This article has been accepted for publication in IEEE Transactions on Cognitive Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2026.3662678

PreConfig: A Unified Language Model Framework for Network Configuration Automation

Fuliang Li, *Member, IEEE*, Bocheng Liang, Haozhi Lang, Jiajie Zhang, Jiaxing Shen
Chengxi Gao, *Member, IEEE*, and Xingwei Wang, *Member, IEEE*

***Abstract*—Manual network configuration tools are constrained by their reliance on extensive domain expertise and rigid, single-purpose designs, limiting their adaptability to diverse scenarios and complex applications. This paper introduces PreConfig, a novel language model-based framework for automating network configuration tasks. By framing tasks such as configuration generation, translation, analysis, and completion as text-to-text transformations, PreConfig unifies these processes under a single versatile model. Leveraging advancements in natural language processing, PreConfig eliminates the need for extensive manual re-engineering by automatically learning domain-specific patterns through continued training on a specialized network configuration corpus. To address the lack of domain knowledge in general language models, we construct a comprehensive dataset from vendor manuals and community forums and fine-tune a programming language model for robust performance across various tasks. Additionally, we propose ConfigBLEU, a novel evaluation metric that incorporates syntax-aware features to assess the accuracy of generated configurations. Experimental results demonstrate that PreConfig significantly outperforms existing tools and general-purpose language models in both syntactic accuracy and semantic correctness across diverse network configuration tasks. This work establishes a unified and adaptable approach for advancing network configuration automation.**

***Index Terms*—Network Configuration Automation, Network Management, Configuration Generation.**

I. Introduction

NETWORK configuration automation (NCA) has become an essential component of modern network management, enabling organizations to efficiently configure and manage network devices such as routers, switches, and firewalls. By automating repetitive configuration tasks like generation, translation, and analysis, NCA reduces manual effort, enhances operational efficiency, and supports rapid scaling in increasingly complex networking environments.

Despite its potential, existing NCA tools face significant limitations due to their reliance on manual design. Tools such as NetComplete [1], Propane [2], Juniper2Cisco [3], and Batfish [4] are highly specialized for specific tasks like configuration synthesis, translation, or analysis. This single-purpose nature necessitates combining multiple tools for real-world scenarios, increasing learning curves and maintenance costs. Furthermore, these tools struggle with scalability in complex networks due to inherent protocol intricacies and state explosion issues [1].

Unlike traditional methods, large language models possess strong generalization capabilities, making them not confined to a specific protocol, vendor, or scenario. Furthermore, large language models enable natural language interaction with a low learning curve. Additionally, they can effectively circumvent the issue of state explosion. Moreover, existing large language model technologies have already achieved promising results in code generation and analysis. So, can current general-purpose large language models accomplish network configuration automation?

The answer is no. Existing language models can produce outputs for network configuration automation tasks. However, there are serious shortcomings in the correctness of their outputs. Network configuration demands extremely high standards for accuracy. At present, overall, the direct application of existing large language models to network configuration automation faces the following shortcomings:

Limitation 1: Deficiency in domain knowledge. Language models lack specialized domain knowledge required for robust performance in network configuration tasks [6].

Limitation 2: Insufficiency in evaluation metrics. Existing evaluation metrics like BLEU or ROUGE fail to capture syntax-specific features unique to configurations, making them inadequate for assessing model outputs.

To solve all aforementioned problems, we propose PreConfig, a novel language model-based framework for network configuration automation. Unlike traditional approaches, PreConfig frames key NCA tasks—such as configuration generation, translation, analysis, and completion—as text-to-text transformations. For example, configuration generation transforms natural language requirements into device configurations, while translation converts configurations between vendor-specific syntaxes (Figure 1). We construct a domain-specific corpus by extracting data from vendor manuals and community forums. We then fine-tune a programming language model on this corpus to equip it with specialized knowledge for generating, analyzing, translating, and completing configurations across diverse networking environments. Additionally, we propose ConfigBLEU—a novel evaluation metric that combines n-gram matching with syntactic tree analysis—to better assess

This work is supported by the National Natural Science Foundation of China under Grant Nos. U22B2005 and 62572105, as well as the LiaoNing Revitalization Talents Program under Grant No. XLYC2403086. *(Corresponding authors: Jiaxing Shen, Chengxi Gao, and Xingwei Wang.)*

Fulinag Li, Bocheng Liang, Haozhi Lang, Jiajie Zhang, and Xingwei Wang are with the School of Computer Science and Engineering, Northeastern University, Shenyang 110169, China (e-mail: lifuliang@cse.neu.edu.cn; 2110666@stu.neu.edu.cn; 571329288@qq.com; 13353980863@163.com; wangxw@mail.neu.edu.cn).

Jiaxing Shen is with the School of Data Science, Lingnan University, Hong Kong SAR (e-mail: jiaxingshen@LN.edu.hk).

Chengxi Gao is with the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences (e-mail: chengxi.gao@siat.ac.cn).

Authorized licensed use limited to: Lingnan University Library. Downloaded on February 11,2026 at 02:30:35 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cognitive Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2026.3662678

