tails (e.g., TCP packet sizes are at least bimodal as ACK packets are mixed with data packets) – small/mid-sized cardinality – tabular dataset are usually significantly more contained compared image and text-related generative models (e.g., public dataset hardly go beyond 1M samples compared to the billion of samples of image datasets like LAION¹). To address the above challenges, over the last couple of years we witnessed an increasing interest in the application of DM on tabular data coupled with advancement in multinomial synthesis for effective modeling of categorical columns. This rapid evolution is well portrayed in recent surveys (Li et al., 2025). In the following we focus only on the prominent works summarized in Table 8.2 to offer a broad view on the design options.

Input representation. Preprocessing plays a crucial role in tabular data synthesis due to the coexistence of categorical and numerical features. While standard techniques such as one-hot encoding for categorical variables and Min/Max or quantile scaling for numerical features are widely used (Kotelnikov et al., 2023), alternatives

¹<https://laion.ai/blog/laion-5b/>

Name	Latent	Diffusion Model (DM)			Neural Network model	cond.	Generation		#D
		DM	cat noise	NFE			trick4scratch	imput.	
TabDDPM (Kotelnikov et al., 2023)	-	DDPM	\mathcal{U}	Search	MLP	✓	-	n.a.	16
CoDi (Lee et al., 2023a)	-	DDPM	\mathcal{U}	50	2 UNet	-	-	n.a.	17
ForestDiffusion (Jolicoeur-Martineau et al., 2024)	-	VPSDE/CFM	As ctn	50	#C \times NFE Xgb	-	-	RePaint	27
TabSyn (Zhang et al., 2024a)	✓	EDM	n.a.	20	MLP	-	-	RePaint	6
TabDiff (Shi et al., 2025)	-	VESDE (LN)	[MASK]	50	Transformer+MLP (Shi et al., 2025)	-	Restart	RePaint+CFG	7

NFE: number of function evaluations
LN: feature-wise learnable noise schedule $\sigma(t)$

TABLE 8.2: Diffusion models for tabular data: from TabDDPM to recent advances

have been proposed. For example, CTGAN (Xu et al., 2019) applies a Gaussian Mixture Model to decompose numerical columns into two components: a standardized continuous value and a one-hot encoded mode indicator. A similar decomposition strategy appears in networking literature DoppelGANger (Lin et al., 2020b) which introduces an per-sample scaling method, yet this was used for modeling time series rather than tabular records. Going beyond input encoding, TabSyn (Zhang et al., 2024a) proposes projecting the transformed features into a continuous latent space using β -VAE, which helps unify heterogeneous feature types into a shared representation. In this setting, the coefficient β controls the weight of the KL divergence term during training. By gradually increasing β , TabSyn relaxes the Gaussian prior constraint in early stages, improving the expressiveness of the latent space for downstream diffusion modeling.

Scheduler and score modeling. With the exception of early methods like CTGAN and VAE, which face challenges such as training instability, mode collapse, or restrictive latent assumptions, recent tabular synthesis approaches increasingly adopt diffusion models thanks to their stability and flexibility in handling mixed-type data distributions. TabDDPM (Kotelnikov et al., 2023) is (one of) the first method introducing a diffusion-based approach where a single network handles both numerical and categorical features — numerical features are corrupted with Gaussian noise and trained using MSE loss as in standard discrete-time DDPM, while categorical features are corrupted by mixing the one-hot vector with a uniform distribution and trained using cross-entropy loss. Notably, TabDDPM is the only method in this line of work that conditions the neural network on an external class label, while the subsequent work considers class label as an additional categorical column and generate unconditionally. By introducing diffusion modeling tailored to mixed-type tabular data, TabDDPM paved the way for later methods that have extended in various angles. For instance, CoDi (Lee et al., 2023a) extends this by using separate networks for numerical and categorical features, conditioning them on each other at every timestep and aligning their representations via an additional contrastive loss. ForestDiffusion (Jolicoeur-Martineau et al., 2024) replaces NN with XGBoost regressors as a lightweight, interpretable, and CPU-efficient alternative for score approximation, using a continuous-time scheduler (VPSDE or CFM) but with predefined noise levels

t , enabling the training of one model per noise level. TabDiff (Shi et al., 2025) retains a unified network but applies a continuous-time Variance Exploding (VE) SDE with feature-wise learnable noise schedules to numerical features, allowing each numerical feature to adapt its own noise dynamics for more precise modeling. Additionally, categorical noise is implemented by extending the one-hot vector with an extra [MASK] dimension representing the noised state, inspired by recent advances in discrete diffusion for language modeling. TabSyn also uses a continuous-time EDM where the noise level is linear w.r.t. time.

Generation from scratch. For generation from scratch, most methods follow the standard denoising process, except TabDiff, which adopts a stochastic restart sampling strategy (Xu et al., 2023b), where each sampling step consists of a stochastic forward transition that injects noise — Gaussian noise for continuous columns and masked values for categorical ones — followed by a deterministic reverse denoising step. This design aims to combine the exploratory benefits of stochastic forward transitions with the low discretization error of deterministic reverse steps.

Imputation. Imputation aims to predict the unknown part of a sample based on the known observable part. While this task is meaningful for tabular data, it was not considered by early methods like TabDDPM and CoDi. In 2024, ForestDiffusion and TabSyn began to address imputation by employing a pretrained unconditional model with the RePaint approach — originally developed for image inpainting — during sampling, where the reverse step is a mixture of the known part’s forwarding and the unknown part’s denoising. TabDiff later integrates classifier-free guidance (CFG) to the RePaint process, but this requires training an extra model to estimate the unconditional distribution of the unknown part, and in their experiments the unknown part was limited to a single column — the class label.

Evaluation. Synthetic data is typically evaluated along three dimensions, each motivated by a distinct goal. i) Fidelity reflects how well the synthetic data captures the real data distribution. This includes low-order metrics, such as marginal distributions of individual columns, and high-order metrics that assess global sample-level structure — e.g., fidelity and coverage based on distances, and discriminative scores that measures how easily a classifier can distinguish real from synthetic samples. ii) Privacy measures how close synthetic samples are to real training data, with Distance to Closest Records (DCR) used to assess potential information leakage. iii) Downstream utility evaluates whether synthetic data is useful for tasks like classification/regression (utility test) and missing value imputation (if supported). Despite its importance, data augmentation has not been considered in existing work.

8.2.4 Generative time series synthesis

For time series data, a key application of generative models is *imputation*, where missing values are predicted based on observed parts. Forecasting can be seen as a

Name	Neural Network			Diffusion Model (DM)		Cond. Guidance
	Backbone	Cond.	Noisy Inp	DM	Prior	
CSDI (Tashiro et al., 2021)	Transformer	(x_{obs}, M)	$(x_{\text{tar}}^t, 0\text{-pad})$	DDPM	$N(0, 1)$	-
SSSD (Alcaraz and Strodthoff, 2022)	S4	(x_{obs}, M)	$(x_{\text{tar}}^t, x_{\text{obs}})$	DDPM	$N(0, 1)$	-
ImagenTime (Naiman et al., 2024)	UNet (TS→Img)	-	$(x_{\text{tar}}^t, x_{\text{obs}})$	EDM	$N(0, 1)$	-
TSDiff (Kollovieh et al., 2023)	S4	- or (x_{obs}, M)	x^t	DDPM	$N(0, 1)$	Asym. Laplace $p_\theta(x_{\text{obs}} x^t)$
TSFlow (Kollovieh et al., 2025)	S4	- or (x_{obs}, M)	x^t	CFM	$\mathcal{GP}(0, K) +$ cond. guided prior sampling	Asym. Laplace $p_\theta(x_{\text{obs}} x^t)$
TMDM (Li et al., 2024b)	MLP	$\hat{x}_{\text{tar}} = f(x_{\text{obs}})$	x_{tar}^t	DDPM	$N(\hat{x}_{\text{tar}}, 1)$	-
NSDiff (Ye et al., 2025)	MLP	$(\hat{x}_{\text{tar}}, g(x_{\text{obs}}))$	x_{tar}^t	DDPM	$N(\hat{x}_{\text{tar}}, g(x_{\text{obs}}))$	-

x_{obs} : observed part
 x_{tar} : missing target part to be imputed
 $x = (x_{\text{obs}}, x_{\text{tar}})$: concatenation of observed part and missing part
 M : binary mask indicating which parts are observed and which are missing
 $\mathcal{GP}(0, K)$: Gaussian process (GP) with a kernel function $K(\tau, \tau')$
 f : pre-trained Transformer for forecasting x_{tar}
 g : pre-trained MLP for predicting variance Var (SlidingWindow (Y_0))

TABLE 8.3: Diffusion models for time series data.

special case of imputation, where the missing portion is not randomly distributed but instead corresponds to the continuous segment at the end of the sequence, representing “future” values to be forecasted.

Given an observed part x_{obs} , the objective is to infer the missing part x_{tar} , i.e., to model the conditional distribution $p(x_{\text{tar}} | x_{\text{obs}})$.

While some early methods use VAEs and GANs, and recent ones explore autoregressive models including pretrained LLMs, this chapter focuses more on diffusion models for time series, with diffusion-based time series imputation work summarized in Table 8.3.

Unlike in tabular data, directly applying RePaint at inference to an unconditional model — where the predicted noisy observed part is replaced by the forward-diffused version of x_{obs}^t at each reverse step — leads to unsatisfactory results for time series according to the experiments in Alcaraz and Strodthoff, 2022, as it disrupts temporal dependencies and introduces inconsistencies in the evolving sequence. To better incorporate conditional information from x_{obs} , the literature proposes following strategies:

1. **Conditional Architecture:** The denoising network is explicitly conditioned on the information given by x_{obs} . In early work (Tashiro et al., 2021; Alcaraz and Strodthoff, 2022; Kollovieh et al., 2023; Kollovieh et al., 2025), this conditioning input is simply the concatenation of the clean observed part x_{obs} and a binary mask M indicating which parts are observed and which are missing. In recent work (Li et al., 2024b; Ye et al., 2025), instead of directly using x_{obs} , the condition signal is a learned embedding from a pretrained network that encodes x_{obs} , such as a Transformer that predicts \hat{x}_{tar} from x_{obs} .
2. **Condition by Classifier Guidance:** In most of the literature, diffusion is applied exclusively to the missing part of the data during both training and inference. That is, the neural network receives inputs in which only the missing portion x_{tar}^t is noised, and the training loss is computed only over this part so

that the model learns to denoise x_{tar} specifically. In practice, to ensure consistent input dimensions, some methods concatenate x_{tar}^t with either the clean x_{obs} or zero-padding.

In contrast, to support multiple downstream tasks with a single model, some recent works model the entire time series x using a task-agnostic unconditional model. Conditional generation is then enabled through classifier guidance. Specifically, they assume $p_{\theta}(x_{\text{obs}} \mid x^t)$ as an asymmetric Laplace distribution, whose score function can be analytically computed using the denoising output $\epsilon_{\theta}(x^t, t)$ and the observed part x_{obs} .

3. **Conditional Prior:** Recent works replace the standard isotropic Gaussian prior with priors informed by x_{obs} , allowing generation to start from a more informative and easier endpoint. TSFlow (Kollovich et al., 2025) employs Gaussian Process (GP) priors $\mathcal{GP}(0, K)$, where the kernel $K(\tau, \tau')$ is designed to capture temporal patterns (e.g., periodicity), and prior sampling is further conditioned on x_{obs} using classifier guidance. TMDM (Li et al., 2024b) introduces a Gaussian prior whose mean is set to the predicted \hat{x}_{tar} generated by a pretrained Transformer that takes x_{obs} as input. NSDiff extends this idea by replacing the unit variance of the prior Gaussian with a learned one predicted by another pretrained network that takes x_{obs} as input.

Besides how the conditional information from the observed part x_{obs} is integrated, there are other variations across related work. For example, regarding the architectural backbone, SSSD finds that S4 outperforms Transformer previously used in CSDI, and the S4-based architecture is adopted in subsequent works such as TSDiff and TSFlow. ImagenTime, on the other hand, employs a U-Net backbone to process an image representation of the time series, arguing that image format handles both short- and long-term time series more efficiently. As for the noise scheduler, most works adopt DDPM, while ImagenTime uses EDM and TSFlow employs CFM.

8.3 Research goals

By contrasting the networking Table 8.1 with the ML about tabular data Table 8.2 and time series data Table 8.3, we gather the following observations which lead to our research goals:

Observation 1: Sparsity. Both ML and networking literature offer a variety of options, but we argue that networking literature is *sparse* with respect to multiple dimensions. First of all, while the networking datasets used are larger than the common datasets used in ML, the data scarcity is evident with studies having only up to 3 datasets in networking, which restricts the ability to assess model generalization across diverse traffic types and deployment scenarios. Moreover, networking

literature tends to do not reuse datasets from previous publications preferring investigation of complementary scenarios often resulting in apple-vs-oranges comparisons between different works. In contrast, the ML community benefits from a well-established set of benchmark datasets that are consistently reused across studies. New proposals are evaluated on the consistent datasets, enabling rigorous and fair comparisons. This culture of standardized evaluation — both in terms of data and accompanying code — remains largely absent in networking research.

We also observe a preference to focus on input representation in networking rather than tapping into the rich design options for the modeling itself. The availability of multiple input formats for network data is likely contributing to this separation, but at the same time we argue that little attention has been placed on ablating the importance of each representation. For instance, considering works based on input series, NIDS-CVAE (Aceto et al., 2024) does not ablate the importance of the binning strategy presented, while NetDiffus (Sivaroopan et al., 2024) does not consider training directly on input series rather than GASF. Yet, NetShare was evaluated on both tabular and raw packet bytes – the evaluation can result opinionated rather than based on a common “protocol”.

Overall, this fragmented landscape makes it difficult to compare different methods. At the core of the problem there is a lack of a reference benchmark, covering both data and code, for evaluating new proposals. ML literature is significantly more mature in this regard as, beside usually evaluating methods across a variety of datasets, often the repositories associated to literature publications are benchmark on their own as they incorporate the code of the alternative methods used for comparison.

Research goal 1: Benchmark. To address this fragmented landscape, our first goal is to build a unified benchmark for network traffic modeling. This benchmark will include a curated suite of datasets, standardized pre-processing and post-evaluation pipelines, and a set of baseline methods. By providing both data and code in a reproducible framework, we aim to enable fair comparisons, support rigorous empirical evaluations, and promote methodological clarity in this emerging intersection between machine learning and networking.

Observation 2: Going beyond fidelity. While any generative method needs to be assessed for fidelity, we argue that there is the need for going beyond the mere generation. Indeed, many networking proposal do not consider an actual downstream task (except for small ablation or toycases scenarios) and only episodically investigate generation for augmentation (Jiang et al., 2024b; Zhang et al., 2024b; Xu et al., 2021; Wang et al., 2020b). Conversely, ML literature always includes at least an utility test that assesses the downstream classification effectiveness of generated data, yet still augmentation test — where synthetic data is used to enhance real data for classifier’s training — has almost never been considered in diffusion-based tabular/time-series generation. In other words, as many of the proposal only tackle generation

for “ease of sharing”, we argue that the integration of generative methods to solve a specific downstream task is still lacking in the literature.

This is partially justified considering that the fidelity-vs-variety is a design problem cored into contradicting requirements. In fact, training a generative model entails learning the (unknown) distribution of a training set and the model aims at being as close as possible to the training set without memorizing it – maximum fidelity – but using it for augmentation requires complementing the training data with out of distribution data useful for a downstream task – additional task-specific variety. That said, we believe that both aspects should be integral part of the evaluation of generative models which, beside task-agnostic fidelity should also integrate a concrete downstream task. Where possible, the evaluation should also include *hand-crafted augmentations* to verify if the generative models offer superior performance of “cheaper” ways of integrating additional variety in the original data.

Research goal 2: Evaluation metrics. We aim to establish a comprehensive evaluation protocol for generative models in networking, moving beyond fidelity alone. Specifically, given that our target downstream task is TC where x being the input packet series and y being the target label, we define three key evaluation axes: (i) *class-agnostic fidelity*, which assesses how well the model captures the overall data distribution $p(x)$; (ii) *class-conditional fidelity*, or utility test, which evaluates how effectively the model captures the conditional distribution $p(x | y)$ and supports downstream classification; and (iii) *augmentation utility*, which measures whether generated data can provide additional useful variety beyond that embedded in real training data and enhance classification performance. This multi-part evaluation helps fairly assess the usefulness of generative methods in network traffic classification.

8.3.1 Baselines

Based on the different branches of related work and our research goal of establishing a comprehensive evaluation protocol, we select a set of baseline methods covering both generative models and hand-crafted augmentations. The three generative approaches we consider are NetDiffusion (Jiang et al., 2024b), which is designed for packet-level traffic synthesis using pre-trained text-to-image diffusion; TabD-DPM (Kotelnikov et al., 2023), a tabular data diffusion model; and ImagenTime, a time series generative model operates image-transformed inputs. Notice that ImagenTime is conceptually similar to NetDiffus (Sivaroopan et al., 2023b) which uses GASF rather than delay embedding; yet we did not consider NetDiffus in our benchmark because its design is unconditional (although we think it could be adapted to our usecase). Similarly, we exclude also NetDiff (Zhang et al., 2024b) because it relies on flow summary statistics and, while an adaptation is possible, this would require significant redesign. To support the evaluation in terms of utility and augmentation effectiveness,, we also include three hand-crafted augmentations — Translate,

Wrap, and Gaussian that performed well in [Part I](#). In this section, we introduce and compare their key properties.

The 3 generative model baselines are originally designed for tabular data, time series, and network pcap data, respectively:

1. **TabDDPM** applies quantile transformation for input scaling. It adopts a DDPM scheduler, with the neural network implemented as a conditional MLP in which the class label and timesteps are added only to the noisy input before the first MLP layer. We follow the processes used by the original authors to search the hyperparameters separately for each dataset using Optuna.
2. **ImagenTime** converts a univariate 1-D packet time series of length 30 into a squared 2-D matrix using delay embedding, with an embedding size of 16 and a delay step of 1:

$$X_{\text{Img}} = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_{15} & x_{16} \\ x_2 & x_3 & x_4 & \dots & x_{16} & x_{17} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{15} & x_{16} & x_{17} & \dots & x_{29} & x_{30} \\ x_{16} & x_{17} & x_{18} & \dots & x_{30} & 0 \end{bmatrix} \in \mathbb{R}^{16 \times 16}. \quad (8.1)$$

An example of the converted image is shown in [Figure 8.1](#). It is then unsqueezed along the last dimension, resulting in a 3-D image $X_{\text{Img}} \in \mathbb{R}^{16 \times 16 \times 1}$. The input is first scaled using min-max normalization from the range $[-1500, 1500]$ to $[0, 1]$, and then standardized to have a mean of 0.5 and a standard deviation of 0.5, following common practice for preparing image inputs for neural networks. The model uses a continuous-time EDM scheduler and a U-Net architecture with Conv2D layers, where the class label and timestep are injected into each layer as conditioning signals. As illustrated in [Figure 8.2](#), the model is trained to reconstruct the clean image from a noisy input image. At inference, a clean image is generated by iteratively denoising through the reverse process. Finally, the generated image is transformed back into the time series domain since the delay embedding mapping is invertible. Sampling is performed using only 20 denoising steps (consistent with the original paper’s setting), which is 5 times fewer than TabDDPM and DDPMs4 (both use 100). To compensate, we scale up ImagenTime’s model size to be 6 times larger, ensuring comparable representational capacity.

3. **NetDiffusion** (Jiang et al., [2024b](#)) targets the generation of raw pcap traces representing each packet with an nPrint (Holland et al., [2021](#)) encoding, i.e., a 3-level bit sequences based on a predefined superset of packet header fields. Given a raw sequence of bytes, if a header fields is found in the sequence, its value is one-hot encoded, otherwise the related bits are set to -1. NetDiffusion is vertically stacking 1,024 nPrint encoding into a 2D matrix mapping

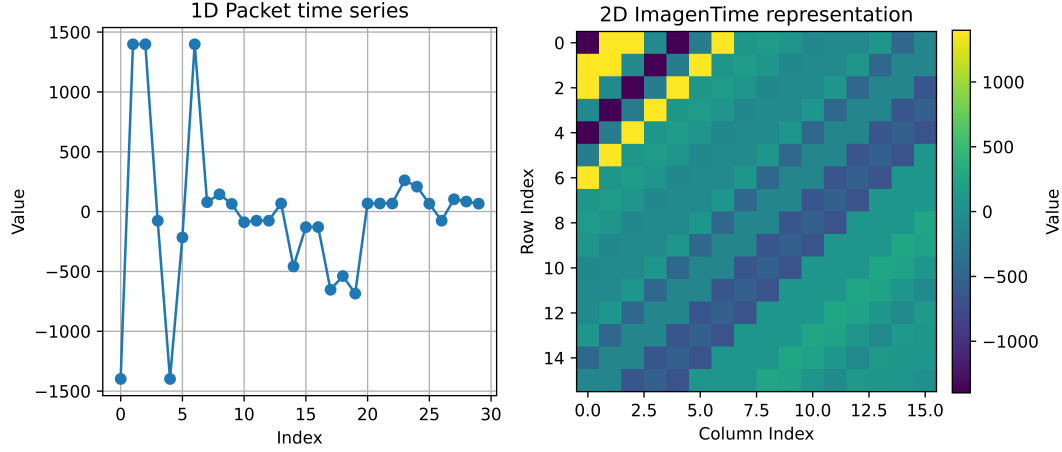


FIGURE 8.1: ImagenTime delay embedding.

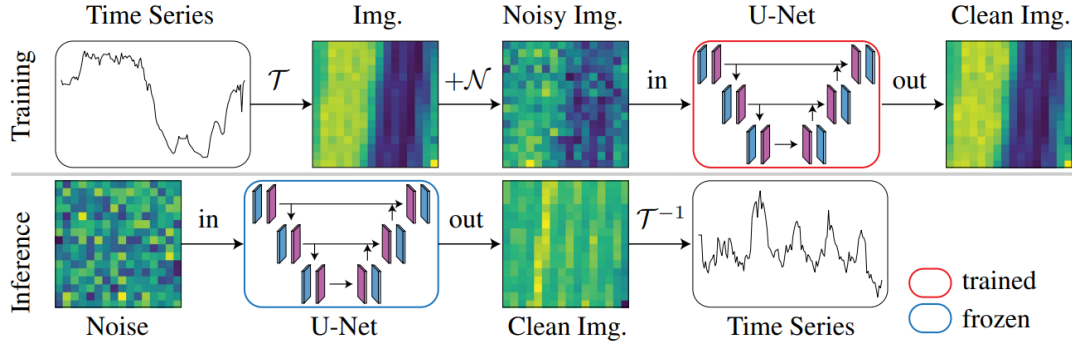


FIGURE 8.2: ImagenTime pipeline.

the 3-level bits to the red, green and blue channels of a picture, as shown in the leftmost panel of Figure 8.3. The resulting images are paired with text prompts (e.g., “pixelated network traffic type-0”) and used to fine-tune the LoRA weights of Stable Diffusion 1.5. Sampling involves two steps: ControlNet (Zhang et al., 2023) is applied using a Canny edge mask derived from a training nPrint image to provide structural guidance during generation, followed by ad-hoc correction rules to ensure syntactic validity at both packet and flow levels. NetDiffusion was originally evaluated on a private dataset of 4 video streaming services, thus we consider a public dataset providing similar services, namely UNSW24, which is described in Section 8.5.

The 3 hand-crafted augmentations selected from Part I, among which 2 are top-performing time sequence transformations (Translate and Wrap), and 1 does amplitude change (Gaussian). Here, we briefly recap their definition:

1. **Translation** shifts a segment of the time series left (simulating packet drops) or right (simulating duplication or retransmission). The shift length n is sampled as $n \sim \mathcal{U}[1, N]$, where $N = 1 + \arg \max_i \{a_i \leq \alpha\}$ and $a_i \in \{0.15, 0.3, 0.5, 0.8\}$. A direction $b \in \{\text{left}, \text{right}\}$ and a start index $t \sim \mathcal{U}[0, T]$ are also sampled. If left, the feature values are shifted left by n steps from t , and zeros are padded

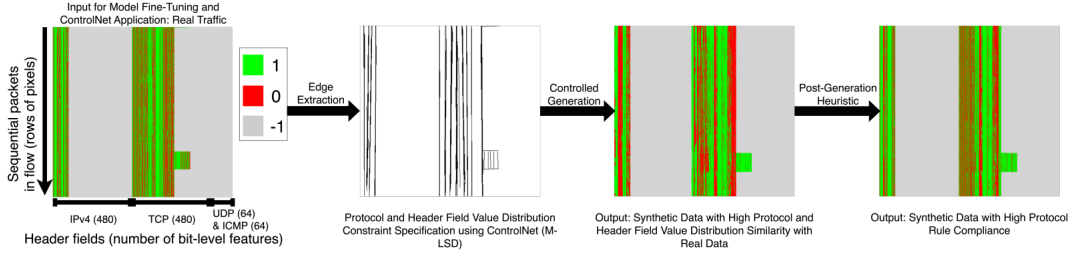


FIGURE 8.3: Example of NetDiffusion: canny edges are extracted from the original Nprint image, serving as additional control signal when using ControlNet. Post-generation heuristics are applied to refine field details for protocol conformance.

in the tail. If right, values are shifted right and padded with the original value at $x_{(d,t)}$.

2. **Wrap** augments a time series by randomly applying one of three operations — interpolation, drop, or no change — to each time step $x_{(:,t)}$. For each time step, one of the three actions is selected with probabilities $P_{\text{interpolate}} = P_{\text{drop}} = 0.5\alpha$ and $P_{\text{no change}} = 1 - \alpha$. Interpolation replaces the value with the average of itself and the next step; drop removes the step. The process continues until the output reaches length T , padding if needed.
3. **Gaussian** adds independent Gaussian noise to each time step, where the standard deviation of the noise is proportional to the standard deviation of all sample values at that time step.

8.4 Our method

8.4.1 Diffusion scheduler

As explained in [Chapter 5](#), DDPM models continuous data ($x_t \in \mathbb{R}^n$) as a Markov chain with Gaussian forward and reverse transition kernels:

$$\begin{cases} q(x_t | x_{t-1}) := \mathcal{N}(x_t; \sqrt{1 - \beta_t}x_{t-1}, \beta_t I) \\ p_\theta(x_{t-1} | x_t) := \mathcal{N}\left(x_{t-1}; \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right), \sigma_t\right) \end{cases}$$

where $\alpha_t := 1 - \beta_t$, $\bar{\alpha}_t := \prod_{i \leq t} \alpha_i$ and the model $\epsilon_\theta(x_t, t)$ is trained to predict the actual noise component ϵ that was used to generate the noisy sample x_t . Under the variational lower bound (ELBO) framework, the training objective can be simplified to a weighted sum of MSE between the predicted and actual noise across all timesteps t , effectively turning the optimization into a denoising score matching problem:

$$L_t^{\text{simple}} = \mathbb{E}_{x_0, \epsilon, t} \|\epsilon - \epsilon_\theta(x_t, t)\|_2^2 \quad (8.2)$$

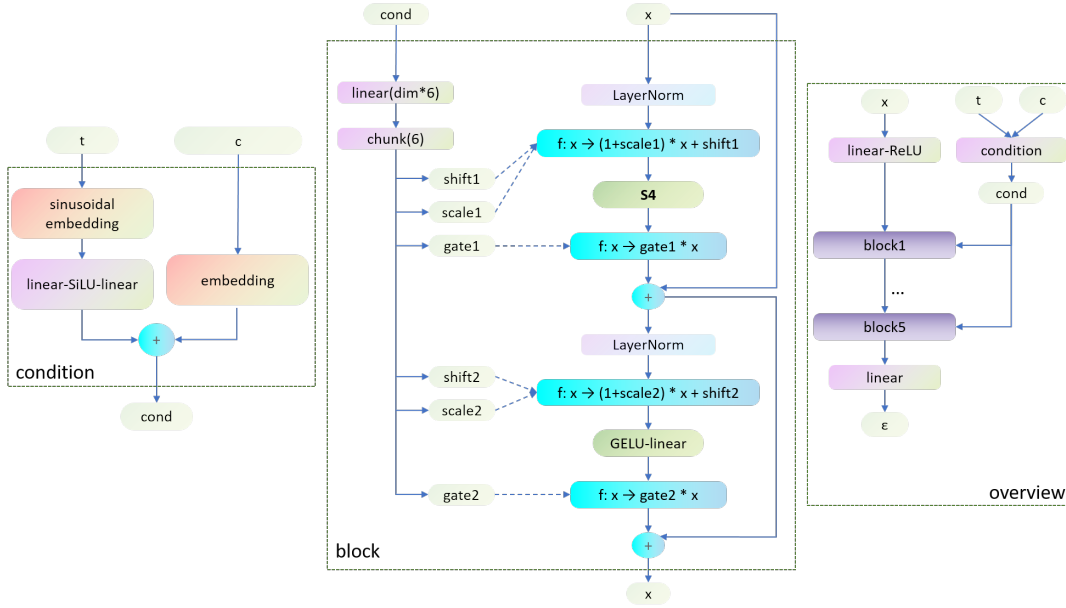


FIGURE 8.4: DDPMS4 neural network architecture

8.4.2 Neural network architecture

In [Figure 8.4](#), we present the architecture of the denoising neural network, which is a DiT (Peebles and Xie, 2023) variant where attention is replaced by S4 (Gu et al., 2022) to better capture temporal dependencies in the input sequence. Overall, the backbone consists of five DiT layers, and the estimated noise $\epsilon_\theta(x_t, t)$ is produced by the final linear layer and used in the training objective [Equation \(8.2\)](#).

Condition block.

1. Class label c : An embedding table of shape (N_{classes}, d) is learned from scratch, where d is the embedding dimension. This is practical since our conditioning is based on a small number of discrete class labels, rather than free-form text prompts with much higher variability.
2. Time t : to represent a scalar timestep t as a 1D vector $\gamma(t) \in \mathbb{R}^d$, we use sinusoidal position encoding: $\gamma(t) = [\cos(t \cdot \omega_i), \sin(t \cdot \omega_i)]_{i=1}^{d/2}$, where $\omega_i = 10000^{-\frac{i}{d}}$.

