SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

September 2026, Vol. 69, Iss. 9, 190303:1–190303:17
https://doi.org/10.1007/s11432-025-5015-4

Special Topic: User-Centric Intelligent Networking

Empowering a lightweight large language model agent for intent-based networking: harnessing curriculum learning and LoRA fine-tuning

Qing LI, Xinghui YANG, Yunhan QIN, Yusong ZHOU, Zonghang LI, Long LUO* & Hongfang YU*

School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China

Received 18 October 2025/Revised 24 March 2026/Accepted 21 June 2026/Published online 2 September 2026

Abstract Intent-based networking (IBN) automates network configuration by translating high-level user intents into executable policies. In this work, we focus on intent-driven network topology deployment, a critical scenario in IBN. While large language models (LLMs) show significant promise in enhancing intent interpretation and policy translation, current approaches grapple with abstract intent understanding and the prohibitive costs associated with large-scale proprietary models. To address this limitation, we propose a lightweight IBN agent powered by a novel fine-tuning strategy for open-source medium-scale LLMs, integrating curriculum learning (CL) with low-rank adaptation (LoRA). To address the scarcity of training data for topology deployment, we construct an expert-verified dataset tailored to intent-driven scenarios for robust model fine-tuning. Our proposed fine-tuning algorithm dynamically schedules training samples based on intent ambiguity, thereby enhancing model stability and generalization, particularly under limited data conditions. Experimental results demonstrate that the proposed method achieves a superior task success rate, outperforming GPT-4 (with prompts) by 2.4 times, while maintaining superior explainability and function-call similarity. Crucially, it accomplishes this with significantly lower economic inference costs compared to GPT-4's API usage. This research provides a practical and efficient solution for intent-driven topology deployment in IBN systems, effectively balancing performance, cost, and scalability.

Keywords intent-based networking, large language model, curriculum learning, low-rank adaptation, network automation, agent system

Citation Li Q, Yang X H, Qin Y H, et al. Empowering a lightweight large language model agent for intent-based networking: harnessing curriculum learning and LoRA fine-tuning. Sci China Inf Sci, 2026, 69(9): 190303, https://doi.org/10.1007/s11432-025-5015-4

1 Introduction

For large-scale complex networks, the manual configuration process is often complicated, time-consuming, and error-prone [1]. This challenge is further amplified by computing integration networking and increasingly heterogeneous network services, which make network management more complex [2]. Intent-based networking (IBN) automatically configures networks from user intents, improving the efficiency of network setup and management [3,4]. It supports a wide range of application scenarios, including smart factories, vehicular networks, and emerging intelligent wireless systems [5–7]. Intents may come from a variety of roles, including network operators, administrators, application developers, and end-users. Even within the same business context, individuals may differ in their knowledge and ways of expressing themselves. Traditional IBN technologies typically rely on methods such as intent classification, entity extraction, template filling, and configuration rule mapping. However, these approaches struggles to meet the ever-changing and diverse user demands.

Large language models (LLMs), with their strong natural-language understanding and reasoning capabilities, have shown great potential in network management and optimization, thereby providing new opportunities for improving intent-driven network automation [8–10]. However, existing IBN research remains largely limited to intent translation [11–14], which maps abstract user intents to structured intent languages or network configurations such as routing policies. More advanced methods adopt prompt-based network management agents that convert natural-language intents into executable configuration instructions and drive network automation through function calls. The performance of these methods typically relies on the capabilities of large-scale general LLMs, such as GPT-4 with 1.8 trillion parameters. These approaches have several limitations. LLM APIs are costly, and reliance

* Corresponding author (email: llong@uestc.edu.cn, yuhf@uestc.edu.cn)

on cloud-hosted models raises security and connectivity concerns that can hinder broad deployment [15–17]. Local deployment of large LLMs is also resource-intensive [18], while smaller models often underperform because of constraints such as limited context length [19, 20].

Fine-tuning effectively adapts LLMs to specialized tasks [21–24], yet its application to IBN agents is limited by the scarcity of task-specific data [25, 26]. In this work, we study intent-driven network topology deployment, a representative and challenging IBN task. We construct a dedicated dataset and develop a fine-tuning method that combines curriculum learning (CL) with low-rank adaptation (LoRA) to better exploit limited training data and improve LLM performance on topology deployment. We further design a lightweight IBN agent that outperforms agents built on larger LLMs while requiring only modest resources. Our contributions are as follows.

• We design an LLM-based agent architecture for intent-driven network topology deployment in IBN, enabling automatic generation and configuration of virtual network topologies based on abstract user intents expressed in natural language. The resulting lightweight agent substantially reduces economic and resource overhead.

• We propose an LLM fine-tuning method that integrates ambiguity-aware curriculum learning with LoRA, improving data efficiency and model performance on intent-driven topology deployment tasks.

• We construct an expert-validated dataset for intent-driven topology deployment, addressing the lack of structured training data for topology deployment in IBN.

• We develop multidimensional evaluation metrics for IBN tasks and conduct extensive experiments, showing that the proposed method improves success rate while reducing cost overhead.

2 Related work

IBN aims to automatically translate human users' high-level intents into underlying network policies and device configurations, thereby achieving automated and intelligent network management. However, accurately and efficiently understanding and translating diverse and potentially ambiguous human intents remains a key bottleneck in realising fully autonomous networks [27]. The application of LLMs within IBN systems has significantly advanced network automation, facilitating more efficient and dynamic network management [28–30]. This section reviews key research on LLM-based intent comprehension, LLM-based agents for IBN, and the use of fine-tuning (FT) and CL to enhance performance.

2.1 LLM-based intent comprehension

Recent advances in LLMs have created new opportunities for intent comprehension in IBN, especially for intent classification and intent translation. Through prompt-based interaction, LLMs can interpret natural-language user intents and support downstream network automation.

Intent classification is a fundamental task in IBN, aiming to identify user intents from natural-language inputs. Prior studies have evaluated models such as BERT, GPT-2, RoBERTa, and LLaMA3 for intent classification and have shown their effectiveness in real-world applications such as tourism chatbots [31]. When labeled data are limited, LLMs such as ChatGPT and LLaMA2 can also be used to generate labeled samples for training intent classifiers [32].

Intent translation maps high-level user intents to executable network configurations and is central to IBN. Existing approaches have explored graph embedding, graph attention networks, and deep reinforcement learning for router configuration generation [33]. Prompt engineering has also been widely used for this task [34–38]. For example, LLMs have been applied through few-shot prompting to translate high-level intents into detailed configurations for 5G and 6G networks, including radio access and core networks [34]. Other frameworks use LLMs as translators that convert user intents into machine-readable configuration descriptions with the help of few-shot prompting and supplementary context [35]. Similar ideas have also been adopted in wireless network testbeds, where LLMs support automated experimentation and optimization by generating experiment configurations and control commands [36].

2.2 LLM-based agent for IBN

LLM-based agents for IBN enable end-to-end automated configuration, automating the entire process from intent requirement input to network configuration and management [39]. They typically use LLMs to interpret user natural language requirements, converting them into executable network configuration commands and triggering tools to automate the network configuration. Through diverse prompt engineering designs, agents tailored for various network scenarios can be developed. Examples include agents for automatically configuring P4 routing tables [19], and agents for satellite communication configuration and network task planning [40]. Our previous work,

KlonetAI [41], introduced a network management agent that implements network automation configuration based on user intents expressed in natural language, thereby reducing the complexity of network management. Recent studies have further investigated multi-agent LLM collaboration for network deployment and management [42]. Collectively, these studies highlight the potential of prompt engineering in IBN, while also emphasizing the challenge of managing the limitations of LLMs, particularly the impact of limited context window size on processing larger and more complex inputs [19].

2.3 Fine-tuning and curriculum learning used to enhance performance

Although general LLMs exhibit strong capabilities, networking tasks involve specialized terminology, strict configuration syntax, and complex logical dependencies. Prompt engineering can partially alleviate these challenges through in-context examples, but its effectiveness often degrades in long or multi-turn contexts as adherence to earlier instructions weakens. FT is therefore important for adapting LLMs to networking tasks.

Existing studies have explored both full FT and parameter-efficient FT. LIT [43] combines full FT with RAG and MoE for intent translation and network policy sequence generation, but this design incurs substantial compute and memory overhead and remains prone to overfitting under limited data. Parameter-efficient FT offers a more practical alternative by approaching full FT performance with much lower training cost. NetLLM [44] equips a shared base model with task-specific encoders, adapters, and heads to support routing, traffic control, and fault diagnosis, but it is sensitive to distribution shifts and requires extensive training data. PTC [45] studies secure network script generation with prefix tuning, which has a more restricted capacity to capture task-specific knowledge under complex intent variations. By contrast, LoRA provides a stronger balance between efficiency and performance through low-rank adaptation, making it well suited to low-resource settings and specialized networking tasks. LoRA has also been applied to joint intent classification and slot filling for medium-scale LLMs [46]. Despite these advances, FT for task-specific IBN agents remains largely underexplored. A key reason is the severe lack of high-quality and publicly available annotated datasets in the IBN domain. This gap motivates more data-efficient FT strategies for intent-driven network automation.

To address data scarcity, curriculum learning (CL) offers a promising strategy for improving sample efficiency and generalization. By organizing training samples from easy to hard, CL guides models to learn complex patterns more effectively [47]. This strategy has been widely adopted in natural language processing, where it improves response selection in dialogue systems [48], accelerates model pretraining through text-length-based sequence scheduling [49], and stabilizes the training of GPT models with billions of parameters [50]. These results suggest that CL can also benefit intent-based networking by enabling more robust learning from limited data when intent complexity is properly defined.

Existing research in IBN has made significant progress, particularly in leveraging LLMs for intent understanding and intent translation. However, in the context of intent-driven network automation and configuration, current methods face several key challenges. On the one hand, the context window limitation of LLMs affects the ability of prompt-engineered agents to handle complex requirements. On the other hand, the lack of high-quality, large-scale annotated datasets limits the effectiveness of model fine-tuning for network automation configuration tasks.