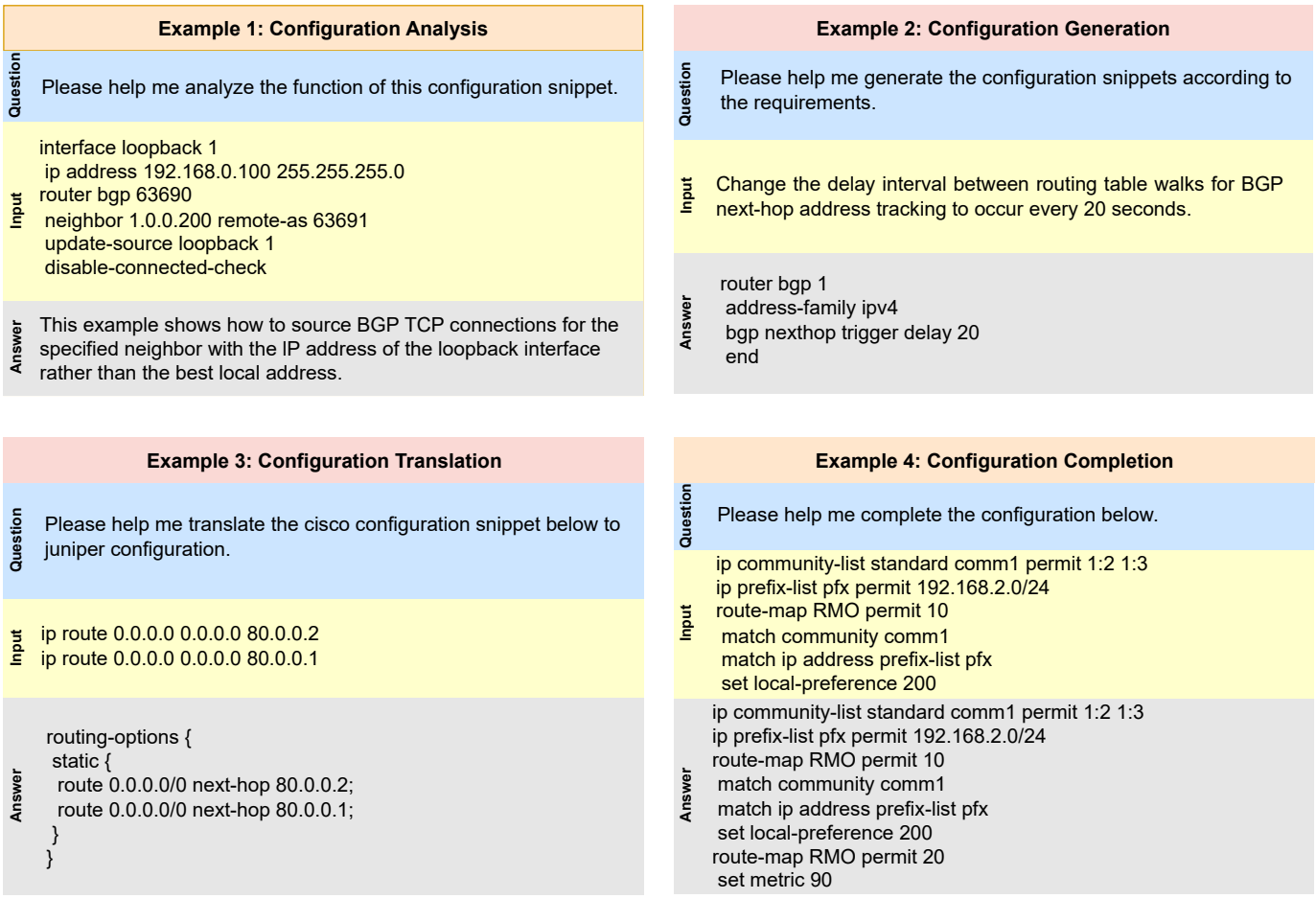

Fig. 1. An example of text transformation for configuration tasks. Configuration generation and analysis tasks involve the transformation between network configuration and natural language, The configuration translation task involves the transformation of configurations from different vendors. The configuration completion task involves the automatic filling in of missing or incomplete configuration elements.

the accuracy of generated configurations.

Our contributions are summarized as follows:

- We propose PreConfig, a unified language model tailored specifically for diverse NCA tasks through continued training on domain-specific data.
- We design ConfigBLEU, an evaluation metric that incorporates syntactic features to more effectively evaluate network configurations compared to traditional metrics.
- Extensive experiments demonstrate PreConfig's superior performance over general-purpose language models across multiple NCA tasks in terms of syntactic accuracy and semantic correctness.

II. Background

In this section, we first explore relevant tasks in the context of network configuration automation, then discuss the limitations of existing NCA tools, and finally introduce robotic process automation with LMs.

A. Network Configuration Automation

In recent years, with the continuous expansion of network scale and the enhancement of functional requirements, network managers often need to face complex and diverse network configuration tasks. For example, to modify the route propagation, it is necessary to add configuration elements such as route policies and access control lists. To avoid potential issues like route interruptions, or to meet requirements such as device replacement or preventing vendor monopolies, it is necessary to use a router A' from another vendor as a backup for router A. For business requirements, it is essential to analyze the current configuration for enhanced operational efficiency and business adaptability. These processes correspond to the tasks that network operators often need to complete. The operators have begun to realize the limitations of traditional network configuration methods in meeting the requirements of complex network operations. To simplify network configuration management, introducing automated methods has become increasingly critical [7], [8].

In this context, the paper focuses on four common tasks in network configuration automation: configuration analysis, generation, translation, and completion. Figure 1 shows four examples about the four text transformation tasks of configuration analysis, generation, translation, completion. We use the examples in Figures 1 to illustrate these concepts.

Configuration Analysis. Configuration analysis aims at

Authorized licensed use limited to: Lingnan University Library. Downloaded on February 11,2026 at 02:30:35 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cognitive Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2026.3662678

assisting network operators in analyzing and extracting the functional description of device configuration. Network configuration can be complex and extensive. In the core routers of large networks, configuration files can reach thousands of lines. Unlike the simple routing policies, the elements in real network configurations, such as routing policies and access control lists, interact with each other. This mutual interaction poses significant challenges for network operators in analyzing network behaviors. Configuration analysis task provides a comprehensive analysis of network configuration to extract configuration intent. As shown in the Example 1 of Figure 1, different from formatted configuration information mining [4], our target is to accomplish the transformation from network configuration to natural language through configuration analysis task.

Configuration Generation. The input of example 2 in Figure 1 illustrates an example of the configuration intent. To change the delay interval between routing table walks for BGP next-hop address tracking, in the practical process of configuring network devices, network operators formulate configuration intent, as depicted in example 2. Following this, they input configuration commands relying on manuals or personal experience. Finally, the configuration files are generated by network devices. This process is undoubtedly laborious, particularly in the configuration of large networks. For the configuration generation task, as shown in example 2, our target is to achieve the transformation from natural language to network configuration.

Configuration Translation. As shown in the example 3 of Figure 1, the configuration translation task involves the transformation between configurations from different vendors. For the backup between Cisco router and Juniper router, network operators need to translate the configuration from Cisco to Juniper and make sure that the functionality of the configuration remains consistent before and after translation. We provide configuration texts for CIso routers and Juniper routers in example 3. Both configuration paragraphs specify the static routing of two different manufacturers. It is evident that there are significant syntactic differences between configurations from different vendors. This is a challenging task as the operational commands and logic between configurations of different vendors are not entirely consistent [5].

Configuration Completion. The example 4 in Figure 1 illustrates an example of the configuration completion task. When network operators configure network devices, they often provide partial configurations based on their requirements or the network's needs, network operators rely on tools or automated systems to complete the configuration, ensuring that all necessary parameters are properly set. The configuration completion process involves automatically filling in the gaps in a configuration, often using contextual information or best practice guidelines. This process is crucial, especially for large-scale networks, where manual completion can be time-consuming and error-prone. For the configuration completion task, as shown in example 4, our goal is to achieve the automatic generation of missing or incomplete configuration elements based on the existing configuration context.

B. Limitations of Existing NCA Tools

Existing tools are primarily single-purpose: each excels in a narrow task but cannot handle the full spectrum of configuration workflows. This necessitates the integration of multiple tools in practice, increasing operational complexity and learning overhead. Furthermore, their reliance on manually crafted rules or formal models leads to poor scalability in large, heterogeneous networks and struggles with state explosion in complex policy interactions. Lastly, they are often vendor- or protocol-specific, lacking the flexibility to adapt to diverse network environments.