This encoding has two advantages: i) in diffusion models, low-frequency components capture the overall denoising stage (e.g., early vs. late), while high-frequency components distinguish fine-grained timestep differences, enabling precise behavior across the trajectory. ii) It allows the model to reason about the relative difference between two timesteps t_1 and t_2 , since the inner product of their embeddings depends only on $t_1 - t_2$: $\langle \gamma(t_1), \gamma(t_2) \rangle = \sum_{i=1}^n (\sin(\omega_i t_1) \sin(\omega_i t_2) + \cos(\omega_i t_1) \cos(\omega_i t_2)) = \sum_{i=1}^n \cos(\omega_i (t_1 - t_2))$.

The sinusoidal timestep encoding is passed through two linear layers with SiLU activation in between, and the resulting timestep embedding is added

to the class embedding to condition the network blocks.

Main block.

1. Conditioning via Adaptive LayerNorm-Zero (adaLN-Zero): In standard LayerNorm $\text{LN}(x) = \gamma \cdot \frac{x - \mu}{\sigma} + \beta$, the scale γ and shift β are learnable static parameters. In adaLN, γ and β are instead dynamically regressed from the conditioning inputs: $\text{AdaLN}(x \mid c) = \gamma(c) \cdot \frac{x - \mu}{\sigma} + \beta(c)$. These conditioning scale and shift parameters are the same for all positions (packets) in the input sequence (packet series), enabling efficient conditioning without introducing position-specific complexity.

Additionally, a dimension-wise gate factor α is also regressed from the conditioning inputs, to scale the output of a residual block before it's added to the skip connection: $\text{output} = x + \alpha \cdot \mathcal{F}(x)$. Initialized to zero, this gate makes all blocks to initially behave like an identity function, which stabilizes early training and allows the model to gradually learn meaningful residual contributions as needed.

Overall, adaLN-Zero provides fine-grained control over how conditioning influences each block, both through adaptive pre-normalization and by regulating how much residual output is added to the main signal path.

2. S4: In continuous time, a state space model (SSM) maps an input signal $u(t)$ to a latent state $x(t)$, which is then projected to an output signal $y(t)$:

$$\begin{cases} x'(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases}$$

To apply this model to a discrete input sequence $u = (u_k = u(k\Delta))_{k=1,2,\dots}$ sampled from the underlying continuous signal $u(t)$, by using the Euler approximation and assuming the input remains constant over a small step size Δ , we have the discretized SSM:

$$\begin{cases} x_k = \bar{A}x_{k-1} + \bar{B}u_k \\ y_k = \bar{C}x_k + \bar{D}u_k \end{cases} \quad (8.3)$$

where the discrete system matrices are defined as:

$$\begin{cases} \bar{A} = (I - \frac{\Delta}{2}A)^{-1} (I + \frac{\Delta}{2}A) \\ \bar{B} = (I - \frac{\Delta}{2}A)^{-1} \Delta B \\ \bar{C} = C \\ \bar{D} = D \end{cases} \quad (8.4)$$

Furthermore, although Equation (8.3) defines a recurrent formulation which is fast at inference, it is slow to train because the hidden state at each time step depends on the previous one, introducing a sequential dependency. To accelerate

training, assuming the initial state $x_0 = \bar{B}u_0$, Equation (8.3) can be computed in closed-form as a convolution, where the output y_k is a convolution of the input sequence u with a SSM kernel \bar{K} :

$$y_k = (\bar{K} * u)_k + \bar{D}u_k, \text{ where } \bar{K} = (\bar{C}\bar{B}, \bar{C}\bar{A}\bar{B}, \dots, \bar{C}\bar{A}^{k-1}\bar{B}) \in \mathbb{R}^L \quad (8.5)$$

This convolution form of discrete SSMs enables parallel computation across time steps, allowing efficient GPU acceleration and faster training compared to sequential recurrence.

In practice, the state matrix A is fixed using a HiPPO formulation, rather than trained from a random initialization. HiPPO is designed to track the input history by projecting it onto Legendre polynomials, enabling the hidden state to store a compressed summary of the past. This enables the model to remember long sequences. It also helps prevent the vanishing and exploding gradient problems, which can happen in linear systems due to the exponential form of their solutions. Finally, the Diagonal Plus Low-Rank (DPLR) reparameterization of the HiPPO matrix A allows efficient computation of its discretized form \bar{A} without requiring matrix inversion, enabling efficient kernel computation.

8.5 Experiments

8.5.1 Datasets and curation

Dataset			Raw samples		Curated samples					
name	environment	traffic	classes	samples	classes	all	largest	(%) smallest	(%) *imbalance	
UNSW24	testbed	manual	4	10.6k	4	6.6k	3k (45.14)	443 (6.67)	6.8	
Mirage22	testbed	manual	10	59.1k	9	11.5k	2.9k (25.66)	655 (5.71)	4.5	
UTMOBILENET21	testbed	synthetic	17	34.4k	9	5.6k	1.5k (27.26)	223 (3.96)	6.9	
CESNET-TLS22	real network	real	198	141.7M	50	2.1M	164.6k (7.66)	9.1k (0.42)	18.1	
					100	2.4M	164.6k (6.84)	2.6k (0.11)	63.4	
					135	2.5M	164.6k (6.70)	505 (0.02)	325.9	
CESNET-QUIC22	real network	real	105	153.2M	50	1.5M	190.1k (12.90)	3.4k (0.23)	55.9	
					81	1.5M	190.1k (12.48)	548 (0.04)	347.0	
Enterprise	real network	real	3,126	272M	50	9.3M	1.7M (18.78)	45.7k (0.49)	38.1	
					100	10.8M	1.7M (16.12)	21.3k (0.20)	81.6	
					300	12.2M	1.7M (14.32)	2.1k (0.02)	839.8	
					500	12.4M	1.7M (14.06)	649 (0.01)	2,682.5	

*ratio between largest and smallest class.

TABLE 8.4: Dataset properties.

Table 3.4 summarizes the properties of the datasets we consider in our experiments.

UNSW24 (Wang et al., 2024) gathers traffic (available in raw pcap format) of 4 video streaming services (namely YouTube, Amazon Prime, Netflix and Disney+) when

using native apps on various devices (laptops, mobile devices, PlayStation consoles) in a controlled environment.

Mirage22 (Guarino et al., 2022) gathers traffic of 9 video meeting Android apps using 3 instrumented devices in a test-bed setup, covering scenarios like webinars, audio/video calls, and video conferences. Each session was post-processed into a bidirectional flow log containing both aggregate metrics and per-packet time series, and label was obtained from strace logs that map sockets to app names.

UTMOBILENET21 (Heng et al., 2021b) gathers traffic of 9 Android apps obtained by synthetically controlling 3 devices via Android Debug Bridge (ADB) interface from a laptop. Unlike Mirage22, the dataset lacks strace logs and is released as per-packet csv logs obtained running tshark on the collected pcap.

CESNET-TLS22 (Luxemburk and Čejka, 2023b) and CESNET-QUIC22 (Luxemburk et al., 2023b) are two large scale datasets collected from CESNET2, the Czech Republic’s national research and education network. CESNET-TLS22 includes 2 weeks of traffic, while CESNET-QUIC22 spans 4 weeks and captures a major data shift, with hundreds of apps using the two different transport protocols. Each flow (offered as CSV) includes aggregate stats, packet time series for the first 30 packets, metadata from TLS/QUIC handshakes (e.g., SNI, JA3), and a service label derived via ad-rule-based processing.

Enterprise is a private dataset collected from residential and enterprise networks, capturing flow-level logs with aggregate metrics (e.g., bytes, packets, TCP flags, RTT) and packet time series (size, direction, IAT). Each flow is enriched with an app label using commercial DPI software covering hundreds of apps.

The datasets were preprocessed before training, involving 3 key aspects:

Input extraction and data filtering. We process all datasets to extract per-flow packet series and represent each flow as a 1D univariate array by multiplying the size and direction of its first 30 packets. TCP flows without a valid 3-way handshake are discarded, while we keep UDP flows expect DNS traffic identified based on standard port numbers. For the three small datasets (UNSW24, Mirage22, UTMOBILENET21), we remove classes with fewer than 200 flows, and for larger datasets, the threshold is raised to 500. In the case of CESNET-QUIC22 and CESNET-TLS22, which span multiple weeks of traffic, we use only the first day to avoid time-related variability — this day is sufficiently large and representative. As shown in Table 8.4, this curation is significantly reducing the number of flows in each dataset, but those are “mice” flows as the filtering retain 98% of the bytes in each dataset.

Datasets ground-truth. To train models we need to define a label for each flow. At the same time, some of the datasets considered are generated from mobile devices

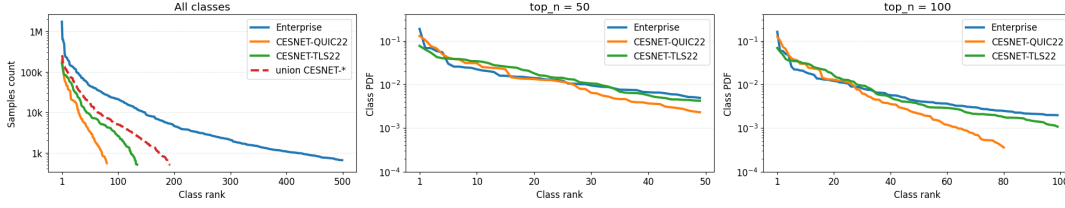


FIGURE 8.5: Class imbalance comparison among Enterprise, CESNET-QUIC22 and CESNET-TLS22.

which notoriously create “background” traffic which, if not removed, can create label conflicts (e.g., traffic for the OS or generic services appear across different experiments of different apps introducing confusion). The three large datasets raw data include reliable labels from real-world environments so there is extra curation to be operated. Conversely, the small datasets require ad-hoc processing. For UNSW24, we rely on the TLS SNI fields (extracted when processing the raw pcap) verifying that it was related to the name of the expected name of the app encoded in the pcap file names. For Mirage22, raw logs have a column reporting the application associated to each flow that was retrieved using OS-level strace, so we use this information to remove background traffic defined as any flow associated to an app/service different from the app reported in the file name of each experiment. Last, for UTMOBILENET21 the labels rely solely on experiment filenames, i.e., we are unable to remove any background due to lack of information.

Datasets imbalance. Real-world large datasets, as shown in Table 3.4, exhibit higher class imbalance (ratio between the most and least popular class) than test-bed small ones due to collection constraints. To simulate progressively harder classification tasks, we segment the large datasets in “tiers”, each with increasingly greater class imbalance. Figure 8.5 illustrates this imbalance by comparing the absolute sample counts per class across entire datasets (left), and the normalized distributions within the top-50 (center) and top-100 (right) classes. Notice how when focusing on the most frequent classes the distributions are fairly similar across datasets, but Enterprise shows a very long tail, indicating a broader and more imbalanced class distribution.

8.5.2 Hyperparameters and training details

Table 8.5 provides an overview of the hyperparameters. DDPMS4 is optimized using the AdamW optimizer with a learning rate of 7.5×10^{-4} , scheduled with 10% linear warmup followed by cosine decay to 0. Training runs for 300 epochs. We use a large batch size of 4096 and apply gradient clipping with a threshold of 0.5 for stability. The network architecture includes 5 residual layers with 512 channels and a time embedding dimension of 128. Diffusion is performed over 100 steps using a cosine *beta* schedule. The same hyperparameters are used for all datasets, as they serve as reliable defaults that perform well across all tested cases. Regarding to the

Hyperparam.	DDPMS4	TabDDPM	ImagenTime
LR	7.5×10^{-4}	1×10^{-3}	1×10^{-4}
LR scheduler	warmup+cosine	infinite cosine	constant
Optimizer	AdamW	AdamW	AdamW
Weight decay	0	0	1×10^{-5}
Batch size	4096	4096	1024
Epochs	300	300	300
Grad. clip thresh.	0.5	n.a.	1.0
#Layers	5	search in [2, 4, 6, 8]	3 resolution levels
#Feat. in backbone	512×30	search in [256, 512, 1024, 2048]	U-Net
#Feat. of time emb.	128	128	128
NFE	100	search in [50, 100, 500, 1000]	20
DM	DDPM(cosine β_t)	DDPM(cosine β_t)	EDM

TABLE 8.5: Hyperparameters

baseline generative models TabDDPM and Imagen, their hyperparameters are also noted in Table 8.5, from which we could tell their similarities and differences.

To ensure robust evaluation, we apply 5-fold stratified cross-validation with random splits of 80%/10%/10% for training, validation, and testing, respectively; hyperparameters are tuned on the validation splits, and final performance is reported on the test splits. To provide confidence in the results, we report the mean and standard deviation over 3 random seeds per fold, resulting in 15 runs per experiment. For evaluation, from each method, we build a synthetic dataset matching the size of the real training set — by generating new samples conditioned on class labels sampled from the real training label distribution (for generative models) or by applying augmentations once per sample (for hand-crafted methods).

8.6 Replication of NetDiffusion

As a representative attempt to apply pretrained image diffusion models (Stable Diffusion) to network traffic, NetDiffusion (Jiang et al., 2024b) offers an interesting point of comparison. As the original evaluation was based on private datasets, we adapt its pipeline to the publicly available UNSW24 dataset, enabling a reproducible and fair comparison. In this section, we discuss the replication results of NetDiffusion.

As the datasets used in Jiang et al., 2024b are not public, for this analysis we rely only on UNSW24 for which we have raw pcap and with traffic similar to what used in the original paper, namely video streaming services. The authors consider 1024 packets for each flow as a 1024×1088 nPrint/pixel image, where each row corresponds to a packet and each column to a header bit. Since UNSW24 dataset is small (only up to 443 samples for the smallest class when considering flows with more than 30 packets) and filtering for flows with $\geq 1,024$ packets would result $< 1,000$ samples for all classes, we opt for considering flows with ≥ 100 packets (leaving 351 samples for the smallest class while all the others have $\geq 1,500$). To match the 1024-rows format used in the original paper, we replicate these 100 rows nine times and then pad the remaining rows with “vacant” placeholders (set to -1). While this approach would

not be effective to create real flows, it enable us to fine-tune SD1.5 using the approach of NetDiffusion.

We train a LoRA adapter on Stable Diffusion 1.5 for two days on an NVIDIA H100 GPU (i.e., more than twice the computational budget used for the other methods in our benchmark). Since neither the paper nor the official GitHub repository specifies any hyperparameter, we adopt a constant learning rate of 5×10^{-5} and a batch size of 10 (the maximum supported by the GPU) with 5 gradient accumulation steps (so effectively 50 samples before operating an update). We also experimented with a learning rate of 1×10^{-3} , which just resulted in faster convergence but no substantial performance improvement, as shown in [Figure 8.7](#) and [Figure 8.8](#). At sampling, we apply a Canny-based ControlNet to the fine-tuned LoRA adapter, with prompt guidance scale and canny guidance scale being 3 and 1 respectively. We do not apply images down scaling neither for finetuning nor at sampling time. The canny edge input obtained from the real sample and the corresponding sampled image are shown in [Figure 8.6](#). Although the training loss decreases, visual inspection of the sampled image suggests that the LoRA adapter only learns the overall three-color distribution to some extent, while prior knowledge from the pretrained SD1.5 still persists — for example, the fish-scale patterns that should not be present. This indicates that the model fails to capture the key semantics. That said, this outcome is expected, given the significant domain shift between nPrint images and the pretraining data of SD.

After obtaining a generated image, NetDiffusion applies a sequence of manually designed post-processing rules to align the generated image with detailed transport- and network-layer protocol rules at both inter- and intra-packet levels, despite the paper’s ablation results showing that the performance remains unchanged with or without these rules. The effect of each post-processing rule on the generated image is illustrated in [Figure 8.6](#). While these rules do make the overall image more visually similar to real traffic, they remain insufficient for preserving the most critical semantics to downstream classification — specifically, the size and direction of the first 30 packets. Specifically, NetDiffusion is “overwriting” packets directions by constructing a 2-state markov chain modeling the “state” of the source IP field based on the real nPrint sample used for the ControlNet signaling, i.e., given a packet at time t , what is the probability that the probability that the following packet has the same IP address or the direction is reversed. Since there are only 2 possible IPs, this result in a markov chain with 2 states and 4 transitions (srcIP→srcIP, srcIP→dstIP, dstIP→srcIP, dstIP→dstIP). While this approach can guarantee that for a long flow the overall distribution of the direction is reasonably close to the expected one, when considering the first few packets of a flow the distortion can be significant (e.g., it is very unlikely that the first 3 packets of a TCP flow resemble the expected directions of the 3-way handshake).

Finally, we map generated nPrint back to packet series representation, focusing on

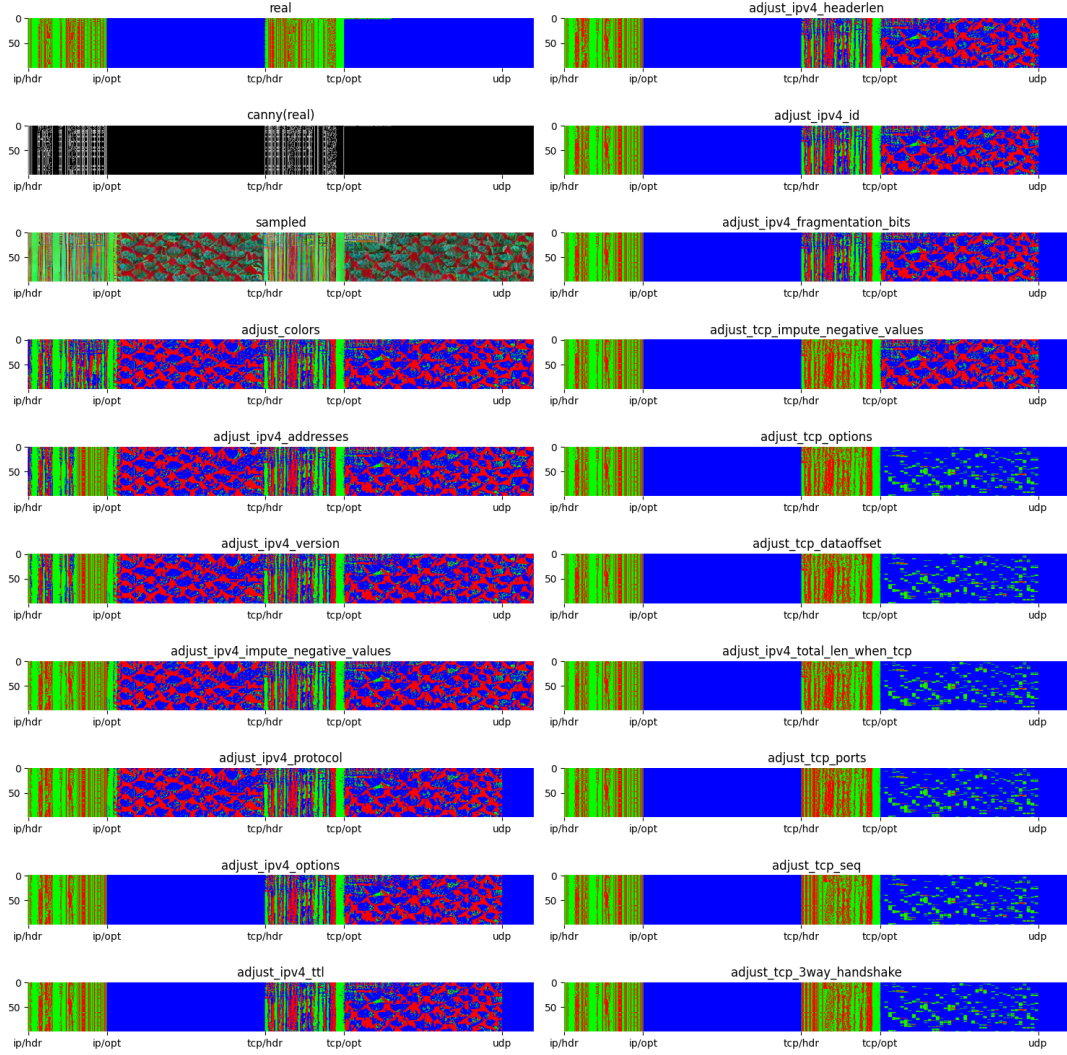


FIGURE 8.6: Example of NetDiffusion set of postprocessing rules applied sequentially (left column first, top to bottom). After selecting a real image from the dataset (top-left picture) and extracting the edges via a canny filter (2nd image in the left column), a sample is extracted (3rd image in the left column) and the rules are applied sequentially.

the size and direction of the first 30 packets. The resulting visualization for 200 samples per class are shown in Figure 8.8, where the four columns correspond to: real training data, synthetic samples generated by our DDPMS4 model, and NetDiffusion outputs with and without ControlNet. It is evident that the samples generated by DDPMS4 closely resemble the real data and successfully preserve class-relevant semantics. In contrast, NetDiffusion samples differ clearly from the real data and show little distinction between classes. In particular, we observe that packet sizes are clipped to 1500. In fact, despite multiple rules focus on having header field syntactically correct (e.g., when “cleaning” the TCP options, the IP total length field is adjusted accordingly), the underlying base values are not learned properly by SD1.5, so a rule simply clips them to the expected MTU=1500.

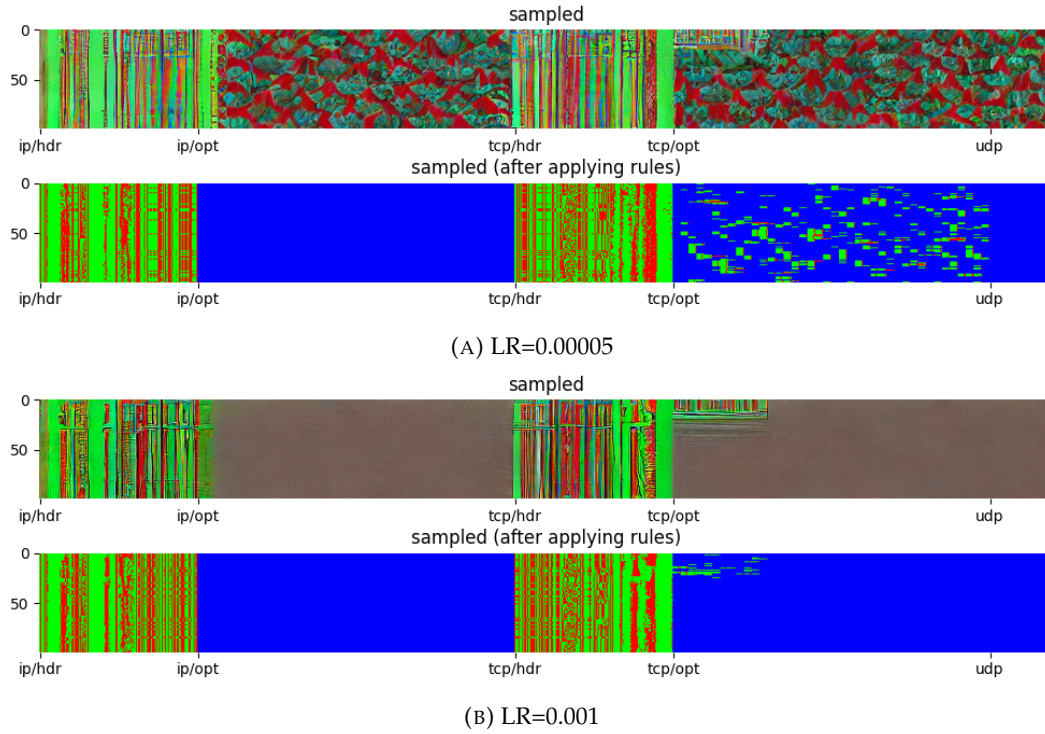


FIGURE 8.7: Example of NetDiffusion sampling (with ControlNet).

Overall, despite our best efforts, we have not being able to use NetDiffusion effectively so we excluded it from our benchmark.

8.7 Classification evaluation

Since classification is our target task, our primary goal is to assess how effectively synthetic data supports downstream classification. To do this meaningfully, we first need to decide which classifier to use, and establish baseline performance on real data to understand dataset difficulty and characteristics. We therefore begin this section by comparing a ML classifier (XGBoost) and a DL model (ResNet), both trained on real data. While this comparison is almost always ignored in TC literature which tends to focus only on DL solutions, our results instead show that the two models can provide similar classification performance but have different tradeoffs considering training costs. Moreover, by using multiple datasets we observe that dataset cleanliness and class imbalance can play a significant role. Since our primary goal is general classification performance, we choose XGBoost for its significantly faster training time and use it to evaluate the synthetic data performance (Section 8.7.1).

To assess the effectiveness of synthetic data for downstream classification, we consider two settings: the utility test, where a classifier is trained only on synthetic data to evaluate whether it can substitute real training data (Section 8.7.2); and the augmentation test, where synthetic data is combined with real data to evaluate

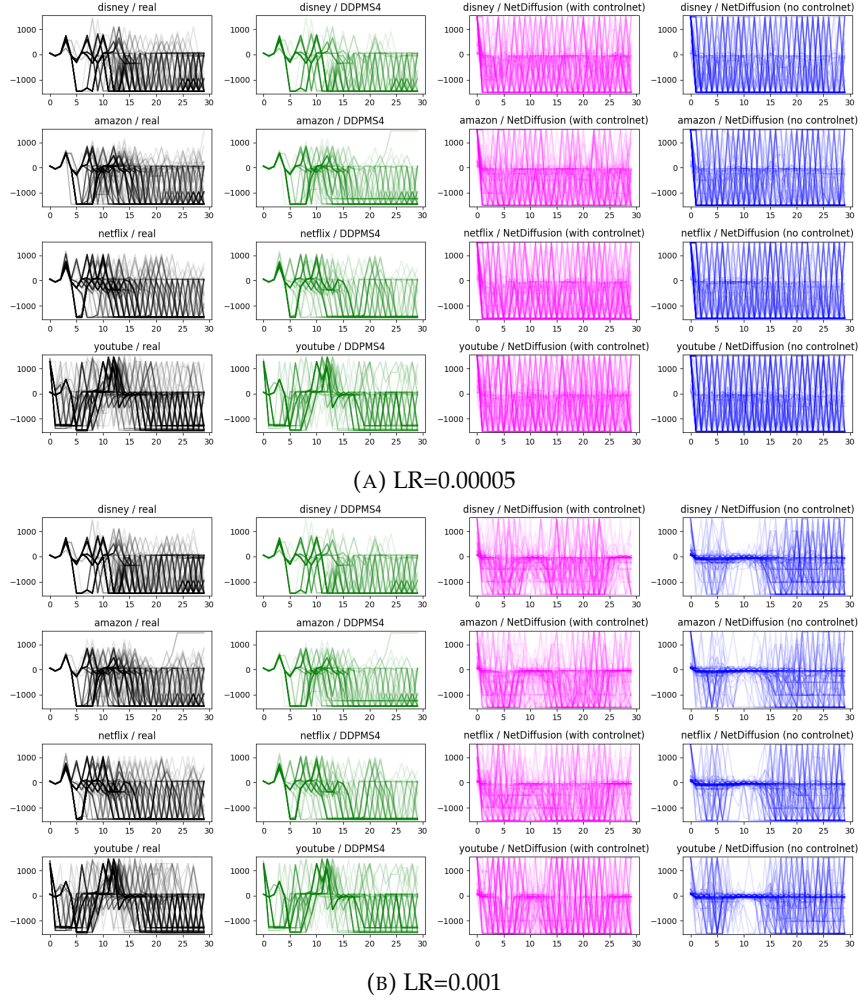


FIGURE 8.8: Qualitative comparison of packet time series generated from UNSW24.

whether it provides additional variability that improves classification performance (Section 8.7.3). Each subsection presents the corresponding metrics and results.

8.7.1 Base classifier

Most of TC literature only focus on DL classifiers, yet most of ML literature on generative tabular data consider ML models. To embrace both points of view we contrast XGBoost against a ResNet-based model with 1D-CNN blocks.

Classifiers configuration. For XGBoost, we search the best hyper parameters via Optuna (Akiba et al., 2019) considering the grid reported in Table 8.6. Specifically, we performed 30 trials on a single split and evaluated the performance on the validation set, guiding the search to maximize the classification weighted F1. Conversely, for ResNet we experienced a significantly longer training time, so we resorted to run small-scale experiments (without Optuna) to explore architectural choices, batch sizes, and learning rate schedules. We settled on two variants which we simply name ResNet (one block of size 64 followed by one of 128) and ResNet(2×) (two blocks of

Name	Range
n_estimators	LogUniform[50, 300]
learning_rate	LogUniform[0.1, 1]
max_depth	Uniform[4, 10]
min_child_weight	Uniform[1, 50]
subsample	Uniform[0.5, 1]
lambda	Uniform[0, 3]

TABLE 8.6: XGBoost classifier Optuna search hyper params.

ResNet learning rate scheduler.		ResNet batch size (with infinite cosine LR).	
Name	Weighted F1	Batch size	Weighted F1
Infinite Cosine	97.09 \pm 0.04		
Linear annealing	96.30 \pm 0.03		
Cosine annealing	96.31 \pm 0.04		
Constant (0.001)	95.44 \pm 0.11		
Exponential	95.93 \pm 0.04		
		512	97.09 \pm 0.04
		1024	96.96 \pm 0.05
		10240	96.10 \pm 0.03

TABLE 8.7: ResNet classifier hyper params.

64 followed by two of 128), to assess the effect of increased model capacity. We trained the ResNet models using AdamW with a batch size of 512 and adopted the Infinite Cosine scheduler (Singh et al., 2025), a composite learning rate policy combining warmup, cosine decay, and exponential annealing. Models were trained for 200 and 400 epochs (ResNet and ResNet(2 \times), respectively) without early stopping, as the schedule was designed to operate over the full number of epochs. Table 8.7 reports some results about the different config for a single seed for one split (just to provide a qualitative sense).

Table 8.8 collects the performance of the base classifiers across multiple dimensions. The results reflect 15 models (5 splits with 3 training seeds for each split).