As summarized in Table 1, unlike prior prompt-based IBN agents and general fine-tuning approaches, this paper proposes an LLM-based agent architecture for intent-driven network topology deployment within IBN systems and constructs a high-quality network intent dataset to fill the data gap in this task. Additionally, we design an efficient large-model fine-tuning approach that combines LoRA with CL aware of intent ambiguity, specifically tailored to the variability of intent arising from diverse user backgrounds and expression patterns in IBN tasks.

3 LLM agent architecture for IBN

This section introduces the IBN architecture powered by a specialized LLM, providing a detailed description of its components and the interactions among them. As shown in Figure 1, the proposed architecture consists of an user interaction interface, the IBN agent, and a virtual network management platform.

The users can be various roles from different business scenarios, such as network operators, network administrators, application developers, and end users. Each user role has distinct requirements and levels of expertise. The user interaction interface allows users to input their network requirements in natural language, which are then processed by the IBN agent. The agent translates abstract intents into network policies and, through a series of tools, calls the interfaces of the virtual network management platform to execute these policies, ultimately ensuring that the execution results align with the users requirements. Therefore, the key innovation of this architecture lies in the design of the IBN agent. This agent utilizes a medium-scale LLM fine-tuned with IBN expertise and data.

Table 1 Comparison of related work and our approach.

Ref.	Task	Core technique	Agent/tool support	Core challenges & limitations
[32]	Intent classifier	Few-shot prompting	No	Weak constraint retention over long/multi-turn contexts; no autonomous action support
[39]	Intent-based SDN automation	Few-shot prompting + iterative refinement	Yes	Weak constraint retention over long/multi-turn contexts
[40]	Satellite network operation & control	Zero-shot chain-of-thought	Yes	Constraint decay in extended dialogues; unfaithful or inconsistent reasoning chains
[44]	Adapting LLMs for general networking	LLM adaptation (encoder + head + low-rank)	No	High adaptation complexity; large compute & data requirement; sensitivity to data distribution; no autonomous action support
[46]	Intent classification and slot filling	LoRA	No	Sensitivity to limited & diverse training data; no autonomous action support
[45]	Secure network script generation	Prefix-tuning	No	Limited expressivity under broad distribution; domain instability; no autonomous action support
[43]	Wireless network intent translation	Full-parameter fine-tuning + RAG + MoE	No	Very high compute & memory costs; overfitting risk under scarce & diverse data; no autonomous action support
Ours	Intent-driven network topology deployment	Ambiguity-aware CL + LoRA	Yes	Increased training pipeline complexity; longer overall training time

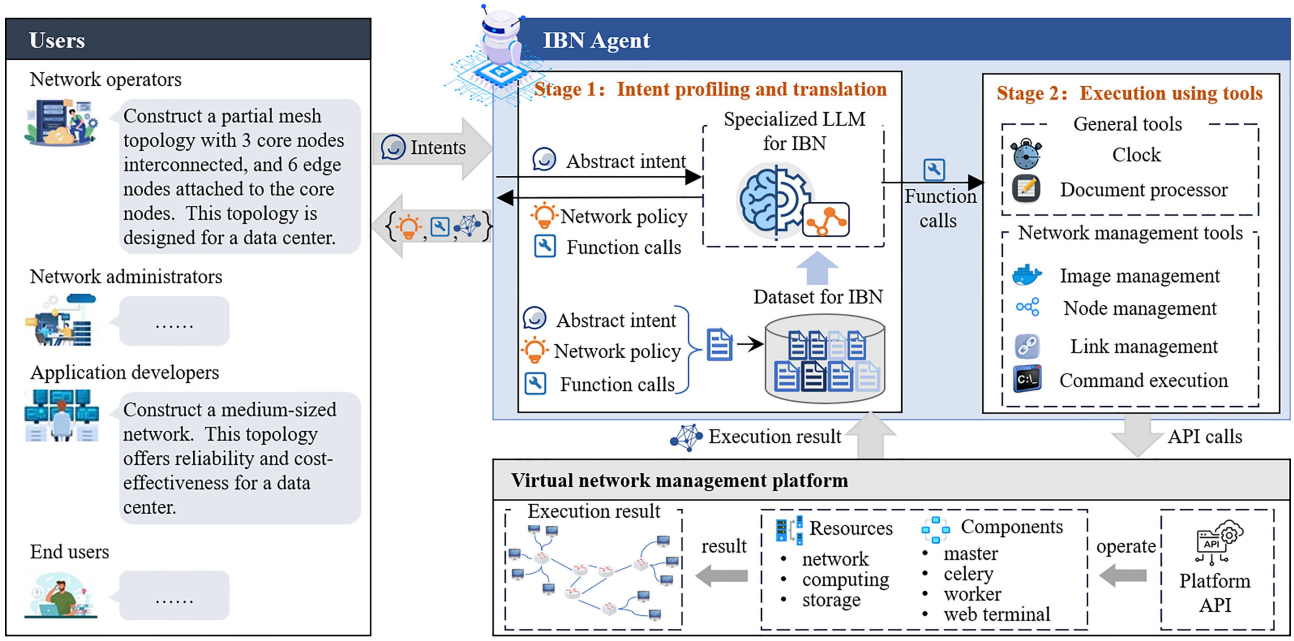

Figure 1 (Color online) The IBN architecture powered by a specialized LLM.

Compared with agent designs optimized through prompt engineering, the proposed agent offers several advantages. First, it does not rely on the performance of large-scale general LLMs, which reduces deployment costs. Second, it eliminates the need to embed specialized knowledge into the prompts, thus avoiding performance issues caused by the context length limitations of medium-scale models. Third, the fine-tuning process, based on domain-specific knowledge and IBN-related data, enhances the agent's specialized performance.

The operation of the proposed IBN agent consists of two main stages that leverage a specialized LLM to automate network configurations based on abstract user intents.

In **stage 1: intent profiling and translation**, the specialized LLM interprets the abstract user intent, formulates the corresponding network policy, and generates a series of appropriate function calls. The network policy and function call plans generated by the agent are then presented to the user for confirmation before execution.

In **stage 2: execution using tools**, the agent proceeds to execute the generated network policy using various tools. These tools are designed to handle tasks such as image management, link management, node management, and command execution. The agent interacts with the virtual network management platform's API through these tools to carry out the necessary operations. The results of the execution are then fed back to the user.

Each piece of execution data, including the user intent, network policy, and function call plan, is stored in a dataset for further optimization of the specialized large model.

Under the proposed IBN architecture, user experience is closely tied to the agent's ability in intent profiling and

translation. If the network policy and function calls generated by the agent deviate from the user's true intent during the initial generation, the user is forced to repeatedly clarify and manually correct, leading to an increase in interaction cycles and a significant drop in satisfaction. Therefore, we consider the successful generation of network policies and function calls that align with the user's intent in the first round as a parsing success, and improving this success rate is a key metric for optimizing user experience. To address this, we propose a cost-effective and efficient solution that leverages a high-quality, small-scale intent parsing dataset, combined with data-efficient fine-tuning methods to enhance the LLM's capability in intent profiling and translation within the agent.

4 Methodology

4.1 Problem formulation

Let $I = \{x_i \mid i = 1, \ldots, |I|\}$ denote the set of user intents in a network scenario, where x_i is the i-th intent. These intents may belong to different categories, such as network topology deployment, network configuration, and network security. Let $I_k \subseteq I$ denote the set of intents in the k-th category. Each category is associated with a category-specific set of informational elements, denoted by $E_k = \{e_{k,m} \mid m = 1, \ldots, |E_k|\}$, where $e_{k,m}$ is the m-th element relevant to category k. If the k-th intent category corresponds to topology deployment intents, the relevant elements are the number of nodes, node types, connection methods, topology type, application scenarios, and functional requirements, giving $|E_k| = 6$.

Even within the same intent type, the expression of intents from different users is likely to vary. For LLMs, the extent and accessibility of key task information provided in the input context have a significant impact on the models reasoning trajectory and output quality. To quantify the level of detail in user expressions, we introduce the concept of ambiguity. We define the ambiguity score $H \in [0, 1)$ of an intent as the proportion of informational elements not included in the intent relative to the total number of relevant elements in its corresponding category. For an intent $x_i \in I_k$, let $E(x_i) = \{e_{k,m} \in E_k \mid e_{k,m} \text{ is specified in } x_i\}$ denote the set of informational elements explicitly specified in x_i. The ambiguity score of x_i is computed as

$$H(x_i) = 1 - \frac{|E(x_i)|}{|E_k|}, \tag{1}$$

where $|E(x_i)|$ denotes the number of specified informational elements in x_i. For simplicity and consistency across intent samples, we use equal weights. The ambiguity score measures structural under-specification as the proportion of missing informational elements, rather than element-wise semantic importance. Addressing unequal contributions would necessitate a principled weighting scheme, which we leave for future work.

Based on the above model, the problem of maximizing the success rate can be formulated as

$$\max \frac{1}{|I|} \sum_{i=1}^{|I|} \text{success}(x_i), \text{where } \text{success}(x_i) = \begin{cases} 1, & \text{if } \forall e_{k,m} \in E(x_i) \text{ satisfied,} \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Let the success function be a binary indicator that takes the value 1 if and only if the generated network policy and its execution outcome satisfy all elements of the user intent, and 0 otherwise. The ultimate objective is to maximize the success rate of fulfilling user requirements within a single dialogue, thereby optimizing user experience. To this end, we propose a method that integrates CL with FT techniques to enhance the performance of a locally trained LLM on end-to-end automatic configuration tasks, thus improving the success rate.