In contrast, PreConfig is designed as a unified framework that frames all core NCA tasks as text-to-text transformations. By leveraging a language model trained on diverse configuration corpora, it inherently gains vendor-agnostic understanding, robust scalability through parallel generation, and the ability to handle multiple tasks within a single model, thereby addressing the core limitations of prior work.

C. Robotic Process Automation with LMs

Robotic Process Automation (RPA) is a sophisticated form of process automation technology, primarily aimed at using machines, software scripts, or other technological means to mimic and simulate human interactions with digital systems. The primary goal of RPA is to automate repetitive, time-consuming tasks that would otherwise require human intervention, thus increasing efficiency and reducing the potential for errors. By deploying RPA tools, organizations can streamline a wide range of processes, including those in software engineering, operational log management, and network configuration.

LMs are a type of technology utilizing statistics and machine learning. The introduction of Language Models (LMs), a category of artificial intelligence that relies heavily on advanced machine learning and statistical methods, has opened up new possibilities for automating more complex processes. These models, such as GPT and BERT , excel in understanding and generating natural language, allowing them to carry out intricate text-based tasks. Over recent years, LMs have become increasingly proficient in text generation, semantic understanding, and natural language processing (NLP). This robust progress has unlocked significant opportunities for transforming text-based tasks across various domains.

Existing research, such as ProAgent, PreSQL, and CodeT5, has incorporated LMs into RPA, making advancements in automation processes within their respective domains.

III. Design

Motivated by the limitations of existing NCA tools, we introduce PreConfig, a language model to complete NCA tasks involving configuration generation, translation, completion and analysis. The overall framework of PreConfig is shown in Figure 2.

A. Overview

Data Acquisition: The Data Acquisition consists of product manuals, and the forum corpora from different manufacturers.

Authorized licensed use limited to: Lingnan University Library. Downloaded on February 11,2026 at 02:30:35 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cognitive Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2026.3662678

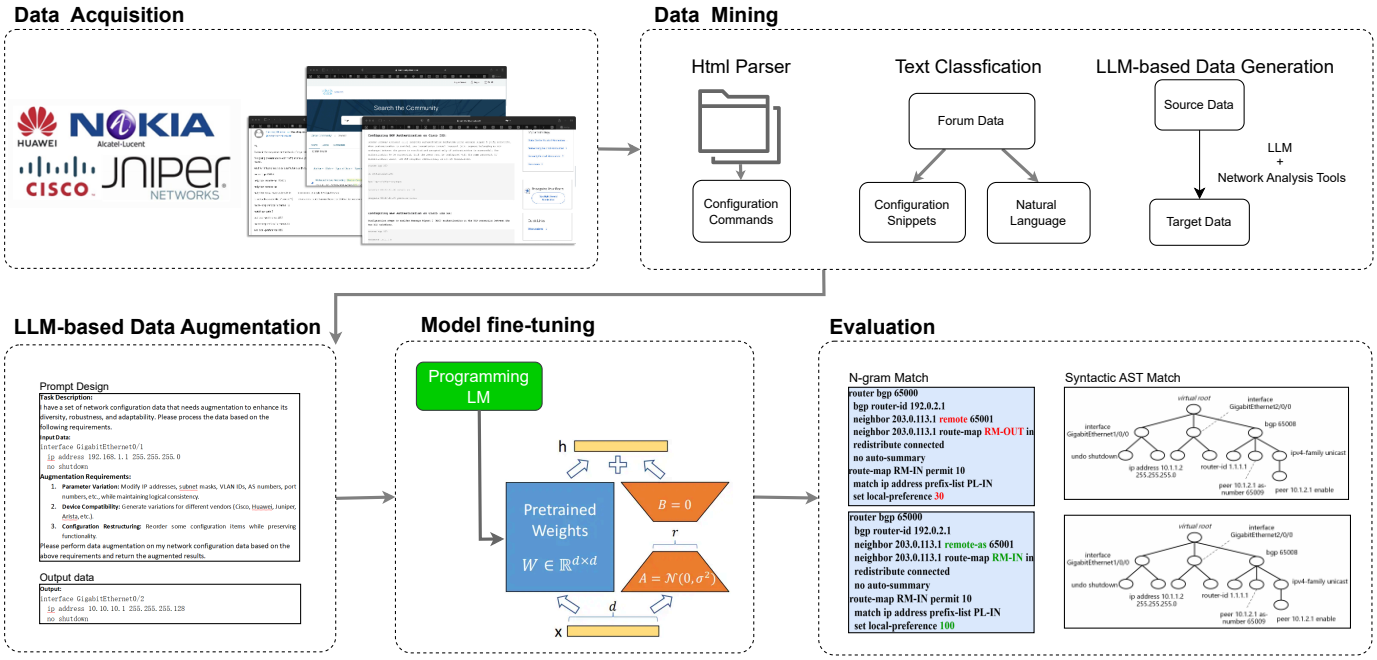

Fig. 2. Overall framework of PreConfig. The workflow is as follows: First, obtain the official tutorial websites and community forums of different network vendors. Then, design various data mining methods to acquire network configuration task data. Next, use a language model for data augmentation to expand the task data. Finally, perform model training through LoRA fine-tuning and complete the evaluation of the model.

Algorithm 1: Configuration Extraction Algorithm Based on Text Classification.

Data: User community website L, high quality configuration data $D2$
Result: Generated datasets D (a list of saved data)

$D \leftarrow []$;
$D1 \leftarrow \text{DataProcess}(L)$;
$model \leftarrow \text{ModelPretrain}(D1, D2)$;
for $d \in D2$ **do**
  $candidates \leftarrow \text{DataSelection}(d, model, n)$;
  for $c \in candidates$ **do**
    $D \leftarrow D.\text{append}(\text{DataJudgment}(c))$;
  end
end

Data Mining: There is a scarcity of supervision data in the network configuration field. In order to train the model for network configuration tasks, we design different methods to help us obtain task supervision data from the manual and the forum corpora.

Data Augmentation: In order to effectively scale up the volume of task-specific data, we employ a data augmentation strategy leveraging large language models (LLMs). This method involves using the generative capabilities of LLMs to synthetically create additional data samples that closely mirror the characteristics of the original dataset.

Model Fine-tuning: As for the scarcity of network configuration data, leveraging the thought of transfer learning, we initialize PreConfig with a pretrained programming language model, to accelerate model fine-tuning and enhance its comprehension of configuration languages.

Evaluation: In order to evaluate the quality of the output network configuration of the model, we refer to the evaluation metric CodeBLEU in the code domain and design Config-BLEU to evaluate the performance of the model in different configuration tasks.

B. Data Mining

Learning on high-quality task supervision data is essential for pretrained models to handle specific tasks [9]. Different tasks have specific requirements for data. For example, configuration generation and analysis tasks require transforming text of natural language and configuration language. Configuration translation task, on the other hand, requiring transforming configuration text of different vendors. In this section, we present our methods of constructing supervision data for network configuration tasks.