Dataset Tier	Weighted F1			Macro F1			Model size [k]			Train time		
	XGB	ResNet	ResNet(2 \times)	XGB	ResNet	ResNet(2 \times)	XGB	ResNet	ResNet(2 \times)	XGB	ResNet	ResNet(2 \times)
UNSW24 *	99.56 \pm 0.21	99.15 \pm 0.37	99.30 \pm 0.40	99.38 \pm 0.35	98.71 \pm 0.47	98.78 \pm 0.78	10	115	238	1s	2m	5m
Mirage22 *	96.17 \pm 0.26	93.07 \pm 0.41	93.18 \pm 0.52	96.16 \pm 0.33	92.88 \pm 0.52	93.02 \pm 0.67	37	118	241	2s	1m	6m
UTMOBILENET21 *	87.67 \pm 1.81	84.77 \pm 2.29	84.47 \pm 2.13	83.00 \pm 2.24	79.87 \pm 3.26	79.75 \pm 2.67	44	118	241	3s	1m	5m
CESNET-TLS22	50 98.85 \pm 0.02	97.80 \pm 0.06	98.53 \pm 0.02	98.64 \pm 0.03	97.27 \pm 0.08	98.19 \pm 0.03	974	144	268	1m	1h	12h
	100 98.44 \pm 0.02	97.30 \pm 0.05	98.18 \pm 0.03	97.64 \pm 0.04	95.99 \pm 0.11	97.25 \pm 0.08	3,054	176	300	4m	1h	14h
	* 98.26 \pm 0.03	97.10 \pm 0.06	98.05 \pm 0.03	96.62 \pm 0.09	94.81 \pm 0.11	96.37 \pm 0.10	4,126	199	322	6m	1h	14h
CESNET-QUIC22	50 91.75 \pm 0.08	90.32 \pm 0.11	91.70 \pm 0.09	88.81 \pm 0.15	86.72 \pm 0.17	88.79 \pm 0.17	4,136	144	268	2m	45m	3h
	* 90.79 \pm 0.06	89.37 \pm 0.06	90.86 \pm 0.08	82.61 \pm 0.15	80.66 \pm 0.20	83.17 \pm 0.24	5,078	164	288	3m	36m	3h
	50 97.58 \pm 0.01	95.95 \pm 0.73	97.52 \pm 0.07	96.26 \pm 0.02	93.60 \pm 1.19	96.10 \pm 0.11	7,309	151	268	3m	11h	19h
Enterprise	100 96.30 \pm 0.02	94.55 \pm 0.05	96.27 \pm 0.23	94.24 \pm 0.03	91.33 \pm 0.08	94.02 \pm 0.31	15,160	176	300	8m	8h	22h
	300 92.62 \pm 0.03	91.51 \pm 0.48	94.07 \pm 0.32	83.87 \pm 0.05	80.59 \pm 0.88	85.76 \pm 0.61	5,553	304	428	22m	10h	25h
	* 92.48 \pm 0.05	91.12 \pm 0.04	93.87 \pm 0.06	78.88 \pm 0.16	75.94 \pm 0.12	81.86 \pm 0.16	10,894	433	556	41m	14h	26h

TABLE 8.8: Performance of ML and DL base classifiers.

Small datasets. On UNSW24, performance is saturated across models. On Mirage22 and UTMObILENET21, ResNet has about -3% gap w.r.t. XGBoost, even when increasing the training budget. UTMObILENET21 shows the lowest performance and the highest

variability across runs, suggesting it is a very “noisy” dataset. In terms of training complexity and model size, XGBoost trains faster and yields compact models with moderate depth and estimator count.

Large datasets. Considering the large datasets we observe a mix of scenarios. Focusing on datasets up to 100 classes, Classifiers achieve similar performance on CESNET-TLS22 and Enterprise datasets, yet both CESNET-QUIC22 tiers seem harder tasks, resulting in lower scores. As expected, the long tail of Enterprise significantly complexity the task as evident from the -15/20% drop of the macro F1 between top-50 and all 500 classes.

While ResNet is consistently under performing compared to XGBoost, using larger neural networks allows to recover the gap but the weighted-vs-macro F1 comparison enables a few observations. For instance, XGBoost and ResNet(2×) offers a comparable weighted F1, but the macro F1 is showing a +2/3% advantage for ResNet(2×). Results also highlight an evident trade-off between model size and training time. Given the large number of samples, Optuna tends to prefer XGBoost configurations with `n_estimators` very close to the allocated budget. In other words, the search is suggesting that “the more the capacity, the better the classification performance”. This results in incredibly large models, hardly useful for any practical deployment (unless, perhaps, in the presence of on-device hardware accelerations). As from Table 8.9, investigating all XGBoost Optuna trials² we find configurations resulting in smaller models with respect to the absolute best and within a 1% classification performance gap from the best, yet those models are not as compact as ResNet models.

At the same time, ResNet models have a significantly higher training cost. This is due to the very different training dynamics of the two algorithms. Specifically, training XGBoost on GPU requires pinning the whole dataset to the GPU as the algorithm does not support training in batches. Conversely, for training ResNet we used the traditional DL approach of sending individual batches to the GPU, which introduces system overheads. Likewise, using a batch size of 512 (selected as result of a grid search) contributes to the overhead given the large size of the datasets.

Max F1 penalty	Weighted	Weighted F1			Model size [k]		
		CESNET-QUIC22	CESNET-TLS22	Enterprise	CESNET-QUIC22	CESNET-TLS22	Enterprise
none		90.94	98.27	92.53	5,075	4,128	10,880
-0.5%		90.60	97.89	92.24	2,709	2,082	8,760
-1.0%		90.04	97.41	91.85	1,761	774	7,427

TABLE 8.9: Investigating XGBoost model size -vs- performance.

Takeaways. From the results we gather the following observations. First of all, we argue there is a significant value in integrating ML baselines when evaluating

²Results in Table 8.9 only qualitatively match Table 8.8 results as the Optuna trials are run on a single split and are assessed on the validation split, while the models are finally evaluated across 5 test splits.

classifiers. Unfortunately, most of the TC literature is focusing on small datasets and do not take in consideration ML models. Our analysis instead suggests the need to reverse this trend: while small datasets can be useful to ease prototyping, they should not be the primary tool for studying new DL algorithms/models unless proven to beat off-the-shelf ML solutions.

Second, model evaluation should consider a broader scope of metrics not just mere classification performance. For instance, in our analysis we intentionally investigate the model capacity to show that both XGBoost and ResNet can be configured to be equivalent/better/worse than each other. A less often investigated aspect is the trade-off between model size and training cost. Without knowing the computational budget available in environment where the models are expected to run (e.g., a network monitoring environment may or not integrate an hardware accelerator) it is difficult to optimize models. For instance, on large datasets, ResNet offers efficient inference due to its compact size, but it requires significantly longer training time.

Ultimately, given the significantly faster training speed, in the following we rely on XGBoost.

8.7.2 Class-semantics fidelity (utility test)

In this subsection, we evaluate whether synthetic data alone can capture the class-conditional structure well enough to support accurate classification. In other words, we aim to answer the question: **Does the synthetic data accurately model the class-conditional distribution for effective classification?**

Metric. One of the most common approaches to assess the class-semantics fidelity of a generative model is to train a model with synthetic data (obtained from generative models or via hand-crafted augmentations) and evaluate it with real data. In our case, we use TC as the downstream task by training an XGBoost classifier using solely synthetic samples and evaluating it on test split of real data. To better highlight differences, we measure the performance gap of training the downstream classifier on synthetic data with respect to real data. The smaller the gap, the better the generative model has effectively captured the class-conditional patterns in the real data, reflecting its ability to preserve task-relevant semantics. Notably, to the best of our knowledge, this utility test is the only evaluation metric that makes use of the class labels y .

Given the class imbalance in our TC datasets, we report both weighted and macro F1 scores. As an upper bound reference, we also include the F1 score of a classifier trained only on the real data.

Results. Utility test results (Table 8.10) are based on 5-fold cross-validation, with each fold repeated using 3 different seeds. Synthetic dataset is at the same size of the real training set. Similarly, we transform each training sample three times to obtain the datasets related to hand-crafted augmentations. We report the absolute

Weighted F1 Gap \uparrow										Macro F1 Gap \uparrow						
		Ref/real	Fake/generated			Fake/hand-crafted			Ref/real	Fake/generated			Fake/hand-crafted			
Dataset	Tier	XGB	DDPMS4	TabDDPM	ImagenTime	Translate	Wrap	Gaussian	XGB	DDPMS4	TabDDPM	ImagenTime	Translate	Wrap	Gaussian	
UNSW24	*	99.56	-0.21	-0.45	-0.39	-0.72	-0.45	-0.34	99.38	-0.35	-0.65	-0.63	-0.87	-0.62	-0.40	0.59
	*	96.17	-0.26	-1.77	0.38	-3.94	0.52	-4.60	96.16	0.33	-1.94	0.37	-4.34	0.61	-4.92	0.45
	*	87.67	1.81	-1.81	1.20	-3.40	1.14	-4.73	83.00	2.24	-1.94	1.88	-3.50	1.24	-5.96	2.68
Mirage22		98.85	0.02	-1.04	0.03	-3.98	0.06	-2.85	98.64	0.03	-1.30	0.05	-4.91	0.07	-3.58	0.05
		98.44	0.02	-1.27	0.04	-4.72	0.05	-3.28	97.64	0.04	-1.81	0.08	-6.52	0.04	-4.53	0.03
		98.26	0.03	-1.30	0.03	-5.00	0.09	-3.65	96.62	0.09	-2.23	0.09	-8.13	0.13	-6.26	0.04
CESNET-TLS22	50	91.75	0.08	-3.05	0.07	-8.80	0.14	-4.63	88.81	0.15	-4.51	0.22	-12.84	0.20	-7.08	0.17
	*	90.79	0.06	-3.27	0.07	-9.37	0.13	-5.00	82.61	0.15	-5.82	0.24	-17.71	0.36	-9.97	0.20
CESNET-QUIC22	50	97.58	0.01	-2.31	0.03	-5.07	0.04	-6.28	96.26	0.02	-3.56	0.06	-7.80	0.07	-9.63	0.05
	100	96.30	0.02	-3.10	0.02	-7.03	0.06	-8.43	94.24	0.03	-4.78	0.03	-10.88	0.09	-13.06	0.08
	300	92.62	0.03	-3.54	0.06	-8.73	0.11	-10.50	83.87	0.05	-7.59	0.10	-18.89	0.16	-22.89	0.13
Enterprise	*	92.48	0.05	-3.94	0.03	-9.69	0.12	-11.30	78.88	0.16	-9.32	0.17	-23.68	0.22	-27.53	0.17
avg across		94.16	0.40	-2.09	0.35	-5.35	0.41	-4.94	89.44	0.55	-3.65	0.56	-9.71	0.53	-9.17	0.69

Negative utility test = training with synthetic data underperforms compared to training with real data.

TABLE 8.10: Utility test of generative models and hand-crafted augmentations.

weighted/macro F1 as from Table 8.8 as reference and the gap (i.e., generated F1 – real F1) on synthetic data, so higher values indicate better performance.

The key observations are described below:

1. Compared to real training data, all types of synthetically generated data — whether generated or hand-crafted — result in negative scores. This is to be expected as modeling is imperfect and some patterns present in the real data are not fully captured, leading to some information loss.

Despite DMs have shown incredible generalization capability, our results are in line with findings from ML literature, i.e., even in the best case, the generated data is still a few percentage point behind real data. This can be justified considering that (i) DMs capacity is fundamentally limited by the information available in the real data they’re trained on, and (ii) perfectly estimating the score function without error is very challenging. Therefore, it is reasonable to consider the performance of a classifier trained on real data as an upper bound. With this in mind, DDPMS4-generated data achieves competitive weighted F1 scores on real test data, with an average -2% performance gap. This indicates that class-dependent patterns are largely preserved in the synthetic packet sequences and generalize well to unseen samples. Compared to generative base-lines such as TabDDPM and ImagenTime, DDPMS4 consistently outperforms them across all the datasets, demonstrating a stronger ability to model the conditional distribution.

For hand-crafted transformations, it is important to underline that (i) the intensity of the transformation depends on both the designed rules and their configuration and (ii) differently from generative models, the rules operated directly on real samples so the less an original sample is altered, the higher the utility score will be — at the extreme, an identity mapping would yield the same performance as using real training data. Based on this, the utility test performance is ranking *translate* > *wrap* > *gaussian*, matching our benchmark in Part I, which

is justified based on the properties of these transformations. Specifically, Translate and Wrap are time-domain transformations that tend to preserve many of the original original values, while variations occurs by “shifting” values left or right, adding zeros (which are out of distribution values in our cases as packet sizes in our datasets capture what observed as IP level) or, for Wrap, interpolating consecutive values. In contrast, the gaussian transformation operates on amplitude, altering the value distribution across all packets. More specifically, the configuration used for Gaussian results in variation of ± 20 -30 bytes for each packet. This seems small in absolute scale, yet the results show that this perturbation is sufficient to disrupt class semantic.

2. Comparing weighted and macro F1 scores, the performance gap between synthetic and real data is larger for macro F1 in both generated and hand-crafted cases. This suggests that class imbalance makes it harder to model tail classes, leading to greater information loss for minority classes. Moreover, as more tail classes are included in a dataset, the gap tends to be larger.

Extended analysis: Scaling in Number of Generated Data. Since generative models can produce unlimited samples, we test the scalability of our DDPMS4 by increasing the number of generated examples for utility test, as shown in [Figure 8.9](#).

We considered Mirage22 as it representative for the small datasets, but it is also the least imbalanced dataset, and relatively easy to model according to the an XGBoost trained on real data. DDPMS4 provides very good utility test, but we obtain a noticeable +2% improvement when expanding the synthetic samples up to $15\times$ original training set size. This suggests that the stochastic variation introduced by sampling more data can help recover underrepresented patterns.

Conversely, on the large, more imbalanced, and more challenging dataset CESNET-QUIC22-50, we also observe only a +1% improvement, and adding more synthetic data does not allow to recover the gap with respect to the upper bound. On the one hand, this confirms that generating more samples is beneficial — although low-density regions get generated with low probability, increasing the number of synthetic samples raises the expected number of samples falling into these regions, making it more likely that the synthetic dataset includes these rare but informative patterns. On the other hand, increasing the number of samples alone is insufficient to eliminate the gap, as it does not address systematic errors introduced by imperfect score estimation in the reverse diffusion process. These errors can lead to biased generation in specific regions of the data space, limiting the overall fidelity and utility of the synthetic data.

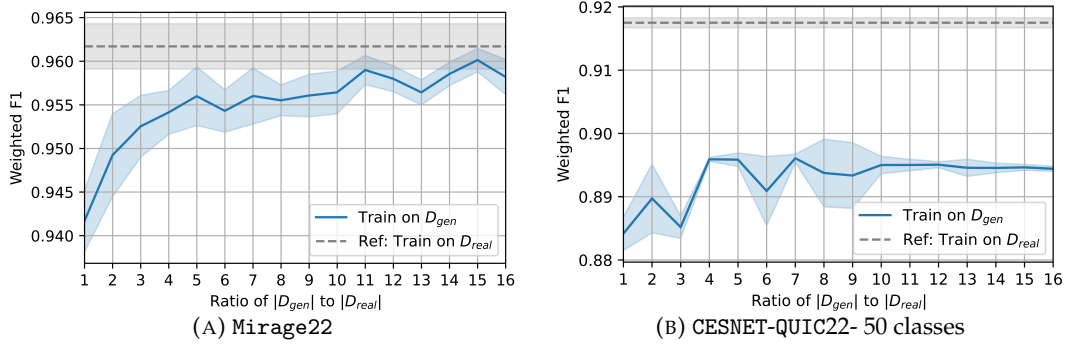


FIGURE 8.9: Scaling in number of generated data - utility test

		Relative Weighted F1 \uparrow										Relative Macro F1 \uparrow												
		Ref/real	Fake/generated					Fake/hand-crafted					Ref/real	Fake/generated					Fake/hand-crafted					
Dataset	Tier	XGB	DDPMS4	TabDDPM	ImagenTime	Translate	Wrap	Gaussian		XGB	DDPMS4	TabDDPM	ImagenTime	Translate	Wrap	Gaussian		XGB	DDPMS4	TabDDPM	ImagenTime	Translate	Wrap	Gaussian
UNSW24	*	99.56 _{0.21}	-0.12 _{0.24}	-0.10 _{0.19}	0.03 _{0.17}			0.37 _{0.18}	0.29 _{0.14}	0.04 _{0.33}	99.38 _{0.35}	-0.16 _{0.38}	-0.10 _{0.28}	0.10 _{0.23}	0.46 _{0.26}	0.31 _{0.23}	0.10 _{0.50}							
	Mirage22	* 96.17 _{0.26}	0.07 _{0.22}	-0.49 _{0.37}	-0.00 _{0.41}			0.58 _{0.26}	0.51 _{0.46}	-0.93 _{0.49}	96.16 _{0.33}	-0.02 _{0.24}	-0.55 _{0.37}	-0.03 _{0.53}	0.57 _{0.30}	0.51 _{0.40}	-1.10 _{0.51}							
	UTMOBILENET21	* 87.67 _{1.81}	0.49 _{0.94}	0.08 _{0.72}	-0.47 _{0.70}			1.01 _{0.81}	0.36 _{0.44}	-0.78 _{0.74}	83.00 _{2.24}	0.89 _{1.22}	0.58 _{0.79}	-0.28 _{0.79}	1.55 _{0.87}	0.77 _{0.48}	-0.73 _{1.05}							
CESNET-TLS22	50	98.85 _{0.02}	-0.13 _{0.02}	-0.86 _{0.03}	-0.35 _{0.01}			0.06 _{0.02}	-0.18 _{0.02}	-0.93 _{0.04}	98.64 _{0.03}	-0.16 _{0.03}	-1.01 _{0.03}	-0.41 _{0.02}	0.10 _{0.03}	-0.18 _{0.04}	-1.05 _{0.05}							
	100	98.44 _{0.02}	-0.04 _{0.02}	-0.68 _{0.02}	-0.11 _{0.01}			0.23 _{0.02}	0.11 _{0.02}	-0.53 _{0.03}	97.64 _{0.04}	-0.00 _{0.05}	-0.89 _{0.05}	-0.06 _{0.03}	0.38 _{0.03}	0.24 _{0.04}	-0.59 _{0.06}							
	*	98.26 _{0.03}	-0.02 _{0.03}	-0.77 _{0.03}	-0.16 _{0.01}			0.25 _{0.03}	0.12 _{0.03}	-0.62 _{0.03}	96.62 _{0.09}	0.10 _{0.06}	-1.11 _{0.07}	-0.11 _{0.09}	0.53 _{0.08}	0.41 _{0.08}	-0.76 _{0.06}							
CESNET-QUIC22	50	91.75 _{0.08}	-0.03 _{0.09}	-1.63 _{0.07}	0.06 _{0.03}			0.72 _{0.06}	0.28 _{0.07}	-0.70 _{0.09}	88.81 _{0.15}	-0.05 _{0.15}	-2.22 _{0.12}	0.15 _{0.03}	1.07 _{0.12}	0.57 _{0.10}	-0.80 _{0.17}							
	*	90.79 _{0.06}	-0.04 _{0.10}	-1.86 _{0.08}	-0.05 _{0.04}			0.64 _{0.05}	0.10 _{0.06}	-0.94 _{0.04}	82.61 _{0.15}	0.39 _{0.22}	-2.86 _{0.29}	0.13 _{0.22}	1.55 _{0.17}	0.69 _{0.15}	-1.13 _{0.18}							
Enterprise	50	97.58 _{0.01}	-0.49 _{0.02}	-0.99 _{0.01}	-0.42 _{0.00}			-0.12 _{0.01}	-0.41 _{0.01}	-0.51 _{0.01}	96.26 _{0.02}	-0.76 _{0.04}	-1.48 _{0.02}	-0.60 _{0.00}	-0.14 _{0.02}	-0.53 _{0.02}	-0.74 _{0.02}							
	100	96.30 _{0.02}	-0.67 _{0.02}	-1.41 _{0.01}	-0.52 _{0.01}			-0.05 _{0.02}	-0.64 _{0.02}	-0.81 _{0.02}	94.24 _{0.03}	-1.02 _{0.03}	-2.08 _{0.03}	-0.74 _{0.03}	-0.01 _{0.02}	-0.73 _{0.03}	-1.11 _{0.04}							
	300	92.62 _{0.03}	-0.71 _{0.10}	-1.93 _{0.09}	-1.17 _{0.03}			0.11 _{0.05}	-0.79 _{0.05}	-1.34 _{0.10}	83.87 _{0.05}	-1.02 _{0.18}	-3.26 _{0.14}	-1.55 _{0.08}	0.89 _{0.09}	-0.55 _{0.07}	-1.54 _{0.18}							
avg across		94.18 _{0.40}	-0.06 _{0.27}	-0.85 _{0.24}	-0.30 _{0.23}			0.49 _{0.23}	0.10 _{0.20}	-0.76 _{0.29}	90.27 _{0.54}	0.03 _{0.38}	-1.22 _{0.32}	-0.29 _{0.32}	0.93 _{0.29}	0.35 _{0.24}	-0.86 _{0.41}							

TABLE 8.11: Augmentation test.

8.7.3 Additional variety beyond the real training data (augmentation test)

We complement the utility test by evaluating the benefit of using synthetic data alongside real training data during training. By training the classifier on the concatenation of real training data and synthetic data, we aim to answer the question: **Does synthetic data add useful variability beyond real data, such that combining them improves classification performance?**

Metric. In addition to training a classifier solely on synthetic data, we also evaluate performance when training classifier on the concatenation of real and synthetic data. This setup, referred to as the augmentation test, aims to assess whether the synthetic samples provide additional variability beyond the real training data and improve downstream classification.

Results. Table 8.11 presents the augmentation test results, where the classifier is trained on the concatenation of fake data and real data. Results are gathered using the data generated for the utility test, hence Table 8.11 reflects an aggregation of 15 runs for each dataset. From the results we gather the following observations:

1. Even the best generated data only occasionally improves performance, and the gains are generally marginal. In most cases, adding generated data alongside real data actually reduces F1 scores. This suggests that generative models introduce noise by failing to fully capture the real class-conditional distribution. As a result, the synthetic data may conflict with real examples, which distorts

the training data distribution and degrades test performance. The drop is more noticeable in macro F1 than in weighted F1, indicating that modeling minority classes is more challenging and therefore this negative impact is more severe. Among the generative models, DDPMS4 is on par with ImagenTime, while TabDDPM is the worst across all datasets.

2. In contrast, by incorporating useful domain knowledge into the design, hand-crafted augmentation can be more reliable. Translate consistently brings significant improvements across all datasets, confirming its effectiveness. Wrap is also beneficial on small datasets but provides smaller gains; on large datasets, it tends to lower weighted F1 while improving macro F1, suggesting that it introduces variety for minority classes at the cost of slightly disrupting majority class patterns. Gaussian, which performed poorly in the utility test (indicating it breaks class-relevant structure), causes the most significant drop in the augmentation test as well — an expected outcome. Overall, the ranking $\text{translate} > \text{wrap} > \text{gaussian}$ aligns with the findings from [Part I](#).

Extended analysis: Scaling in Number of Generated Data. As in the utility test, we expand the number of generated samples in the augmentation test to assess potential gains. Results are shown in [Figure 8.10](#).

On the small dataset Mirage22, since the generative model is trained solely on real training data, the amount of information it can capture is inherently upper-bounded by that data. As a result, adding more synthetic samples cannot exceed this information ceiling, and the performance remains on par with training on real data alone.

On the large dataset CESNET-QUIC22- 50, increasing low-quality synthetic data harms performance, as more samples lead to lower weighted F1 scores. This indicates that when generation quality is not good enough, adding more data amplifies noise rather than improving learning. By evaluating classification accuracy on the real training data, we find that the classifier trained on real data plus $16\times$ synthetic data performs 3% worse than the one trained with only real plus $1\times$ synthetic data. This suggests that when the model underfits the real data — as we will show in the next section, where fitting is noticeably worse on large datasets than on small ones — excessive imperfect synthetic samples can overwhelm the real data signal, leading the classifier to fit a distorted distribution biased by the synthetic data.

8.7.4 Concluding remarks

This section evaluated generative models and hand-crafted transformations for TC. We have shown empirically that class-conditional generative models can produce synthetic data that preserves most class-relevant semantics and effectively supports downstream classification. Our method, DDPMS4, achieves the best performance among all generative models considered. While a gap remains compared to training

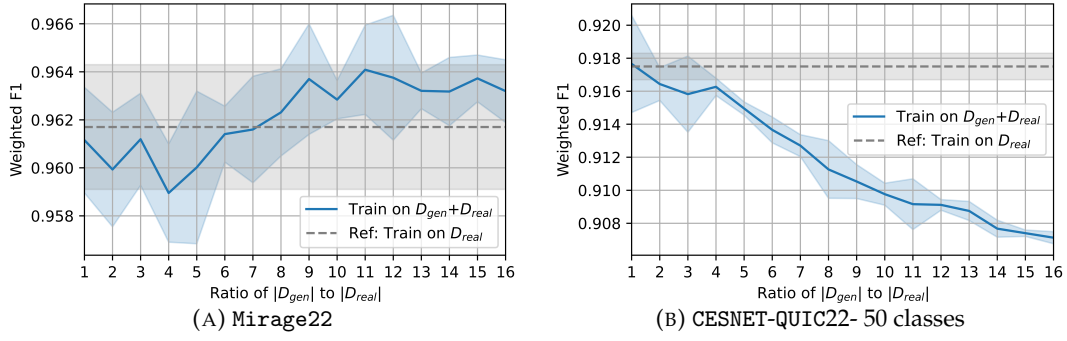


FIGURE 8.10: Scaling in number of generated data - augmentation test

on real data due to imperfect score estimation, we believe this gap can be further reduced in future work.

While diffusion models are primarily designed to optimize in-distribution fidelity, results show that for effective data augmentation in downstream classification, they must be re-engineered to include a complementary mechanism that promotes diversity beyond the training distribution, as relying solely on accurate modeling of real training data to yield additional useful variety is unrealistic — generation does not provide a “free lunch”. This points to a clear future work direction: designing generative models that incorporates task-relevant variability alongside realism to better support downstream classification.

While we already started to investigating this path, the key for an effective generative model for augmentation resides in having the right “signaling” from a classifier. In other words, the effectiveness of hand-crafted transformations for augmentation remains unclear for packet series, primarily due to the lack of understanding of class semantics. In computer vision, humans can easily identify class-relevant features (e.g., a cow’s face for the class “cow”), which supports the design of transformations that preserve such features while removing irrelevant cues (e.g., blurring background texture). In contrast, for packet series, it is unclear which segments or timesteps encode class-relevant information, making it challenging to design principled transformations or draw analogies to generative augmentation strategies. This highlights an important direction for future research: better understanding class semantics to enable more effective and interpretable augmentation methods.

8.8 Similarity evaluation

While the previous section evaluated synthetic data based on downstream classification performance, this section complements that analysis by assessing how well the synthetic data matches the whole training distribution, independent of class labels — that is, from the perspective of the unconditional distribution. We examine several complementary metrics: the discriminative score (real/fake distinguishability)

(Section 8.8.1), α -precision and β -recall (high-order sample-level fidelity and coverage) (Section 8.8.3), and feature-wise and pairwise distribution estimation (low-order distribution-level alignment) (Section 8.8.2). These metrics assess similarity without relying on class labels, providing a broader picture of how well the synthetic data captures the structure of the real data.

8.8.1 Realism via real/fake discrimination (discriminative score)

This subsection evaluates the realism of synthetic data by testing how well a classifier can distinguish real samples from synthetic ones. The evaluation targets the distribution $p(x)$ of the data itself, without considering class semantics. The question we aim to answer is: **How difficult is it to tell apart real data from synthetic data?**

Metric. We train an XGBoost model for binary classification to distinguish between the original and synthetic data in a supervised setting. The discriminative score is defined as $DS = accuracy - 0.5$ on the test set. A score close to 0 is better, indicating that the synthetic data is difficult to distinguish from the real data.

Results. In Table 8.12, we report the results of using XGBoost for binary classification to distinguish real from synthetic data. The discriminative scores vary significantly across generative methods, indicating this metric’s superior discriminative power. Moreover, the ranking of methods is consistent across all datasets, suggesting stable relative performance. Main insights are as follows:

1. The proposed DDPMS4 consistently outperforms all other generative models and hand-crafted augmentations across all datasets, and achieves notably low discriminative scores on small datasets. This indicates that DDPMS4-generated data closely aligns with the real data distribution and is particularly difficult to distinguish from real samples, highlighting its strong capacity for generating high-fidelity synthetic data.
2. Notably, ImagenTime-generated data is surprisingly easy to distinguish from real data, with discriminative scores approaching the worst possible value (0.5) across all datasets. However, its utility test score in downstream classification remains competitive, especially on small datasets and CESNET datasets. This suggests that while the overall distribution of the generated data deviates significantly from the real one, it still captures class-relevant features that are useful for classification. In fact, a classification model only needs to learn the conditional distribution $p(\text{label} \mid \text{input})$, not the full data distribution $p(\text{input})$. So, even if $p(\text{input})$ is not faithfully reproduced, as long as $p(\text{label} \mid \text{input})$ remains informative, the synthetic data can still be beneficial. For example, teaching a student to recognize cats using only AI-generated images — though slightly unnatural — still works if the images include key features like ears and whiskers, because the student only needs to learn to identify cats, not

to judge realism. Similarly, classifiers focus on label-relevant signals rather than authenticity. In summary, ImagenTime’s samples, while so statistically different from real data that a binary classifier can easily detect the distributional shift, preserve enough discriminative information to support classification tasks. From this perspective, the utility test and discriminative score are complementary, each capturing a different facet of data quality.