4.2 Dataset generation pipeline

Existing studies show that domain-specific fine-tuning can improve LLM performance in specialized tasks. Since no dataset exists for end-to-end automatic configuration in IBN, we construct a network topology deployment intent dataset to evaluate our approach. It contains 1297 samples, each verified by three domain experts, with user intent, corresponding network policy, and function calls.

The intents in the dataset vary along three dimensions, namely topology type, network scale, and intent ambiguity. Topology type characterizes how nodes and links are organized in a network. As shown in Table 2, the dataset covers 22 topology types, ranging from classic structures such as star, ring, tree, and fully connected topologies to more complex multi-domain networks. Network scale is defined by the total number of nodes and links. Networks with fewer than 40 nodes and links are categorized as small-scale, while those with more than 40 are categorized as large-scale. Topology deployment intents are characterized by six key elements, namely the number of nodes, node

Table 2 Twenty-two network topology types.

No.	Topology type	No.	Topology type
1	Active hub	12	High availability redundant internet egress topology
2	Binary tree	13	Industrial control and IT integrated network
3	Campus multi-service partition topology	14	Linear bus
4	Centralized star	15	Medical multi-domain secure isolation topology
5	Cloud service multi-tenant isolation topology	16	Multi-domain hierarchical secure enterprise network
6	Cross-domain VPN secure interconnection	17	Partial mesh
7	Distributed star	18	Single ring
8	Dual ring	19	Smart campus multi-service integrated topology
9	Financial multi-domain compliance security topology	20	Star-ring hybrid
10	Full mesh	21	Three-layer multi-domain data center topology
11	Hierarchical tree	22	Tree-star hybrid

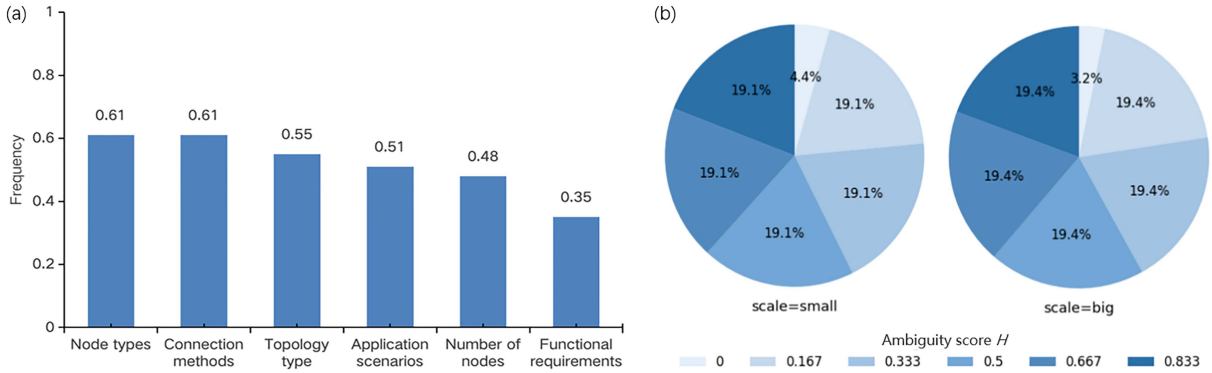

Figure 2 (Color online) Dataset statistics for topology deployment intents. (a) Frequencies of the six elements in user intents; (b) ambiguity score distributions under small-scale and large-scale topology deployment scenarios.

types, connection methods, topology type, application scenarios, and functional requirements. The frequencies of these six elements are shown in Figure 2(a). By varying their combinations, we cover diverse forms of user expression and capture different levels of intent ambiguity. Figure 2(b) further shows the distribution of the dataset across the two network scales under different ambiguity levels.

Each data sample includes not only a user intent but also the corresponding network policy and function calls. To reduce manual annotation cost, these outputs are first generated by the LLM-based agent KlonetAI [41] through prompt engineering. Specifically, we input diverse user intents into KlonetAI to generate network policies and function calls under predefined specifications. We use GPT-4 32k as the base model and augment the prompts with tool descriptions for topology deployment, representative use cases, and interaction guidelines.

The generated outputs are then reviewed and validated by domain experts to ensure semantic accuracy and deployability. Experts correct generation errors and use the virtual network management platform Klonet [51] to verify that the generated policies can be successfully deployed and that the resulting network satisfies all elements specified in the user intent. Overall, nearly 80% of the GPT-4-generated samples required expert correction or refinement. Most revisions were caused by unsupported loop-based tool usage or naming violations that conflicted with platform constraints, while a smaller portion involved structural corrections because the generated topology did not fully match the intended deployment requirements. This process improves both the executability and the semantic reliability of the final dataset. Finally, each validated result is paired with its user intent to form an input-output sample, which is stored in ShareGPT format.

Figure 3 presents two example intents together with their corresponding network policies, function calls, and successfully deployed virtual networks on the network management platform. In Figure 3(a), the user intent describes a small-scale multi-domain topology and explicitly specifies all six elements. The agent first generates a network policy to describe the planned actions and methods, then produces executable function calls, automatically executes them on the platform, and returns the deployed virtual network, which satisfies all requirements in the user intent. In Figure 3(b), the user intent describes a large-scale multi-domain topology and specifies only the topology type. Despite this high level of abstraction, the agent still generates fine-grained executable network policies and implements them accurately.

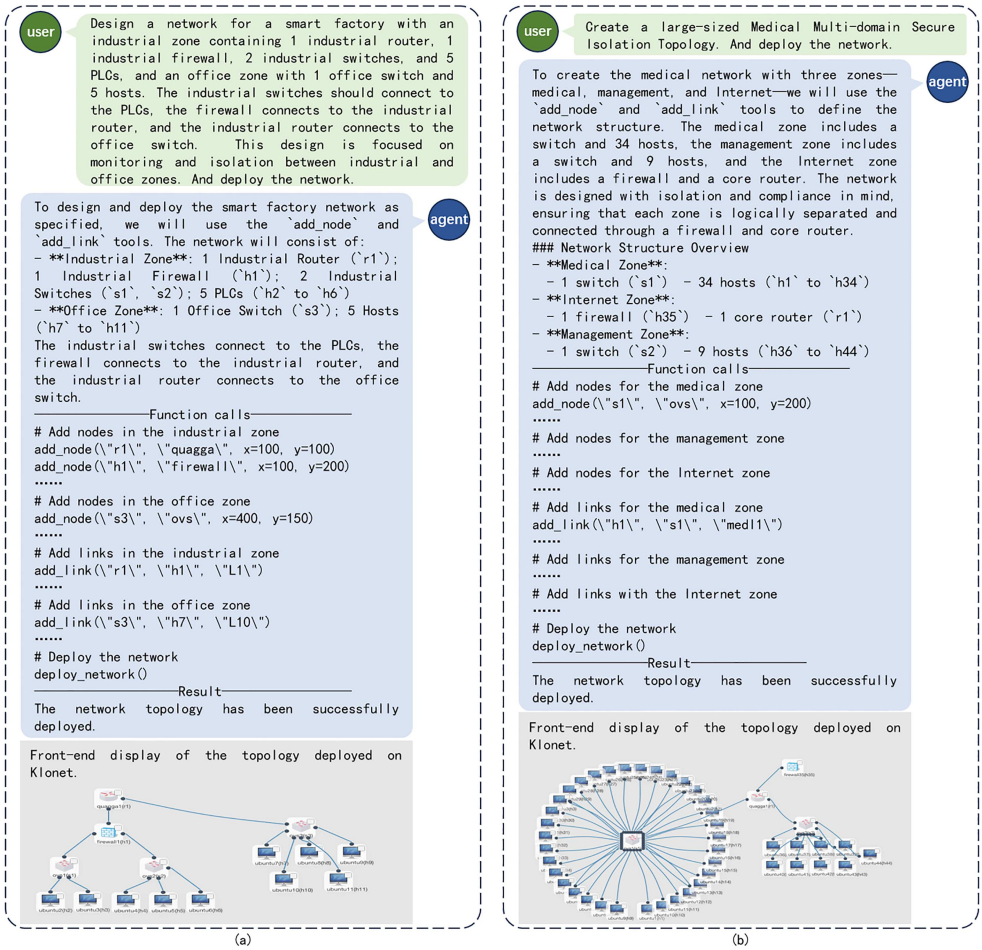

Figure 3 (Color online) Intent representations with corresponding network policies and function calls. (a) Clear-case intent; (b) ambiguous-case intent.

4.3 Fine-tuning algorithm combining CL with LoRA

Leveraging our constructed dataset, we fine-tune a pre-trained LLM for intent-driven network topology deployment. However, direct LoRA fine-tuning on this dataset proves suboptimal. This stems from the dataset's limited scale and the significant variation in intent ambiguity, factors that most existing fine-tuning methods fail to address, thereby hindering model performance and generalization.

To address this issue, we propose a novel fine-tuning method that combines ambiguity-aware curriculum learning with LoRA. As shown in Figure 4, this approach integrates the key components of curriculum learning, a difficulty measurer and a training scheduler, to strategically sample data for LoRA fine-tuning. The fundamental principle of CL involves training models on progressively more challenging samples based on a task-specific criterion. This structured progression, where the difficulty measurer assesses sample complexity and the training scheduler selects data for different stages, enhances stability, accelerates convergence, and improves generalization, particularly with limited datasets.

First, we design a difficulty measurer for IBN tasks based on the intent ambiguity score H, which is computed according to (1). In this work, H serves as the curriculum signal and quantifies the degree of under-specification in topology deployment intents. A sample with a lower H contains more specified informational elements and thus provides richer supervision in the early fine-tuning stage, so it is treated as an easy sample. By contrast, a sample with a higher H leaves more deployment details unspecified and requires stronger inference, so it is treated as a hard sample. The easy-to-hard progression is therefore defined by semantic completeness rather than sample-level training loss. The difficulty measurer then sorts samples in descending order of H and partitions them into B approximately equal buckets, so that samples with similar ambiguity levels are grouped together.

