LLM-enabled Intent-driven Service Configuration for Next Generation Networks

Abdelkader Mekrache, *Member, IEEE,* and Adlen Ksentini, *Senior Member, IEEE.*

***Abstract*—Intent-Based Networking (IBN) is a promising paradigm for next generation networks, enabling automated network management based on user-defined business network requirements (Intents). However, current IBN approaches consider that users require expertise in some formal and technical models (e.g., Network Service Descriptors - NSDs) to define these Intents, necessitating substantial effort. A natural progression of IBN systems is to define Intents using natural language instead of structured models. However, dealing with this becomes challenging due to the unstructured and ambiguous nature of natural language. Fortunately, Large Language Models (LLMs) are becoming very powerful in understanding human language, making them well-suited for this task. This paper proposes an LLM-based Intent translation system that allows users to express Intents in natural language, which the system subsequently converts into NSDs. Moreover, we employ a Human Feedback (HF) loop that enables the system to learn from past experiences. Evaluations conducted at the EURECOM 5G facility [1] confirm the effectiveness of our approach in generating accurate NSDs suitable for deployment on an edge computing cluster.**

***Index Terms*—Intent-based networking, next generation networks, natural language, large language models, human feedback.**

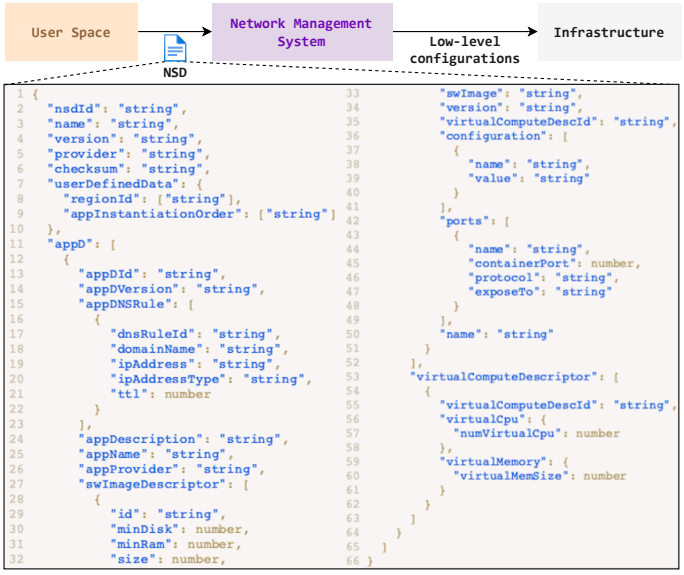

Fig. 1. Network Service Descriptor (NSD) JSON structure.

I. Introduction

Intent-Based Networking (IBN) empowers autonomous networks by enabling users to define desired network outcomes (Intents) at a high level of abstraction. These Intents communicate goals and constraints to the Network Management System (NMS) [2]. Subsequently, The NMS generates the necessary low-level configurations to achieve the specified outcomes. Despite being relatively new, IBN is actively undergoing standardization efforts by organizations such as the 3rd Generation Partnership Project (3GPP), European Telecommunications Standards Institute (ETSI), and TM Forum [3]. It involves five steps, including Intent translation, activation, and assurance. During Intent translation, high-level user Intents are converted into low-level configurations that the NMS can execute. This abstraction minimizes the complexity of low-level configuration details by providing standardized interfaces to express high-level user Intents, known as Northbound Interfaces (NBIs). These NBIs use human-readable languages like JSON or YAML. For instance, in ETSI standards [4], this high-level Intent is defined using the Network Service Descriptor (NSD), a JSON structure designed to deploy network services comprising multiple applications. Using the NSD, users can specify the necessary applications within the network service, their resource consumption, and how the applications are exposed. The NSD is subsequently transmitted to the Network Function Virtualization Orchestration (NFVO) in the NMS, responsible for deploying and updating these services within the infrastructure. In Fig. 1, the NSD structure is presented, and more in-depth information is available in [5].

However, the current model of expressing Intents still requires significant effort in writing the human-readable structures, demanding a detailed comprehension of the format and model specified by the NBI. This process is not always straightforward, and adhering to the structure of these NBIs is time-consuming. A natural evolution for IBN is to move beyond the use of human-readable languages and transition towards natural language. To illustrate, users could simply state *"We request a network service that contains two applications, a Graphical User Interface (GUI), and a database. Both should have at least 1Gb of RAM."*, then the IBN system would translate this statement into low-level configurations, and seamlessly deploy the services on the underlying infrastructure. However, translating natural language into low-level configurations poses a significant challenge. The unstructured nature and the ambiguity of natural language can make it difficult for IBN systems to extract the necessary information and generate accurate low-level configurations.