3. Among the three hand-crafted augmentations, Translate achieves the highest utility score and the lowest discriminative score, indicating relatively good alignment with real data. However, its discriminative score is still not close to zero, suggesting that it may not be as minimally altering as intended—possibly due to artifacts like zero padding. Gaussian, which introduces large-amplitude perturbations, performs the worst in both utility and discriminative metrics, often reaching a saturated score of 0.5. Wrap achieves slightly lower utility than Translate but still results in a high discriminative score. One possible explanation is that while Wrap may preserve some label-relevant features, it alters the temporal structure in a way that makes the data statistically easier to distinguish from real samples, similar to what we observe with ImagenTime. However, further analysis would be needed to confirm the exact source of this behavior.

The 3 hand-crafted augmentations have much higher discriminative scores than the best generative method, as expected — these transformations are designed to introduce detectable shifts that preserve label-relevant features, not to fool a class-agnostic discriminator. Among them, Translate achieves the best utility and lowest discriminative score, suggesting relatively good alignment with the real data distribution. Gaussian performs poor in both metrics. Wrap shows a contradictory behavior: its utility score is only slightly lower than Translate’s, yet it has a much higher discriminative score — a pattern also observed with ImagenTime. This may be due to temporal distortions introduced by Wrap that, while preserving label-relevant information, introduce statistical artifacts that make the data easier to distinguish from real samples.

8.8.2 Low-order distribution-level similarity (feature-wise density estimation and pairwise features correlation estimation)

This section focuses on evaluation metrics that consider individual features or feature pairs in isolation — specifically, feature-wise density estimation and pairwise correlation. Given that these metrics ignore higher-order dependencies across features, we aim to explore: **Are these low-order distribution-level statistics reliable for evaluating synthetic packet series quality?**

Metrics. For this analysis, we rely on metrics commonly adopted in the tabular data generation literature. This is motivated by the fact that a univariate packet

		Discriminative score ↓											
		Fake/ <i>generated</i>						Fake/ <i>hand-crafted</i>					
		DDPMS4	TabDDPM	ImagTime	Translate	Wrap	Gaussian						
UNSW24	*	6.70	1.22	11.84	0.73	49.87	0.06	32.84	0.47	50.00	0.00	49.92	0.04
Mirage22	*	6.59	1.18	23.42	0.77	45.52	0.18	26.90	0.36	49.95	0.06	47.32	0.31
UTMOBILENET21	*	12.33	1.08	22.33	0.61	49.37	0.20	32.23	0.79	50.00	0.00	49.31	0.14
CESNET-TLS22	50	13.25	0.18	26.41	0.50	46.94	0.18	28.91	0.05	50.00	0.00	47.65	0.21
	100	13.06	0.30	27.05	0.19	46.69	0.14	28.96	0.05	50.00	0.00	47.30	0.15
	*	12.72	0.31	26.22	0.30	47.08	0.09	28.92	0.08	49.88	0.00	47.40	0.32
CESNET-QUIC22	50	13.10	0.33	30.99	0.23	49.83	0.04	29.66	0.07	49.99	0.00	49.95	0.01
	*	12.95	0.17	31.65	0.30	49.85	0.01	29.57	0.08	49.99	0.00	49.94	0.01
Enterprise	50	9.50	0.41	21.59	0.38	49.30	0.05	31.26	0.09	50.00	0.00	49.70	0.03
	100	9.54	0.22	23.46	0.16	48.96	0.05	31.00	0.08	49.99	0.00	49.37	0.03
	300	10.03	0.39	22.41	0.17	48.79	0.04	30.71	0.07	50.00	0.00	49.15	0.04
	*	10.08	0.44	23.89	0.22	48.67	0.09	30.65	0.06	49.99	0.00	49.18	0.04
<i>avg across</i>		*10.82	2.43	24.27	4.90	48.41	1.41	30.13	1.60	49.98	0.04	48.85	1.06

TABLE 8.12: Discriminative score (%)

time series can be viewed as a table row, where each packet corresponds to a column. Specifically, we apply feature-wise density estimation, which assesses how closely the distribution of each single feature is matched, and pairwise features correlation estimation, which evaluates the similarity of linear correlations between feature pairs. Detailed formulations are given below.

- Feature-wise density estimation: to assess how well the synthetic data preserves the marginal distribution of individual continuous feature x , we use the Kolmogorov–Smirnov Test (KST). This test measures the distance between the real and synthetic distributions (denoted as $p_R(x)$ and $p_S(x)$ respectively), by first computing the Cumulative Distribution Function (CDF) for each distribution

$$F(x) = \int_{-\infty}^x p(x)dx$$

and then calculating the upper bound of the discrepancy between the CDFs of the real and synthetic data:

$$\text{KST} = \sup |F_R(x) - F_S(x)|$$

A smaller score indicates that the synthetic data more accurately captures the feature-wise (i.e., per-variable) distributional properties of the real data.

- Pairwise features correlation estimation: we use Pearson correlation coefficient to assess the linear correlation degree between a pair of continuous features. Given two continuous features x and y , it is defined as:

$$\rho_{x,y} = \frac{\text{Cov}(x,y)}{\sigma_x \sigma_y}$$

where Cov is the covariance, and σ is the standard deviation.

Then, the quality of correlation estimation is measured by averaging, over all

		Feature-wise density estimation \uparrow						Pairwise features correlation estimation \uparrow					
		Fake/generated			Fake/hand-crafted			Fake/generated			Fake/hand-crafted		
Dataset	Tier	DDPMS4	TabDDPM	ImagenTime	Translate	Wrap	Gaussian	DDPMS4	TabDDPM	ImagenTime	Translate	Wrap	Gaussian
UNSW24	*	84.93 _{0.20}	97.87 _{0.29}	86.02 _{0.15}	83.14 _{0.08}	76.91 _{0.03}	80.74 _{0.07}	99.23 _{0.09}	99.28 _{0.02}	99.38 _{0.10}	96.57 _{0.00}	96.60 _{0.06}	99.85 _{0.00}
Mirage22	*	96.30 _{0.07}	93.94 _{0.40}	96.32 _{0.06}	93.86 _{0.13}	91.05 _{0.16}	94.11 _{0.02}	99.48 _{0.03}	95.76 _{0.16}	99.29 _{0.05}	98.08 _{0.05}	97.62 _{0.06}	99.78 _{0.00}
UTMOBILENET21	*	97.86 _{0.08}	97.75 _{0.07}	86.55 _{0.64}	92.44 _{0.20}	85.75 _{0.04}	86.53 _{0.03}	99.12 _{0.08}	98.61 _{0.06}	99.20 _{0.11}	97.11 _{0.14}	95.45 _{0.06}	99.89 _{0.00}
CESNET-TLS22	50	92.16 _{0.01}	98.87 _{0.07}	93.55 _{0.02}	89.48 _{0.01}	88.44 _{0.00}	88.03 _{0.00}	99.92 _{0.01}	99.67 _{0.08}	99.80 _{0.00}	98.15 _{0.00}	98.37 _{0.00}	99.88 _{0.00}
	100	91.76 _{0.00}	98.88 _{0.10}	93.47 _{0.00}	89.21 _{0.00}	88.18 _{0.01}	87.89 _{0.01}	99.92 _{0.02}	99.72 _{0.09}	99.84 _{0.00}	98.12 _{0.00}	98.31 _{0.00}	99.88 _{0.00}
	*	91.74 _{0.00}	98.82 _{0.07}	93.74 _{0.02}	89.18 _{0.01}	88.17 _{0.00}	87.85 _{0.01}	99.91 _{0.00}	99.69 _{0.04}	99.82 _{0.01}	98.12 _{0.00}	98.32 _{0.00}	99.88 _{0.00}
CESNET-QUIC22	50	99.25 _{0.02}	98.61 _{0.09}	91.23 _{0.01}	96.11 _{0.01}	93.66 _{0.01}	89.05 _{0.01}	99.93 _{0.00}	99.71 _{0.04}	99.84 _{0.00}	98.53 _{0.00}	98.47 _{0.00}	99.91 _{0.00}
	*	99.28 _{0.03}	98.59 _{0.08}	92.24 _{0.00}	96.12 _{0.01}	93.69 _{0.01}	89.03 _{0.00}	99.93 _{0.00}	99.73 _{0.02}	99.84 _{0.00}	98.54 _{0.01}	98.49 _{0.00}	99.91 _{0.00}
	100	98.81 _{0.01}	99.08 _{0.03}	86.74 _{0.01}	93.94 _{0.00}	88.45 _{0.00}	84.68 _{0.00}	99.87 _{0.01}	99.76 _{0.04}	99.54 _{0.01}	96.61 _{0.00}	95.90 _{0.00}	99.90 _{0.00}
Enterprise	100	98.93 _{0.01}	98.98 _{0.06}	86.24 _{0.01}	94.03 _{0.00}	88.58 _{0.00}	84.48 _{0.00}	99.85 _{0.00}	99.66 _{0.07}	99.60 _{0.00}	96.89 _{0.00}	96.30 _{0.00}	99.90 _{0.00}
	300	98.99 _{0.01}	99.05 _{0.07}	86.50 _{0.01}	94.07 _{0.00}	88.64 _{0.00}	84.48 _{0.00}	99.86 _{0.01}	99.77 _{0.05}	99.65 _{0.00}	97.14 _{0.00}	96.69 _{0.00}	99.90 _{0.00}
	*	99.00 _{0.00}	98.99 _{0.02}	85.20 _{0.01}	94.07 _{0.00}	88.64 _{0.00}	84.46 _{0.00}	99.87 _{0.01}	99.74 _{0.05}	99.68 _{0.00}	97.17 _{0.00}	96.73 _{0.00}	99.90 _{0.00}
avg across		95.75 _{1.46}	98.29 _{1.40}	89.82 _{3.85}	92.14 _{3.64}	88.35 _{1.16}	86.78 _{3.27}	99.74 _{0.28}	99.26 _{1.12}	99.62 _{0.22}	97.59 _{0.72}	97.27 _{1.08}	99.88 _{0.03}

TABLE 8.13: Low-order distribution-level evaluation (%)

feature pairs (x, y) , the differences between the Pearson correlation coefficients in the real data $\rho^R(x, y)$ and the synthetic data $\rho^S(x, y)$:

$$\text{Pearson Score} = \frac{1}{2} \mathbb{E}_{x,y} \left| \rho^R(x, y) - \rho^S(x, y) \right|$$

Since $\rho \in [-1, 1]$, dividing the absolute difference by 2 ensures the final score lies in $[0, 1]$. A smaller score indicates that the synthetic data better preserves the correlation structure of the real data.

Results. Table 8.13 presents the results of low-level density estimation, including both feature-wise and feature-pair metrics. The main observations are as follows:

Although the metrics theoretically range from $[0, 1]$, all methods achieve a feature-wise density estimation score above 84% a pairwise feature correlation score above 95%, and almost no difference between the datasets tiers, indicating that these metrics are not sensitive enough to meaningfully distinguish between models.

Importantly, the feature-wise density score only evaluates how well each individual feature $p(x_i)$ is modeled, without accounting for interactions between features. In contrast, the discriminative score — obtained from a binary classifier trained on full packet sequences $x = (x_1, \dots, x_{30})$ — reflects how well the model captures the joint distribution $p(x)$, including inter-feature dependencies. While most methods score well on the feature-wise metric, their discriminative scores show substantial differences in how well they model the full sequence. This difference matters, because a model can match individual feature histograms yet still miss to preserve the relationships between features. Therefore, we focus our analysis on discriminative scores for distribution-level evaluation, rather than drawing conclusions from these lower-order density metrics that only capture individual or pairwise feature statistics and may overlook important structural differences.

8.8.3 High-order sample-level similarity (α -precision and β -recall)

This subsection evaluates whether synthetic data matches real training data at a sample level by measuring L2 distances of (real, synthetic) points pairs in the input space. The distances are then also used to compute α -precision and β -recall scores to assess fidelity and coverage. Overall, we aim to answer the following question: **Does the synthetic data match the real training data with respect to high-order sample-level metrics?**

Metrics. As outlined in (Alaa et al., 2022), the quality of synthetic data should be evaluated from two complementary perspectives: *fidelity*, which assesses how closely the generated samples resemble real data, and *coverage*, which evaluates whether the generated data cover all the modes of the real dataset. To this end, the authors propose α -Precision and β -Recall to measure fidelity and diversity respectively.

Given a synthetic dataset $S = \{x_j^s\}_{j \in [1, m]}$ and a real training dataset $R = \{x_i^r\}_{i \in [1, n]}$ (where $m = n^3$), the metrics are defined as following:

- α -Precision: First, we compute real data's empirical center: $\mu^r = \frac{1}{n} \sum_{i=1}^n x_i^r$, and the Euclidean distances from each real point to the real center μ^r : $r_i^r = \|x_i^r - \mu^r\|_2$. For each quantile level $\alpha \in [0, 1]$, we compute the radius $R_\alpha^r = \text{Quantile}_\alpha(\{r_i^r\}_{i=1}^n)$. This means that the Euclidean ball $B_\alpha^r = \{x \in \mathbb{R}^d \mid \|x - \mu^r\|_2 \leq R_\alpha^r\}$ contains approximately an α proportion of the real data points, i.e., $\mathbb{P}(x \in B_\alpha^r) = \alpha$. Intuitively, this region B_α^r centered at μ^r with radius R_α^r can be interpreted as the α -support of the real data distribution, encompassing the most "typical"/"normal" real samples, while real points outside B_α^r are considered "outliers"; the parameter α controls the fraction of data considered "normal."

Then, α -precision at level α is defined as the proportion of synthetic points that lie within α -support region B_α^r of the real data distribution:

$$P(\alpha) = \frac{1}{m} \sum_{j=1}^m \mathbf{1}(\|x_j^s - \mu^r\|_2 \leq R_\alpha^r).$$

By conditioning on α , this metric deems a synthetic sample to be of a high fidelity not only if it looks "realistic", but also if it looks "typical". If the synthetic and real densities are equal $P^s = P^r$, we would expect $P(\alpha) = \alpha$ for all $\alpha \in [0, 1]$.

³In practice, both datasets are upper-bounded by 15,000 samples to limit computational cost. For hand-crafted augmentations, the down-sampling indices for the real training set and the augmented set are kept identical to preserve the one-to-one correspondence between each real sample and its augmented version. For generative models, since samples are generated from scratch rather than derived from real inputs, there is no one-to-one correspondence between real and synthetic data. Therefore, the real and synthetic datasets are down-sampled independently.

- β -Recall: As a preliminary step, For each real sample x_i^r , we find its nearest synthetic neighbor $x_{j_i^*}^s$ among S , and its nearest real neighbor $x_{k_i^*}^r$ among $R \setminus \{x_i^r\}$. Denote their respective distances to the real anchor x_i^r as $d_i^r = \|x_i^r - x_{k_i^*}^r\|_2$, $d_i^s = \|x_i^r - x_{j_i^*}^s\|_2$.

First, we compute synthetic data's empirical center: $\mu^s = \frac{1}{m} \sum_{j=1}^m x_j^s$, and the Euclidean distances from each nearest synthetic neighbor $x_{j_i^*}^s$ to the synthetic center μ^s : $r_i^s = \|x_{j_i^*}^s - \mu^s\|_2$. For each quantile level $\beta \in [0, 1]$, we compute the radius $R_\beta^s = \text{Quantile}_\beta(\{r_i^s\}_{i=1}^n)$.

Then, β -recall at level β is defined as the proportion of real points for which: (i) its nearest synthetic neighbor is closer than its nearest real neighbor, and (ii) its nearest synthetic neighbor is within the β -support region of the synthetic distribution:

$$R(\beta) = \frac{1}{n} \sum_{i=1}^n \mathbf{1} \left(d_i^s \leq d_i^r \wedge r_i^s \leq R_\beta^s \right).$$

The first condition ensures that the synthetic data offers a closer approximation to the real point x_i^r than the real data itself, reflecting strong local fidelity. The second condition ensures that this nearest synthetic neighbor lies within the β -support of the synthetic distribution, meaning it belongs to a region where synthetic samples "typically" concentrate. Together, these conditions ensure that x_i^r is not only covered by a synthetic point, but that such coverage is provided by a "typical"/"representative" synthetic sample. In this sense, β -recall at level β reflects the proportion of real samples effectively captured within the β -support of the generative distribution. If the synthetic and real densities are equal $P^s = P^r$, we would expect $R(\beta) = \beta$ for all $\beta \in [0, 1]$.

While $P(\alpha)$ and $R(\beta)$ are functions of α and β , respectively, and these curves reveal fine-grained information about a model's fidelity and coverage, it is often more convenient to summarize performance using a single number. To this end, Alaa et al., 2022 quantifies the deviation from ideal precision and recall using the mean absolute deviation of each curve from the identity line:

$$\Delta P(\alpha) = \int_0^1 |P(\alpha) - \alpha| d\alpha, \quad \Delta R(\beta) = \int_0^1 |R(\beta) - \beta| d\beta,$$

Since $\Delta P(\alpha)$ and $\Delta R(\beta)$ lie in the range $[0, \frac{1}{2}]$, they are rescaled to the range $[0, 1]$ by defining

$$\text{IP}_\alpha = 1 - 2\Delta P(\alpha), \quad \text{IR}_\beta = 1 - 2\Delta R(\beta),$$

with higher values indicating better alignment between the generative and real distributions. If $P_g = P_r$, $\text{IP}_\alpha = \text{IR}_\beta = 1$.

Results. In Table 8.14, we report the scores of α -precision and β -recall. In Table 8.14, we present the fine-grained curves of α -precision and β -recall on small and large datasets.

The key findings are as follows:

		α -precision \uparrow								β -recall \uparrow							
		Ref/real		Fake/generated		Fake/hand-crafted				Ref/real		Fake/generated		Fake/hand-crafted			
Dataset	Tier	XGB	DDPMS4	TabDDPM	ImagTime	Translate	Wrap	Gaussian	XGB	DDPMS4	TabDDPM	ImagTime	Translate	Wrap	Gaussian		
UNSW24 Mirage22 UTMOBILENET21	*	97.06 _{0.40}	97.27 _{0.49}	96.74 _{0.56}	99.02 _{0.22}	77.51 _{0.25}	75.74 _{0.53}	99.00 _{0.02}	51.89 _{5.33}	61.85 _{0.43}	55.25 _{0.71}	42.07 _{0.83}	30.60 _{0.58}	5.65 _{0.22}	48.22 _{0.28}		
	*	96.94 _{1.71}	95.18 _{0.52}	94.35 _{0.35}	98.64 _{0.33}	98.28 _{0.06}	97.53 _{0.09}	98.36 _{0.02}	54.10 _{2.30}	69.69 _{0.32}	66.05 _{0.43}	48.54 _{0.47}	45.12 _{0.70}	22.30 _{0.46}	59.96 _{0.32}		
	*	96.84 _{0.64}	97.02 _{1.21}	97.15 _{0.62}	98.90 _{0.11}	97.87 _{0.08}	95.11 _{0.15}	98.97 _{0.05}	59.11 _{1.45}	61.00 _{0.11}	54.76 _{0.80}	38.72 _{0.36}	24.74 _{0.47}	8.92 _{0.29}	45.95 _{0.62}		
CESNET-TLS22	50	99.57 _{0.16}	98.79 _{0.78}	99.33 _{0.22}	97.30 _{0.31}	98.25 _{0.38}	97.70 _{0.22}	99.58 _{0.03}	50.18 _{0.66}	47.14 _{0.65}	43.37 _{0.55}	44.72 _{0.20}	49.03 _{0.57}	18.34 _{0.32}	75.50 _{0.43}		
	100	99.26 _{0.32}	99.11 _{0.28}	99.28 _{0.21}	97.19 _{0.60}	98.25 _{0.12}	97.65 _{0.13}	99.59 _{0.05}	50.59 _{0.23}	46.03 _{0.44}	43.47 _{0.27}	44.14 _{0.21}	49.24 _{0.36}	18.84 _{0.22}	75.91 _{0.20}		
	*	99.38 _{0.03}	98.74 _{0.86}	98.28 _{0.54}	98.02 _{0.56}	98.37 _{0.01}	97.71 _{0.21}	99.53 _{0.03}	50.00 _{0.81}	46.30 _{0.09}	43.68 _{0.24}	43.86 _{0.28}	49.40 _{0.13}	18.57 _{0.31}	76.98 _{0.91}		
CESNET-QUIC22	50	99.34 _{0.15}	98.02 _{0.24}	99.10 _{0.31}	98.24 _{0.64}	98.12 _{0.12}	98.81 _{0.11}	99.11 _{0.06}	49.40 _{0.52}	47.86 _{0.49}	44.96 _{0.87}	45.78 _{0.42}	48.07 _{0.21}	24.94 _{0.81}	86.97 _{0.19}		
	*	98.61 _{0.60}	98.51 _{0.59}	98.48 _{0.77}	98.36 _{0.52}	98.19 _{0.06}	98.53 _{0.34}	99.13 _{0.06}	48.89 _{0.59}	47.39 _{0.28}	44.57 _{0.90}	46.31 _{0.23}	48.20 _{0.44}	24.73 _{0.85}	87.81 _{0.23}		
	50	99.23 _{0.20}	99.30 _{0.36}	99.20 _{0.39}	98.98 _{0.34}	98.24 _{0.18}	95.64 _{0.10}	98.37 _{0.04}	53.57 _{0.49}	47.88 _{0.33}	42.42 _{0.24}	43.29 _{0.63}	39.07 _{0.39}	15.35 _{0.33}	62.10 _{0.76}		
Enterprise	100	99.48 _{0.08}	98.92 _{0.44}	99.32 _{0.10}	99.05 _{0.77}	98.44 _{0.26}	95.33 _{0.50}	98.29 _{0.04}	52.57 _{0.27}	47.38 _{0.49}	41.22 _{0.61}	43.58 _{0.36}	40.59 _{0.72}	16.24 _{0.06}	64.84 _{0.10}		
	300	99.20 _{0.49}	98.96 _{0.48}	99.21 _{0.10}	98.94 _{0.24}	98.15 _{0.13}	95.38 _{0.24}	98.38 _{0.06}	52.11 _{0.27}	48.03 _{0.67}	43.04 _{0.55}	44.59 _{0.33}	40.94 _{0.16}	17.30 _{0.08}	66.87 _{0.53}		
	*	99.02 _{0.15}	99.39 _{0.19}	99.29 _{0.04}	98.90 _{0.14}	98.37 _{0.32}	95.27 _{0.21}	98.39 _{0.06}	52.83 _{0.35}	48.08 _{0.47}	41.20 _{0.36}	44.50 _{0.38}	42.00 _{0.20}	17.30 _{0.60}	67.84 _{0.45}		
avg across		98.66 _{1.15}	98.27 _{1.29}	98.31 _{1.52}	98.46 _{0.74}	96.50 _{5.81}	95.03 _{6.05}	98.89 _{0.50}	52.10 _{3.06}	51.55 _{7.69}	47.00 _{7.43}	44.17 _{2.34}	42.25 _{7.65}	17.38 _{5.53}	68.25 _{12.90}		

TABLE 8.14: High-order sample-level evaluation (%)

1. All the generative models have almost saturated α -precision, with only subtle differences across methods and datasets. This indicates that the generated samples consistently exhibit high fidelity, i.e., they tend to lie within the high-density regions of the real data distribution, as measured by α -precision using quantile-based distance thresholds from the real data centroid.
2. Across all methods and datasets, β -recall is substantially lower than α -precision, indicating that while synthetic samples tend to lie in high-density regions of the real data distribution, they do not sufficiently cover the full support of the real distribution. In other words, the generative models produce data that looks realistic but fail to capture the full diversity of the real dataset.
3. Different generative models vary more significantly in β -recall than in α -precision, highlighting that coverage is a more discriminative property across models. Among them, DDPMS4 consistently achieves the highest β -recall scores across all datasets. Given that broad coverage of the real distribution is a crucial factor for improving downstream classification utility, this provides a compelling explanation of why DDPMS4 also achieves the highest utility scores.
4. On the other hand, we observe some inconsistencies for hand-crafted augmentations. For example, Translate on Enterprise datasets, despite achieving slightly lower α -precision and significantly lower β -recall than DDPMS4, consistently outperforms DDPMS4 on both the utility test and augmentation test. Gaussian, however, despite achieving the highest β -recall on large datasets, has the worst classification utility among all methods. This indicates that α -precision and β -recall do not capture the full picture of generalization. In particular, these metrics do not penalize methods that create near-duplicates of training data by adding small noise, and they do not care whether the noise preserves class semantics or not. In addition, the considered metrics do

not necessarily reward methods that generate novel, class-consistent examples that deviate from the original data and introduce additional variety but still preserve semantic meaning.

Fundamentally, both α -precision and β -recall are class-agnostic: they operate purely on the geometry of the data space without considering class labels. As a result, they fail to detect whether generated samples introduce inter-class confusion or reinforce class-separability. Overall, these limitations help explain why Translate, despite poor distributional alignment, achieves higher utility: it introduces task-relevant, label-preserving variation that improves classifier generalization. Conversely, they also help explain why Gaussian, despite achieving high β -recall by broadly covering the data space, performs poorly in classification: it perturbs real samples in a way that can easily distort class semantics and increase ambiguity near decision boundaries.

These findings highlight the need for evaluation metrics that go beyond geometric fidelity and distributional coverage — ones that explicitly account for semantic consistency and task relevance, especially in class-conditional settings.

Extended analysis. In β -recall, each real training sample is matched to its nearest synthetic sample. Here, we do the opposite: for up to 10,000 synthetic samples, we take each synthetic sample as a query and find its nearest neighbor from the entire real training set, recording both the distance and the index of the selected real sample. We then plot the Cumulative Distribution Function (CDF) of these distances, and the CDF of how often each real sample is selected as the nearest neighbor.

We use two small datasets (UTMOBILENET21 and Mirage22) and three large ones (CESNET-QUIC22-50, CESNET-TLS22-50, and Enterprise-50). Results are shown in [Figure 8.11](#). As a reference, we also show the distribution obtained using real test samples as queries instead of synthetic ones.

We highlight the following key findings:

1. Among the hand-crafted transformations, Gaussian amplitude perturbation yields distances mostly in the range $[0.01, 0.03]$, consistent with its high β -recall score. However, its poor performance in the utility test suggests that adding i.i.d. Gaussian noise to each feature — despite a small overall L2 difference — breaks inter-feature dependencies and disrupts the joint feature distribution. Furthermore, since XGBoost relies on threshold-based splits, such small perturbations can lead the model to learn decision paths that fail to generalize to clean test data, resulting in misclassification.

On the other hand, for Translate, its distance CDF consistently lies below the reference CDF of the real test data, indicating that it introduces noticeable differences, yet it achieves the best performance in the utility test. Combined with

the close CDF and poor utility score of Gaussian, this suggests that smaller L2 distances do not necessarily imply preserved class semantics, and larger distances do not imply semantic disruption. Overall, input-space L2 distance captures structural similarity but fails to reflect whether the underlying class information is retained — a conclusion consistent with α -precision and β -recall results.

2. For DDPMS4 on small datasets, although its synthetic samples are very close to their nearest real training neighbors, around 50% of the real training samples are selected more than once as nearest neighbors. Since the number of synthetic samples matches the small real training set, this means roughly half of the real samples are never selected. This pattern indicates that the model tends to overfit to easy modes in the data while missing to capture harder or less frequent ones.

On large datasets, the distance distribution shows that DDPMS4-generated samples are generally farther from real training samples compared to real test samples, as reflected by the DDPMS4 CDF curve lying below that of the real test set. However, DDPMS4 still performs best among all methods, as shown by its CDF being the leftmost. Regarding real index frequency, nearly all real training samples are selected only once as nearest neighbors, suggesting there is no significant overfitting when the dataset is large. Instead, the main issue appears to be underfitting, likely due to the greater complexity of the data.

8.8.4 Concluding remarks

Capturing the full picture of synthetic data quality requires multiple and diverse metrics, including fidelity, diversity, and semantic utility. This section complemented the class-conditional utility evaluation by assessing the quality of unconditional distribution modeling—that is, how well the generated data aligns with the overall real data distribution, regardless of class labels. Using multiple metrics from tabular data synthesis literature, the analysis reveals several key insights.

As a starting point, we highlight two important aspects: (1) unconditional fidelity is not equivalent to conditional fidelity — a dataset can be globally unrealistic yet still useful for classification if it preserves label-relevant semantics; (2) unconditional distribution modeling relates to both *fidelity*, which measures how closely synthetic samples resemble real ones, and *coverage*, which assesses how well the synthetic data captures all the diversity of the real training set.