During the model training process, we do not expect the model to encounter difficult samples only in the later training stages, since this would expose them primarily during the low learning-rate period, leading to weaker

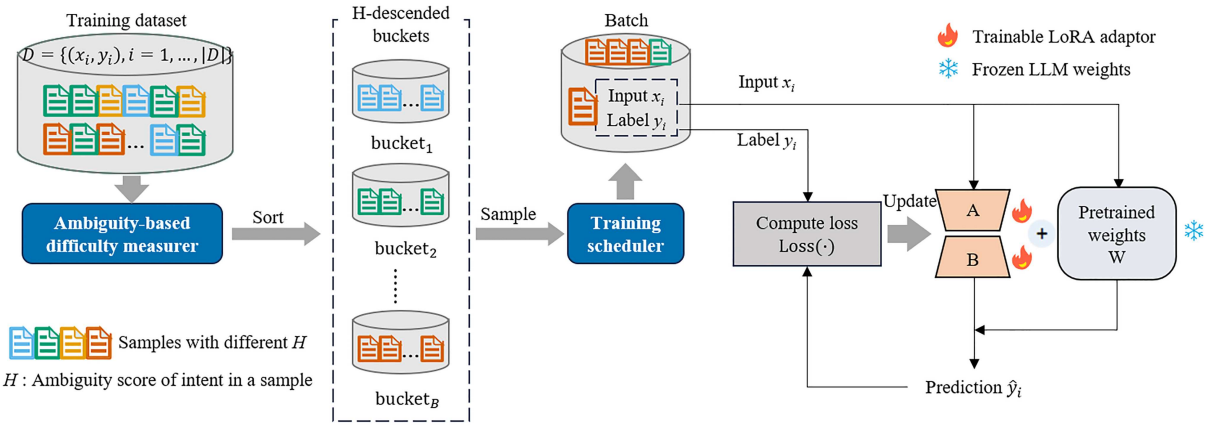

Figure 4 (Color online) Overview of the proposed fine-tuning algorithm combining CL with LoRA.

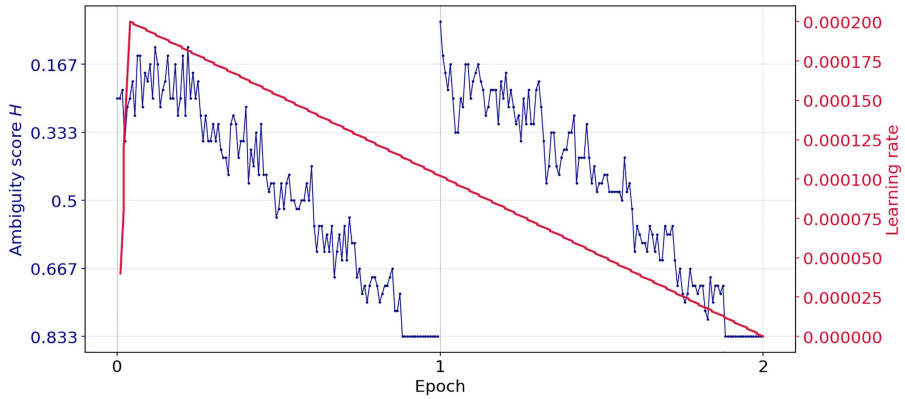

Figure 5 (Color online) Variation of the data ambiguity and the learning rate during the fine-tuning process.

generalization in the final model. Thus, we design the training scheduler based on cyclical curriculum learning, which periodically reintroduces data of varying difficulty levels. This enables the model to repeatedly experience an "easy → difficult → easy" cycle during fine-tuning, rather than a one-way progression of increasing or decreasing difficulty. To achieve this, the training scheduler assigns sampling weights to each bucket using a growth multiplier Grow-P2 [49]. Suppose that the training data are partitioned into B buckets. For the b-th bucket, the multiplier is defined as

$$o_b = 2^{b-1}, \tag{3}$$

and the corresponding step-level sampling probability is computed as

$$p_b = \frac{o_b}{\sum_B o_b}. \tag{4}$$

The training scheduler then samples data from each bucket according to their respective probabilities p_b to form batches for fine-tuning. For example, when $B = 6$, the probabilities are $p_1 = 1.59\%$, $p_2 = 3.17\%$, $p_3 = 6.35\%$, and so on. The training scheduler then samples data without replacement according to the bucket-level sampling probabilities p_b to form each batch. Once all data in a given bucket have been exhausted, its weight is reassigned to zero, meaning that no further samples will be drawn from that bucket.

Figure 5 illustrates the changes in the ambiguity scores of the samples within the batches and the learning rate throughout the fine-tuning process. The ambiguity-based curriculum and learning rate schedule are designed as independent mechanisms to ensure stable training while addressing higher ambiguity in the later stages of training. The blue line represents the variation in ambiguity across the batch, while the red line tracks changes in the learning rate. This ensures that, during different learning rate phases, the model is exposed to user intents with higher ambiguity, thereby improving overall training effectiveness.

All training data is sampled into batches by the training scheduler for model fine-tuning. LoRA is a reparameterization method designed to efficiently fine-tune LLMs by applying low-rank decomposition to reduce the number

of trainable parameters. In the fine-tuning process, LoRA updates the model's weight matrices by introducing low-rank matrices, specifically the LoRA adaptor in Figure 4, which includes matrices A and B. Despite the reduction in parameters, LoRA preserves performance comparable to full-parameter fine-tuning while minimizing both memory consumption and inference overhead.

The overall flow of the proposed fine-tuning algorithm is depicted in Algorithm 1. Let the training dataset be denoted as $D = \{(x_i, y_i), i = 1, \ldots, |D|\}$, where x_i represents the i-th intent in the training dataset, y_i represents the corresponding network policy and function calls. Each instance is associated with a quantified ambiguity score $H(x_i)$. The pretrained model weight matrix is denoted by $W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$, which remains frozen during fine-tuning. Instead of updating W, the LoRA method introduces two trainable low-rank matrices: $A \in \mathbb{R}^{d_{\text{in}} \times r}$ and $B \in \mathbb{R}^{r \times d_{\text{out}}}$, where r is the LoRA rank. First, the dataset is reorganized using the ambiguity-based difficulty measure according to the ambiguity scores $H(x_i)$ and partitioned into B buckets. At the beginning of each epoch, data inside each bucket are shuffled, and bucket weights $\mathbf{O} = [o_1, \ldots, o_B]$ are reset by (3). Sampling probabilities $\mathbf{P} = [p_1, \ldots, p_B]$ are then computed by (4). During training, batches of size s are constructed by probabilistically sampling instances from the non-empty buckets according to $\mathbf{P}$. If a bucket becomes empty, its corresponding weight $\mathbf{O}$ is set to zero and the probability vector $\mathbf{P}$ is recomputed to ensure proper redistribution over the remaining buckets. After each batch is sampled, training is carried out using the LoRA algorithm. Given a sampled batch (x, y), the model output is computed via a forward pass as $\hat{y} \leftarrow xW + xAB$, where xW corresponds to the frozen pretrained transformation and xAB represents the LoRA low-rank update. The discrepancy between the predicted output $\hat{y}$ and the ground-truth label y is quantified by the loss function, and backpropagation is applied to update the adaptive matrices A and the low-rank parameters B. The above procedure is repeated iteratively until all samples in the buckets are consumed, completing one epoch. The curriculum-based sampling and LoRA parameter updates continue for E epochs until convergence, yielding the learned low-rank matrices A and B.

Algorithm 1 LLM fine-tuning algorithm combining CL and LoRA.

Input: Dataset $D = \{(x_i, y_i), i = 1, \ldots, |I|\}$, frozen pretrained weight $W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$, LoRA rank r, learning rate η, buckets classified according to the degree of data ambiguity $Bucket = \{bucket_b, b = 1, \ldots, B\}$, number of epochs E, batch size s;
Output: Trained LoRA matrices $A \in \mathbb{R}^{d_{\text{in}} \times r}$, $B \in \mathbb{R}^{r \times d_{\text{out}}}$;

Initialize $A \in \mathbb{R}^{d_{\text{in}} \times r}$ with $\mathcal{N}(0, 0.02^2)$, $B \leftarrow \mathbf{0}$;
for $epoch = 1$ **to** E **do**
  Reload all buckets (shuffle data inside each $bucket_b$);
  Reset weights $\mathbf{O} \leftarrow [o_1, \ldots, o_B]$ via (3);
  Compute probabilities $\mathbf{P} \leftarrow [p_1, \ldots, p_B]$ via (4);
  while $\exists\, bucket_b \in Bucket$ such that $bucket_b \neq \emptyset$ **do**
    $batch = \emptyset$;
    while $|batch| < s$ **do**
      Based on $\mathbf{P}$, randomly and without replacement extract data d from $bucket_b$;
      $batch = batch \cup d$;
      if $|bucket_b| == 0$ **then**
        $o_b = 0$, and compute probabilities $\mathbf{P} \leftarrow [p_1, \ldots, p_B]$ via (4);
      end if
    end while
    $(x, y) \leftarrow batch$;
    Calculate the predicted output $\hat{y} \leftarrow xW + xAB$;
    $\mathcal{L} \leftarrow \text{Loss}(\hat{y}, y)$;
    Compute gradients $\nabla_A \mathcal{L}$, $\nabla_B \mathcal{L}$;
    Updated parameters $A \leftarrow A - \eta \nabla_A \mathcal{L}$, $B \leftarrow B - \eta \nabla_B \mathcal{L}$;
  end while
end for

5 Performance evaluation

5.1 Experimentation setup

Our experiments were conducted across two distinct server environments to host the agent and the network management platform separately.

Hardware environment: The proposed agent was deployed on an Ubuntu 20.04.6 LTS server, featuring an AMD Ryzen 9 5950X 16-Core Processor CPU at 3.40 GHz, two NVIDIA GeForce RTX 3090 GPUs, and 62 GiB RAM. The virtual network management platform Klonet [51] was deployed on a separate server running Ubuntu 18.04.6 LTS, equipped with two Intel Xeon Gold 5218 CPUs at 2.30 GHz and 125 GB of RAM.