A. Mekrache is with EURECOM, France (e-mail: abdelkader.mekrache@eurecom.fr).
A. Ksentini is with EURECOM, France (e-mail: adlen.ksentini@eurecom.fr).

Moreover, users from different regions can further complicate this task, as it requires the development of an approach that can comprehend Intents using various human languages, even in the presence of grammatical errors. Fortunately, with the rapid explosion in the Natural Language Processing (NLP) area, Large Language Models (LLMs) become very powerful in understanding human languages [6].

LLMs, such as OpenAI's ChatGPT, are trained on vast amounts of text data, enabling them to understand and generate human-like text across a wide range of topics. LLMs excel in tasks such as language translation, text completion, and question answering, showcasing their ability to comprehend context and generate relevant responses. Utilizing LLMs to translate natural language based Intent to NSDs and subsequently creating low-level configuration from the NSD by the NFVO is a promising approach, addressing limitations in the structure of standardized Intents (e.g., YAML or JSON). LLMs can understand natural language, providing users the freedom to express their needs without constraints, using different languages. However, challenges still need addressing. Pre-trained open-source LLMs, like Code Llama [7], lack training on network-specific data, making them challenging to employ for network management purposes without further training. Furthermore, it is crucial for the system's design to prioritize meeting users' needs. Essentially, the system should understand users' requirements, adapt to their writing styles, and provide NSDs that align with each user's specific desires.

To tackle the outlined challenges, we introduce an Intent translation system based on LLMs to convert natural language Intents into NSDs. However, other IBN procedures such as activation and assurance, are beyond the scope of this paper. Our approach leverages an open-source LLM, trained using few-shot examples from a Knowledge Base (KB), which includes user Intents and their corresponding NSDs. Additionally, we integrate a Human Feedback (HF) mechanism to refine the KB and improve the system's performance over time. The main contributions are:

- We designed an LLM-based Intent translation system that leverages few-shot learning to generate NSDs from natural language. Moreover, we developed a validation agent to ensure that the outputs respect the NSD structure.
- We designed a HF loop to improve the quality of our system over time. The feedback is directly injected into KB, without modifying the LLMs weights. This ensures compatibility even with closed-source LLMs.
- We deployed the solution on a single A100 Nvidia GPU 40GB using the Code Llama LLM [7]. Subsequently, we performed a real-world test on an edge computing cluster managed by the EURECOM 5G facility [1].

The remaining sections of this paper are structured as follows: Section II describes related works on Artificial Intelligence (AI)-based Intent translation in IBN. In section III, we present our Intent translation methodology. Section IV provides an analysis of the solution's performance. Finally, section V concludes the paper.

II. Background and Related Works

IBN systems have attracted significant research attention in recent years, with researchers proposing new AI-based methods for Intent translation to simplify network management. For example, researchers in [8] presented an Intent-based framework for deploying network services, combining NLP and Case-Based Reasoning (CBR) techniques to enhance Intent translation. Additionally, tools like chatbot interfaces have emerged to simplify Intent translation, such as LUMI [9], which employs Google Dialogflow and learning methods to translate user Intents into Nile Intents, which are compiled into programs for network configuration changes. Furthermore, *Mahtout et al.* proposed EVIAN in [10], which utilizes NLP to create semantic Resource Description Framework (RDF) graphs, which are translated into network commands.

At the same time, LLMs are being developed to support a wide range of tasks, including text generation, machine translation, and information retrieval [6]. These models are trained on extensive datasets and can generate text based on input, which could be a question or an instruction referred to as a prompt. LLMs exhibit two fundamental functionalities: semantic comprehension and generative capabilities. Semantic comprehension enables LLMs to understand word meanings, sentence structures, and contextual details. Meanwhile, generative capabilities allow LLMs to produce contextually relevant text based on prompts. Despite excelling in various NLP tasks, LLMs are not explicitly designed for network infrastructure management. To adapt pre-trained LLMs to specific domains, two primary approaches are utilized: supervised fine-tuning and in-context learning [11].