Capturing these two aspects requires multiple complementary metrics, yet it is not very common to find prior works providing a comprehensive view, especially in networking literature. Specifically, point (1) is assessed via the discriminative score, which captures global distinguishability, while point (2) is evaluated using sample-level similarity metrics such as α -precision and β -recall. Regarding (1), we observe

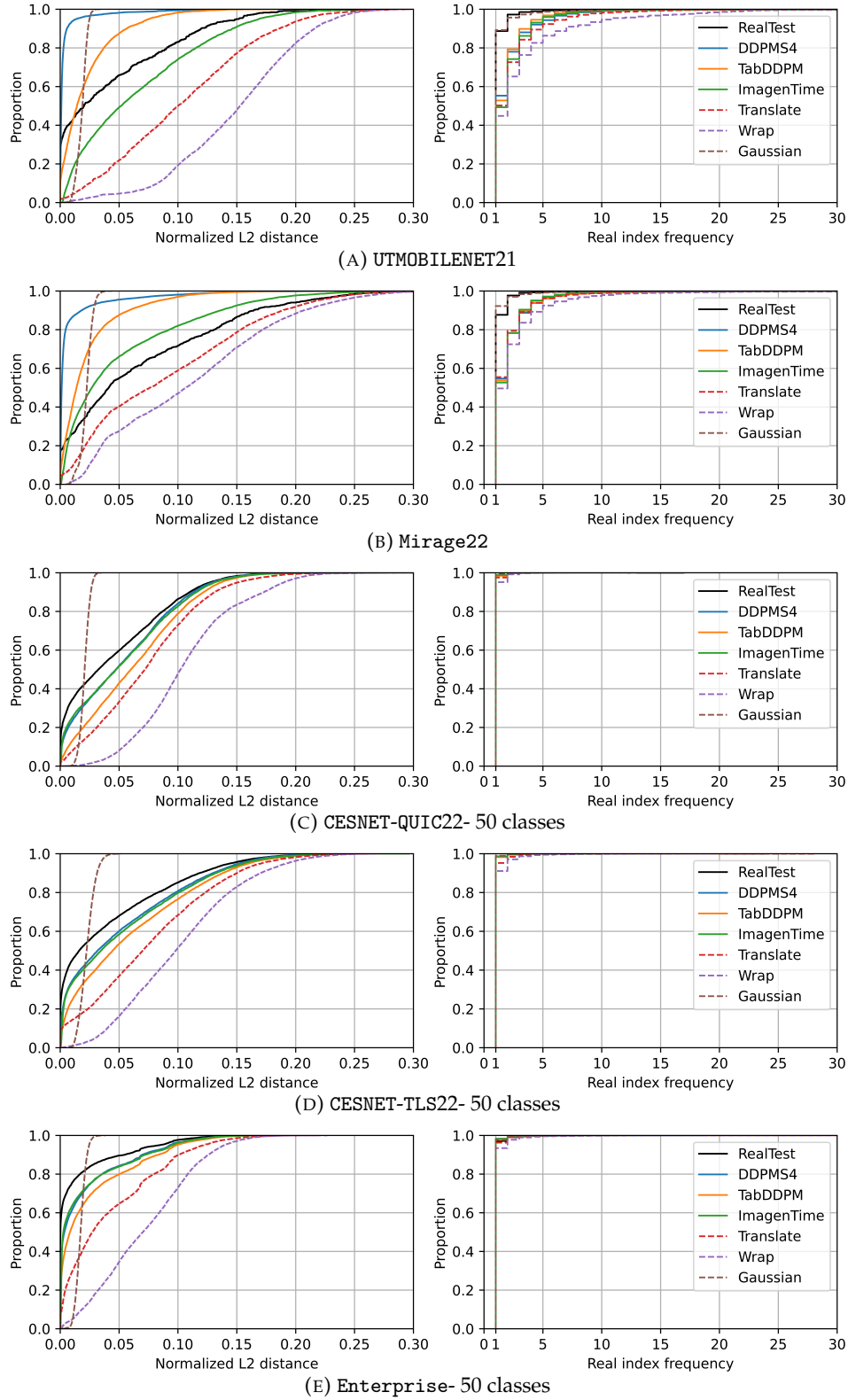


FIGURE 8.11: Closest real training record statistic.

that methods like ImagenTime and Wrap perform poorly in discriminative evaluation yet effectively support classification, indicating they retain class-relevant features despite global distributional differences. Similarly, Translate produces synthetic samples that are distant from real data in input space but achieve the best performance in utility and augmentation tests. This further shows that L2 distance in

input space does not reliably reflect semantic preservation. Regarding (2), DDPMS4-generated data achieves around 98% α -precision but 52% β -recall, suggesting it captures well high-density/easy-modes, while missing to cover low-density/harder-modes. This gap in coverage helps explaining the utility test gap of synthetic versus real training data.

Still, among the three generative models, DDPMS4 consistently performs best across all similarity metrics on all datasets. Meanwhile, its β -recall scores on large datasets are noticeably lower than those on small datasets, indicating that further improvement is needed in the likelihood estimation to better fully cover more complex packet series distributions.

Finally, low-order distribution-level metrics commonly used in tabular data synthesis often saturate and do not capture the full-sequence structure of packet series, making them inadequate for our setting.

Chapter 9

Discussion and future work

In [Chapter 8](#), aiming toward automated DA for TC, we investigated generative modeling of packet series. Specifically, we adopted a discrete-time DDPM framework and designed a Transformer-based architecture where attention is replaced by S4 to model temporal dependencies in univariate time series. The resulting model, DDPMS4, was evaluated on multiple TC datasets with varying sizes and class imbalance degrees. Evaluation covered classification utility, augmentation effectiveness, and class-agnostic fidelity and coverage. While DDPMS4 outperforms all other generative baselines, its utility and augmentation scores remain inferior to the best-performing hand-crafted method, Translate. By using the multi-facet evaluation for probing, we identify the following key limitations:

1. Collapse error: high fidelity but limited coverage of low-density regions

The generated data appears similar to real samples but covers only a limited portion of the real distribution, concentrating on high-density regions while missing low-density ones. Here, the missing low-density regions refer not only to general less frequent modes such as points near class boundaries, but also to minority classes, as indicated by the larger gap in macro F1 compared to weighted F1 across all datasets.

Furthermore, on small datasets, we observe signs of memorization, suggesting the generative model overfits to high-density regions. This aligns with prior findings on DMs, which generally achieve better mode coverage than GANs by explicitly modeling the full data distribution through iterative denoising but may still exhibit imperfect coverage in practice, especially when model capacity is large relative to the data size (Karras et al., [2022](#)). Additionally, Biroli et al., [2024](#) theoretically identifies three regimes in diffusion model’s reverse process: from white noise, to class separation, and last collapse to memorization. Specifically, if the score function is estimated perfectly, the model will collapse to a training point at late timesteps, meaning it can only memorize training samples.

2. Additional meaningful variety for augmentation does not come for free

At first glance, one might expect that a well-trained diffusion model, due to its inherent stochasticity, would naturally provide useful data augmentation. However, our experiments show that even our best-performing diffusion model fails to improve downstream classification performance. Why does this happen? According to information theory, the information a generative model can learn is fundamentally limited by the information contained in its training data. Therefore, even in an ideal case where the model learns the data distribution perfectly without collapse, it can merely memorize the training set, but cannot introduce genuinely new information beyond what is already present in the real training data.

This raises a natural question: why are generative models widely used for DA in traditional ML domains like CV? First, diffusion models in vision are often pretrained on internet-scale datasets (e.g., LAION), giving them access to knowledge far beyond the target classification task. Second, in images, class-relevant and irrelevant patterns are easier to distinguish, enabling the design of prompt-based augmentations with pretrained text-to-image models. In contrast, for packet series, we lack both large-scale pretrained models and a clear understanding of class-semantic subsequence patterns, making effective generative augmentation much more challenging.

With the key problems now clarified, new opportunities and challenges emerge for future research. We highlight several of them below.

First, having a broader coverage is essential. For larger coverage, we could:

1. Augment the training data size via hand-crafted DA:

Increasing the training set can help mitigate overfitting and collapse errors. One approach is to apply hand-crafted augmentations, such as the Translate function, which has proven effective. Aiming to mimic the transformation, we trained a diffusion model solely on translated data. However, this led to higher training loss and reduced downstream utility, suggesting that the transformed distribution is more complex and harder to model, thereby reducing fidelity. To address this, we propose providing augmentation parameters (e.g., start index, translate direction, translate steps) as additional conditioning inputs to the NN. This enriches the conditioning context, helping the model better capture the structure of the augmented data.

2. More conditioning context via imputation:

Another way to improve coverage is through imputation. Rather than always modeling the entire time series given only the class label, we could train the model to reconstruct the missing part of a sequence conditioned on the class label, the observed part, and a binary mask. This shifts the task to modeling

$p(\text{missing part}|\text{class label, observed part, mask})$, which provides more informative context and makes distribution to learn easier, as our preliminary results showed that it can improve utility test performance. In particular, during generation, real training samples from low-density regions can be combined with randomly selected masks to define the observed part. This provides meaningful context, enabling the model to generate more accurate completions of rare or underrepresented patterns.

3. Reinforcement fine-tuning with coverage-based rewards:

Improving coverage through reinforcement fine-tuning requires a reward signal that promotes coverage among generated samples. Since coverage r_D is evaluated over a generated set \mathbf{X}_g relative to real data set \mathbf{X}_r , assigning rewards to individual generated samples is nontrivial. To assign per-sample rewards from a set-level coverage metric, monotonicity of r_D is required. Under this condition, the marginal contribution of each generated sample $x_g \in \mathbf{X}_g$ can be defined as $\hat{r}_D(x_g; \mathbf{X}_r) = r_D(\mathbf{X}_g; \mathbf{X}_r) - r_D(\mathbf{X}_g \setminus \{x_g\}; \mathbf{X}_r)$. For computation efficiency, r_D could be analytical metrics such as Maximum Mean Discrepancy (MMD) (Miao et al., 2024), β -coverage, or mutual information, which support efficient and scalable fine-grained rewards.

Then, to further improve sampling efficiency for DA, it is desirable to guide generation toward low-density regions, where the downstream classifier is more likely to struggle. In contrast, passively following the original distribution tends to produce samples mostly fall into high-density areas. To address this, we can apply guidance in the reverse process to push generation toward low-density regions. The central question is: what kind of guidance can effectively achieve this?

At the current reverse step t , after predicting an estimate \hat{x}_0 from x_t , we can feed \hat{x}_0 into either a pre-trained classifier or back into the diffusion model to assess its likelihood. This helps identify whether \hat{x}_0 lies in a high- or low-likelihood region, and several strategies can be used:

1. High cross-entropy based on the classifier's output: if \hat{x}_0 leads to a high cross-entropy loss when passed through a pre-trained classifier, it likely lies in a region where the classifier is uncertain or prone to errors.
2. Distance from class prototypes in the classifier's embedding space: samples that lie far from class prototypes tend to be ambiguous and harder to classify, making them valuable for augmentation.
3. Low likelihood based on noise estimation error of the diffusion model: diffuse \hat{x}_0 to a noisy version x_s , then let the diffusion model predict the added noise. A large noise estimation error suggests that the sample lies in a low-likelihood region of the learned data distribution.

4. Low likelihood measured by re-denoising distance in the diffusion model: compute the distance between $\hat{x}_0(x_t)$ and the re-denoised output after strong noise perturbation to a large timestep s (e.g. $s = 0.8T$) (Um and Ye, 2024; Zhang and Zou, 2024): $d(\hat{x}_0(x_t), \text{sg}(\hat{x}_0(\hat{x}_s(x_t))))$. A smaller distance means the model can still recover the original sample after adding strong noise, suggesting the sample comes from a region the model knows well, i.e., a high-likelihood region. A larger distance means the model struggles to recover the sample, indicating it lies in a region the model hasn't learned well, i.e., a low-likelihood region.

Still, selecting guidance hyperparameters is a challenge, as they should be effective while adhering to the manifold of $p(x_t)$.

Overall, these directions open new possibilities for improving generative data augmentation in TC. Through richer conditioning, reinforcement-based objectives, or principled guidance, future work can achieve better coverage and enhance the effectiveness of generative models for downstream use.

Appendix A

Appendix for Chapter 4

A.1 Layout of DL Network Architectures

We list here our implementation of the network architectures for the 32×32 flowpic resolution (see also Fig.7 in Horowicz et al., 2022). The Listings A.1-A.5 are obtained via the torchsummary python package. For code flexibility, our architectures are designed to use Pytorch `nn.Identity()` modules to mask out layers that are not needed from a given architecture. When this masking is applied, our training framework takes care of recreating the network optimizers to reflect the architecture modifications.

LISTING A.1: Supervised network (with dropout).

```
flowpic_dim: 32
num_classes: 5
with_dropout: True
-----
Layer (type)          Output Shape          Param #
-----
Conv2d-1              [-1, 6, 28, 28]       156
ReLU-2                [-1, 6, 28, 28]       0
MaxPool2d-3           [-1, 6, 14, 14]       0
Conv2d-4              [-1, 16, 10, 10]      2,416
ReLU-5               [-1, 16, 10, 10]      0
Dropout2d-6           [-1, 16, 10, 10]      0
MaxPool2d-7           [-1, 16, 5, 5]        0
Flatten-8             [-1, 400]             0
Linear-9              [-1, 120]             48,120
ReLU-10              [-1, 120]             0
Linear-11             [-1, 84]             10,164
ReLU-12              [-1, 84]             0
Dropout1d-13          [-1, 84]             0
Linear-14             [-1, 5]              425
-----
Total params: 61,281
Trainable params: 61,281
Non-trainable params: 0
-----
Input size (MB): 0.00
Forward/backward pass size (MB): 0.13
Params size (MB): 0.23
Estimated Total Size (MB): 0.36
-----
```

LISTING A.2: Supervised network (without dropout).

```
flowpic_dim: 32
num_classes: 5
with_dropout: False
-----
Layer (type)          Output Shape          Param #
-----
Conv2d-1              [-1, 6, 28, 28]       156
ReLU-2                [-1, 6, 28, 28]       0
MaxPool2d-3           [-1, 6, 14, 14]       0
Conv2d-4              [-1, 16, 10, 10]      2,416
ReLU-5               [-1, 16, 10, 10]      0
Identity-6            [-1, 16, 10, 10]      0 <-- masked
MaxPool2d-7           [-1, 16, 5, 5]        0
Flatten-8             [-1, 400]             0
Linear-9              [-1, 120]             48,120
ReLU-10              [-1, 120]             0
Linear-11             [-1, 84]             10,164
ReLU-12              [-1, 84]             0
Identity-13           [-1, 84]             0 <-- masked
Linear-14             [-1, 5]              425
-----
```

```

=====
Total params: 61,281
Trainable params: 61,281
Non-trainable params: 0
=====
Input size (MB): 0.00
Forward/backward pass size (MB): 0.13
Params size (MB): 0.23
Estimated Total Size (MB): 0.36
=====

```

LISTING A.3: SimCLR pre-train (small projection layer).

```

flowpic_dim: 32
num_classes: 5,
projection_layer_dim: 30
with_dropout: False
=====
Layer (type)      Output Shape      Param #
=====
Conv2d-1          [-1, 6, 28, 28]   156
ReLU-2            [-1, 6, 28, 28]   0
MaxPool2d-3       [-1, 6, 14, 14]   0
Conv2d-4          [-1, 16, 10, 10]  2,416
ReLU-5            [-1, 16, 10, 10]  0
Identity-6        [-1, 16, 10, 10]  0
MaxPool2d-7       [-1, 16, 5, 5]    0
Flatten-8         [-1, 400]          0
Linear-9          [-1, 120]          48,120
ReLU-10           [-1, 120]          0
Linear-11         [-1, 120]          14,520 <- proj layer 1
ReLU-12           [-1, 120]          0
Identity-13       [-1, 120]          0
Linear-14         [-1, 30]           3,630 <- smaller proj layer
=====
Total params: 68,842
Trainable params: 68,842
Non-trainable params: 0
=====
Input size (MB): 0.00
Forward/backward pass size (MB): 0.13
Params size (MB): 0.26
Estimated Total Size (MB): 0.39
=====

```

LISTING A.4: SimCLR pre-train (large projection layer).

```

flowpic_dim: 32
num_classes: 5,
projection_layer_dim: 84
with_dropout: False
=====
Layer (type)      Output Shape      Param #
=====
Conv2d-1          [-1, 6, 28, 28]   156
ReLU-2            [-1, 6, 28, 28]   0
MaxPool2d-3       [-1, 6, 14, 14]   0
Conv2d-4          [-1, 16, 10, 10]  2,416
ReLU-5            [-1, 16, 10, 10]  0
Identity-6        [-1, 16, 10, 10]  0
MaxPool2d-7       [-1, 16, 5, 5]    0
Flatten-8         [-1, 400]          0
Linear-9          [-1, 120]          48,120
ReLU-10           [-1, 120]          0
Linear-11         [-1, 120]          14,520 <- proj layer 1
ReLU-12           [-1, 120]          0
Identity-13       [-1, 120]          0
Linear-14         [-1, 84]           10,164 <- larger proj layer
=====
Total params: 75,376
Trainable params: 75,376
Non-trainable params: 0
=====
Input size (MB): 0.00
Forward/backward pass size (MB): 0.13
Params size (MB): 0.29
Estimated Total Size (MB): 0.42
=====

```

LISTING A.5: Fine-tune network.

```

flowpic_dim: 32
num_classes: 5,
projection_layer_dim: 30
with_dropout: False
=====
Layer (type)      Output Shape      Param #
=====
Conv2d-1          [-1, 6, 28, 28]   156
ReLU-2            [-1, 6, 28, 28]   0
MaxPool2d-3       [-1, 6, 14, 14]   0
Conv2d-4          [-1, 16, 10, 10]  2,416
ReLU-5            [-1, 16, 10, 10]  0
Identity-6        [-1, 16, 10, 10]  0
MaxPool2d-7       [-1, 16, 5, 5]    0
Flatten-8         [-1, 400]          0
Linear-9          [-1, 120]          48,120
ReLU-10           [-1, 120]          0
Identity-11       [-1, 120]          0 <- masked
Identity-12       [-1, 120]          0 <- masked
Identity-13       [-1, 120]          0 <- masked
Linear-14         [-1, 5]            605 <- final classifier
=====
Total params: 51,297

```

```
Trainable params: 51,297
Non-trainable params: 0
-----
Input size (MB): 0.00
Forward/backward pass size (MB): 0.13
Params size (MB): 0.20
Estimated Total Size (MB): 0.33
-----
```


Appendix B

Appendix for Chapter 7

B.1 Details on user study

Our user studies are based on small focus groups participants only, with (lightly) guided discussions led by a moderator. In those campaigns we elicited feedback from users regarding the comparison of different alignment metrics, aiming to understand if MI is a *plausible* choice. Although launching large-scale survey campaigns would be desirable, this would require a completely different organization and implementation with respect to what we adopted for this work.

The survey web app. Beside *punctually* comparing alignment metrics [Section 7.2](#) and methods [Section 7.5](#), we designed a web app to collect *subjective* feedback, in the form of mini surveys, from real users. Each survey is composed of multiple tests, each showing a prompt and a set of images generated from it. Under the hood, the web app corresponds to a jupyter notebook with `ipywidgets`¹ for UI controls, rendered via the `voila`² framework and deployed live via a docker-ized HuggingFace space. Via the web app we run campaigns to *compare alignment metrics* and to *compare alignment methods*.

[Figure B.1](#) shows an example screenshot of the alignment metric comparison survey [Section 7.2](#). As from the example, users are free to select from 0 to up to 3 images for each prompt. However, to stress users subjectivity, we intentionally did not provide guidelines on how to handle “odd” cases (e.g., if the prompt asks for a picture of “an apple”, but the picture show more than one apple). Last, each survey is saved as a separate CSV with the timestamp of its creation which also serves as unique identifier of the survey, i.e., neither a user identifier nor cookies are required by the web app logic, so users’ privacy and anonymity is preserved.

B.1.1 Comparing alignment metrics

In the first surveys campaign we aimed to understand how users perceive images pre-selected by BLIP-VQA, HPS and MI. Specifically, we run surveys composed of 10

¹<https://ipywidgets.readthedocs.io/en/stable/>

²<https://voila.readthedocs.io/en/stable/using.html>

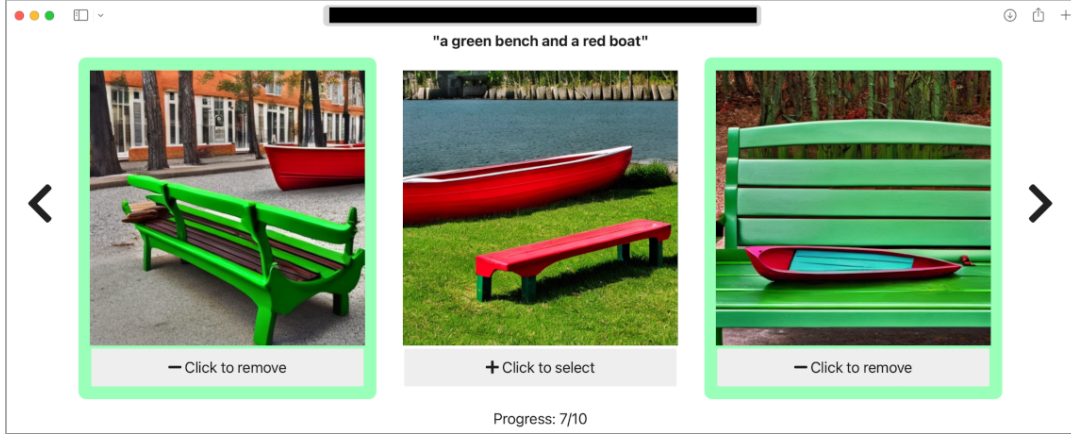


FIGURE B.1: Web app screenshot example of the alignment metric comparison survey.

Metric			Campaign answers (%)			
MI	BLIP-VQA	HPS	Academic users	Random users	Students	avg
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	14.7	16.9	25.0	18.9
<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	1.8	14.0	2.7	6.2
<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	10.4	22.0	4.1	12.2
<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	4.0	7.4	3.6	5.0
<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	4.0	10.6	0.9	5.2
<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	6.0	2.5	2.3	3.6
<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	18.7	10.6	16.8	15.4
<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	40.4	16.0	44.6	33.7
<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	69.1	39.7	64.6	57.8
<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	73.5	56.0	69.1	66.2
<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	52.2	39.9	53.2	48.2

☒ (selected) ☐ (not selected) ☐ (indifferent to the selection)

TABLE B.1: User study about comparing alignment metrics.

tests, each showing a prompt and the related best image among 50 generations (using SD-2.1-base) as ranked according to each metric separately (Section 7.2). Each of the 10 prompts is randomly selected from a pool of 700 prompts for the T2I-Combench color category, and at each test the order in which the 3 pictures is shown is also randomized.

We run surveys across three user groups: *Academic users* (5 members) are representative of highly informed and tech savvy users, who are familiar with how generative models work; *Random users* (25 members) are representative of illiterate users who are not familiar with computer-based image generation; *Students* (16 members) are representative of masters' level students who are familiar with image generation tools, and who have attended introductory-level machine learning classes.

Overall, we collected 102 surveys (45, 35 and 22 surveys across 3 days for Academics, Random users and Students respectively) which we detail in Table B.1. The top part of the table breaks down all possible answers combinations. The results, although with some differences between user groups, clearly highlight that the three alignment metrics we consider in this work are roughly equivalent, with MI and BLIP-VQA being preferred over HPS. For the Academics and Students groups, all the three images are considered sufficiently aligned with the prompt in almost half

Alignment		Category (%)					
Methodology	Model	Color	Shape	Texture	2D-Spatial	Non-spatial	Complex
<i>none</i>	SD-2.1-base	29.76	11.90	40.48	35.71	66.67	29.76
Inference-time	A&E	◊31.95	◊15.48	◊52.38	32.14	65.48	30.95
	SDG	26.19	◊15.48	38.10	38.10	61.90	29.76
	SCG	20.24	11.90	33.33	◊40.48	◊69.05	◊39.29
Fine-tuning	DPOK	23.81	16.67	◊47.62	34.52	◊70.24	◊38.10
	GORS	◊34.52	14.29	48.81	◊36.90	65.48	30.95
	HN-ITM	23.81	◊19.05	30.95	20.24	47.62	23.81
	MI-TUNE	46.43	25.01	53.19	45.24	73.81	46.43

TABLE B.2: Users study comparing alignment methods. **Bold** shows best performance; ◊shows the best method per-family.

of the cases (40.4% and 44.6% respectively). Interestingly, random users select only one of the three images about $10\times$ much more frequently than the other two groups (on average 14.2% for real users while 5.4% and 3% for Academics and Students respectively). We hypothesise that being previously exposed (or not) to the technical problems of image generation from the alignment perspective, or simply being literate (or not) about machine learning can influence the selection among the three pictures.

The bottom part of the table summarizes the answers for each individual metric. Despite the general preference for BLIP-VQA, the results corroborate once more that MI provides a meaningful alignment signal (possibly compatible with aesthetics too).

Finally, we recall that our goal in this section is to study whether **MI is a plausible alignment measure**, rather than electing the “best” alignment metric. Indeed, this analysis does not indicate the final performance of alignment methods, which instead we report in Table 7.1.

B.1.2 Comparing alignment methods

In this second survey campaign we aimed to understand how users perceive images generated by the 8 methods we considered in our study, i.e., “vanilla” SD, A&E (Chefer et al., 2023b), SDG (Feng et al., 2023b) and SCG (Salimans and Ho, 2022) DPOK (Fan et al., 2023), GORS (Huang et al., 2023), HN-ITM (Krojer et al., 2023) and our method Mutual Information Fine Tuning (MI-TUNE) (when used with a single round of fine-tuning).

To do so, we run surveys composed of 2 tests for each T2I-Combench category (12 rounds in total). Each test shows a prompt and the 8 pictures generated using a different method. For each category, we randomly selected 100 prompts from T2I-Combench test set to pre-generate the pictures. At run time, the web app randomly selects 2 prompts for each category, and also randomly selects images from the related pool. Last, it randomly arranges both the tests (so that categories are shuffled) and the methods (so that pictures of a method are not visualized in the same position in the visualized grid).

Table B.2 collects the results of a campaign with 42 surveys. Specifically, the table shows the percentage of answers where the picture of a given method was selected (no matter if other methods were also selected) – these results are integrated in right side of **Table 7.1** and are duplicated here for completeness.

B.2 Experimental protocol details

Diffusion model. We report in **Table B.3** all the hyper-parameters we used for our experiments.

Name	Value
Trainable model	UNET
Trainable timesteps	$t \sim U(500, 1000)$
PEFT	DoRA (Liu et al., 2024a)
Rank	32
α	32
Learning rate (LR)	$1e - 4$
Gradient norm clipping	1.0
LR scheduler	Constant
LR warmup steps	0
Optimizer	AdamW
AdamW - β_1	0.9
AdamW - β_2	0.999
AdamW - weight decay	$1e - 2$
AdamW - ϵ	$1e - 8$
Resolution	512×512
Classifier-free guidance scale	7.5
Denoising steps	50
Batch size	400
Training iterations	300
GPUs for Training	$1 \times \text{NVIDIA A100}$

TABLE B.3: Training hyperparameters on SD-2.1-Base.

Next, we provide additional details on the computational cost of MI-TUNE. In our approach, there are two distinct phases that require computational effort:

The first is the construction of the fine-tuning set \mathcal{S} based on point-wise MI. As a reminder, for this phase, we use a pre-trained SD model (namely SD-2.1-base at a resolution 512×512) and, given a prompt, conditionally generate 50 images, while at the same time computing point-wise MI between the prompt and each image. This is done for all the prompts in the set \mathcal{Y} . Specifically for T2I-Combench, each category training set has 700 prompts, and for each prompt we generate 50 images from which we select the one with highest MI. The generation of the 700×50 fine-tuning set requires roughly *24 hours*, *i.e., about 2min per-prompt* on a single A100-80GB GPU – the 50 images are generated together (as they roughly require 50GB of the 80GB available VRAM), while each prompt is processed sequentially.

The second is the parameter efficient fine-tuning of the pre-trained model. Using the configuration discussed above, MI-TUNE requires *8 hours* when using a single A100-80GB GPU.

Rectified flow model. Table B.4 includes all the hyper-parameters we used for our experiments.

Name	Value
Trainable model	MMDiT
PEFT	LoRA
Rank	32
α	32
Learning rate (LR)	$5e - 6$
Gradient norm clipping	0.005
LR scheduler	Constant
LR warmup steps	400
Optimizer	AdamW
AdamW - β_1	0.9
AdamW - β_2	0.999
AdamW - weight decay	$1e - 4$
AdamW - ϵ	$1e - 8$
Resolution	1024×1024
CFG scale	4.5
Denoising steps	100
M	50
k	1
Global batch size	240
Training iterations	2000

TABLE B.4: Training hyperparameters on SD-3.5-Medium.

To construct a fine-tuning set \mathcal{S} based on point-wise MI, we use the pre-trained SD3.5-M and, given a prompt, conditionally generate 50 images with CFG = 4.5 at resolution 1024×1024 , while at the same time computing point-wise MI between the prompt and each image latent. Only the image with the highest MI is kept. This process is done twice for all the 700 fine-tuning prompts \mathcal{Y} defined by T2I-Combench. Given the constructed fine-tuning set, we finetune SD3.5-M for 2000 iterations with LoRA adaptation.

Note that (i) there is no overhead at image generation time: once a pre-trained model has been fine-tuned with MI-TUNE, conditional sampling takes the same amount of time of “vanilla” SD and (ii) the time to process the workloads scales down (almost linearly) with the number of GPUs used according to our observations.

B.3 HPS scores range

Wu et al. (2023a) report a detailed benchmark of their metrics across 20+ models in the HPS-v2 GitHub repository <https://github.com/tgxs002/HPSv2>. These details are hidden by default when loading the repository home page and need to be explicitly “opened” expanding collapsed menus (e.g., ► v2 benchmark). To ease discussion, in Table B.5 we report an extract of these benchmarks focusing on StableDiffusion as other models are out scope for our study.

Benchmark	Model	Animation	Concept-Art	Painting	Photo	(avg)
v2	SDXL Refiner (0.9)	28.45	27.66	27.67	27.46	(27.80)
	SDXL Base (0.9)	28.42	27.63	27.60	27.29	(27.73)
	SD (2.0)	27.48	26.89	26.86	27.46	(27.17)
	SD (1.4)	27.26	26.61	26.66	27.27	(26.95)
v2.1	SDXL Refiner (0.9)	33.26	32.07	31.63	28.38	(31.34)
	SDXL Base (0.9)	32.84	31.36	30.86	27.48	(30.63)
	SD (2.0)	27.09	26.02	25.68	26.73	(26.38)
	SD (1.4)	26.03	24.87	24.80	25.70	(25.35)

TABLE B.5: HPS benchmark across multiple Stable Diffusion models extracted for HPS-v2 GitHub repo.

Results refer to two benchmark and are visually split between SD and SDXL. The columns Animation, Concept-Art, Painting and Photo are different images style, while (avg) reflects average by row.

Both versions of the benchmark present similar takeaways which we can summarize in two main observations. Specifically, (i) different versions of the same model present < 0.5 differences and (ii) SDXL outperforms SD of about +1 point – the variation of HPS scores is extremely contained even if these models are different generations apart.

Our HPS scores in Table 7.1 present similar properties, but other literature (e.g., Table 2 in Zhao et al., 2024) present similar evidence.

B.4 Additional results and ablations

B.4.1 Ablation: Fine-tuning set selection strategies

Fine-tuning set selection strategy. It is important to stress that creating a fine-tuning dataset using the very same metric used for the final evaluation can artificially introduce a bias as stated in (Huang et al., 2023): “calculating the rewards for GORS with the automatic evaluation metrics can lead to biased results”.