Dataset and base model: Our dataset comprises 1297 samples, which are distributed as 1024 training samples,

Table 3 Fine-tuning parameters.

Parameter	Value	Parameter	Value
Training epochs	2	Gradient accumulation steps	4
Load 4-bit model	True	Learning rate	2e–4
Number of buckets	6	Warmup steps	5
Max sequence length	1024	Optimizer	"adamw_8bit"
Dataset text field	"text"	Weight decay	0.01
Packing	False	LR scheduler type	"linear"
Train batch size	2	Random seed	3407

249 validation samples, and 24 test samples. For fine-tuning, we utilized Meta-Llama-3-8B-Instruct as the base model. The model fine-tuned using Algorithm 1 is subsequently referred to as LLaMA3-FT&CL. All relevant fine-tuning parameters for this process are detailed in Table 3.

Comparison methods: To thoroughly evaluate the performance of our LLaMA3-FT&CL model, we established the following comparison methods.

• **GPT-4-Prompt:** This baseline employs the GPT-4 model, interacting via its API, using carefully designed prompts consistent with those utilized by recent related work KlonetAI [41].

• **LLaMA3-Prompt:** This method uses the Meta-Llama-3-8B-Instruct model with the identical prompt template and task instructions as GPT-4-Prompt.

• **LLaMA3-FT:** This variant represents Meta-Llama-3-8B-Instruct fine-tuned exclusively with the LoRA method on our training set. Its fine-tuning parameters are consistent with those shown in Table 3.

It is important to note that due to hardware limitations, we cannot include a full fine-tuning (without LoRA) variant with CL. Instead, our experimental design focuses on isolating the effect of CL by directly comparing LoRA-only fine-tuning (LLaMA3-FT) with LoRA combined with CL (LLaMA3-FT&CL).

Deployment Environment: Regarding model deployment, GPT-4-Prompt accessed the GPT-4 API for all its operations. In contrast, LLaMA3-Prompt, LLaMA3-FT, and our proposed LLaMA3-FT&CL model were all deployed locally within the same server environment as our agent for consistent evaluation.

5.2 Performance metrics

General metrics such as BLEU and ROUGE commonly used in the literature are not suitable for IBN tasks [34]. To comprehensively evaluate the agent designed for the end-to-end automated configuration task described in this paper, we have designed a set of multi-dimensional evaluation metrics: (1) explainability, (2) similarity, (3) success rate, and (4) cost.

• **Explainability:** To quantify the explainability of the generated network policies, we introduce the E-score. This metric measures the clarity of the network policies generated by the agent, specifically how intuitively and accurately the user can understand these policies. To achieve this, we define a binary metric U_i for each policy: if the network policy output by the agent for the i-th task is judged to be sufficiently clear and easy to understand through manual evaluation, then $U_i = 1$; otherwise, $U_i = 0$. After independently reviewing the N_{total} policies generated by the same agent, the E-score is computed as the average of the corresponding U_i values:

$$E\text{-score} = \frac{1}{N_{\text{total}}} \sum_{i=1}^{N_{\text{total}}} U_i. \tag{5}$$

• **Similarity:** To quantify the similarity between the agent's generated function calls and the ground truth labels, we introduce the S-score. This metric is designed to evaluate the quality of the generated function calls by directly reflecting their consistency with the dataset labels. We employ CodeBLEU [52] for this purpose, as it is commonly employed to measure the combined semantic and syntactic similarity in code generation, translation, and repair tasks, and provides a more accurate reflection of code correctness and usability compared to traditional BLEU. Owing to the stochastic nature of LLMs, functionally equivalent implementations may differ in their realization paths. When scores are computed against a single reference, such implementation variations may lead to abnormal deviations. Therefore, we first normalize the format of the function calls to ensure consistency, and then compute the CodeBLEU score. All weighting coefficients in CodeBLEU are assigned a value of 0.25. Furthermore, certain APIs in Klonet impose strict naming requirements; non-compliant naming may directly result in execution failure. While normalization helps, such issues can sometimes be obscured. Therefore, we incorporate a naming validation mechanism: if a naming issue is detected in the generated function calls, a fixed penalty of $P_{\text{weight}} = -0.2$ is applied

to the CodeBLEU score; otherwise, $P_{\text{weight}} = 0$. We evaluate $P_{\text{weight}} \in \{-0.1, -0.2, -0.3\}$ and observe that the relative ranking of methods remains unchanged, indicating robustness to this hyperparameter choice. The S-score is ultimately calculated using

$$S\text{-score} = \text{CodeBLEU}(y_i, \text{normalized}(\hat{y_i})) + P_{\text{weight}}. \tag{6}$$

• **Success rate:** The success rate is defined as the percentage of tasks successfully completed out of the total number of tasks. This metric serves as a primary indicator of the agent's overall capability and practical utility. Let N_{total} denote the total number of tasks. A task is considered successful only if the generated network policy can be executed on the virtual network management platform (i.e., the deployment completes without errors) and the resulting virtual network satisfies all elements specified in the user intent, as verified by platform feedback and expert validation. The success rate can be computed using

$$\text{Success rate} = \frac{1}{N_{\text{total}}} \sum_{i=1}^{N_{\text{total}}} \text{success}(x_i), \text{where success}(x_i) = \begin{cases} 1, & \text{if } \forall\, e_{k,m} \in E(x_i) \text{ satisfied,} \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

• **Cost:** Cost is defined as the expense incurred for successfully completing a task, measured in \$/intent. This metric quantifies the economic efficiency of the agent, especially when relying on external LLM APIs. In this study, we utilize GPT-4(8k) as the online LLM, which charges \$0.03 per 1k tokens for the input and concatenated prompt section, and \$0.06 per 1k tokens for the output section. We calculate the total token-based cost for each task by summing the costs of input and output tokens. To reflect the cost of a successfully delivered intent, this total token-based cost is then divided by the overall success rate, as shown in

$$\text{Cost} = \frac{0.03 \times (\text{prompt tokens} + \text{input tokens}) + 0.06 \times \text{output tokens}}{\text{Success rate}}. \tag{8}$$

5.3 Experimental results and analysis

We independently repeated the experiment three times on the same test set and recorded and analyzed the results for the four metrics mentioned above. The test set contains 24 structurally selected samples, covering two network scales, four topology types per scale, and three ambiguity levels for each topology.

The proposed method exhibits better explainability compared to other LLaMA3-based agents. As shown in Figure 6(a), the E-score of LLaMA3-FT&CL reaches 84.7%, closely matching GPT-4-Prompt and significantly outperforming LLaMA3-FT, which only achieves 65.3%. GPT-4, a powerful pre-trained model, excels at understanding complex language and generating content aligned with human understanding, allowing GPT-4-Prompt to effectively parse user intents and generate accurate network policies. After fine-tuning with a CL strategy based on the constructed dataset, the proposed method achieves an E-score similar to that of GPT-4, highlighting the effectiveness of our approach. In contrast, LLaMA3-FT suffers from high-ambiguity intents in the dataset, which reduces explainability. As shown in Figure 6(b), the ranking of the four agents E-scores under varying levels of intent ambiguity aligns with overall test results. Low, medium, and high ambiguity levels are defined as $H \in [0, 0.33]$, $H \in (0.33, 0.66]$, and $H \in (0.66, 1)$, respectively. Agents exhibit the lowest E-score on high-ambiguity intents, which lack clear goals and context, leading to more divergent results. In contrast, medium-ambiguity intents offer a moderate degree of uncertainty, allowing the model to perform optimally, resulting in slightly higher E-scores than low-ambiguity intents. In summary, the proposed training approach outperforms other LLaMA3-based agents in understanding user intents and generating interpretable network policies. It strikes a balance between usability and controllability, enabling quick adoption by non-experts and efficient planning by professional network operators, which is essential for IBN.

The LLM fine-tuned with task-specific training data shows a significant improvement in similarity. As illustrated in Figure 7(a), both LLaMA3-FT&CL and LLaMA3-FT achieve an average S-score of around 0.7, which is more than twice that of the comparison methods using prompts. As shown in Figure 7(b), the similarity of the fine-tuned models remains consistently high across both small-scale and large-scale network intent scenarios. However, for intents requiring large-scale network topology deployment, the average S-score of GPT-4-Prompt drops sharply to 0.17. The main reasons for this discrepancy are twofold: (1) GPT-4-Prompt tends to use macro-style generation patterns, such as for loops, to replace repetitive tool invocations, which significantly reduces its output length. However, these patterns are non-executable and explicitly prohibited in the prompt, and (2) GPT-4-Prompt frequently encounters naming non-compliance issues, which trigger the penalty weight in the S-score computation, thereby reducing its score. Although the deployment rules are explicitly specified in the prompt, the generative

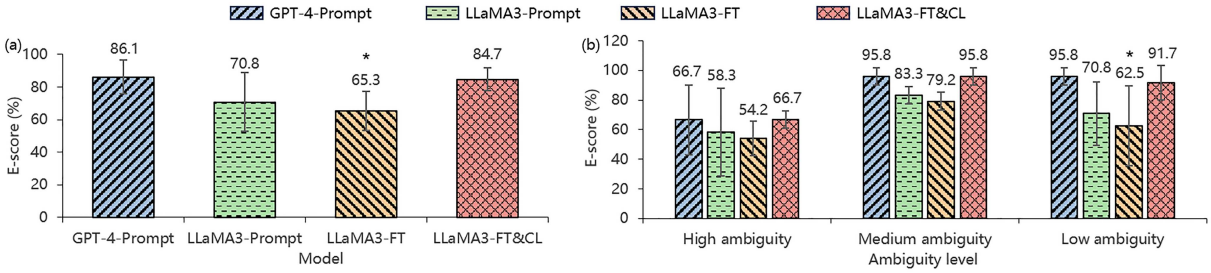

Figure 6 (Color online) *E*-score comparison across agents. (a) Overall performance; (b) performance under varying intent-ambiguity levels. Error bars denote standard deviations across three independent runs. Statistical significance was evaluated using the McNemar test on sample-level paired outcomes. $*p < 0.05$ vs. LLaMA3-FT&CL group.