1) Html Parser: There are a large number of example tutorials on configuration commands on the official websites of network device manufacturers, which are often embedded in HTML pages with the same tags. We first extract configuration commands and corresponding natural language descriptions from HTML pages of different manufacturers by writing crawler scripts to construct configuration generation and analysis task datasets.

2) Network configuration extraction based on text classification: The manufacturer's forum website contains rich configuration examples that reflect the configuration habits of different users. These data are very suitable for network configuration completion tasks. However, in the manufacturer forum webpage, configuration fragments often coexist with natural language fragments, making it difficult to separate

Authorized licensed use limited to: Lingnan University Library. Downloaded on February 11,2026 at 02:30:35 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cognitive Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2026.3662678

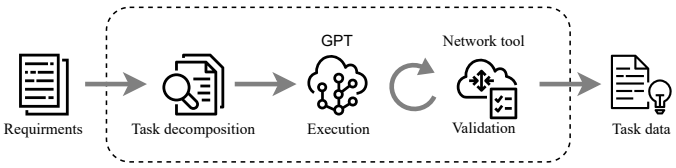

Fig. 3. LLM-based task data generation tool for task supervision data mining. The input is the prompt templete of data mining requirements for a specific task, the output is task supervision data.

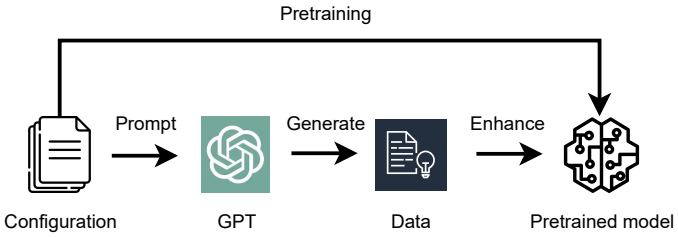

Fig. 4. Implementation of configuration data augmentation utilizing prompt engineering. The inputs to GPT are configuration snippets, prompt templates, and the outputs of GPT are expanded configuration data.

them manually. We consider this problem as a text classification task and design a Configuration extraction algorithm based on a bag-of-words language model [11]. We describe the workflow in Algorithm 1. The algorithm mainly consists of three steps:

- The DataProcess function extracts text from the URL L. This function further preliminarily separates natural language text from configuration snippets and returns the initial community data D1.
- The ModelPretrain function pretrains a bag-of-words model using data D1 and standard configuration snippets D2 to obtain embeddings for all data.
- For each standard configuration snippet d in D2, the DataSelection function selects the top n most similar candidate texts from the overall data utilizing a K-Nearest Neighbors (KNN) algorithm. The DataJudgment function then returns the filtered configuration snippets.

Through the above method, we efficiently collect network configurations snippets and build the dataset for configuration completion task.

3) LLM-based task data generation: Manufacturer manuals and community websites rarely include examples of configuration translation tasks. It is difficult to quickly obtain the data required for training translation tasks through manual writing. Inspired by prompt engineering and intelligent agents, we design a task data generation method based on large language models, which combines large language models and network configuration analysis tools to obtain supervised data for configuration translation task.

As shown in Figure 3, we first design precise prompt templates and use large language models to help us generate initial task supervision data. To ensure the correctness of the task supervision data, we utilize campion, a network configuration analysis tool to ensure the syntax and semantic consistency across configurations from different vendors. If the generated data is incorrect, we will provide feedback to GPT and iteratively modify the data.

C. LLM-based Data Augmentation

Leveraging LMs for text process can be a novel and practical method for data augmentation. It can effectively expand the scale of data while improving data diversity [12], [13]. After completing task data collection, we propose a method for configuration data augmentation utilizing LMs. How to ensure the quality of configuration text generated by the LMs is a core issue for us. We address this challenge by designing various detailed prompt templates. As shown in Figure 4, we utilize prompt engineering to guide GPT to expand configuration data. Firstly, we design prompt templates to present detailed task requirements to GPT. As shown in Table I, we incorporate domain knowledge into the prompt templates [14], including vendor names, configuration attributes, and model roles. Secondly, we employ Standard Operating Procedures (SOP) to standardize the operation process of the GPT, generating high-quality configuration data [10]. Finally, we use the data generated by GPT and the original data together as the pretraining data for the model.

TABLE I
THE DESIGN OF PROMPT TEMPLATES FOR CONFIGURATION DATA AUGMENTATION.

Methods	Prompt Templates
Raw	Please help me enhance this configuration text: {INPUT_CONFIG}
Raw+DSP	You are an expert in network configuration domain, Please help me enhance this configuration text: {INPUT_CONFIG}
Raw+DSP+SOP	You are an expert in configuration domain, Please help me enhance this configuration text, considering various combinations of protocol parameters and statements: {INPUT_CONFIG}

D. Fine-tuning

After obtaining task supervision data, we leverage llama factory framework to train the model to handle multiple configuration tasks.

Model Selection. In recent years, large-scale language models (LLMs) have developed rapidly. This growth trend has also promoted the development of various open-source LLMs. Based on the idea of transfer learning, we have chosen the Qwen2.5-Coder model as the base LLM for downstream task fine-tuning. Qwen2.5-Coder is built on the Qwen2.5 architecture and utilizes an advanced Transformer structure, optimized specifically for code generation and understanding. By incorporating a multi-layer self-attention mechanism, the model can efficiently capture long-range dependencies and complex syntactical structures in code. Leveraging the thought of transfer learning, we initialize PreConfig with pretrained

Authorized licensed use limited to: Lingnan University Library. Downloaded on February 11,2026 at 02:30:35 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cognitive Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2026.3662678

parameters of Qwen2.5-Coder, to accelerate model fine-tuning and enhance its comprehension of configuration languages.

Fine-tuning. In order to enable the model to handle different configuration tasks from multiple vendors simultaneously, we use instruction fine-tuning to train the model. Instruction tuning is a fine-tuning technique designed for large pre-trained language models. Its objective is to enhance the model's ability to understand and execute specific instructions, enabling it to generate appropriate outputs or perform relevant tasks accurately based on user-provided natural language commands. Meanwhile, in order to reduce training costs, we adopted Lora fine-tuning technique. LoRA fine-tuning is an efficient model adaptation technique that falls under the category of PEFT (Parameter-Efficient Fine-Tuning). The core idea of LoRA is to introduce low-rank matrices into the weight matrices of large pre-trained models to enable fine-tuning. The main advantage of this approach is that it can effectively adjust the model's behavior for specific downstream tasks without significantly increasing computational overhead.

E. Evaluation Metric

In order to evaluate the output performance of the model in different configuration tasks, we propose an evaluation metric ConfigBLEU, designed for network configuration tasks, which not only considers text matching, but also syntactic matching.

First, we use BLEU as the foundation for our evaluation metric. The BLEU score can be used to analyze text matching by calculating the percentage of overlapping n-grams between the candidate text and the reference text, which reflects the degree of match between them. The evaluation of the model's configuration accuracy also begins with basic text matching, and other metrics are added on top of this.