The selection strategy to compose the fine-tuning dataset is directly related to alignment scores and different fine-tuning methods opt for different choices. Specifically: HN-ITM uses an ad-hoc dataset with real positive and negative pairs; GORS uses a synthetic dataset with no selection, but the fine-tuning loss of each sample is weighted by BLIP-VQA DPOK synthesizes new images at each training iteration since it is an online RL fine-tuning approach, and uses a pre-trained human preference model for reward. Table 7.1, in the main paper, shows alternative fine-tuning strategies based on synthetic generated data using a variety of selection scores:

GORS and DPOK are the closest methods to MI-TUNE from this point of view, yet generally underperforming compared to it.

TABLE B.6: FT set selection.		For completeness, we perform an experiment where we fine-tune based on a dataset selected via HPS scores. Results in Table B.6 (same as Table 7.4, but duplicated here for simplicity) show that selecting fine-tuning samples based on MI outperforms such an alternative strategy, using BLIP-VQA.	
Strategy	BLIP-VQA		HPS
	Color	Shape	
MI only	65.04	50.08	29.13 25.57
HPS only	59.43	46.87	<i>n.a.</i> <i>n.a.</i>
MI+Real(0.25)	61.34	48.47	29.16 25.87
MI+Real(0.5)	61.63	49.50	29.38 25.92
MI+Real(0.9)	59.83	48.92	28.60 25.60

Next, another natural question to ask is whether the self-supervised fine-tuning method we suggest in this work is a valid strategy. Indeed, instead of using synthetic image data for fine-tuning the base model, it is also possible to use real-life,

captioned image data. Then, we present an ablation on the use of real samples, along with synthetic images, in the fine-tuning procedure. In Table B.6(bottom) we report the experimental results obtained by composing the fine-tuning dataset by imposing the ratio of images generated by the SD model to x , and the ratio of real images taken from the CC12M dataset (Changpinyo et al., 2021) to $(1 - x)$, where in both cases we select the candidate images to be used in the fine-tuning set \mathcal{S} using MI. So, for example, MI+Real(0.25) indicates that we use 25% of real images. Interestingly, we observe the following trend. Complementing the synthetically generated samples with few real ones does not benefit alignment (lower BLIP-VQA) but might have a positive effect for aesthetics (higher HPS).

Fine-tuning set size. We continue by reporting an ablation on the fine-tuning set \mathcal{S} size.

Specifically, based on Algorithm 1, two parameters determine both the quality and the associated computational cost related to the fine-tuning set \mathcal{S} : the number of candidate images M , and how many k are selected to be included in \mathcal{S} .

Hyper-params		Category	
M	k	Color	Shape
30	1	58.12	47.48
50	7	59.31	47.26
50	1	61.57	48.40
100	1	60.12	47.80
500	1	59.28	46.79

TABLE B.7: BLIP-VQA alignment results on T2I-CompBench’s Color and Shape categories varying size and composition of fine-tuning set. Results obtained using $R=1$.

Table B.7 shows that the best performance is obtained selecting 2% images (1 image out of 50). We repeated the finetuning experiments on the categories Color and Shape by varying the selection ratio in the ranges $\{7/50, 1/30, 1/100, 1/500\}$. Results indicate that the best selection ratio is the middle-range corresponding to the baseline MI-TUNE. We hypothesise that higher selection ratios pollute the finetuning set with lower quality images, while a more selective threshold favours images which have the highest alignment but possibly lower realism. Additionally, we remark that the number M of candidate images has a negligible impact, above $M = 50$, whereas fewer candidate images induce degraded performance. Hence, the value $M = 50$ is, in our experiments, a sweet-spot that produces a valid candidate set, while not imposing a large computational burden.

B.4.2 Ablation: Fine-tuning model adapters and modalities

In this Section, we provide additional results (Table B.8) on MI-TUNE, concerning which part of the pre-trained SD model to fine-tune. In particular, we tried to fine-tune the denoising UNET network alone and both the denoising and the text encoding (CLIP) networks. The baseline results are obtained, as described in the main paper, with Do-RA (Liu et al., 2024a) adapters. Switching to Lo-RA layers Hu et al., 2021 incurs in a performance degradation, a trend observed also for other tasks in the literature (Liu et al., 2024a). Interestingly, joint fine-tuning of the UNET backbone together with the text encoder layers degrades performance as well, which has also been observed in the literature Huang et al., 2023. Even if, in principle, a joint fine-tuning strategy should provide better results, as the amount of information transferred from the prompt to the image is bottle-necked by the text encoder architecture, we observed empirically more unstable training dynamics than the variant where only the score network backbone is fine-tuned, resulting in degraded performance.

Model	Category	
	Color	Shape
MI-TUNE DoRA	61.57	48.40
MI-TUNE LoRA	58.25	48.27
MI-TUNE UNet+Text(joint)	57.88	47.79

TABLE B.8: BLIP-VQA alignment results on T2I-CompBench’s Color and Shape categories finetuning different portions of the model.

B.4.3 Ablation: Combining categories into a single model

The design space for T2I alignment improvement has many options and this should call not only to investigate alignment performance but also operational and computational costs. For instance, fine-tuning methods require to create ad-hoc models

while one can argue that a single/multi-purpose model might be a more lean and general solution.

This calls for investigating if/how different task-specific fine-tuned models can be combined into a single model to address the different tasks at once. For the T2I-Combench, we considered two design options:

1. **Weights merging:** the DoRA weights of the 6 distinct per-category models are “merged” doing their arithmetic means forming a new “meta” model.
2. **Joint optimization:** we create a new “meta” model by running a single fine-tuning process but using the union of the category-specific fine-tuning set.

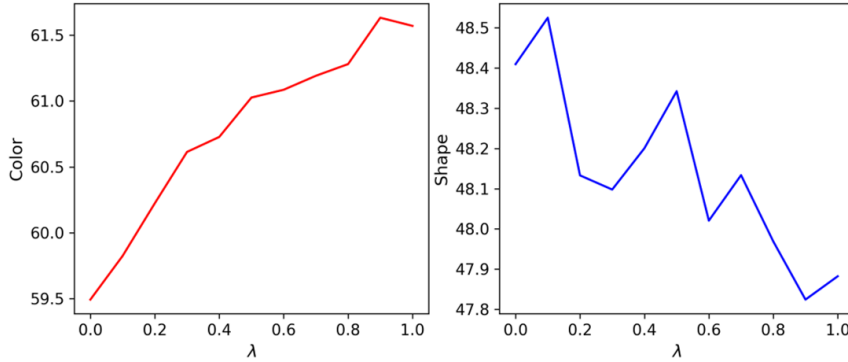


FIGURE B.2: Weights merging: $\lambda \times \text{Color} + (1.0 - \lambda) \times \text{Shape}$.

To start from a reference example, Fig. B.2 reports the BLIP-VQA obtained when testing on the color and shape test sets on the merged model obtained of the two task-specific models. The hyper-parameter λ is used to balance the merging. For instance, at $\lambda = 0$, the performance on color (left plot) are obtained using the shape-only model. Overall, the results show that these two categories are (partially) conflicting across all λ values. Yet, a performance trade off might be sufficient in some scenarios.

MI-TUNE	Color	Shape	Texture	2D-Spatial	Non-spatial	Complex
<i>variants</i>	$\pm(\text{BLIP-VQA})$	$\pm(\text{BLIP-VQA})$	$\pm(\text{BLIP-VQA})$	$\pm(\text{UNIDET})$	$\pm(\text{BLIP-VQA})$	$\pm(\text{BLIP-VQA})$
<i>from table 7.1</i>	61.57	48.40	58.27	18.51	67.77	53.54
Model weighting	58.50	48.23	58.22	16.72	68.28	54.35
Joint optimization	60.35	47.73	57.96	18.44	69.68	54.88

TABLE B.9: Benchmarking strategies for combining models.

We then extended the analysis across all categories using a simple arithmetic mean for model merging, i.e., all models have the same weight. Results are reported in Table B.9 using MI-TUNE as reference. Overall, for most categories, the single “meta” model has degraded performance and neither weights merging nor joint optimization are the best alternative across all categories.

B.5 Qualitative examples for T2I-CompBench using SD-2.1-base

B.5.1 Color prompts

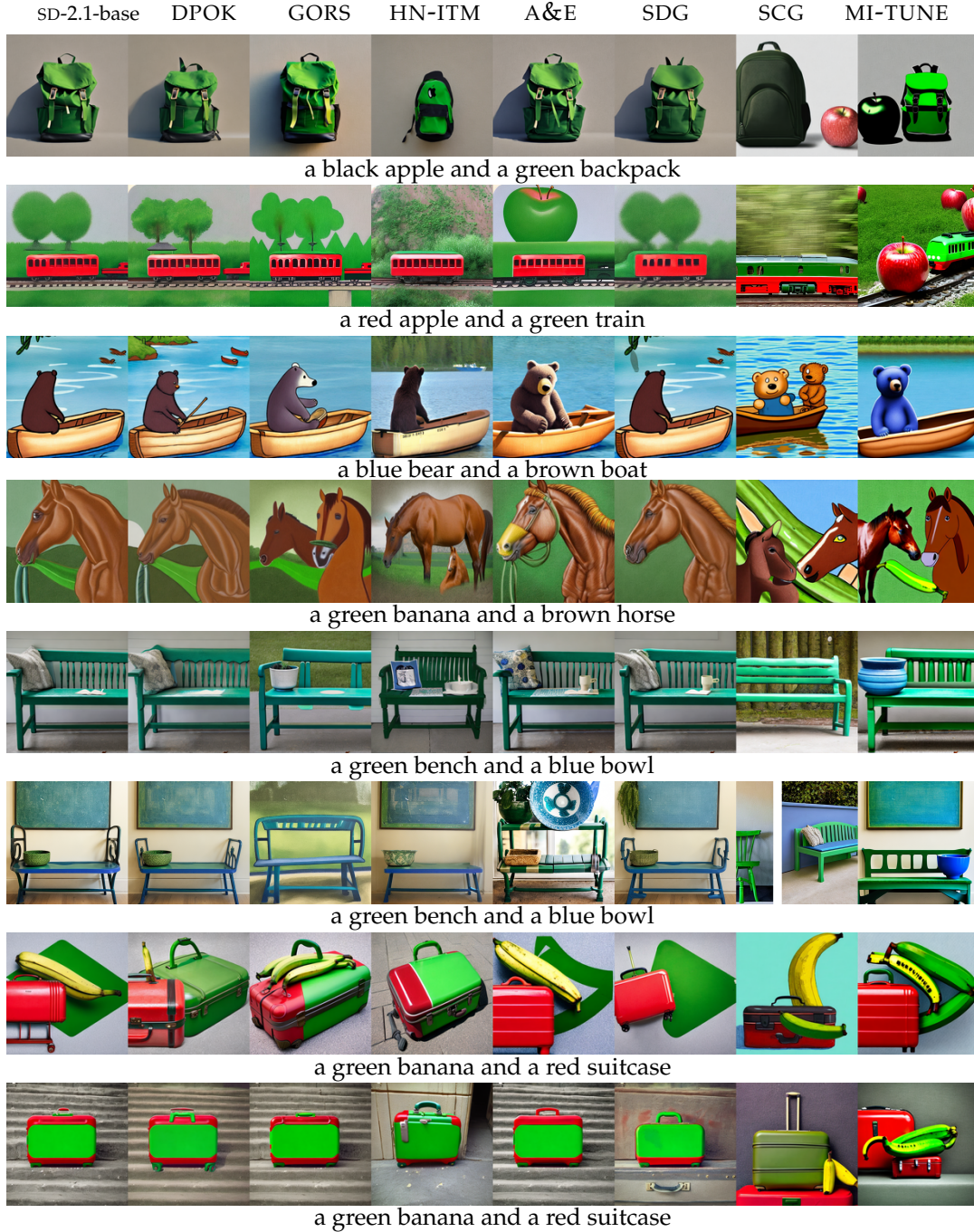


FIGURE B.3: Qualitative examples for Color from Table 7.1 (same seed used for a given prompt).

B.5.2 Shape prompts



FIGURE B.4: Qualitative examples of the Shape category from Table 7.1 (same seed used for a given prompt).

B.5.3 Texture prompts



FIGURE B.5: Qualitative examples of the Texture category from Table 7.1 (same seed used for a given prompt).

B.5.4 2D-Spatial prompts

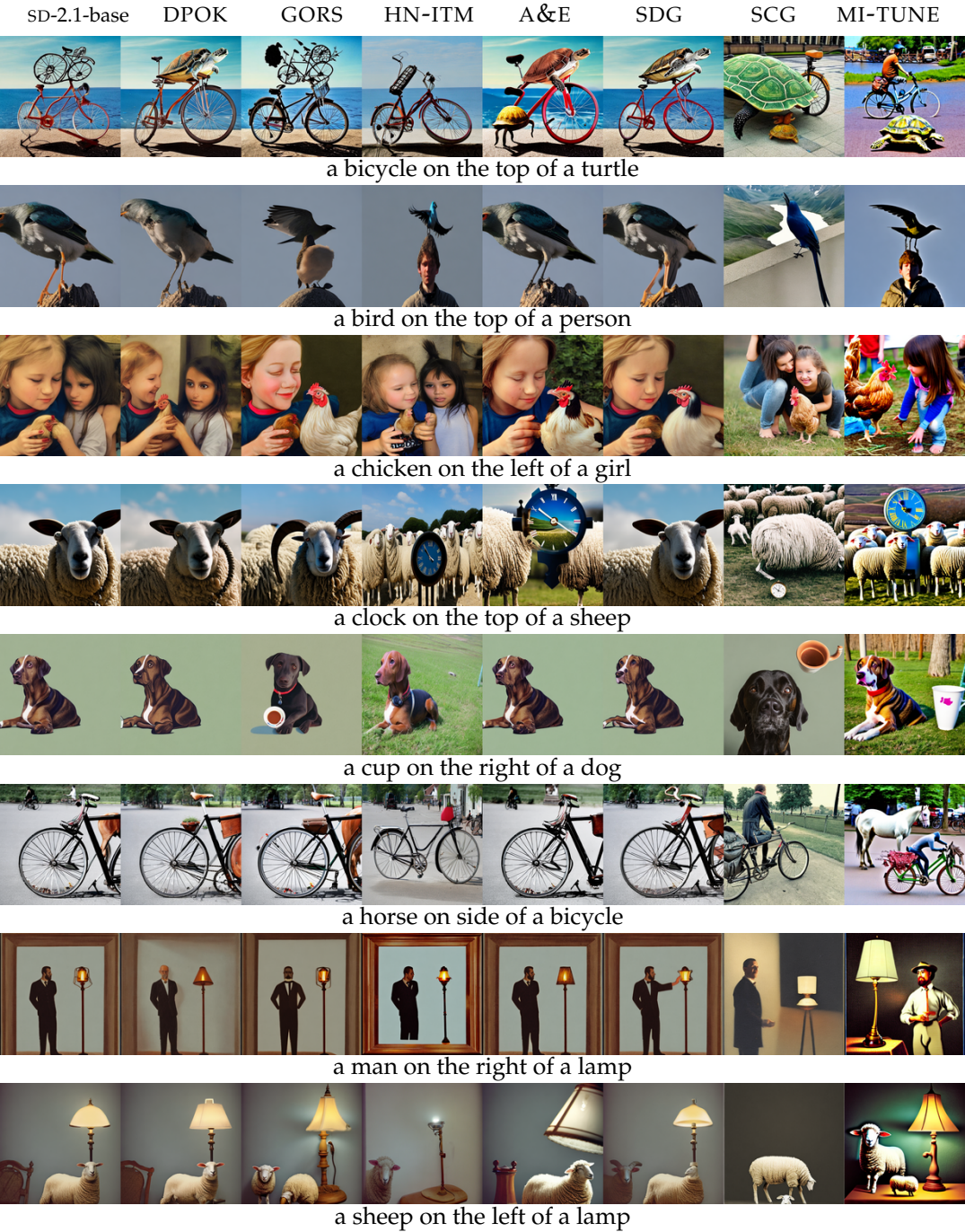


FIGURE B.6: Qualitative examples of the 2D-Spatial category from Table 7.1 (same seed used for a given prompt).

B.5.5 Non-Spatial prompts

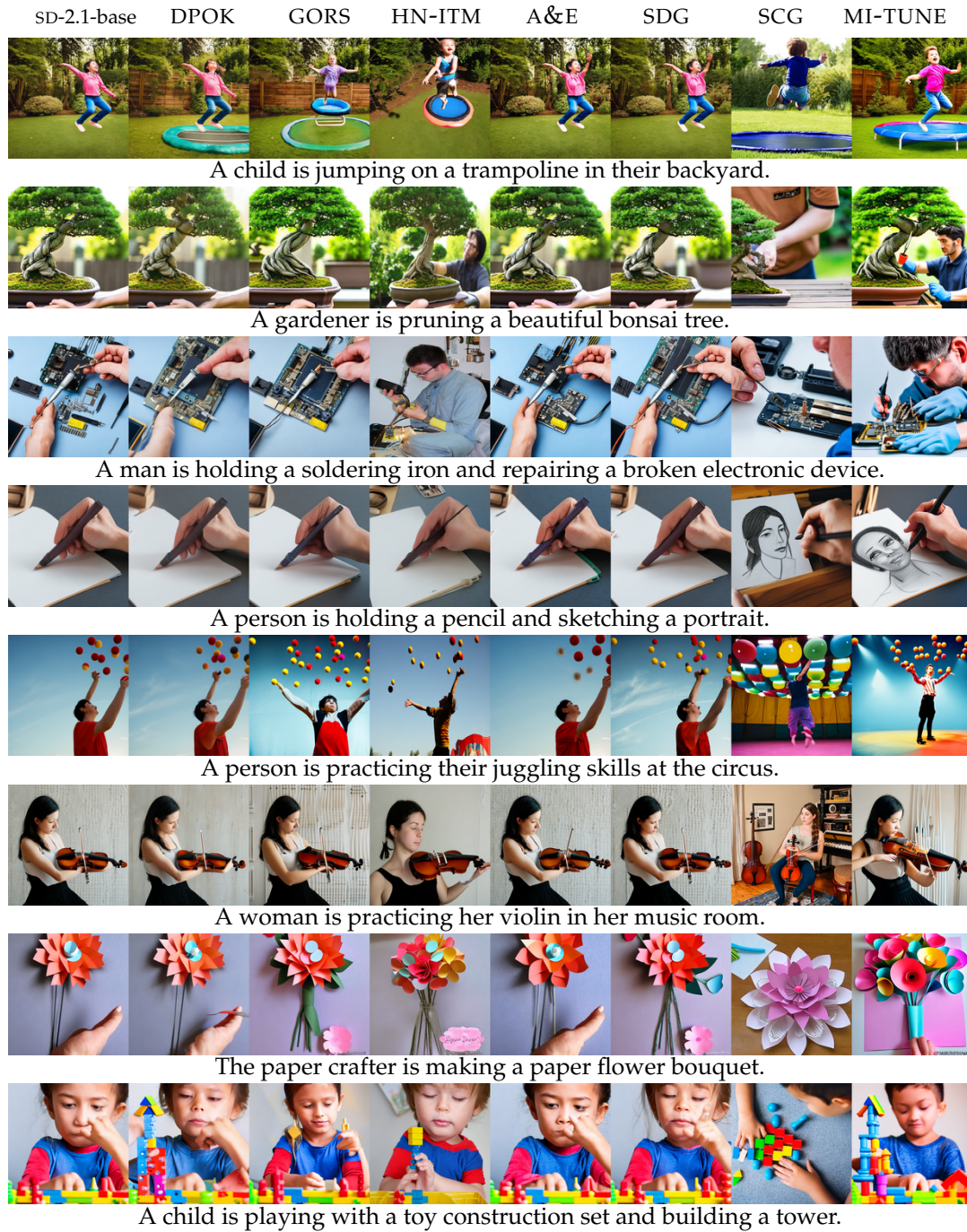


FIGURE B.7: Qualitative examples of the Non-spatial category from Table 7.1 (same seed used for a given prompt).

B.5.6 Complex prompts

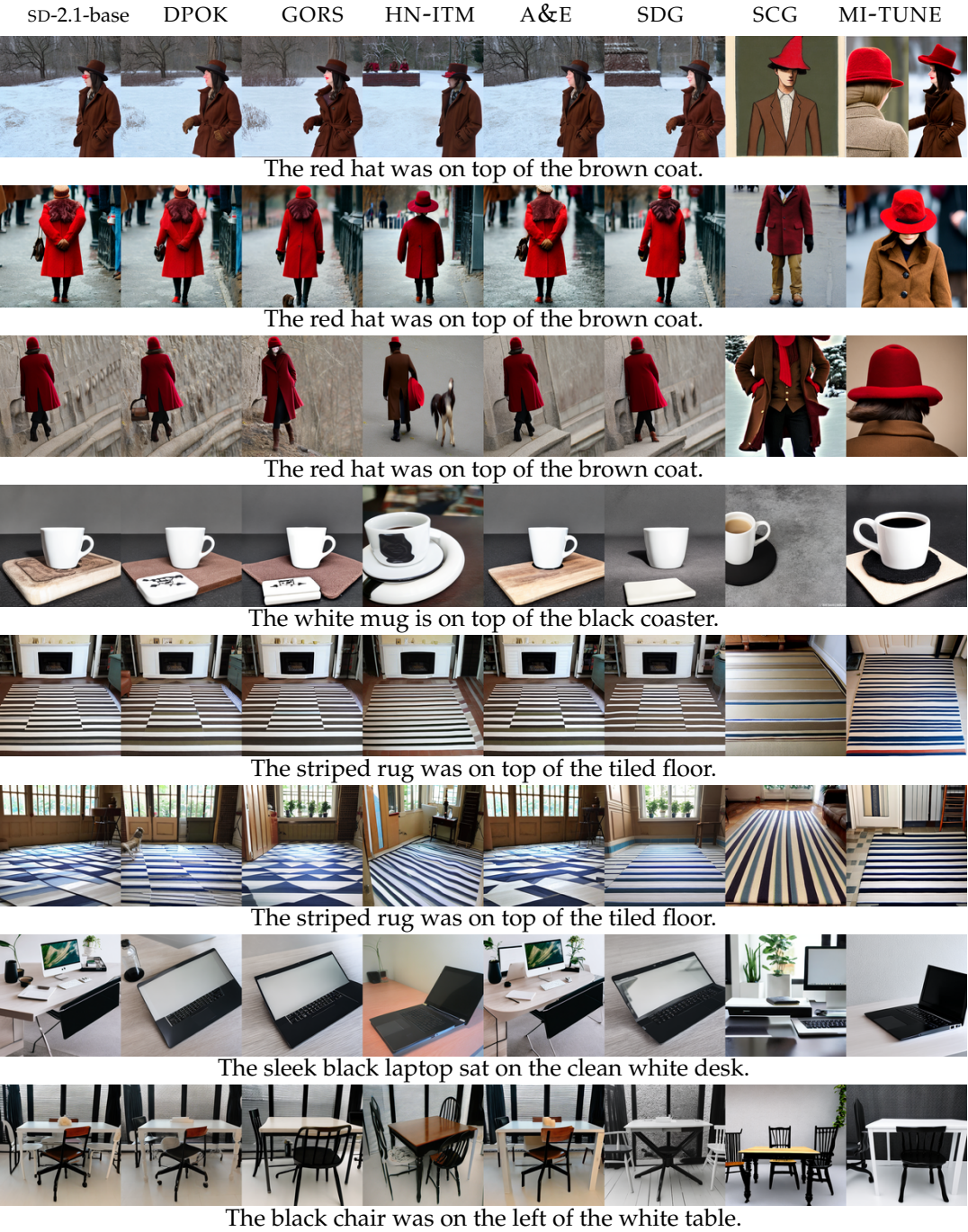


FIGURE B.8: Qualitative examples of the Complex category from Table 7.1 (same seed used for a given prompt).

B.6 Qualitative examples for T2I-CompBench using SDXL













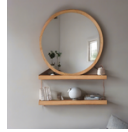









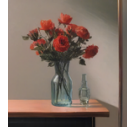

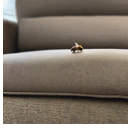
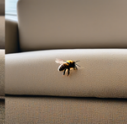



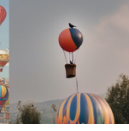


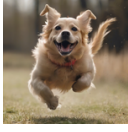
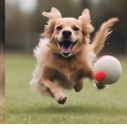














	SDXL	MI-TUNE	SDXL	MI-TUNE	SDXL	MI-TUNE	SDXL	MI-TUNE
Color prompts.								
	a black cat and a gray mouse		a red backpack and a blue chair		a red bowl and a blue train		a blue bench and a green bowl	
Shape prompts.								
	a big elephant and a small dog		a big lion and a small mouse		a circular mirror and a triangular shelf unit		a tall skyscraper and a short cottage	
Texture prompts.								
	a fabric bag and a glass vase		a fabric hat and a glass mirror		a fabric jacket and a glass plate		a leather jacket and a glass vase	
2D-Spatial prompts.								
	a bee on side of a couch		a bicycle on the bottom of a girl		a bird on the top of a balloon		a candle on the top of a chicken	
Non-Spatial prompts.								
	A dog is chasing after a ball and wagging its tail		A gardener is pruning a beautiful bonsai tree		A person is looking at a display of vintage clothing and admiring the fashion		A person is looking at a sculpture garden and appreciating the artwork	
Complex prompts.								
	The black chair is on top of the blue rug		The black pencil was next to the green notebook		The blue mug is on top of the green coaster		The bright yellow banana contrasted with the dull brown apple	

FIGURE B.9: Qualitative examples from [Table 7.5](#) (same seed used for a given prompt).

Model	HPS
SD-2.1-base	23.99
DiffusionDB	24.35
MI-TUNE	25.32
MI-TUNE \ominus base	1.33
MI-TUNE \ominus DiffusionDB	0.97

$A \ominus B$ shows the abs. difference between A and B.

TABLE B.10: DiffusionDB.

B.7 Fine-tuning with DiffusionDB dataset

B.7.1 Selecting images and BLIP-VQA prompts decomposition

In this section, we provide additional details about using prompts created by real users, i.e., DiffusionDB.

Dataset properties. DiffusionDB was collected scraping the StableDiffusion discord channels “[...] We download chat messages from the Stable Diffusion Discord channels with DiscordChatExporter, saving them as HTML files. We focus on channels where users can command a bot to run Stable Diffusion Version 1 to generate images by typing a prompt, hyperparameters, and the number of images [...]” (Wang et al., 2022c). The scraped data is then packaged into parquet files (containing metadata such prompt, image filenames and hyperparams) and zip files (containing the actual images in WebP format) and made available on HuggingFace.

Fine-tuning with DiffusionDB. We fine-tune SD-2.1-base on 1,250 prompts randomly sampled and compare two different scenarios. A first dataset is composed using images provided by DiffusionDB itself. As each prompt in DiffusionDB is paired to (about) 4 generated images we obtain a 5,000 prompt-image pairs reference dataset. For the second dataset, we use the 1,250 prompts to generate $M = 50$ images for each prompt and selecting the $k = 1$ image with the highest MI. We repeat this procedure 4 times to construct a complementary fine-tuning dataset with prompt-image 5,000 pairs. We fine-tune SD-2.1-base on each of the two datasets with pre-trained loss, then test on 500 DiffusionDB prompts (again, randomly selected and disjoint from the training set prompt-image pairs) generating 10 images for each test prompt.

Table B.10 (which is duplicating here Table 7.6 for simplicity) shows the results. Fine-tuning either using the DiffusionDB images or MI-TUNE can improve HPS score alignment with respect to the SD-2.1-base baseline. Yet, MI-TUNE improves upon using directly DiffusionDB images, i.e., using MI is very competitive compared to (expensive) manual labeling.

BLIP-VQA prompts decomposition. Our evaluation considers only HPS as we find

that the higher prompt complexity does not well suit the BLIP-VQA prompt decomposition. Recall that BLIP-VQA requires to split the prompt into “noun phrases”, each used to create a VQA for the BLIP model. Specifically, BLIP-VQA uses spaCy’s English pipeline `en_core_web_sm` to extract noun phrases from the prompt which result complex when the prompt is complex. Below we report some examples related to extracting first three noun phrases extracted from human prompts.