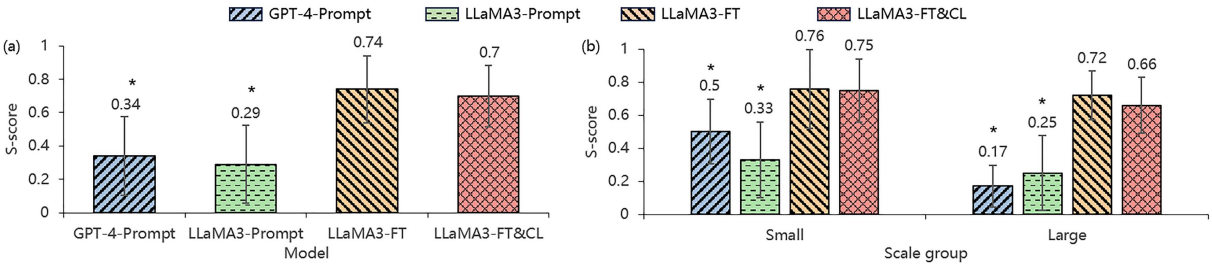

Figure 7 (Color online) *S*-score performance across agents. (a) Overall comparison; (b) breakdown by network-scale intent. Error bars denote standard deviations across three independent runs. Statistical significance was evaluated using the Wilcoxon signed-rank test on sample-level paired outcomes. $*p < 0.05$ vs. LLaMA3-FT&CL group.

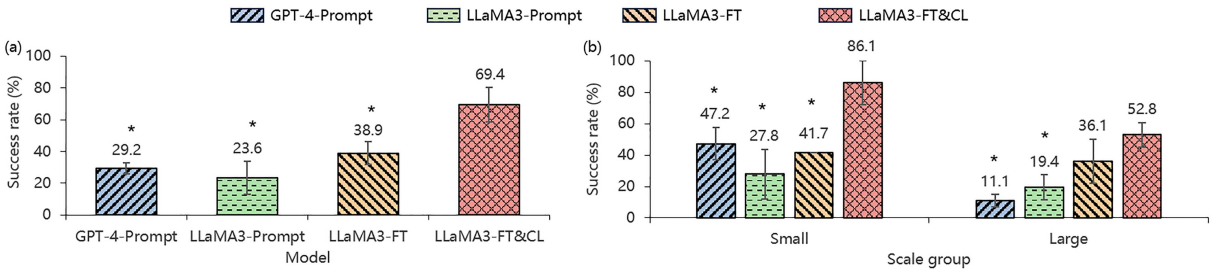

Figure 8 (Color online) Success rate performance across agents. (a) Overall comparison; (b) breakdown by network-scale intent. Error bars denote standard deviations across three independent runs. Statistical significance was evaluated using the McNemar test on sample-level paired outcomes. $*p < 0.05$ vs. LLaMA3-FT&CL group.

models inherent randomness and omission risk often lead to missed constraints, resulting in unstable outputs. It can be anticipated that as the prompt size increases, this issue will become more pronounced. In summary, fine-tuning with task-oriented, domain-specific datasets yields superior and more robust performance compared to prompt-based approaches.

The proposed method greatly improves the task success rate. As shown in Figure 8(a), LLaMA3-FT&CL achieves the highest success rate overall, reaching 2.4 times that of GPT-4-Prompt, 2.9 times that of LLaMA3-Prompt, and 1.8 times that of LLaMA3-FT. The fine-tuned model outperforms models trained using prompt engineering overall. This finding provides two key insights: (1) future research can focus on domain-specific fine-tuning of smaller models to reduce the cost and dependency associated with large-model prompt engineering; (2) it confirms the strong adaptability potential of smaller models within specific domains, offering valuable guidance for subsequent research and industrial applications.

Meanwhile, the quality of the constructed dataset is validated. Although the dataset size is limited, each sample has been carefully reviewed and verified by experts. Experimental results demonstrate that a high-quality, small-scale dataset can significantly enhance the performance of fine-tuned models, reinforcing the papers contribution in dataset construction.

The success rate of LLaMA3-FT&CL is notably higher than that of LLaMA3-FT, which confirms the motivation for incorporating CL. CL helps mitigate the interference caused by high-difficulty data during fine-tuning and improves model performance. As illustrated in Figure 8(b), LLaMA3-FT&CL demonstrates the best performance in both small-scale and large-scale network deployments. In contrast, GPT-4-Prompt still exhibits a significant

Table 4 Failure mode rates across all test cases for different methods.

Failure category	Failure mode	GPT-4-Prompt	LLaMA3-Prompt	LLaMA3-FT	LLaMA3-FT&CL
Unsupported/invalid tool usage	Use of loops	45.83%	9.72%	4.17%	1.39%
	Missing deploy command	0.00%	2.78%	1.39%	0.00%
	Code error	0.00%	1.39%	0.00%	1.39%
Naming violations	Wrong node naming	40.28%	18.06%	20.83%	0.00%
	Duplicate node names	0.00%	0.00%	1.39%	0.00%
	Wrong link naming	4.17%	9.72%	1.39%	0.00%
Topology construction errors	Extra nodes	4.17%	8.33%	26.39%	5.56%
	Wrong node images	0.00%	26.39%	9.72%	1.39%
	Missing nodes	1.39%	9.72%	1.39%	8.33%
	Extra links	1.39%	0.00%	4.17%	0.00%
	Wrong links	2.78%	13.89%	6.94%	6.94%
	Missing links	4.17%	5.56%	12.50%	8.33%

performance degradation on large-scale tasks, due to the same underlying reason as observed in the similarity metric, which is violations of system rules during tool invocation and naming. These violations result in undeployable outputs and consequently a sharp drop in success rate. In summary, the combination of curriculum learning and fine-tuning enables LLaMA3-FT&CL to produce more stable and reliable results.

To better characterize model failures beyond aggregate success rate, we categorize unsuccessful cases into three groups: unsupported/invalid tool usage, naming violations, and topology construction errors. Table 4 reports the occurrence rate of each failure mode across all test cases for different methods. The results show clear differences in failure patterns across methods. To provide a more intuitive understanding of the observed failure patterns, Figure 9 presents a representative case study on a multi-zone financial network deployment intent. For GPT-4-Prompt, the dominant failure modes are loop-based tool usage and wrong node naming. In particular, loop-based errors mainly occur in large-scale deployment intents, where repetitive tool invocations are required. This is noteworthy because the prompt explicitly states that loop-based tool usage is unsupported and that node names must follow the platform constraints, i.e., they may only consist of letters and digits without special characters. The fact that these errors still occur frequently suggests that, under long-context generation, prompt-level constraints alone are insufficient to reliably enforce platform rules. The failures of LLaMA3-Prompt and LLaMA3-FT are more evenly distributed across wrong node image, wrong node naming, wrong links, missing links, and extra nodes. LLaMA3-Prompt is more prone to wrong node image errors, mainly caused by hallucinating unsupported images. The most frequent failure mode of LLaMA3-FT is extra nodes, especially in multi-domain topology intents, suggesting that LoRA-only fine-tuning improves executability but tends to over-generate topology components when the intent is structurally complex or partially under-specified. By contrast, LLaMA3-FT&CL substantially reduces failures in unsupported/invalid tool usage and completely eliminates naming violations, indicating that ambiguity-aware curriculum fine-tuning improves the models ability to follow platform constraints while better controlling topology generation.

Using a locally deployed model eliminates the costs associated with using cloud-based LLMs. Table 5 presents a comparison of operational costs among agents powered by different LLMs. Among them, only GPT-4-Prompt uses the cloud-based large model via API. The economic cost of successfully executing an intent can be calculated using (8), amounting to 0.33758 USD. The prompt contains descriptive information about the Klonet tools associated with the users intent. For the intent types tested in this study, the prompt length reaches approximately 1900 tokens. As the range of user intent categories expands, the prompt length is expected to grow linearly, thereby causing a further increase in the invocation cost of GPT-4-Prompt. Long-term reliance on online models for network deployment and management would ultimately result in significant economic costs. In contrast, the other three agents are based on locally deployed LLMs, which incur no additional expenditure and can therefore significantly reduce operational costs.