Supervised fine-tuning involves refining the model's performance by training it on specialized datasets that contain explicit reasoning tasks. For example, authors of [12] fine-tuned BERT for Intent detection task. However, fine-tuning requires a large amount of labeled data, which is difficult and time-consuming to collect. In-context learning (e.g., zero-shot and few-shot learning) is an alternative approach to adapting LLMs to new tasks [13]. LLMs can be given a few examples of the desired input-output behavior, and they learn to perform the task based on this information. Few-shot learning has been shown to be effective for a variety of tasks, including code summarization [14], question answering [15], and machine translation [16]. For instance, *Lin et al.* [17] used few-shot learning with GPT-3 to translate natural language into Python code. Nonetheless, previous research has not focused on using open-source LLMs to translate natural language Intents with a HF feature. In our approach, by leveraging recent LLMs, we break away from the constraints of predefined structures, enabling users to express their Intents using their own language, even with grammatical errors. We also introduced a HF loop to help the system learn from previous interactions. In addition, our approach is adaptable to closed-source LLMs, like GPT-3, as we do not modify their internal weights. Moreover, our approach can also be applied to other IBN layers, using the corresponding KB.

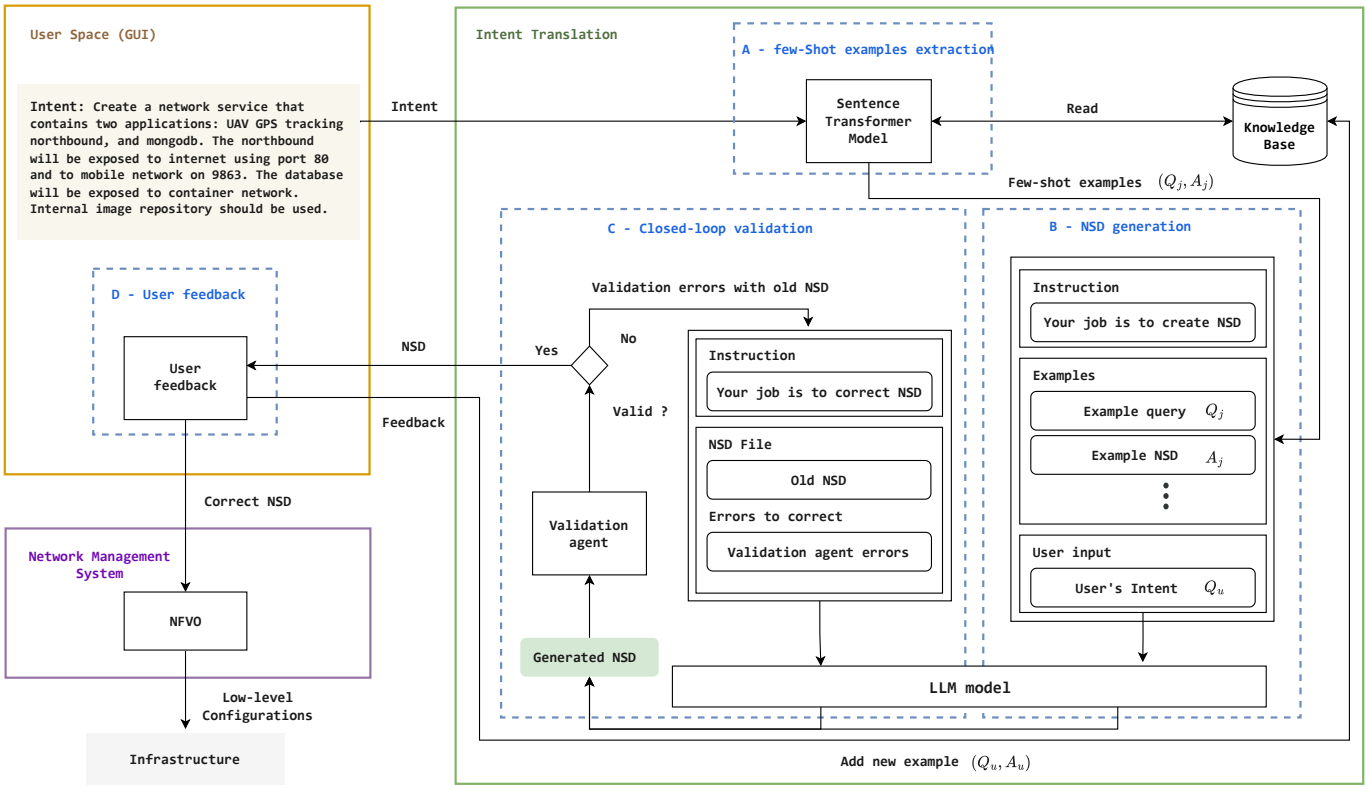

Fig. 2. LLM-based Intent translation design.