Examples of good/easy segmentations:

- **concept art** of **a silent hill monster**. painted by **edward hopper**.
- **anthropomorphic shark**, **digital art**, **concept art**
- **geodesic landscape**, **john chamberlain**, **christopher balaskas**, tadao ando, 4k

Examples of segmentations with missing/broad subjects:

- **a realistic architectural visualization** of **a sustainable mixed - use post - modern post - growth walkable people** oriented **urban development**.
- **a realistic wide angle painting** of **a vintage cathode ray tube**, in **a park**, in and advanced state of decay, psychedelic mushrooms all around, in a post apocalyptic city, ghibli, daytime, dynamic lighting
- render of **dreamy beautiful landscape**, dreamy, by **herbaceous plants**, **artger**, large scale, detailed vintage photo hyper realistic ultra realistic photo realistic photography, unreal engine, high detailed, 8 k

B.7.2 Qualitative examples for DiffusionDB

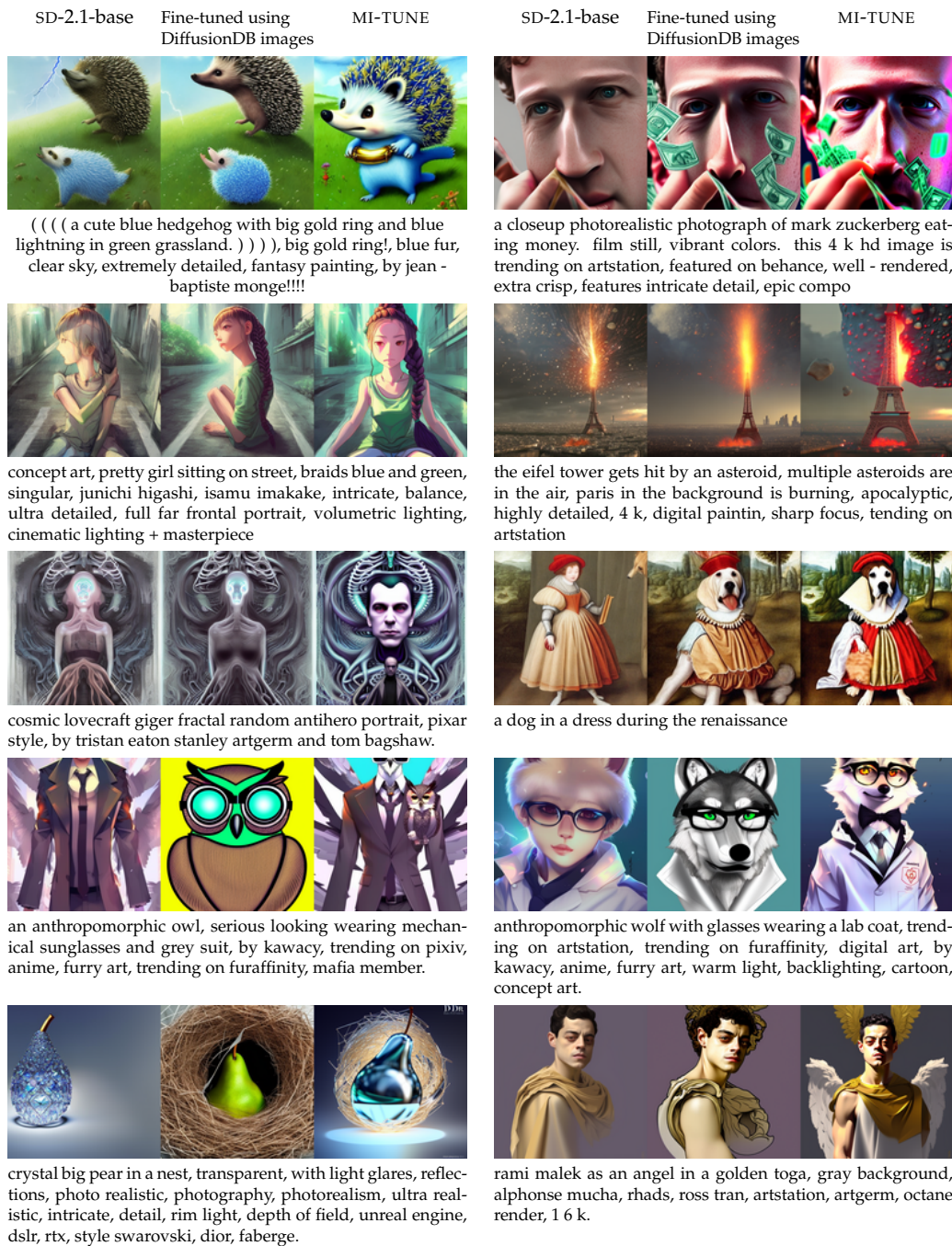


FIGURE B.10: Qualitative examples from Table 7.6 (same seed used for a given prompt).

B.8 Qualitative analysis of MI as an alignment measure

Figure B.11 is expanding Figure 7.1 to include qualitative examples for all categories in T2I-CompBench.

Color binding:
"A blue car and a red horse"



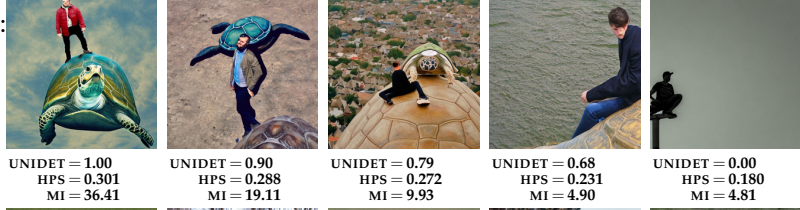
Texture binding:
"A fabric dress and a glass table"



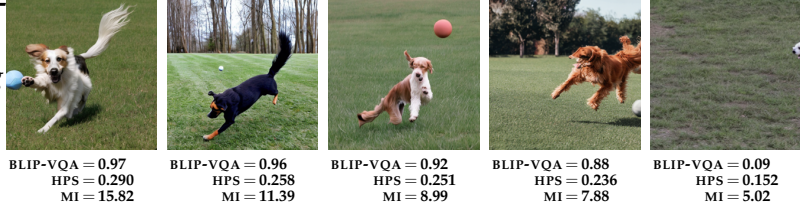
Shape binding:
"A round bag and a rectangular wallet"



Spatial relation:
"a man on the top of a turtle"



Non-spatial relation:
"A dog is chasing after a ball and wagging its tail"



Complex prompt:
"The red hat was on top of the brown coat"



FIGURE B.11: Qualitative analysis of MI as an alignment measure (all metrics decrease from left to right).

B.9 BLIP-VQA, HPS and MI score distributions

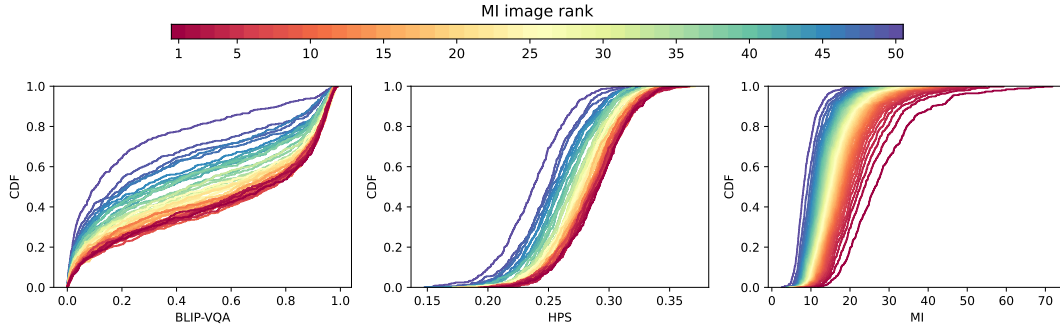


FIGURE B.12: CDF of alignment scores. Color reflect images rank based on MI.

The analysis presented in Section 7.2 shows that BLIP-VQA, HPS and MI relate to each other. However, two aspects not discussed in Section 7.2 are (i) the support of each metric and (ii) how the distribution of the scores compare between well and poor aligned images. In this ablation we address both aspects using the following protocol.

We considered all 700 training prompts for the color category (the consideration presented in this ablation extends to the other T2I-CompBench categories too), we generated 50 images for each prompt, and computed the 3 metrics for each of the 50 images. Last, for each prompt, we rank the images based on MI (1:highest, 50:lowest) – overall we obtained a $700 \text{ prompts} \times 50 \text{ images} \times 4$ (3 metrics + 1 rank) tensor.

We then investigated if/how the MI rank affects the distribution of the scores for BLIP-VQA and HPS. Intuitively, given the highest-ranked (viz lowest-ranked) images based on MI, also BLIP-VQA and HPS should show very high values (viz low values). In practice, we first reordered the scores of the three metrics for each prompt based the MI rank and then we derived 50 distributions for each metric, one for each column in the tensor collecting the scores of each metric. Figure B.12 shows the obtained distributions color coded based on the MI rank.

Considering the metrics support, we can notice a few differences among the three metrics. Specifically, BLIP-VQA is in the $[0,1]$ range and for all rank values, the whole support is always used. Conversely, despite HPS is also in the $[0, 1]$ range,³ the actual support is more skewed – this corroborates the discussion presented in Appendix B.3. Last, while MI is unbounded, the scores are mostly contained in the $[0-40]$ range.

Considering the relationship between the rank and the scores, all metrics show very similar patterns. Specifically, all distributions are very smooth no matter the rank.

³HPS is defined as the cosine similarity between image and text embeddings, similarly to CLIP. As such, theoretically, the score is in $[-1, 1]$ range. However, in practice, and for the T2I-CompBench dataset, the score is effectively only in the $[0, 1]$ range.

Moreover, as expected, for all metrics the distributions smoothly shift horizontally with respect to their rank – the color gradient separates very well red/high rank, yellow/middle rank, blue/low rank.

The kendal τ analysis reported in [Section 7.2](#) considers the 1st, 25th, 50th image for a prompt, selected by ranking the images based on their MI score. This is consistent with the analysis presented in [Figure B.12](#) and based on the figure we argue that our selection of 3 pictures (having the highest, mid, lowest scores for each prompt) is a reasonable choice for the results reported in [Section 7.2](#) as they are representative of the spectrum of values observed by the metrics.

Bibliography

- Aceto, Giuseppe et al. (2019a). "MIMETIC: Mobile encrypted traffic classification using multimodal deep learning". In: *Computer Networks* 165, p. 106944. ISSN: 1389-1286.
- (2019b). "MIMETIC: Mobile encrypted traffic classification using multimodal deep learning". In: *Computer Networks* 165, p. 106944. ISSN: 1389-1286. DOI: <https://doi.org/10.1016/j.comnet.2019.106944>. URL: <https://www.sciencedirect.com/science/article/pii/S1389128619304669>.
- Aceto, Giuseppe et al. (2019c). "MIRAGE: Mobile-app Traffic Capture and Ground-truth Creation". In: *IEEE International Conference on Computing, Communication and Security (ICCCS)*.
- Aceto, Giuseppe et al. (2019d). "MIRAGE: Mobile-app Traffic Capture and Ground-truth Creation". In: *ICCCS*.
- Aceto, Giuseppe et al. (2024). "Synthetic and privacy-preserving traffic trace generation using generative AI models for training Network Intrusion Detection Systems". In: *Journal of Network and Computer Applications* 229, p. 103926. ISSN: 1084-8045. DOI: <https://doi.org/10.1016/j.jnca.2024.103926>.
- Adeleke, Oluwamayowa Ade, Nicholas Bastin, and Deniz Gurkan (Jan. 2022). "Network Traffic Generation: A Survey and Methodology". In: *ACM Comput. Surv.* 55.2.
- Agarwal, Aishwarya et al. (2023). "A-STAR: Test-time Attention Segregation and Retention for Text-to-image Synthesis". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2283–2293.
- Akbari, Iman et al. (2021a). "A Look Behind the Curtain: Traffic Classification in an Increasingly Encrypted Web". In: *Proc. ACM Meas. Anal. Comput. Syst.* 5.1.
- (2021b). "A Look Behind the Curtain: Traffic Classification in an Increasingly Encrypted Web". In: *ACM Measurement and Analysis of Computing Systems* 5.1.
- Akiba, Takuya et al. (2019). "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD '19*. Association for Computing Machinery.

- Alaa, Ahmed M. et al. (2022). *How Faithful is your Synthetic Data? Sample-level Metrics for Evaluating and Auditing Generative Models*. arXiv: 2102.08921 [cs.LG]. URL: <https://arxiv.org/abs/2102.08921>.
- Albergo, Michael S. and Eric Vanden-Eijnden (2023). *Building Normalizing Flows with Stochastic Interpolants*. arXiv: 2209.15571 [cs.LG]. URL: <https://arxiv.org/abs/2209.15571>.
- Alcaraz, Juan Lopez and Nils Strodthoff (2022). “Diffusion-based Time Series Imputation and Forecasting with Structured State Space Models”. In: *Transactions on Machine Learning Research*. ISSN: 2835-8856. URL: <https://openreview.net/forum?id=hHiIbk7ApW>.
- Balaji, Yogesh et al. (2022a). “eDiff-I: Text-to-Image Diffusion Models with Ensemble of Expert Denoisers”. In: *arXiv preprint arXiv:2211.01324*.
- Balaji, Yogesh et al. (2022b). “eDiff-I: Text-to-Image Diffusion Models with Ensemble of Expert Denoisers”. In: *arXiv preprint arXiv:2211.01324*.
- Belghazi, Mohamed Ishmael et al. (2021). *MINE: Mutual Information Neural Estimation*. arXiv: 1801.04062 [cs.LG]. URL: <https://arxiv.org/abs/1801.04062>.
- Biroli, Giulio et al. (Nov. 2024). “Dynamical regimes of diffusion models”. In: *Nature Communications* 15.1. ISSN: 2041-1723. DOI: 10.1038/s41467-024-54281-3. URL: <http://dx.doi.org/10.1038/s41467-024-54281-3>.
- Bovenzi, G. et al. (2021). “A First Look at Class Incremental Learning in Deep Learning Mobile Traffic Classification”. In: *IFIP Traffic Measurement and Analysis (TMA)*.
- Burg, Max F. et al. (2023). “A data augmentation perspective on diffusion models and retrieval”. In: *arXiv:2304.10253*. arXiv: 2304.10253 [cs.CV].
- Changpinyo, Soravit et al. (2021). “Conceptual 12m: Pushing web-scale image-text pre-training to recognize long-tail visual concepts”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3558–3568.
- Chatterjee, Agneet et al. (2024). *Getting it Right: Improving Spatial Consistency in Text-to-Image Models*. arXiv: 2404.01197 [cs.CV].
- Chawla, N. V. et al. (2002). “SMOTE: Synthetic Minority Over-sampling Technique”. In: *Journal of Artificial Intelligence Research* 16, pp. 321–357.
- Chefer, Hila et al. (2023b). *Attend-and-Excite: Attention-Based Semantic Guidance for Text-to-Image Diffusion Models*. arXiv: 2301.13826 [cs.CV].
- (2023a). “Attend-and-Excite: Attention-Based Semantic Guidance for Text-to-Image Diffusion Models”. In: *ACM Trans. Graph.* 42.4.

- Chen, Ricky T. Q. et al. (2019a). *Neural Ordinary Differential Equations*. arXiv: 1806.07366 [cs.LG]. URL: <https://arxiv.org/abs/1806.07366>.
- Chen, Ting et al. (2020). “A Simple Framework for Contrastive Learning of Visual Representations”. In: *arXiv:2002.05709*. arXiv: 2002.05709 [cs.LG].
- Chen, Wei-Yu et al. (2019b). “A Closer Look at Few-shot Classification”. In: *ICLR*.
- Claffy, KC et al. (July 2021). “Workshop on Overcoming Measurement Barriers to Internet Research (WOMBIR 2021) final report”. In: *SIGCOMM Comput. Commun. Rev.* 51.3, 33–40. ISSN: 0146-4833. DOI: 10.1145/3477482.3477489. URL: <https://doi.org/10.1145/3477482.3477489>.
- Clark, Kevin et al. (2024). “Directly Fine-Tuning Diffusion Models on Differentiable Rewards”. In: *The Twelfth International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=1vmSEVL19f>.
- Conwell, Colin and Tomer Ullman (2022). *Testing Relational Understanding in Text-Guided Image Generation*. arXiv: 2208.00005 [cs.CV].
- Cubuk, Ekin D. et al. (2019a). “AutoAugment: Learning Augmentation Policies from Data”. In: *arXiv:1805.09501*. arXiv: 1805.09501 [cs.CV].
- Cubuk, Ekin D. et al. (2019b). “RandAugment: Practical automated data augmentation with a reduced search space”. In: *arXiv:1909.13719*. arXiv: 1909.13719 [cs.CV].
- Cubuk, Ekin Dogus et al. (2021). “Tradeoffs in Data Augmentation: An Empirical Study”. In: *International Conference on Learning Representations (ICLR)*.
- Czyż, Paweł et al. (2023). *Beyond Normal: On the Evaluation of Mutual Information Estimators*. arXiv: 2306.11078 [stat.ML]. URL: <https://arxiv.org/abs/2306.11078>.
- Dahary, Omer et al. (2024). *Be Yourself: Bounded Attention for Multi-Subject Text-to-Image Generation*. arXiv: 2403.16990 [cs.CV].
- Dai, Wenyuan et al. (2007). “Boosting for Transfer Learning”. In: *ICML*.
- Dalva, Yusuf, Kavana Venkatesh, and Pinar Yanardag (2024). *FluxSpace: Disentangled Semantic Editing in Rectified Flow Transformers*. arXiv: 2412.09611 [cs.CV]. URL: <https://arxiv.org/abs/2412.09611>.
- Demšar, Janez (2006). “Statistical comparisons of classifiers over multiple data sets”. In: *The Journal of Machine learning research* 7, pp. 1–30.
- Dhariwal, Prafulla and Alexander Nichol (2021). “Diffusion Models Beat GANs on Image Synthesis”. In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., pp. 8780–8794.

- Eldele, Emadeldeen et al. (2021). “Time-Series Representation Learning via Temporal and Contextual Contrasting”. In: *arXiv:2106.14112*. arXiv: [2106.14112 \[cs.LG\]](#).
- Emmerich, Paul et al. (2015). “MoonGen: A Scriptable High-Speed Packet Generator”. In: *Proceedings of the 2015 Internet Measurement Conference*. IMC ’15. New York, NY, USA: Association for Computing Machinery, 275–287. ISBN: 9781450338486. DOI: [10.1145/2815675.2815692](#). URL: <https://doi.org/10.1145/2815675.2815692>.
- Esser, Patrick et al. (2024). “Scaling Rectified Flow Transformers for High-Resolution Image Synthesis”. In: *Proceedings of the 41st International Conference on Machine Learning (ICML)*. URL: <https://arxiv.org/abs/2403.03206>.
- Fan, Ying et al. (2023). “Reinforcement Learning for Fine-tuning Text-to-Image Diffusion Models”. In: *Thirty-seventh Conference on Neural Information Processing Systems*. URL: <https://openreview.net/forum?id=80TPepXzeh>.
- Feng, Tongtong et al. (2021). “Few-Shot Class-Adaptive Anomaly Detection with Model-Agnostic Meta-Learning”. In: *IFIP Networking*.
- Feng, Weixi et al. (2023a). “Training-Free Structured Diffusion Guidance for Compositional Text-to-Image Synthesis”. In: *The Eleventh International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=PUIqjT4rzq7>.
- (2023b). “Training-Free Structured Diffusion Guidance for Compositional Text-to-Image Synthesis”. In: *The Eleventh International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=PUIqjT4rzq7>.
- Ferriol-Galmés, Miquel et al. (2023). “RouteNet-Fermi: Network Modeling With Graph Neural Networks”. In: *IEEE/ACM Transactions on Networking* 31.6, pp. 3080–3095.
- Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *ICML*.
- FLUX (2023). *Black Forest Labs*. <https://github.com/black-forest-labs/flux>.
- Franzese, Giulio, Mustapha Bounoua, and Pietro Michiardi (2024). “MINDE: Mutual Information Neural Diffusion Estimation”. In: *The Twelfth International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=0kWd8SJq8d>.
- Gafni, Oran et al. (2022). *Make-A-Scene: Scene-Based Text-to-Image Generation with Human Priors*. arXiv: [2203.13131 \[cs.CV\]](#).
- Gao, Kaihui et al. (2023). “DONS: Fast and Affordable Discrete Event Network Simulation with Automatic Parallelization”. In: *Proceedings of the ACM SIGCOMM 2023 Conference*. ACM SIGCOMM ’23, 167–181.

- Gao, Shuyang, Greg Ver Steeg, and Aram Galstyan (2015). "Efficient Estimation of Mutual Information for Strongly Dependent Variables". In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Guy Lebanon and S. V. N. Vishwanathan. Vol. 38. Proceedings of Machine Learning Research. San Diego, California, USA: PMLR, pp. 277–286. URL: <https://proceedings.mlr.press/v38/gao15.html>.
- Goodfellow, Ian et al. (Oct. 2020). "Generative adversarial networks". In: *Commun. ACM* 63.11, 139–144.
- Goodfellow, Ian J. et al. (2014). *Generative Adversarial Networks*. arXiv: 1406.2661 [stat.ML]. URL: <https://arxiv.org/abs/1406.2661>.
- Gordon, Brian et al. (2023). *Mismatch Quest: Visual and Textual Feedback for Image-Text Misalignment*. arXiv: 2312.03766 [cs.CL].
- Grill, Jean-Bastien et al. (2020). "Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 21271–21284.
- Grimal, Paul et al. (2024). "TIAM - A Metric for Evaluating Alignment in Text-to-Image Generation". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 2890–2899.
- Gu, Albert, Karan Goel, and Christopher Ré (2022). *Efficiently Modeling Long Sequences with Structured State Spaces*. arXiv: 2111.00396 [cs.LG]. URL: <https://arxiv.org/abs/2111.00396>.
- Guarino, Idio et al. (2021a). "Classification of Communication and Collaboration Apps via Advanced Deep-Learning Approaches". In: *IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*.
- (2021b). "Classification of Communication and Collaboration Apps via Advanced Deep-Learning Approaches". In: *2021 IEEE 26th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pp. 1–6.
- Guarino, Idio et al. (2022). "Contextual counters and multimodal Deep Learning for activity-level traffic classification of mobile communication apps during COVID-19 pandemic". In: *Computer Networks* 219, p. 109452. ISSN: 1389-1286.
- Guarino, Idio et al. (2023). "Many or Few Samples? Comparing Transfer, Contrastive and Meta-Learning in Encrypted Traffic Classification". In: *IFIP Traffic Measurement and Analysis (TMA)*.

- Han, Hui, Wenyuan Wang, and Binghuan Mao (2005). "Borderline-SMOTE: A New Over-sampling Method in Imbalanced Data Sets Learning". In: *Advances in Intelligent Computing*.
- He, Haibo et al. (2008). "ADASYN: Adaptive synthetic sampling approach for imbalanced learning". In: *International Joint Conference on Neural Networks (IJCNN)*.
- He, Kaiming et al. (2015). "Deep Residual Learning for Image Recognition". In: *arXiv:1512.03385*. arXiv: [1512.03385 \[cs.CV\]](#).
- Heng, Yuqiang, Vikram Chandrasekhar, and Jeffrey G. Andrews (2021a). "UTMobileNetTraffic2021: A Labeled Public Network Traffic Dataset". In: *IEEE Networking Letters* 3.3, pp. 156–160.
- (2021b). "UTMobileNetTraffic2021: A Labeled Public Network Traffic Dataset". In: *IEEE Networking Letters* 3.3, pp. 156–160.
- Hessel, Jack et al. (2021). "CLIPScore: A Reference-free Evaluation Metric for Image Captioning". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.
- Heusel, Martin et al. (2017). "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/8a1d694707eb0fefe65871369074926d-Paper.pdf.
- Ho, Jonathan, Ajay Jain, and Pieter Abbeel (2020a). *Denoising Diffusion Probabilistic Models*. arXiv: [2006.11239 \[cs.LG\]](#). URL: <https://arxiv.org/abs/2006.11239>.
- (2020c). "Denoising diffusion probabilistic models". In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS '20.
- (2020b). "Denoising Diffusion Probabilistic Models". In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 6840–6851.
- Ho, Jonathan and Tim Salimans (2021). "Classifier-Free Diffusion Guidance". In: *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*. URL: <https://openreview.net/forum?id=qw8AKxfYbI>.
- (2022). "Classifier-free diffusion guidance". In: *arXiv preprint arXiv:2207.12598*.
- Holland, Jordan et al. (2021). "New Directions in Automated Traffic Analysis". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. CCS '21.
- Hong, Yan et al. (2022). "SAFA: Sample-Adaptive Feature Augmentation for Long-Tailed Image Classification". In: *European Conference on Computer Vision (ECCV)*.

- Horowicz, Eyal, Tal Shapira, and Yuval Shavitt (2022). "A Few Shots Traffic Classification with Mini-FlowPic Augmentations". In: *Proceedings of the 22nd ACM Internet Measurement Conference*. IMC '22. Nice, France: Association for Computing Machinery, 647–654. ISBN: 9781450392594.
- Hu, Edward J. et al. (2021). *LoRA: Low-Rank Adaptation of Large Language Models*. arXiv: 2106.09685 [cs.CL]. URL: <https://arxiv.org/abs/2106.09685>.
- Hu, Yushi et al. (2023). "TIFA: Accurate and Interpretable Text-to-Image Faithfulness Evaluation with Question Answering". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 20406–20417.
- Huang, Kaiyi et al. (2023). "T2I-CompBench: A Comprehensive Benchmark for Open-world Compositional Text-to-image Generation". In: *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. URL: <https://openreview.net/forum?id=weHBzTLXpH>.
- Imagen-Team et al. (2024). *Imagen 3*. arXiv: 2408.07009 [cs.CV].
- IXIA - High-Volume Traffic Generator Products Catalog (n.d.). <https://www.keysight.com/fr/en/assets/7121-1065/catalogs/High-Volume-Traffic-Generator-Products-Catalog.pdf>.
- Jain, Samyak et al. (2023). "DART: Diversify-Aggregate-Repeat Training Improves Generalization of Neural Networks". In: *Computer Vision and Pattern Recognition (CVPR)*.
- Jan, Steve T.K. et al. (2020). "Throwing Darts in the Dark? Detecting Bots with Limited Data using Neural Data Augmentation". In: *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 1190–1206.
- Jayasumana, Sadeep et al. (2024). "Rethinking FID: Towards a Better Evaluation Metric for Image Generation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9307–9315.
- Jiang, Dongzhi et al. (2024a). *CoMat: Aligning Text-to-Image Diffusion Model with Image-to-Text Concept Matching*. arXiv: 2404.03653 [cs.CV]. URL: <https://arxiv.org/abs/2404.03653>.
- Jiang, Xi et al. (2023). "Generative, High-Fidelity Network Traces". In: *ACM Workshop on Hot Topics in Networks (HotNets)*.
- Jiang, Xi et al. (Feb. 2024b). "NetDiffusion: Network Data Augmentation Through Protocol-Constrained Traffic Generation". In: *Proc. ACM Meas. Anal. Comput. Syst.* 8.1.
- Johnson, Justin M. and Taghi M. Khoshgoftaar (2019). "Survey on deep learning with class imbalance". In: *Journal of Big Data* 6.

- Jolicoeur-Martineau, Alexia, Kilian Fatras, and Tal Kachman (2024). *Generating and Imputing Tabular Data via Diffusion and Flow-based Gradient-Boosted Trees*. arXiv: 2309.09968 [cs.LG]. URL: <https://arxiv.org/abs/2309.09968>.
- Jonny Kong, Z. et al. (2024). "High-Fidelity Cellular Network Control-Plane Traffic Generation without Domain Knowledge". In: IMC '24.
- Kang, Wonjun, Kevin Galim, and Hyung Il Koo (2023). *Counting Guidance for High Fidelity Text-to-Image Synthesis*. arXiv: 2306.17567 [cs.CV].
- Karras, Tero et al. (2022). *Elucidating the Design Space of Diffusion-Based Generative Models*. arXiv: 2206.00364 [cs.CV]. URL: <https://arxiv.org/abs/2206.00364>.
- Karthik, Shyamgopal et al. (2023). "If at First You Don't Succeed, Try, Try Again: Faithful Diffusion-based Text-to-Image Generation by Selection". In: *arXiv preprint arXiv:2305.13308*.
- Kendall, M. G. (1938). "A Ner Measure of Rank Correlation". In: *Biometrika* 30.1-2, pp. 81–93.
- Khosla, Prannay et al. (2020). "Supervised Contrastive Learning". In: *Advances in neural information processing systems (NeurIPS)*.
- Kim, Yunji et al. (2023). "Dense Text-to-Image Generation with Attention Modulation". In: *ICCV*.
- Kingma, Diederik P and Max Welling (2022). *Auto-Encoding Variational Bayes*. arXiv: 1312.6114 [stat.ML]. URL: <https://arxiv.org/abs/1312.6114>.
- Kingma, Diederik P et al. (2021). "Variational Diffusion Models". In: *Advances in Neural Information Processing Systems*. Ed. by A. Beygelzimer et al. URL: <https://openreview.net/forum?id=2LdBqxc1Yv>.
- Kirillov, Alexander et al. (2023). "Segment Anything". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4015–4026.
- Kollovieh, Marcel et al. (2023). *Predict, Refine, Synthesize: Self-Guiding Diffusion Models for Probabilistic Time Series Forecasting*. arXiv: 2307.11494 [cs.LG]. URL: <https://arxiv.org/abs/2307.11494>.
- Kollovieh, Marcel et al. (2025). *Flow Matching with Gaussian Process Priors for Probabilistic Time Series Forecasting*. arXiv: 2410.03024 [cs.LG]. URL: <https://arxiv.org/abs/2410.03024>.
- Kong, Xianghao et al. (2024). "Interpretable Diffusion via Information Decomposition". In: *The Twelfth International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=X6tNkN6ate>.

- Kotelnikov, Akim et al. (2023). "TabDDPM: modelling tabular data with diffusion models". In: *Proceedings of the 40th International Conference on Machine Learning*. ICML'23.
- Krojer, Benno et al. (2023). "Are Diffusion Models Vision-And-Language Reasoners?" In: *arXiv preprint arXiv:2305.16397*.
- LAION project (2024). *LAION datasets*. <https://laion.ai/projects/>.
- Lecun, Y. et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.
- Lee, Chaejeong, Jayoung Kim, and Noseong Park (2023a). "CoDi: co-evolving contrastive diffusion models for mixed-type tabular synthesis". In: *Proceedings of the 40th International Conference on Machine Learning*. ICML'23.
- Lee, Kimin et al. (2023b). *Aligning Text-to-Image Models using Human Feedback*. arXiv: 2302.12192 [cs.LG].
- Li, Junnan et al. (2023a). "BLIP-2: bootstrapping language-image pre-training with frozen image encoders and large language models". In: *Proceedings of the 40th International Conference on Machine Learning*.
- Li, Shufan et al. (2024a). *OmniFlow: Any-to-Any Generation with Multi-Modal Rectified Flows*. arXiv: 2412.01169 [cs.MM]. URL: <https://arxiv.org/abs/2412.01169>.
- Li, Yumeng et al. (2023b). "Divide & bind your attention for improved generative semantic nursing". In: *34th British Machine Vision Conference 2023, BMVC 2023*.
- Li, Yuxin et al. (2024b). *Transformer-Modulated Diffusion Models for Probabilistic Multivariate Time Series Forecasting*. URL: <https://openreview.net/forum?id=qae04YACHs>.
- Li, Zhong et al. (2025). *Diffusion Models for Tabular Data: Challenges, Current Progress, and Future Directions*. arXiv: 2502.17119 [cs.LG]. URL: <https://arxiv.org/abs/2502.17119>.
- Liang, Wei et al. (2022). "Variational Few-Shot Learning for Microservice-Oriented Intrusion Detection in Distributed Industrial IoT". In: *IEEE Transactions on Industrial Informatics* 18.8, pp. 5087–5095. DOI: 10.1109/TII.2021.3116085.
- Lin, Tsung-Yi et al. (2015). *Microsoft COCO: Common Objects in Context*. arXiv: 1405.0312 [cs.CV]. URL: <https://arxiv.org/abs/1405.0312>.
- Lin, Xinjie et al. (2022). "ET-BERT: A Contextualized Datagram Representation with Pre-training Transformers for Encrypted Traffic Classification". In: *Proceedings of the ACM Web Conference 2022*. WWW '22. ACM. DOI: 10.1145/3485447.3512217. URL: <http://dx.doi.org/10.1145/3485447.3512217>.