We further report inference time in Table 5, since latency is important for real-time network management. Overall, the locally deployed LLaMA3-based methods are slower than GPT-4-Prompt in terms of wall-clock inference time per generation. This gap is not only due to model differences, but also reflects the deployment setting. GPT-4 is served through a highly optimized cloud infrastructure, whereas the LLaMA3-based methods are deployed locally on a server with an AMD Ryzen 9 5950X CPU, two RTX 3090 GPUs, and 62 GiB RAM. In addition, the local models generate substantially more output tokens than GPT-4-Prompt, which further increases autoregressive decoding time. Among the local methods, LLaMA3-FT&CL achieves the highest success rate, which reduces the average inference time per successful execution to 150.47 s, substantially lower than LLaMA3-Prompt

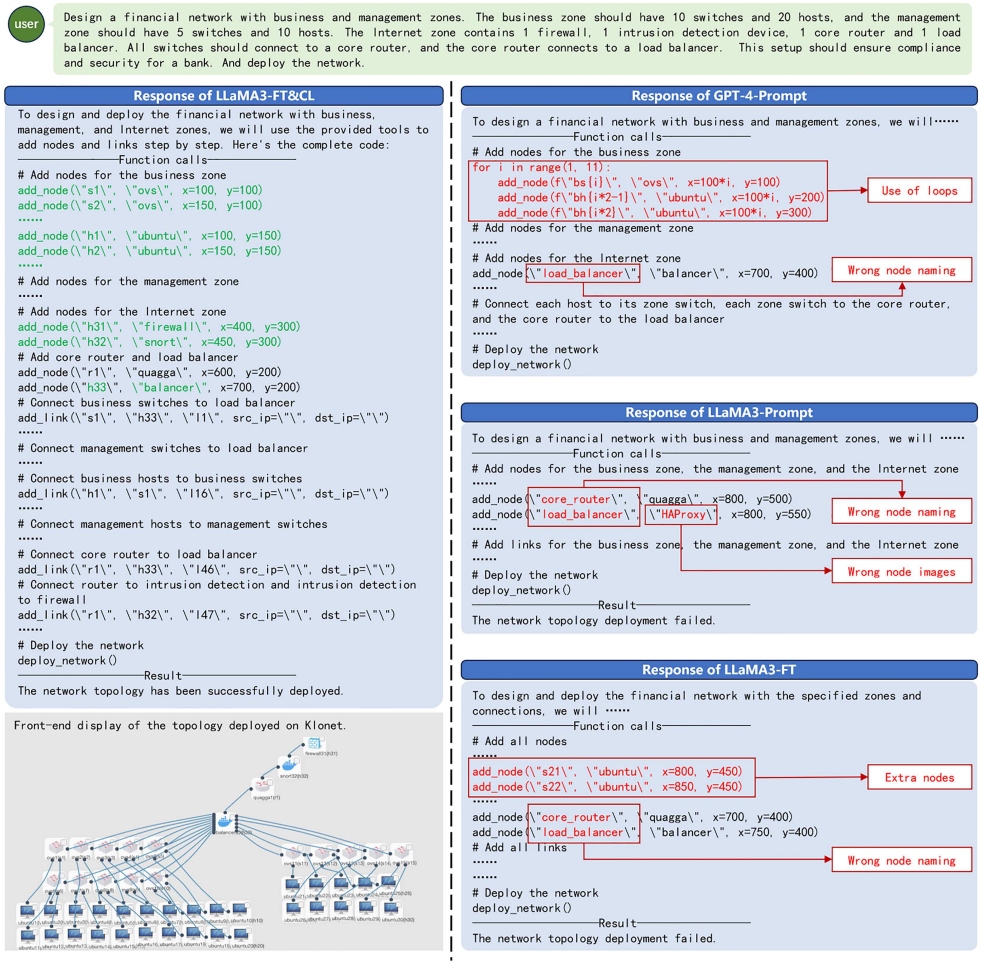

Figure 9 (Color online) Qualitative comparison of model outputs for a financial network deployment intent.

Table 5 Agent operational metrics comparison.

Metric	GPT-4-Prompt	LLaMA3-Prompt	LLaMA3-FT	LLaMA3-FT&CL
Input tokens (mean ± std)	1908 ± 23	1908 ± 23	54 ± 23	54 ± 23
Output tokens (mean ± std)	684 ± 194	1205 ± 705	1698 ± 962	1759 ± 1334
Average cost per generation ($)	0.09828	0	0	0
Inference time per generation (mean ± std, s)	23.52 ± 9.21	76.03 ± 47.71	120.23 ± 73.72	104.43 ± 61.27
Average success rate (%)	29.2	23.6	38.9	69.4
Average cost per successful execution ($)	0.33758	0	0	0
Average inference time per successful execution (s)	80.54	322.16	309.07	150.47

(322.16 s) and LLaMA3-FT (309.07 s). Therefore, although GPT-4-Prompt has the best raw latency, LLaMA3-FT&CL method provides a better balance between effectiveness, deployment autonomy, and economic cost for practical IBN deployment.

5.4 Ablation study: comparison of curriculum learning schedules

To validate the effectiveness of our proposed curriculum learning schedule, we conducted additional experiments comparing it with other approaches. The following methods were considered.

- **LLaMA3-FT&CL:** Our proposed method, where the training schedule cycles through easy, difficult, and easy phases, with 6 buckets.
- **Cyclic Curriculum Learning ($B = 3$):** A variant of our cyclic approach with 3 buckets.
- **Monotonic curriculum learning ($B = 6$):** A classical curriculum learning approach, where training progresses from easy to difficult phases, with 6 buckets.
- **LLaMA3-FT:** A baseline method where the model is fine-tuned without curriculum learning.

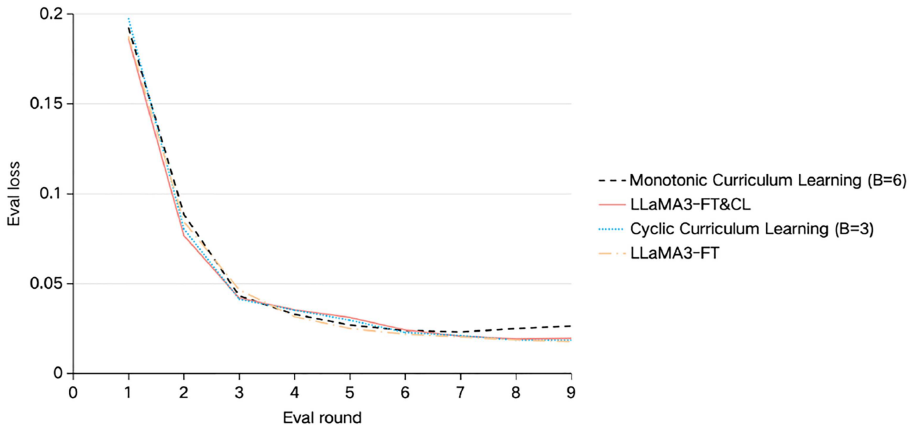

Figure 10 (Color online) Validation loss curves of different ablation methods during training.

Table 6 Performance comparison of ablation methods for curriculum learning schedules.

Method	Total training time (min)	Memory consumption (GB)	Success rate (mean ± std)
LLaMA3-FT&CL	85.61	1.568	69.44% ± 10.94%
Cyclic Curriculum Learning ($B = 3$)	53.44	1.566	56.94% ± 5.20%
Monotonic Curriculum Learning ($B = 6$)	85.24	1.566	45.83% ± 6.80%
LLaMA3-FT	49.65	1.566	38.89% ± 7.08%

These methods were evaluated using the same set of training data, with all other hyperparameters held constant.

In this ablation study, we compared the impact of different curriculum learning strategies on model performance. As shown in Figure 10, overall, all methods exhibit a similar trend of decreasing validation loss during training, indicating that they are all effective in driving model convergence. In contrast, the monotonic curriculum learning ($B = 6$) shows a slightly higher validation loss in the final stages, suggesting that a unidirectional easy-to-difficult training strategy may weaken the model's final convergence and generalization ability when high-difficulty samples are continuously introduced in the later stages. This phenomenon indicates that using only a unidirectional progressive curriculum may not necessarily yield the optimal results. A suitable cyclic scheduling approach is more beneficial for maintaining training stability and achieving better final performance. Table 6 summarizes the total training time, peak GPU memory consumption during training, and final test-set success rate of each method. Overall, LLaMA3-FT&CL achieves the highest success rate, indicating that a carefully designed cyclic curriculum schedule can more effectively improve model performance on the network deployment task. This improvement comes at the cost of a more complex training schedule, which results in a longer total training time and slightly higher memory consumption.

Compared with the fine-tuning-only method, all methods incorporating curriculum learning achieve higher success rates, demonstrating that the ambiguity-aware curriculum learning strategy effectively enhances the performance of fine-tuned models on network deployment tasks. A further comparison of different curriculum scheduling strategies shows that cyclic curriculum learning methods consistently outperform the monotonic curriculum learning ($B = 6$). When considered together with the convergence behavior of validation loss, the monotonic curriculum learning ($B = 6$) exhibits a slightly higher loss in the final stage than the other methods, suggesting that a unidirectional easy-to-difficult training strategy may adversely affect the models final convergence quality and generalization ability when high-difficulty samples are continuously introduced in the later stages. By contrast, cyclic curricula alleviate this issue by reintroducing low-difficulty samples after high-difficulty stages, thereby achieving better final performance.

The cyclic curriculum learning ($B = 3$) reduces the total training time because it adopts a coarser-grained bucket partition, but its success rate is slightly lower than that of LLaMA3-FT&CL. Given that this dataset contains only six discrete ambiguity levels, the 6-bucket partition used in LLaMA3-FT&CL corresponds to the finest-grained partition aligned with the data distribution. The comparison with the cyclic curriculum learning ($B = 3$) further suggests that finer-grained bucket partitioning helps improve model performance.

In summary, the proposed method outperforms the other three comparison methods in terms of similarity and success rate. It ranks second to GPT-4-Prompt in explainability while avoiding the costs associated with accessing cloud-based large models via API. Experimental results demonstrate that the constructed dataset significantly

enhances the performance of fine-tuned models. Additionally, the proposed fine-tuning method, which combines CL and LoRA, effectively mitigates the interference of hard samples during the fine-tuning process, leading to improved final model performance.

6 Conclusion

In this paper, we present a lightweight LLM-based agent for IBN to enable efficient and automated topology deployment. To address high resource costs and limited domain data, we propose an ambiguity-aware curriculum fine-tuning method with LoRA and construct an expert-validated dataset covering diverse topology types, scales, and ambiguity levels. Experiments show that our method achieves the best success rate among baselines while maintaining strong explainability and function-call similarity. The locally deployed agent also delivers performance comparable to cloud-based models at much lower cost, making it a practical option for large-scale deployment. Overall, the results show that combining curriculum learning with LoRA is an effective way to adapt medium-scale LLMs to topology deployment tasks in IBN. Our study is currently limited to topology deployment scenarios, and future work will extend the framework to other intent categories, improve dataset construction, and explore more expressive ambiguity modeling.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant No. 62394324).