III. Methodology

We have devised the Intent translation approach with the capability to accept natural language based Intents as inputs and subsequently produce the NSD. The high-level system design is depicted in Fig. 2, employing a four-stage process to establish an efficient Intent-to-NSD pipeline. In stage 1, historical examples are retrieved from the KB. Moving to stage 2, we use these examples as inputs into an LLM to generate the NSD, enabling the LLM to learn from them. Stage 3 involves validating the generated NSD to ensure its correctness. If any issues are identified, the LLM is requested to correct the NSD. Finally, in stage 4, users provide feedback regarding the quality of the generated NSD. This feedback is incorporated into the KB to improve the system's performance in the future. Below, we discuss each stage in more detail:

A. Few-shot examples extraction

In this stage, the user's Intent, denoted as Q_u, is processed by an AI model that leverages a sentence transformer model. The model's objective is to retrieve relevant examples from the KB where each entry is organized in a tuple structure as (Q_i, A_i). Here, Q_i represents the queries corresponding to historical user Intents, and A_i consists of NSDs previously generated by the LLM, which have been validated either through manual insertion or positive HF. The retrieval of relevant examples involves measuring the cosine similarity between the new input query Q_u and each query Q_i within KB. The cosine similarity indicates the degree of similarity between the queries and is calculated as:

$$S(Q_u, Q_i) = \frac{Q_u \cdot Q_i}{\|Q_u\| \|Q_i\|} \tag{1}$$

where $\cdot$ represents the dot product of the query vectors, and $\|\cdot\|$ denotes the vector's Euclidean norm. Subsequently, the AI model extracts a set of n tuples, denoted as Top_n, representing the most similar historical examples. These tuples, in the form (Q_j, A_j), are then forwarded to the next stage.

B. NSD generation

The previously generated examples are assembled into a prompt, which serves as an input for the LLM. This prompt provides instructions to guide the LLM in its task, with a clear directive stating, *"Your job is to create an NSD."* Alongside this instruction, the prompt includes additional rules defined by the administrator, the few-shot examples (Q_j, A_j), and the Intent Q_u. Upon receiving this prompt, the LLM will generate the NSD as requested. However, it's essential to note that effective performance requires the LLM to have a substantial context window (Input size), particularly when dealing with a large number of few-shot examples. In addition, the LLM should be pre-trained on code-related data, given its primary mission is to generate JSON structures. In this context, we have selected the Code Llama model [7], which was trained on code-related data and has a context window of 100k tokens.

C. Closed loop validation

The validation agent ensures that the LLM's output adheres to the NSD structure illustrated in Fig. 1 through a three-step process: *syntax validation*, the agent examines the NSD JSON structure to ensure its conformity to the NSD schema; *semantic validation*, the agent uses regular expressions to verify the types and values of all parameters in the NSD; *correlation validation*, the agent verifies the consistency of parameter relationships by ensuring that all required parameters are present and valid within their specific contexts. Moreover, it verifies NSDs compatibility with EURECOM's NFVO APIs. Examples of errors detected by the validation agent are explained below:

- *Syntax validation:* These errors include common JSON format errors such as missing opening quotes ("), missing colons (:), and missing closing curly braces (}). They also include errors related to NSD parameters, such as mismatched parameter names (for example, 'apps' instead of 'appD').
- *Semantic validation:* These errors include mismatched data types, errors in accessing nonexistent or duplicated parameters within NSD components, and non respect to regular expressions in defining parameters such as software image path.
- *Correlation validation:* Attributes within NSDs are highly correlated. Common parameters are extracted from outside applications and referenced by an identifier to avoid repetition. For instance, using the "virtualComputeDescriptor" attribute, which specifies virtual memory and CPU requirements, to avoid duplicating the same information for different applications. This field must exist before it is referenced.
- *NFVO's APIs compatibility:* The NFVO requires some additional conditions to be met. For example, the name of the NSD must be in lowercase, and no numerical values are allowed in interface names.

If the NSD passes the validation process, it is forwarded to the users through the GUI. Otherwise, the LLM is requested to correct the NSD using a specific prompt. This prompt provides clear instructions to the LLM, directing it to correct the NSD and including details about the errors detected by the validation agent.

D. User feedback

When users are satisfied with the generated NSDs, they can provide feedback to the administrator for inclusion in KB. Alternatively, the user can correct the NSD and send the corrected one. This mimics the concept of Reinforcement Learning from Human Feedback (RLHF), but indirectly, as we do not alter the LLM's weights. The feedback, i.e., the user's query and the correct NSD (Q_u, A_u), will be used in the future as a few-shot example when there are similar user queries. We envision a future where users can create NSDs directly without requiring human validation, if the trust option is activated.