- Lin, Zinan et al. (2020a). “PacGAN: The Power of Two Samples in Generative Adversarial Networks”. In: *IEEE Journal on Selected Areas in Information Theory* 1.1, pp. 324–335. DOI: [10.1109/JSAIT.2020.2983071](https://doi.org/10.1109/JSAIT.2020.2983071).
- Lin, Zinan et al. (2020b). “Using GANs for Sharing Networked Time Series Data: Challenges, Initial Promise, and Open Questions”. In: *Proceedings of the ACM Internet Measurement Conference*. IMC ’20, 464–483.
- Lipman, Yaron et al. (2023). *Flow Matching for Generative Modeling*. arXiv: [2210.02747](https://arxiv.org/abs/2210.02747) [cs.LG]. URL: <https://arxiv.org/abs/2210.02747>.
- Liu, Nan et al. (2022a). “Compositional Visual Generation with Composable Diffusion Models”. In: *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XVII*, 423–439.
- Liu, Shih-Yang et al. (2024a). *DoRA: Weight-Decomposed Low-Rank Adaptation*. arXiv: [2402.09353](https://arxiv.org/abs/2402.09353) [cs.CL].
- Liu, Vivian and Lydia B Chilton (2022). “Design Guidelines for Prompt Engineering Text-to-Image Generative Models”. In: *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*.
- Liu, Xingchao, Chengyue Gong, and Qiang Liu (2022b). *Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow*. arXiv: [2209.03003](https://arxiv.org/abs/2209.03003) [cs.LG]. URL: <https://arxiv.org/abs/2209.03003>.
- Liu, Xingchao et al. (2024b). *InstaFlow: One Step is Enough for High-Quality Diffusion-Based Text-to-Image Generation*. arXiv: [2309.06380](https://arxiv.org/abs/2309.06380) [cs.LG].
- Luxemburk, Jan, Karel Hynek, and Tomas Cejka (2023a). “Encrypted traffic classification: the QUIC case”. In: *IFIP Traffic Measurement and Analysis (TMA)*.
- Luxemburk, Jan, Karel Hynek, and Tomáš Čejka (2023b). “Encrypted traffic classification: the QUIC case”. In: *2023 7th Network Traffic Measurement and Analysis Conference (TMA)*, pp. 1–10.
- Luxemburk, Jan and Tomáš Čejka (2023a). “Fine-grained TLS services classification with reject option”. In: *Computer Networks* 220, p. 109467. ISSN: 1389-1286.
- (2023b). “Fine-grained TLS services classification with reject option”. In: *Computer Networks* 220, p. 109467. ISSN: 1389-1286.
- Ma, Jian et al. (2023). *Subject-Diffusion: Open Domain Personalized Text-to-Image Generation without Test-time Fine-tuning*. arXiv: [2307.11410](https://arxiv.org/abs/2307.11410) [cs.CV].
- Mahajan, Shweta et al. (2023). *Prompting Hard or Hardly Prompting: Prompt Inversion for Text-to-Image Diffusion Models*. arXiv: [2312.12416](https://arxiv.org/abs/2312.12416) [cs.CV].

- Mañas, Oscar et al. (2024). *Improving Text-to-Image Consistency via Automatic Prompt Optimization*. arXiv: 2403.17804 [cs.CV].
- McAllester, David and Karl Stratos (2020). *Formal Limitations on the Measurement of Mutual Information*. arXiv: 1811.04251 [cs.IT]. URL: <https://arxiv.org/abs/1811.04251>.
- Meng, Xuying et al. (2022). "Packet Representation Learning for Traffic Classification". In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. KDD '22. Washington DC, USA, 3546–3554. ISBN: 9781450393850.
- Meral, Tuna Han Salih et al. (2024). "CONFORM: Contrast is All You Need For High-Fidelity Text-to-Image Diffusion Models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
- Miao, Zichen et al. (2024). "Training Diffusion Models Towards Diverse Image Generation with Reinforcement Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10844–10853.
- Moore, Andrew W and Denis Zuev (2005). "Internet traffic classification using bayesian analysis techniques". In: *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pp. 50–60.
- Mumuni, Alhassan and Fuseini Mumuni (2022). "Data augmentation: A comprehensive survey of modern approaches". In: *Array* 16, p. 100258. ISSN: 2590-0056.
- Müller, Samuel G. and Frank Hutter (2021). "TrivialAugment: Tuning-free Yet State-of-the-Art Data Augmentation". In: *International Conference on Computer Vision (ICCV)*.
- Naiman, Ilan et al. (2024). "Utilizing Image Transforms and Diffusion Models for Generative Modeling of Short and Long Time Series". In: *arXiv preprint arXiv:2410.19538*.
- Negative Prompts (2022). <https://huggingface.co/spaces/stabilityai/stable-diffusion/discussions/7857>.
- Nemenman, Ilya, F. Shafee, and William Bialek (2001). "Entropy and Inference, Revisited". In: *Advances in Neural Information Processing Systems*. Ed. by T. Dietterich, S. Becker, and Z. Ghahramani. Vol. 14. MIT Press. URL: https://proceedings.neurips.cc/paper_files/paper/2001/file/d46e1fcf4c07ce4a69ee07e4134bcef1-Paper.pdf.
- NEOX - Valkyrie Stateless Ethernet Traffic Generation and Analysis up to 800Gbps (n.d.). <https://www.neox-networks.com/en/products/xena-networks/valkyrie-network-traffic-generator-stream-based/>.

- New Brunswick, University of (2016). *VPN-nonVPN dataset (ISCXVPN2016)*. URL: <https://www.unb.ca/cic/datasets/vpn.html>.
- Nguyen, Thuy T.T. and Grenville Armitage (2008). "A survey of techniques for internet traffic classification using machine learning". In: *IEEE Communications Surveys & Tutorials* 10.4, pp. 56–76.
- Nguyen, XuanLong, Martin J. Wainwright, and Michael I. Jordan (2010). "Estimating Divergence Functionals and the Likelihood Ratio by Convex Risk Minimization". In: *IEEE Transactions on Information Theory* 56.11, 5847–5861. ISSN: 1557-9654. DOI: 10.1109/tit.2010.2068870. URL: <http://dx.doi.org/10.1109/TIT.2010.2068870>.
- Nichol, Alex et al. (2022). *GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models*. arXiv: 2112.10741 [cs.CV].
- Odena, Augustus, Christopher Olah, and Jonathon Shlens (2017). "Conditional Image Synthesis With Auxiliary Classifier GANs". In: *arXiv:1610.09585*. arXiv: 1610.09585 [stat.ML].
- Ogezi, Michael and Ning Shi (2024). *Optimizing Negative Prompts for Enhanced Aesthetics and Fidelity in Text-To-Image Generation*. arXiv: 2403.07605 [cs.CV].
- Øksendal, Bernt (2003). *Stochastic differential equations*. Springer.
- Oord, Aaron van den, Yazhe Li, and Oriol Vinyals (2019a). *Representation Learning with Contrastive Predictive Coding*. arXiv: 1807.03748 [cs.LG]. URL: <https://arxiv.org/abs/1807.03748>.
- (2019b). *Representation Learning with Contrastive Predictive Coding*. arXiv: 1807.03748 [cs.LG]. URL: <https://arxiv.org/abs/1807.03748>.
- OpenAI (2024). *ChatGPT (GPT-4) [Large language model]*. URL: <https://chat.openai.com/chat>.
- Oquab, Maxime et al. (2024). "DINOv2: Learning Robust Visual Features without Supervision". In: *Transactions on Machine Learning Research*. ISSN: 2835-8856. URL: <https://openreview.net/forum?id=a68SUt6zFt>.
- OSTINATO - Traffic Generator for Network Engineers (n.d.). <https://ostinato.org>.
- Ouyang, Yuankai et al. (2021). "FS-IDS: A Novel Few-Shot Learning Based Intrusion Detection System for SCADA Networks". In: *ICC*.
- Pacheco, Fannia et al. (2018). "Towards the deployment of machine learning solutions in network traffic classification: A systematic survey". In: *IEEE Communications Surveys & Tutorials* 21.2, pp. 1988–2014.

- Patil, Abhishek G. et al. (2016). "Survey of synthetic traffic generators". In: *2016 International Conference on Inventive Computation Technologies (ICICT)*. Vol. 1, pp. 1–3.
- Peebles, William and Saining Xie (2023). *Scalable Diffusion Models with Transformers*. arXiv: 2212.09748 [cs.CV]. URL: <https://arxiv.org/abs/2212.09748>.
- Piet, Julien, Dubem Nwoji, and Vern Paxson (2023). "GGFAST: Automating Generation of Flexible Network Traffic Classifiers". In: *Proceedings of the ACM SIGCOMM 2023 Conference*. ACM SIGCOMM '23. New York, NY, USA: Association for Computing Machinery, 850–866. ISBN: 9798400702365. DOI: 10.1145/3603269.3604840. URL: <https://doi.org/10.1145/3603269.3604840>.
- Podell, Dustin et al. (2024). "SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis". In: *The Twelfth International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=di52zR8xgf>.
- Pöppelbaum, Johannes, Gavneet Singh Chadha, and Andreas Schwung (2022). "Contrastive learning based self-supervised time-series analysis". In: *Applied Soft Computing* 117, p. 108397. ISSN: 1568-4946.
- Radford, Alec et al. (2021). "Learning Transferable Visual Models From Natural Language Supervision". In: *Proceedings of the 38th International Conference on Machine Learning*. Vol. 139, pp. 8748–8763.
- Ramesh, Aditya et al. (2022). *Hierarchical Text-Conditional Image Generation with CLIP Latents*. arXiv: 2204.06125 [cs.CV].
- Ranftl, René, Alexey Bochkovskiy, and Vladlen Koltun (2021). *Vision Transformers for Dense Prediction*. arXiv: 2103.13413 [cs.CV]. URL: <https://arxiv.org/abs/2103.13413>.
- Rassin, Royi et al. (2023). "Linguistic Binding in Diffusion Models: Enhancing Attribute Correspondence through Attention Map Alignment". In: *Thirty-seventh Conference on Neural Information Processing Systems*. URL: <https://openreview.net/forum?id=A0KU4nRw1W>.
- Redmon, Joseph et al. (2016). "You Only Look Once: Unified, Real-Time Object Detection". In: *arXiv:1506.02640*. arXiv: 1506.02640 [cs.CV].
- Rezaei, Shahbaz and Xin Liu (2019). "How to Achieve High Classification Accuracy with Just a Few Labels: A Semi-supervised Approach Using Sampled Packets". In: *IEEE Industrial Conference Advances in Data Mining - Applications and Theoretical Aspects (ICDM)*.
- (2020). "Multitask Learning for Network Traffic Classification". In: *ICCCN*.

- Riley, George F. and Thomas R. Henderson (2010). "The ns-3 Network Simulator". In: *Modeling and Tools for Network Simulation*. Ed. by Klaus Wehrle, Mesut Güneş, and James Gross. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 15–34.
- Ring, Markus et al. (2019). "Flow-based network traffic generation using Generative Adversarial Networks". In: *Computers & Security* 82, pp. 156–172. ISSN: 0167-4048.
- Rombach, Robin et al. (2022). "High-Resolution Image Synthesis With Latent Diffusion Models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10684–10695.
- Rong, Candong et al. (2021). "UMVD-FSL: Unseen Malware Variants Detection Using Few-Shot Learning". In: *IJCNN*.
- Saharia, Chitwan et al. (2022). "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding". In: *Advances in Neural Information Processing Systems*. Vol. 35, pp. 36479–36494.
- Salimans, Tim and Jonathan Ho (2022). *Progressive Distillation for Fast Sampling of Diffusion Models*. arXiv: 2202.00512 [cs.LG].
- Samuel, Dvir et al. (2024). "Generating images of rare concepts using pre-trained diffusion models". In.
- Schuhmann, Christoph et al. (2022). "LAION-5B: An open large-scale dataset for training next generation image-text models". In: *Neural Information Processing Systems (NeurIPS) - Datasets and Benchmarks Track*.
- SEAGULL - Multi-protocol traffic generator (n.d.). <https://gull.sourceforge.net>.
- Shapira, Tal and Yuval Shavitt (2019). "FlowPic: Encrypted Internet Traffic Classification is as Easy as Image Recognition". In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 680–687.
- Shen, Dazhong et al. (2024). *Rethinking the Spatial Inconsistency in Classifier-Free Diffusion Guidance*. arXiv: 2404.05384 [cs.CV].
- Shen, Ruoqi, Sébastien Bubeck, and Suriya Gunasekar (2022). "Data Augmentation as Feature Manipulation". In: *arXiv:2203.01572*. arXiv: 2203.01572 [cs.LG].
- Shi, Juntong et al. (2025). "TabDiff: a Mixed-type Diffusion Model for Tabular Data Generation". In: *The Thirteenth International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=swvURjrt8z>.
- Shorten, Connor and Taghi M. Khoshgoftaar (2019). "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6.1, pp. 1–48.

- Singh, Vaibhav et al. (2025). *Beyond Cosine Decay: On the effectiveness of Infinite Learning Rate Schedule for Continual Pre-training*. arXiv: 2503 . 02844 [cs.LG]. URL: <https://arxiv.org/abs/2503.02844>.
- Sivaroopan, N. et al. (2023a). “SyNIG: Synthetic Network Traffic Generation through Time Series Imaging”. In: *IEEE Local Computer Networks (LCN)*.
- Sivaroopan, Nirhoshan et al. (2023b). “NetDiffus: Network Traffic Generation by Diffusion Models through Time-Series Imaging”. In: *arXiv:2310.04429*. arXiv: 2310 . 04429 [cs.NI].
- Sivaroopan, Nirhoshan et al. (2023c). “SyNIG: Synthetic Network Traffic Generation through Time Series Imaging”. In: *2023 IEEE 48th Conference on Local Computer Networks (LCN)*, pp. 1–9.
- Sivaroopan, Nirhoshan et al. (2024). “NetDiffus: Network traffic generation by diffusion models through time-series imaging”. In: *Computer Networks* 251, p. 110616. ISSN: 1389-1286.
- Snell, Jake, Kevin Swersky, and Richard Zemel (2017). “Prototypical Networks for Few-shot Learning”. In: *NeurIPS*.
- Sohl-Dickstein, Jascha et al. (2015). “Deep Unsupervised Learning using Nonequilibrium Thermodynamics”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37, pp. 2256–2265.
- Song, Jiaming and Stefano Ermon (2019a). “Understanding the Limitations of Variational Mutual Information Estimators”. In: *International Conference on Learning Representations*.
- Song, Yang and Stefano Ermon (2019b). “Generative Modeling by Estimating Gradients of the Data Distribution”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc.
- (2020). “Improved Techniques for Training Score-Based Generative Models”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 12438–12448.
- Song, Yang et al. (2021). “Score-Based Generative Modeling through Stochastic Differential Equations”. In: *International Conference on Learning Representations*.
- Sun, Danyu et al. (2024). “NetDPSyn: Synthesizing Network Traces under Differential Privacy”. In: *IMC ’24*.
- Sun, Guanglu et al. (2018). “Network traffic classification based on transfer learning”. In: *Computers & Electrical Engineering* 69, pp. 920–927. ISSN: 0045-7906.
- Sun, Jiao et al. (2023). *DreamSync: Aligning Text-to-Image Generation with Image Understanding Feedback*. arXiv: 2311 . 17946 [cs.CV].

- Sung, Flood et al. (2018). “Learning to Compare: Relation Network for Few-Shot Learning”. In: *CVPR*.
- Tang, Raphael et al. (2023). “What the DAAM: Interpreting Stable Diffusion Using Cross Attention”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5644–5659.
- Tashiro, Yusuke et al. (2021). *CSDI: Conditional Score-based Diffusion Models for Probabilistic Time Series Imputation*. arXiv: 2107.03502 [cs.LG]. URL: <https://arxiv.org/abs/2107.03502>.
- Tian, Yonglong et al. (2020). “Rethinking Few-Shot Image Classification: A Good Embedding is All You Need?” In: *ECCV*.
- Towhid, Md. Shamim and Nashid Shahriar (2022). “Encrypted Network Traffic Classification using Self-supervised Learning”. In: *IEEE International Conference on Network Softwarization (NetSoft)*.
- Trabucco, Brandon et al. (2023). “Effective Data Augmentation With Diffusion Models”. In: *arXiv:2302.07944*. arXiv: 2302.07944 [cs.CV].
- Turull, Daniel, Peter Sjödin, and Robert Olsson (2016). “Pktgen: Measuring performance on high speed networks”. In: *Computer Communications* 82, pp. 39–48. ISSN: 0140-3664.
- Um, Soobin and Jong Chul Ye (2024). *Self-Guided Generation of Minority Samples Using Diffusion Models*. arXiv: 2407.11555 [cs.CV]. URL: <https://arxiv.org/abs/2407.11555>.
- Varga, Andras (2010). “OMNeT++”. In: *Modeling and Tools for Network Simulation*. Ed. by Klaus Wehrle, Mesut Güneş, and James Gross. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 35–59.
- Wallace, Bram et al. (2023). *Diffusion Model Alignment Using Direct Preference Optimization*. arXiv: 2311.12908 [cs.CV].
- Wang, Chao et al. (2022a). “AppClassNet: A Commercial-Grade Dataset for Application Identification Research”. In: 52.3, 19–27. ISSN: 0146-4833.
- (2022b). “AppClassNet: A Commercial-Grade Dataset for Application Identification Research”. In: *SIGCOMM Comput. Commun. Rev.* 52.3, 19–27. ISSN: 0146-4833. DOI: 10.1145/3561954.3561958.
- Wang, Pan et al. (2020a). “PacketCGAN: Exploratory Study of Class Imbalance for Encrypted Traffic Classification Using CGAN”. In: *International Conference on Communications (ICC)*.

- (2020b). “PacketCGAN: Exploratory Study of Class Imbalance for Encrypted Traffic Classification Using CGAN”. In: *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pp. 1–7.
- Wang, Yaqing et al. (2020c). “Generalizing from a Few Examples: A Survey on Few-Shot Learning”. In: 53.3.
- Wang, Yifan, Minzhao Lyu, and Vijay Sivaraman (2024). “Characterizing User Platforms for Video Streaming in Broadband Networks”. In: *Proceedings of the 2024 ACM on Internet Measurement Conference*. IMC ’24.
- Wang, Yulin et al. (2020d). “Implicit Semantic Data Augmentation for Deep Networks”. In: *arXiv:1909.12220*. arXiv: 1909.12220 [cs.CV].
- Wang, Zhiguang and Tim Oates (2015). “Imaging time-series to improve classification and imputation”. In: *Proceedings of the 24th International Conference on Artificial Intelligence*. IJCAI’15. AAAI Press.
- Wang, Zijie J. et al. (2022c). “DiffusionDB: A Large-Scale Prompt Gallery Dataset for Text-to-Image Generative Models”. In: *arXiv:2210.14896 [cs]*. URL: <https://arxiv.org/abs/2210.14896>.
- (2023a). “DiffusionDB: A Large-scale Prompt Gallery Dataset for Text-to-Image Generative Models”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.
- Wang, Zirui et al. (2023b). *TokenCompose: Grounding Diffusion with Token-level Supervision*. arXiv: 2312.03626 [cs.CV].
- Wang, ZiXuan et al. (2019). “FLOWGAN: Unbalanced Network Encrypted Traffic Identification Method Based on GAN”. In: *Conference on Parallel and Distributed Processing with Applications, Big Data and Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking (ISPA/BDCloud/SocialCom/SustainCom)*.
- Wen, Qingsong et al. (2021). “Time Series Data Augmentation for Deep Learning: A Survey”. In: *International Joint Conference on Artificial Intelligence (IJCAI)*.
- Wen, Qingsong et al. (2023). “Transformers in Time Series: A Survey”. In: *arXiv:2202.07125*. arXiv: 2202.07125 [cs.LG].
- Witteveen, Sam and Martin Andrews (2022). *Investigating Prompt Engineering in Diffusion Models*. arXiv: 2211.15462 [cs.CV].
- Wu, Qiucheng et al. (2023a). “Harnessing the Spatial-Temporal Attention of Diffusion Models for High-Fidelity Text-to-Image Synthesis”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7766–7776.

- Wu, Xiaoshi et al. (2023b). *Human Preference Score: Better Aligning Text-to-Image Models with Human Preference*. arXiv: 2303.14420 [cs.CV].
- (2023c). *Human Preference Score: Better Aligning Text-to-Image Models with Human Preference*. arXiv: 2303.14420 [cs.CV].
- Wu, Xiaoshi et al. (2023d). *Human Preference Score v2: A Solid Benchmark for Evaluating Human Preferences of Text-to-Image Synthesis*. arXiv: 2306.09341 [cs.CV].
- Wu, Xindi et al. (2024). *ConceptMix: A Compositional Image Generation Benchmark with Controllable Difficulty*. arXiv: 2408.14339 [cs.CV]. URL: <https://arxiv.org/abs/2408.14339>.
- Xie, Renjie et al. (2023a). *Additional material for the paper “Rosetta: Enabling Robust TLS Encrypted Traffic Classification in Diverse Network Environments with TCP-Aware Traffic Augmentation”*. <https://cloud.tsinghua.edu.cn/f/7f250d2ffce8404b845e/?dl=1..>
- (2023b). “Rosetta: Enabling Robust TLS Encrypted Traffic Classification in Diverse Network Environments with TCP-Aware Traffic Augmentation”. In: *USENIX Security Symposium (Security)*.
- Xu, Congyuan, Jizhong Shen, and Xin Du (2020). “A Method of Few-Shot Network Intrusion Detection Based on Meta-Learning Framework”. In: *IEEE Transactions on Information Forensics and Security* 15, pp. 3540–3552. DOI: 10.1109/TIFS.2020.2991876.
- Xu, Jiazheng et al. (2023a). “ImageReward: Learning and Evaluating Human Preferences for Text-to-Image Generation”. In: *Thirty-seventh Conference on Neural Information Processing Systems*. URL: <https://openreview.net/forum?id=JVze0YEx6d>.
- Xu, Lei et al. (2019). *Modeling Tabular data using Conditional GAN*. arXiv: 1907.00503 [cs.LG]. URL: <https://arxiv.org/abs/1907.00503>.
- Xu, Shengzhe et al. (2021). “STAN: Synthetic Network Traffic Generation with Generative Neural Models”. In: *Deployable Machine Learning for Security Defense*. Ed. by Gang Wang, Arridhana Ciptadi, and Ali Ahmadzadeh, pp. 3–29.
- Xu, Yilun et al. (2023b). *Restart Sampling for Improving Generative Processes*. arXiv: 2306.14878 [cs.LG]. URL: <https://arxiv.org/abs/2306.14878>.
- Yang, Huiyuan, Han Yu, and Akane Sano (2022a). “Empirical Evaluation of Data Augmentations for Biobehavioral Time Series Data with Deep Learning”. In: *arXiv:2210.06701*. arXiv: 2210.06701 [cs.LG].
- Yang, Qingqing et al. (2022b). “DeepQueueNet: towards scalable and generalized network performance estimation with packet-level visibility”. In: *Proceedings of the ACM SIGCOMM 2022 Conference*. SIGCOMM ’22, 441–457.

- Yang, Xinyu, Zhenguo Zhang, and Rongyi Cui (2022c). "TimeCLR: A self-supervised contrastive learning framework for univariate time series representation". In: *Knowledge-Based Systems* 245, p. 108606. DOI: [10.1016/j.knosys.2022.108606](https://doi.org/10.1016/j.knosys.2022.108606).
- Ye, Weiwei, Zhuopeng Xu, and Ning Gui (2025). *Non-stationary Diffusion For Probabilistic Time Series Forecasting*. arXiv: [2505.04278](https://arxiv.org/abs/2505.04278) [cs.LG]. URL: <https://arxiv.org/abs/2505.04278>.
- Yin, Chuanlong et al. (2018). "An enhancing framework for botnet detection using generative adversarial networks". In: *IEEE International Conference on Artificial Intelligence and Big Data (ICAIBD)*.
- Yin, Yucheng et al. (2022). "Practical GAN-based synthetic IP header trace generation using NetShare". In: *Proceedings of the ACM SIGCOMM 2022 Conference*. SIGCOMM '22, 458–472.
- Yu, Han and Akane Sano (2022). "Semi-Supervised Learning and Data Augmentation in Wearable-based Momentary Stress Detection in the Wild". In: *arXiv:2202.12935*. arXiv: [2202.12935](https://arxiv.org/abs/2202.12935) [eess.SP].
- Yu, Yingwei and Naizheng Bian (2020). "An Intrusion Detection Method Using Few-Shot Learning". In: *IEEE Access* 8, pp. 49730–49740. DOI: [10.1109/ACCESS.2020.2980136](https://doi.org/10.1109/ACCESS.2020.2980136).
- Yuan, Huizhuo et al. (2024). *Self-Play Fine-Tuning of Diffusion Models for Text-to-Image Generation*. arXiv: [2402.10210](https://arxiv.org/abs/2402.10210) [cs.LG].
- Yue, Zhihan et al. (2022). "TS2Vec: Towards Universal Representation of Time Series". In: *Proceedings of the Association for the Advancement of Artificial Intelligence Conference (AAAI)*.
- Yun, Sangdoo et al. (2019). "CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features". In: *arXiv:1905.04899*. arXiv: [1905.04899](https://arxiv.org/abs/1905.04899) [cs.CV].
- Zhang, Hengrui et al. (2024a). "Mixed-Type Tabular Data Synthesis with Score-based Diffusion in Latent Space". In: *The Twelfth International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=4Ay23yeuz0>.
- Zhang, Hongyi et al. (2018). "mixup: Beyond Empirical Risk Minimization". In: *arXiv:1710.09412*. arXiv: [1710.09412](https://arxiv.org/abs/1710.09412) [cs.LG].
- Zhang, Lvmin, Anyi Rao, and Maneesh Agrawala (2023). "Adding Conditional Control to Text-to-Image Diffusion Models." In: *ICCV*.
- Zhang, Qizhen et al. (2021). "MimicNet: fast performance estimates for data center networks with machine learning". In: *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. SIGCOMM '21. Virtual Event, USA, 287–304.

- Zhang, Shiyuan et al. (Aug. 2024b). “NetDiff: A Service-Guided Hierarchical Diffusion Model for Network Flow Trace Generation”. In: *Proc. ACM Netw.* 2.CoNEXT3. DOI: 10.1145/3676870. URL: <https://doi.org/10.1145/3676870>.
- Zhang, Xinchun et al. (2024c). *RealCompo: Balancing Realism and Compositionality Improves Text-to-Image Diffusion Models*. arXiv: 2402.12908 [cs.CV]. URL: <https://arxiv.org/abs/2402.12908>.
- Zhang, Yang et al. (2024d). *SPDiffusion: Semantic Protection Diffusion for Multi-concept Text-to-image Generation*. arXiv: 2409.01327 [cs.CV]. URL: <https://arxiv.org/abs/2409.01327>.
- Zhang, Yi and Difan Zou (2024). “On the Collapse Errors Induced by the Deterministic Sampler for Diffusion Models”. In: *NeurIPS 2024 Workshop: Self-Supervised Learning - Theory and Practice*. URL: <https://openreview.net/forum?id=HKX2kKF1mn>.
- Zhao, Lirui et al. (2024). “DiffAgent: Fast and Accurate Text-to-Image API Selection with Large Language Model”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6390–6399.
- Zhao, Ruijie et al. (2023). “Yet Another Traffic Classifier: A Masked Autoencoder Based Traffic Transformer with Multi-Level Flow Representation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 37.4, pp. 5420–5427. DOI: 10.1609/aaai.v37i4.25674. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/25674>.
- Zhao, Zijian et al. (2022a). “A Few-Shot Learning Based Approach to IoT Traffic Classification”. In: *IEEE Communications Letters* 26.3, pp. 537–541. DOI: 10.1109/LCOMM.2021.3137634.
- Zhao, Ziyi et al. (2022b). “CL-ETC: A Contrastive Learning Method for Encrypted Traffic Classification”. In: *2022 IFIP Networking Conference (IFIP Networking)*, pp. 1–9.
- Zheng, Qinqing et al. (2023). *Guided Flows for Generative Modeling and Decision Making*. arXiv: 2311.13443 [cs.LG]. URL: <https://arxiv.org/abs/2311.13443>.
- Zheng, Wenbo et al. (2020). “Learning to Classify: A Flow-Based Relation Network for Encrypted Traffic Classification”. In: *WWW*.
- Zhou, Xingyi, Vladlen Koltun, and Philipp Krähenbühl (2022a). “Simple multi-dataset detection”. In: *CVPR*.
- Zhou, Xingyi, Vladlen Koltun, and Philipp Krähenbühl (2022b). *Simple multi-dataset detection*. arXiv: 2102.13086 [cs.CV]. URL: <https://arxiv.org/abs/2102.13086>.

- Zhu, Deyao et al. (2024). “MiniGPT-4: Enhancing Vision-Language Understanding with Advanced Large Language Models”. In: *The Twelfth International Conference on Learning Representations*. URL: <https://openreview.net/forum?id=1tZbq88f27>.
- Zou, Difan et al. (2023). “The Benefits of Mixup for Feature Learning”. In: *arXiv:2303.08433*. arXiv: [2303.08433](https://arxiv.org/abs/2303.08433) [cs.LG].