References

1 Rafiq A, Mehmood A, Ahmed Khan T, et al. Intent-based end-to-end network service orchestration system for multi-platforms. Sustainability, 2020, 12: 2782
2 Zhang H, Yu C, Quan W, et al. Fundamental research on computing integration networking (in Chinese). Acta Electron Sinica, 2022, 50: 2928–2934
3 Leivadeas A, Falkner M. A survey on intent-based networking. IEEE Commun Surv Tutorials, 2022, 25: 625–655
4 Harbaoui M, Martini B, Castoldi P. Intent-based networking: current advances, open challenges, and future directions. In: Proceedings of the 23rd International Conference on Transparent Optical Networks (ICTON), 2023. 1–5
5 Haj Hussein D, Ibnkahla M. Toward intelligent intent-based network slicing for IoT systems: enabling technologies, challenges, and vision. IEEE Trans Netw Serv Manage, 2025, 22: 3480–3495
6 Minhas S, Jaswal R, Sharma A, et al. Revolutionizing networking: a comprehensive overview of intent-based networking. In: Proceedings of the 2024 International Conference on Emerging Innovations and Advanced Computing (INNOCOMP), 2024. 463–468
7 Shao Y, Xu H, Feng W, et al. RIS-enabled integrated sensing and communication: key technologies and research progress (in Chinese). Chinese J Radio Sci, 2026, 41: 213–231
8 Hang C N, Yu P D, Morabito R, et al. Large language models meet next-generation networking technologies: a review. Future Internet, 2024, 16: 365
9 Cui Q, You X, Wei N, et al. Overview of AI and communication for 6G network: fundamentals, challenges, and future research opportunities. Sci China Inf Sci, 2025, 68: 171301
10 Luo L, Huang Y, Chen X, et al. Location matters: LLM-guided joint optimization of in-network aggregation placement and routing for DML workloads. IEEE Trans Netw Sci Eng, 2026, 13: 5978–5991
11 D'Urso S, Fontana M, Martini B, et al. Enhancing intent acquisition and translation with large language models and intelligent chatbots: a DHCP use case. In: Proceedings of the 10th International Conference on Network Softwarization (NetSoft), 2024. 19–24
12 Li Q, Cai J, Xu J, et al. Knowledge-enabled intent-driven network configuration generation for 5G core networks. In: Proceedings of the 24th International Conference on Communication Technology (ICCT), 2024. 827–833
13 Zaid H, Safari P, Shariati B, et al. Multi-agent design for LLM-assisted network management. In: Proceedings of the 2025 Optical Fiber Communications Conference and Exhibition (OFC), 2025. 1–3
14 Van Tu N, Yoo J H, Hong J W K. Towards intent-based configuration for network function virtualization using in-context learning in large language models. In: Proceedings of the 2024 IEEE Network Operations and Management Symposium (NOMS), 2024. 1–8
15 Bang F. Gptcache: an open-source semantic cache for LLM applications enabling faster answers and cost savings. In: Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023), 2023. 212–218
16 Rigaki M, Catania C, Garcia S. Hackphyr: a local fine-tuned LLM agent for network security environments. ArXiv:2409.11276
17 Kim K, Kim J, Chung H, et al. Cost-efficient VM selection for cloud-based LLM inference with KV cache offloading. In: Proceedings of the 18th International Conference on Cloud Computing (CLOUD), 2025. 175–185
18 Li N, Guo S, Zhang T, et al. The MoE-empowered edge LLMS deployment: architecture, challenges, and opportunities. IEEE Commun Mag, 2025, 63: 164–171
19 Wang C, Scazzariello M, Farshin A, et al. NetConfEval: can LLMs facilitate network configuration? In: Proceedings of the ACM on Networking, 2024. 1–25
20 Wang F, Zhang Z, Zhang X, et al. A comprehensive survey of small language models in the era of large language models: techniques, enhancements, applications, collaboration with LLMs, and trustworthiness. ACM Trans Intell Syst Technol, 2024, 16: 1–87
21 Schmirler R, Heinzinger M, Rost B. Fine-tuning protein language models boosts predictions across diverse tasks. Nat Commun, 2024, 15: 7407
22 Wu F, Li Z, Li Y, et al. FedBiOT: LLM local fine-tuning in federated learning without full model. In: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'24), 2024. 3345–3355
23 Xia Y, Kim J, Chen Y, et al. Understanding the performance and estimating the cost of LLM fine-tuning. In: Proceedings of the 2024 IEEE International Symposium on Workload Characterization (IISWC), 2024. 210–223
24 Chen Z, Zhang Z, Liu C, et al. Towards wireless native big AI model: the mission and approach differ from large language model. Sci China Inf Sci, 2025, 68: 170303
25 Fuad A, Ahmed A H, Riegler M A, et al. An intent-based networks framework based on large language models. In: Proceedings of the 10th International Conference on Network Softwarization (NetSoft), 2024. 7–12
26 Mekrache A, Ksentini A, Verikoukis C. Intent-based management of next-generation networks: an LLM-centric approach. IEEE Network, 2024, 38: 29–36
27 Chang Z, Lu F, Zhu Z, et al. A survey of LLM alignment: instruction understanding, intention reasoning, and reliable dialogue generation. Neurocomputing, 2026, 687: 133629

28 Tzanakaki A, Anastasopoulos M, Alevizaki V M. Intent-based control and management framework for optical transport networks supporting B5G services empowered by large language models. J Opt Commun Netw, 2025, 17: A112
29 Ma C, Zhang C, Luo L, et al. Poster: simulation-guided strategy generation for intent-aware distributed LLMs training. In: Proceedings of the 2025 IEEE 33rd International Conference on Network Protocols (ICNP), 2025. 1–3
30 Ma C, Zhang C, Luo L, et al. Unlocking agentic AI service deployment complexity: simulation-guided strategy orchestration and optimization. In: Proceedings of the 2025 IEEE 31st International Conference on Parallel and Distributed Systems (ICPADS), 2025. 1–8
31 Ouaddi C, Benaddi L, Bouziane E, et al. Assessing the effectiveness of large language models for intent detection in tourism chatbots: a comparative analysis and performance evaluation. Sci African, 2025, 28: e02649
32 Benayas A, Miguel-Ángel S, Mora-Cantallops M. Enhancing intent classifier training with large language model-generated data. Appl Artif Intell, 2024, 38: 2414483
33 Dai Y, Zhang H, Wang J, et al. INCS: intent-driven network-wide configuration synthesis based on deep reinforcement learning. Comput Networks, 2024, 251: 110640
34 Dinh L, Cherrared S, Huang X, et al. Towards end-to-end network intent management with large language models. ArXiv:2504.13589
35 Trantzas K, Brodimas D, Agko B, et al. Intent-driven network automation through sustainable multimodal generative AI. EURASIP J Wirel Commun Netw, 2025, 2025: 42
36 Siino M, Giuliano F, Tinnirello I. Integrating large language models into network testbeds: a novel approach for automated experimentation and optimization. J Network Comput Appl, 2025, 243: 104281
37 He X, Dong M, Yan Y, et al. Research on power standard named entity recognition based on improved BERT pre-training model (in Chinese). Electric Power Inform Commun Tech, 2024, 22: 52–59
38 Hu X, Song B, Tong J, et al. Zero-shot entity relationship extraction from massive electric power data with large language model (in Chinese). Electric Power Inform Commun Tech, 2025, 23: 61–67
39 Hossain M K, Aljoby W. NetIntent: leveraging large language models for end-to-end intent-based SDN automation. IEEE Open J Commun Soc, 2025, 6: 10512–10541
40 Sun W, Sun C, Zhang Y, et al. SCNOC-agentic: a network operation and control agentic for satellite communication systems. Electronics, 2025, 14: 3320
41 Li Q, Xiong Y, Li Z, et al. KlonetAI: automating $(com)^2$ nets management with human language intents. IEEE Network, 2025, 39: 12–19
42 Cheng Z, Ma C, Tang M, et al. Poster: LLM multi-agent collaboration for network deployment and management. In: Proceedings of the 2025 IEEE 33rd International Conference on Network Protocols (ICNP), 2025. 1–2
43 Wang J, Guo L, Wu J, et al. Hierarchical index retrieval-driven wireless network intent translation with LLM. IEEE Trans Mobile Comput, 2025, 24: 9837–9851
44 Wu D, Wang X, Qiao Y, et al. NetLLM: adapting large language models for networking. In: Proceedings of the ACM SIGCOMM 2024 Conference, 2024. 661–678
45 Wang Y, Han X. PTC: prefix tuning codeT5 for high-quality secure network measurement script generation. In: Proceedings of the 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2025. 102–115
46 Aguirre M, Méndez A, Del Pozo A, et al. Fine-tuning medium-scale LLMs for joint intent classification and slot filling: a data-efficient and cost-effective solution for SMEs. In: Proceedings of the 31st International Conference on Computational Linguistics: Industry Track, 2025. 251–262
47 Soviany P, Ionescu R T, Rota P, et al. Curriculum learning: a survey. Int J Comput Vis, 2022, 130: 1526–1565
48 Chen G, Zhan R, Wong D F, et al. Dynamic curriculum learning for conversation response selection. Knowledge-Based Syst, 2024, 293: 111687
49 Pouransari H, Li C L, Chang J H, et al. Dataset decomposition: faster LLM training with variable sequence length curriculum. In: Proceedings of the 38th Annual Conference on Neural Information Processing Systems (NeurIPS), 2024. 36121–36147
50 Zhang Y, Mohamed A, Abdine H, et al. Beyond random sampling: efficient language model pretraining via curriculum learning. In: Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics, 2026. 5776–5794
51 Ma T, Luo L, Yu H, et al. Klonet: an easy-to-use and scalable platform for computer networks education. In: Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24), 2024. 2025–2046
52 Ren S, Guo D, Lu S, et al. CodeBLEU: a method for automatic evaluation of code synthesis. ArXiv:2009.10297