The original BLEU compares the n-grams between the candidate and reference texts and calculates the ratio of matching n-grams. Unlike natural language, which has a vast vocabulary and flexible word order, configuration languages are manually designed with only a few key terms, such as "route-map" or "access-list." Applying traditional BLEU directly to configuration generation would overlook the importance of these key terms. Therefore, we introduce weighted n-gram matching to assign different weights to different n-grams, allowing key terms to have higher weights.

In addition to text matching, we also consider the syntactic match in the ConfigBLEU by comparing the syntax tree structures. Unlike natural language, configuration languages have a natural tree-like structure, such as an Abstract Syntax Tree (AST). The AST is a tree representation of the abstract syntax structure of a configuration language. In the AST, each node represents a construct that appears in the configuration, and the leaves represent the variable names or parameter values. We can parse the configuration module into a standard syntax tree structure and then calculate the configuration's syntactic matching accuracy by comparing the candidate and reference subtrees in the syntax tree. The final formula for syntactic tree matching is as follows:

$$Match_{syn} = \frac{Count(Ast_{cand})}{Count(Ast_{ref})} \quad (1)$$

Where Count(ASTcand) is the total number of reference subtrees, and Count(ASTref) is the number of candidate subtrees that match the reference configuration. This score allows us to evaluate the quality of the generated configuration from a syntactic perspective, as syntax errors, such as type errors, can be captured by the differences between their ASTs.

The final ConfigBLEU evaluation metric is designed as follows:

$$ConfigBLEU = \alpha BLEU + \beta BLEU_{weight} + \gamma Match_{syn} \quad (2)$$

The new evaluation method combines the original BLEU, weighted n-gram matching ($BLEU_{weight}$), and syntactic AST matching.

IV. Experiments and Analysis

In this section, we present the implementation of PreConfig, configuration tasks, datasets, experiment setup, and results.

Implementation PreConfig is implemented in Python. We utilize the pretrained Qwen2.5-coder-Instruct model with 1.5B parameters as our base model. The Qwen2.5-coder model is pretrained using extensive programming language and natural language data, resulting in powerful language comprehension and generation capabilities. Due to the similarity between configuration language and programming language, the model can rapidly comprehend configuration language, contributing to its performance in configuration tasks. Through the application of transfer learning, we fine-tuning the Qwen2.5-coder model with configuration data. In the end, we accomplish the implementation of PreConfig. The system environment is Ubuntu 22.04 During the fine-tuning of PreConfig, we set the model's learning rate to 5e-5.

A. Configuration Tasks

Configuration Analysis is the opposite of configuration generation, which involves transforming network configuration into natural language text. We take Cisco and Juniper configuration as input and evaluate the quality of the generated natural language text.

Configuration Generation involves generating configuration for a specific vendor according to the natural language text (English). We set Cisco and Juniper configuration as the output and evaluate the model's performance on this task utilizing evaluation metrics such as BLEU, ROUGE, and Meteor.

Configuration Translation involves translating configuration into equivalent configuration for another vendor. We evaluate the model's performance on translation task utilizing Cisco and Juniper configurations.

Configuration Completion focuses on completing an incomplete configuration based on the context of a given configuration snippets. The task involves inferring and adding missing configuration lines for a specific vendor. For evaluation, we use the Cisco and Juniper configuration as the reference and assess the model's performance using metrics like BLEU, ROUGE, and Meteor to measure the accuracy and completeness of the generated configurations.

Authorized licensed use limited to: Lingnan University Library. Downloaded on February 11,2026 at 02:30:35 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cognitive Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2026.3662678

TABLE II
THE FINE-TUNING DATASETS OF NETWORK CONFIGURATION TASKS.

Task	Language	Train	Test
Configuration Generation	NL-Cisco	2700	300
	NL-Juniper	4500	500
Configuration Analysis	Cisco-NL	2700	300
	Juniper-NL	4500	500
Configuration Translation	Cisco-Juniper	2700	300
	Juniper-Cisco	2700	300
Configuration Completion	Cisco-Cisco	2700	300
	Juniper-Juniper	2700	300

TABLE III
DATA SOURCES OF NETWORK CONFIGURATION

Vendor	Data Source
Cisco	Cisco official configuration documentation [19]
	Cisco technical [20]
Juniper	Juniper official configuration documentation [21]
	Juniper technical forums [22]

TABLE IV
EVALUATION OF PRECONFIG IN CONFIGRATION TASKS.

Task	Sub Task	BLEU	ROUGE	METEOR
Generation	NL-Cisco	35.2	48.86	41.47
	NL-Juniper	22.05	44.41	17.54
Analysis	Cisco-NL	34.87	46.8	43
	Juniper-NL	17.36	27.68	23.8
Translation	Cisco-Juniper	84.17	88.24	85.46
	Juniper-Cisco	85.94	90.51	88.58
Completion	Cisco-Cisco	87.59	89.49	86.03
	Juniper-Juniper	93.07	94.77	93.5

B. Evaluation Metrics.

For configuration task evaluation, we employ machine translation metrics such as BLEU, ROUGE, and Meteor to evaluate the model's performance in configuration tasks.

BLEU measures the degree of matching between the generated results and the reference text based on precision, with a particular focus on the accuracy of the generated results. It evaluates the quality of the generated text by calculating the overlap ratio of n-grams, making it suitable for assessing whether the generated content strictly meets expectations. In network configuration tasks, BLEU can be used to evaluate the precision of generated configuration commands or descriptions, such as checking whether the generated router configuration is highly consistent with the standard configuration.

ROUGE, on the other hand, assesses the completeness of the generated results based on recall, focusing on whether the generated content covers the key information in the reference text. It measures the comprehensiveness of the generated content by calculating the n-gram overlap ratio between the generated text and the reference text. In network configuration tasks, ROUGE can be used to evaluate whether the generated configuration includes all necessary configuration items, such as checking whether the generated security policy is complete and free of omissions.

METEOR further optimizes on the basis of BLEU and ROUGE by balancing both precision and recall, while also incorporating synonym matching, stemming, and sensitivity to word order. This allows it to better capture semantic-level similarities. In network configuration tasks, METEOR can be used to evaluate the semantic accuracy of generated configurations, such as checking whether the generated routing policy is semantically equivalent to the reference configuration, even if the expressions differ.

We use ConfigBLEU to evaluate the tasks of configuration generation, configuration translation, and configuration completion and compare the evaluation results of BLEU and ConfigBLEU to assess their effectiveness in these tasks.

C. Datasets

We obtain the data for model fine-tuning through the methods described in III-B and III-C. Specifically, we collected 7GB of vendor manuals and community forums data. Then we extract the data required for configuration task fine-tuning. Our datasets consists of Cisco and Juniper configuration snippets, along with task-related natural language text. The final statistics for the task datasets of the model are presented in Table II. Our task datasets include configuration related to BGP, OSPF, static route, route policy, ACL, and other elements.