IV. Performance Evaluation

The section is structured into two subsections: *Experimentation Setup,* which details the experimental setup, and *Experimentation Results,* which presents and analyzes the performance of the LLM-based Intent translation mechanism.

A. Experimentation Setup

Our experimental setup consists of two machines, each equipped with 36 Intel(R) Xeon(R) Silver 4314 CPUs running at 2.40GHz. The second machine has one single Nvidia A100 GPU with 40GB of vRAM. The first machine runs the Kubernetes-based test cluster and the EURECOM 5G facility components [1]. The second machine hosts the Intent translation agent, running the LLM on the single GPU using a quantized version of the LLM. Additionally, it hosts the validation agent. For our experiment, we use the LangChain[1] framework to handle LLMs and ChromaDB[2] to store KB embeddings. We gathered our foundational KB data from EURECOM's past and ongoing research projects. We compared several other popular open source LLMs, including Mistral 7B and Llama 13B, and found that the Code Llama Instruct model with 34B parameters[3] produced the most accurate NSDs on the first attempt. We used the MPNet v2 sentence transformer model[4]. We set the maximum number of tokens to 3000, as the longest NSD in our initial KB is 2000 tokens long. We set n to 4, as it is the maximum number of few-shot examples that fit on the single GPU with the LLM and the sentence transformer model. Additionally, we set the LLM temperature to 0.1. It's important to note that these parameters are flexible and can be adjusted to meet future requirements.

B. Experimentation Results

This subsection evaluates the system's performance. First, we present a video example to demonstrate the system's application in NSD generation. Next, given the primary goal of the system, i.e., understand user Intents and adapt to diverse writing styles, we highlight the system's user satisfaction over time. Finally, we examine the impact of the number of requested applications on NSD generation time. To gather performance data, we solicited feedback from 10 volunteer users. Each volunteer evaluated our platform by creating 10 NSDs, resulting in a total of 100 NSDs for evaluation.

1) Example system application: In this video demonstration (https://youtu.be/SDyBge8WMt0), we showcase how to use the system to create a network service. The user's objective is to deploy a UAV GPS service that collects information from drones and displays it on a GUI. This network service comprises two applications: a UAV GPS tracking northbound and a MongoDB database. The user enters a query describing the Intent in natural language. The system then generates a valid and correct NSD, which is then sent to the NFVO. This latter deployed the network service on an edge cluster.

[1]https://www.langchain.com
[2]https://www.trychroma.com/
[3]https://huggingface.co/TheBloke/CodeLlama-34B-Instruct-GPTQ
[4]https://huggingface.co/sentence-transformers/all-mpnet-base-v2

2) User satisfaction: Users created accounts on the 5G facility [1] and received a brief overview of NSD's functionalities, with no detailed explanation of NSD's structure. They then proceeded to generate 10 NSDs each. At the end of each NSD generation, users were asked to rate the NSD on a scale of 0 to 5, with 5 being the highest score, and then submit the HF containing the correct NSD. Fig. 3 presents the average rating of users for each NSD creation, along with the mean cosine similarity between the initially generated NSD and the corrected NSD submitted by the users. Cosine similarity is a metric of similarity between two NSDs, with a score close to 1 indicating very similar NSDs and a score close to 0 indicating dissimilar NSDs. The results indicate that users were initially not fully satisfied, as they had to make some modifications to the generated NSDs before submitting them to the NFVO. However, the average rating was consistently above 3.75, and the average similarity score was above 0.9, demonstrating that only minor adjustments were needed, mostly related to unfamiliar parameters. Over time, we can see from the figure that both graphs are increasing, i.e., the system has learned from past experiences, resulting in an increase in the average rating to approximately 5 and the average similarity to 1.

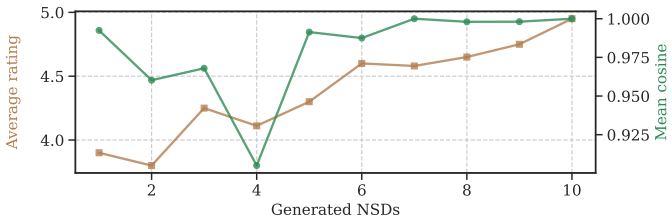

Fig. 3. Mean user satisfaction and similarity score over NSD generations.