After data mining and cleaning of official configuration documents and technical forums, a network configuration corpus containing 2.13 million lines of configuration data was constructed. As shown in Table III, the data sources include official configuration documents and technical forum resources from leading manufacturers such as Cisco and Juniper. Among these, the official configuration documents provide comprehensive equipment configuration standards and functional descriptions, while the technical forum resources accumulate abundant configuration examples and discussions on technical issues. These data sources cover typical configuration task scenarios, including routing protocols and security policies.

D. Experiment Setup

For the four configuration tasks, we conduct a comprehensive exploration of the model's performance on the test dataset after fine-tuning for each specific task. This involves evaluating the model's ability to understand and generate configuration-related content, such as network configurations, system settings, or application parameters, across different scenarios. By fine-tuning the model on task-specific data, we aim to assess its adaptability and effectiveness in handling domain-specific challenges, such as syntax accuracy, semantic coherence, and completeness of generated configurations.

Subsequently, we compare the performance of PreConfig, a specialized model designed for configuration tasks, with generic language models (e.g., GPT, Gemini) on the same set of configuration tasks. We analyze aspects such as precision, recall, and semantic accuracy, as well as the models' ability to handle configuration syntax.

Authorized licensed use limited to: Lingnan University Library. Downloaded on February 11,2026 at 02:30:35 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cognitive Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2026.3662678

Finally, we employ ConfigBLEU to assess the model's outputs in configuration generation, translation, and completion tasks. ConfigBLEU extends the traditional BLEU metric by incorporating domain-specific considerations. We analyze the advantages of ConfigBLEU over standard BLEU, particularly in capturing the nuances of configuration tasks, such as the need for both accuracy and syntaxtic equivalence. This analysis demonstrates how ConfigBLEU provides a more comprehensive and domain-relevant evaluation of model performance in configuration-related tasks.

E. Results and Analysis

1) Configuration Tasks Evaluation.: The evaluation results of PreConfig on various configuration tasks are presented in Table IV. The tasks are categorized into four main types: Generation, Analysis, Translation, and Completion. Each task is further divided into sub-tasks based on the network device type (Cisco or Juniper). The performance is measured using three standard metrics: BLEU, ROUGE, and METEOR.

Generation Task. In the Generation task, the model is evaluated on its ability to generate configuration commands from natural language (NL) descriptions. The results show that the model performs better on the Cisco dataset (BLEU: 35.2, ROUGE: 48.86, METEOR: 41.47) compared to the Juniper dataset (BLEU: 22.05, ROUGE: 44.41, METEOR: 17.54). This discrepancy can be attributed to the fact that the Juniper dataset is more extensive and contains a wider variety of configuration commands, which increases the complexity of the task. Additionally, the language used in the Juniper dataset is more varied, making it more challenging for the model to generate accurate configurations.

Analysis Task. The Analysis task involves converting configuration commands back into natural language descriptions. Similar to the Generation task, the model performs better on the Cisco dataset (BLEU: 34.87, ROUGE: 46.8, METEOR: 43) than on the Juniper dataset (BLEU: 17.36, ROUGE: 27.68, METEOR: 23.8). The larger and more complex nature of the Juniper dataset likely contributes to the lower performance. The model struggles to accurately interpret and translate the more diverse Juniper commands into coherent natural language descriptions.

Translation Task. The Translation task evaluates the model's ability to translate configuration commands between Cisco and Juniper formats. The results indicate that the model performs well in this task, with high scores across all metrics for both Cisco-to-Juniper (BLEU: 84.17, ROUGE: 88.24, METEOR: 85.46) and Juniper-to-Cisco (BLEU: 85.94, ROUGE: 90.51, METEOR: 88.58) translations. This suggests that the model has a strong understanding of the underlying semantics of configuration commands, allowing it to effectively translate between the two formats despite their differences in syntax and structure.

Completion Task. In the Completion task, the model is required to predict the next set of tokens in a configuration command. The model achieves high scores for both Cisco (BLEU: 87.59, ROUGE: 89.49, METEOR: 86.03) and Juniper (BLEU: 93.07, ROUGE: 94.77, METEOR: 93.5) datasets. Notably, the model performs slightly better on the Juniper dataset, despite its complexity. This difference in performance can be attributed to the distinct syntactic structures of the two configuration languages.

Overall, the evaluation results demonstrate that PreConfig is effective in understanding and generating network configuration commands. The model performs well across various tasks, with particularly strong results in the Translation and Completion tasks.

2) Comparisons with generic language model.: The evaluation results of PreConfig and its comparison with general-purpose language models (Gemini and ChatGPT) are presented in Table V. The performance is measured using three standard metrics: BLEU, ROUGE, and METEOR.

Generation Task. The Configuration Generation task evaluates the model's ability to generate configuration commands from natural language (NL) descriptions. PreConfig significantly outperforms both Gemini and ChatGPT in both NL-Cisco and NL-Juniper sub-tasks. For NL-Cisco, PreConfig achieves a BLEU score of 35.2, compared to 27.59 for ChatGPT and 24.29 for Gemini. Similarly, for NL-Juniper, PreConfig achieves a BLEU score of 22.05, while ChatGPT and Gemini achieve only 8.34 and 7.5, respectively. This performance gap can be attributed to PreConfig's domain-specific training, which allows it to better understand the nuances of network configuration syntax and semantics. Additionally, the Juniper dataset's larger size and greater complexity (with more command types and less structured natural language) make the task more challenging for general-purpose models, which lack specialized training in network configurations.

Analysis Task. The Configuration Analysis task involves converting configuration commands back into natural language descriptions. PreConfig again demonstrates better performance compared to general-purpose models. For Cisco-NL, PreConfig achieves a BLEU score of 34.87, significantly higher than ChatGPT (18.11) and Gemini (12.05). For Juniper-NL, PreConfig achieves a BLEU score of 17.36, outperforming ChatGPT (8.37) and Gemini (10.37). PreConfig's domain-specific training enables it to better handle these challenges compared to general-purpose models.

Translation Task. The Configuration Translation task evaluates the model's ability to translate configuration commands between Cisco and Juniper formats. PreConfig achieves good performance in this task, with BLEU scores of 84.17 for Cisco-Juniper and 85.94 for Juniper-Cisco. These results far surpass those of ChatGPT (56.23 and 67.87) and Gemini (60.07 and 69.26). The high performance of PreConfig can be attributed to its ability to understand the semantic similarities between Cisco and Juniper configurations, despite their syntactic differences (e.g., Juniper's Java-like syntax with vs. Cisco's Python-like syntax). General-purpose models, lacking domain-specific knowledge, struggle to achieve similar levels of accuracy.