3) NSD generation time: Fig. 4 illustrates the relationship between the number of requested applications and the time required to generate a valid NSD. As the number of requested applications increases, the creation time also increases. As illustrated in the figure, the creation time exceeds 1 minute when generating NSDs containing 3 applications and more. This is due to the latency of the LLM, which can become a significant bottleneck for a large number of applications. To ensure a seamless experience for all users, enhancing the efficiency of NSD generation is a consideration for future improvements. One potential approach is to implement task decomposition for Intent translation.

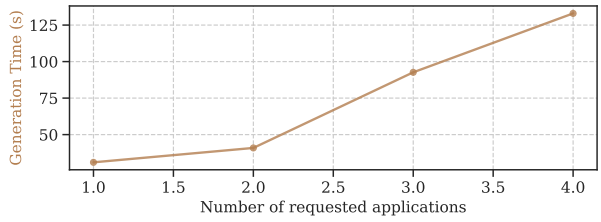

Fig. 4. Effect of varying number of applications on NSD generation time.

V. Conclusion

In this paper, we present an LLM-based approach, addressing the challenge of Intent translation in IBN. Our system enables users to create low-level configurations from natural language, leveraging few-shot learning on an open-source LLM with a HF loop. Performance evaluations conducted on an edge computing cluster demonstrate the system's efficacy in generating accurate NSDs ready for deployment. Additionally, the system demonstrates a remarkable ability to learn from past experiences, continuously adapting and improving its performance to meet the needs of a wide range of users.

Acknowledgment

This work is supported by the European Union's Horizon Program under the 6G-Intense project (Grant No. 101139266).

References

[1] Sagar Arora et al. "A 5G Facility for Trialing and Testing Vertical Services and Applications". In: *IEEE Internet of Things Magazine* 5.4 (2022), pp. 150–155.

[2] A Clemm et al. *Rfc 9315: Intent-based networking-concepts and definitions*. 2022.

[3] Aris Leivadeas and Matthias Falkner. "A survey on intent based networking". In: *IEEE Communications Surveys & Tutorials* (2022).

[4] ETSI. *Network Functions Virtualisation (NFV); Management and Orchestration*. GROUP SPECIFICATION ETSI GS NFV-MAN 001 V1.1.1. Dec. 2014.

[5] Sagar Arora, Adlen Ksentini, and Christian Bonnet. "Lightweight edge slice orchestration framework". In: *ICC 2022-IEEE International Conference on Communications*. IEEE. 2022, pp. 865–870.

[6] Yupeng Chang et al. "A survey on evaluation of large language models". In: *arXiv preprint arXiv:2307.03109* (2023).

[7] Baptiste Rozière et al. "Code Llama: Open Foundation Models for Code". In: *arXiv preprint arXiv:2308.12950* (2023).

[8] R Caldelli et al. "On helping users in writing network slice intents through NLP and User Profiling". In: *2023 IEEE NetSoft*. IEEE. 2023, pp. 545–550.

[9] AS Jacobs et al. "Hey, Lumi! Using Natural Language for Intent-Based Network Management". In: *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association. 2021.

[10] Hocine Mahtout et al. "Using machine learning for intent-based provisioning in high-speed science networks". In: *Proceedings of the 3rd International Workshop on Systems and Network Telemetry and Analytics*. 2020, pp. 27–30.

[11] Jie Huang and Kevin Chen-Chuan Chang. "Towards reasoning in large language models: A survey". In: *arXiv preprint arXiv:2212.10403* (2022).

[12] Sana Shams, Bareera Sadia, and Muhammad Aslam. "Intent Detection in Urdu Queries Using Fine-Tuned BERT Models". In: *2022 16th International Conference on Open Source Systems and Technologies (ICOSST)*. IEEE. 2022, pp. 1–6.

[13] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[14] Toufique Ahmed and Premkumar Devanbu. "Few-shot training LLMs for project-specific code-summarization". In: *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 2022, pp. 1–5.

[15] Zhenyu Li et al. "FlexKBQA: A Flexible LLM-Powered Framework for Few-Shot Knowledge Base Question Answering". In: *arXiv preprint arXiv:2308.12060* (2023).

[16] Xavier Garcia et al. "The unreasonable effectiveness of few-shot learning for machine translation". In: *International Conference on Machine Learning*. PMLR. 2023, pp. 10867–10878.

[17] Jieyu Lin et al. "AppleSeed: Intent-Based Multi-Domain Infrastructure Management via Few-Shot Learning". In: *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*. IEEE. 2023, pp. 539–544.