Completion Task. The Configuration Completion task evaluates the model's ability to predict the next set of tokens in a configuration command. PreConfig achieves the highest scores in both Cisco-Cisco (BLEU: 87.59) and Juniper-Juniper (BLEU: 93.07) sub-tasks, outperforming ChatGPT and Gemini. The performance of PreConfig in the Configuration Com-

Authorized licensed use limited to: Lingnan University Library. Downloaded on February 11,2026 at 02:30:35 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cognitive Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2026.3662678

TABLE V
MODEL PERFORMANCE COMPARISON

	MODEL	BLEU	ROUGE	METEOR	BLEU	ROUGE	METEOR
		Configuration Generation (NL-Cisco)			Configuration Generation (NL-Juniper)		
1	Gemini	24.29	36.75	33.2	7.5	23.73	8.1
2	ChatGPT	27.59	41.89	36.3	8.34	24.05	8.7
3	PreConfig	35.2	48.86	41.47	22.05	44.41	17.54
		Configuration Analysis(Cisco-NL)			Configuration Analysis(Juniper-NL)		
4	Gemini	12.05	17.19	17.94	10.37	15.91	20.09
5	ChatGPT	18.11	25.62	32.23	8.37	13.91	21.52
6	PreConfig	34.87	46.8	43	17.36	27.68	23.8
		Configuration Translation(Cisco-Juniper)			Configuration Translation(Juniper-Cisco)		
7	Gemini	60.07	76.92	71.89	69.26	78.83	81.08
8	ChatGPT	56.23	77.4	74.42	67.87	76.94	75.32
9	PreConfig	84.17	88.24	85.46	85.94	90.51	88.58
		Configuration Completion(Cisco-Cisco)			Configuration Completion(Juniper-Juniper)		
10	Gemini	76.37	82.18	79.63	57.3	74.6	79.48
11	ChatGPT	72.28	83.24	75.86	59.73	74.43	69.18
12	PreConfig	87.59	89.49	86.03	93.07	94.77	93.5

TABLE VI
COMPARISION OF CONFIGBLEU AND BLEU IN NETWORK CONFIGURATION TASKS.

Task	Sub Task	ConfigBLEU	BLEU
Configuration Generation	NL-Cisco	76.51	80.16
Configuration Translation	Juniper-Cisco	81.57	85.94
Configuration Completion	Cisco-Cisco	88.3	87.59

pletion task is a direct result of its domain-specific training, which enables it to capture and replicate user configuration habits effectively. By leveraging its understanding of network configuration syntax and common practices, PreConfig outperforms general-purpose models like ChatGPT and Gemini.

The evaluation results demonstrate that PreConfig outperforms general-purpose language models (Gemini and ChatGPT) in configuration tasks. General-purpose models struggle with the complexity and variability of network configuration, particularly for Juniper configurations.

3) Evaluation configuration tasks with ConfigBLEU.: As a domain-specific evaluation metric for network configuration generation, ConfigBLEU addresses the critical limitations of conventional BLEU through two key enhancements: (1) weighted n-gram matching prioritizing configuration keywords, and (2) syntax tree similarity measurement. Our experimental results in Table VI demonstrate its improved alignment with configuration validity assessment.

The divergence between ConfigBLEU and BLEU scores reveals critical insights. In Configuration Generation (NL-Cisco), while BLEU yields a higher score (80.16 vs 76.51), manual inspection shows this discrepancy stems from BLEU's overrating of textually similar but syntactically invalid outputs. For instance, configurations with misplaced access-control list rules might achieve high lexical overlap yet fail compilation - a pitfall captured by ConfigBLEU's syntax-aware penalty.

Notably, in Configuration Completion (Cisco-Cisco), ConfigBLEU (88.3) surpasses BLEU (87.59), indicating its capacity to reward structurally faithful completions beyond surface-level similarity. This aligns with network configuration's hierarchical nature where command sequence integrity matters more than token ordering alone.

```
[Candidate]:
interface GigabitEthernet0/1
 ip access-group 101 out
access-list 101 permit tcp host 10.1.1.1 any eq 80

[Reference]:
interface GigabitEthernet0/1
 ip access-group 101 in
access-list 101 permit tcp any host 10.1.1.1 eq 80
```

Fig. 5. The example of ConfigBLEU and BLEU in configuration task evaluation.

Our metric particularly shines in cross-vendor translation (Juniper-Cisco). The 4.37-point BLEU-ConfigBLEU gap (85.94→81.57) reflects ConfigBLEU's stricter validation of vendor-specific syntax transformations.

As shown in figure 5. In a Cisco ACL configuration generation task, the candidate configuration achieved a high BLEU score due to lexical overlap but exhibited critical functional flaws. ConfigBLEU analysis revealed substantial discrepancies: 1) Weighted n-gram matching penalized missing directional keyword "in"; 2) Syntax tree matchin detected invalid command hierarchy and reversed ACL parameters through AST subtree comparisons. With weighting parameters, the composite ConfigBLEU score demonstrated superior validity assessment over BLEU by simultaneously evaluating lexical, syntactic dimensions of network configurations. This case highlights ConfigBLEU's capacity to identify syntax mismatches that evade text-based metrics.

Two design principles ensure this effectiveness:

Keyword-aware Weighting: Critical configuration tokens (e.g., router ospf, access-list) receive 5× weights versus generic parameters, reflecting their disproportionate impact on functionality.

Authorized licensed use limited to: Lingnan University Library. Downloaded on February 11,2026 at 02:30:35 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cognitive Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2026.3662678

Hierarchical Validation: Using parsed configuration trees, we compute subtree isomorphism rates rather than token matches, detecting structural errors like misplaced interface declarations.

This approach addresses the limitation of match-based metrics.

V. Related Work

In recent years, with the development of machine learning, the field of software engineering has embarked on the integration of common tasks in program understanding and generation, such as code generation, code summarization, and code translation, with NLP techniques. Pretrained programming language models have achieved excellent performance in these tasks. CodeBERT [16] employs the same transformer-based encoder architecture as the BERT model. It undergoes pre-training using both programming language and natural language data and is utilized for tasks such as code search and code summarization. CodeGPT [17] utilizes the same transformer-based decoder architecture as the GPT model. It undergoes pretraining with programming language data and is employed for tasks such as code completion and code generation. PLBART [15] employs a transformer-based encoder-decoder architecture similar to the BART model. It is used for common code generation and understanding tasks.

For network configuration synthesis, the target is to automatically generate network configurations based on user intent, transforming high-level policies into low-level configurations to avoid manual errors. Several tools have been developed to address this challenge. NetComplete [1] is an configuration synthesis tool that uses SMT solvers to resolve configuration parameters. By leveraging SMT solvers, NetComplete ensures that the generated configurations satisfy both user intent and network constraints, while its incremental approach allows for efficient updates to configurations as requirements change. AED [18] extends this capability by not only performing incremental synthesis but also repairing existing configurations. Additionally, AED supports soft-constrained management objectives, enabling it to optimize configurations for desirable but non-mandatory goals. This flexibility makes AED particularly useful in dynamic network environments where policies and requirements may evolve over time.

VI. Conclusion

This paper introduces PreConfig, a language model for diverse network configuration tasks. First, PreConfig acquires expertise in network configuration through an automated framework for configuration corpora collection. Second, we design an LLM-based agent to mine configuration task supervision data. Through a fine-tuning framework, PreConfig acquires the capabilities to handle various configuration tasks. Finally, we design an evaluation metric for evaluating network configuration tasks Compared to current NCA tools, PreConfig can handle multiple configuration tasks and extend to complex application scenarios. Extensive experiments demonstrate PreConfig's strong performance in configuration generation, translation, analysis and completion tasks. Compared to generic language models, PreConfig exhibits a more powerful capability in handling configuration tasks.

VII. Limitations and Future Work

A. Limitations

Context Length Constraints: Like most Transformer-based language models, PreConfig has a fixed context window. Extremely long configuration files may need to be processed in segments, which could result in the loss of global dependencies.

Dependence on Data Quality: The performance of the framework is heavily dependent on the quality and diversity of the mined supervision data. Noisy or biased data from forums can adversely affect the model.

B. Future Work

Architectural Enhancements: Exploring models with extended context windows or memory mechanisms to handle lengthy configurations.

SDN Network Expansion: The configuration and operational approach of controller-based SDN differs somewhat from that of distributed networks. Based on the characteristics of SDN networks, this study explores methods to extend PreConfig to SDN environments.

Fragment-Level Semantic Verification: Research on fragment-level configuration semantic verification methods to enhance ConfigBLEU.

Acknowledgments

The authors would like to thank the Editors and the Reviewers of IEEE/ACM Transactions on Cognitive Communications and Networking for their review efforts and helpful feedback.

References

[1] A. El-Hassany, P. Tsankov, L. Vanbever, and M. Vechev, *NetComplete: Practical Network-Wide configuration synthesis with autocompletion*, in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018.

[2] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, *Don't mind the gap: Bridging network-wide objectives and device-level configurations*, in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016.

[3] Juniper, *IOS-to-JUNOS Conversion Tool*, [Online]. Available: https://supportportal.juniper.net/s/article/Archive-IOS-to-JUNOS-I2J-Conversion-Tool-Tool-Fact-Sheet?language=en_US, 2009.

[4] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, *A general approach to network configuration verification*, in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017.

[5] A. Tang, S. K. R. Kakarla, R. Beckett, E. Zhai, M. Brown, T. Millstein, Y. Tamir, and G. Varghese, *Campion: debugging router configuration differences*, in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 748–761.

[6] R. Mondal, A. Tang, R. Beckett, T. Millstein, and G. Varghese, *What do LLMs need to Synthesize Correct Router Configurations?*, in *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, 2023, pp. 189–195.

[7] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés-Mulero, D. Meyer, S. Barkai, M. J. Hibbett, et al., *Knowledge-defined networking*, *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, pp. 2–10, 2017.

Authorized licensed use limited to: Lingnan University Library. Downloaded on February 11,2026 at 02:30:35 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Cognitive Communications and Networking. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCCN.2026.3662678

[8] C. Jiang, N. Ge, L. Kuang, *AI-enabled next-generation communication networks: Intelligent agent and AI router*, *IEEE Wireless Communications*, vol. 27, no. 6, pp. 129–133, 2020.
[9] S. Tao, Y. Liu, W. Meng, Z. Ren, H. Yang, X. Chen, L. Zhang, Y. Xie, C. Su, X. Oiao, et al., *Biglog: Unsupervised large-scale pre-training for a unified log representation*, in *2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*, 2023, pp. 1–11.
[10] S. Hong, X. Zheng, J. Chen, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. Yau, Z. Lin, L. Zhou, et al., *Metagpt: Meta programming for multi-agent collaborative framework*, *arXiv preprint arXiv:2308.00352*, 2023.
[11] S. Gururangan, T. Dang, D. Card, N. A. Smith, *Variational pretraining for semi-supervised text classification*, *arXiv preprint arXiv:1906.02242*, 2019.
[12] C. Xu, Q. Sun, K. Zheng, X. Geng, P. Zhao, J. Feng, C. Tao, D. Jiang, *Wizardlm: Empowering large language models to follow complex instructions*, *arXiv preprint arXiv:2304.12244*, 2023.
[13] Y. Wei, Z. Wang, J. Liu, Y. Ding, L. Zhang, *Magicoder: Source Code Is All You Need*, *arXiv preprint arXiv:2312.02120*, 2023.
[14] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, D. C. Schmidt, *A prompt pattern catalog to enhance prompt engineering with chatgpt*, *arXiv preprint arXiv:2302.11382*, 2023.
[15] W. U. Ahmad, S. Chakraborty, B. Ray, K. W. Chang, *Unified pre-training for program understanding and generation*, *arXiv preprint arXiv:2103.06333*, 2021.
[16] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, et al., *Codebert: A pre-trained model for programming and natural languages*, *arXiv preprint arXiv:2002.08155*, 2020.
[17] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, D. Jiang, D. Tang, et al., *Codexglue: A machine learning benchmark dataset for code understanding and generation*, *arXiv preprint arXiv:2102.04664*, 2021.
[18] A. Abhashkumar, A. Gember-Jacobson, A. Akella, *Aed: Incrementally synthesizing policy-compliant and manageable configurations*, in *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, 2020, pp. 482–495.
[19] Cisco Systems, *Networking Software (IOS NX-OS)*, [Online]. Available: https://www.cisco.com/c/en/us/support/ios-nx-os-software/index.html, 2025.
[20] Cisco Systems, *Technology and Support*, [Online]. Available: https://community.cisco.com/t5/technology-and-support/ct-p/technology-support?profile.language=en, 2025.
[21] Juniper Systems, *Documentation*, [Online]. Available: https://www.juniper.net/documentation/#cat=product_documentation&tab=documentation_by_category, 2025.
[22] Juniper Systems, *Juniper Elevate Community*, [Online]. Available: https://community.juniper.net/home, 2025.

Fuliang Li (Member, IEEE) received the BSc degree in computer science from Northeastern University, China in 2009, and the PhD degree in computer science from the Tsinghua University, China in 2015. He is currently a professor at the School of Computer Science and Engineering, Northeastern University, China. He has published more than 50 Journal/conference papers.

Bocheng Liang received the BS degree in Software Engineering from North China Electric Power University, China, in 2018 and MS degree in 2019 from the School of Computer Science and Engineering, Northeastern University, where he is currently working toward the PhD degree. His research interests include network management and measurement, configuration synthesis, and Configuration Verification.

Haozhi Lang received his B.S. degree in communication engineering from the Northeastern University in 2022. He is currently studying for a master's degree with the School of Computer Science and Engineering, Northeastern University, China. His research interests include network configuration management.

Jiajie Zhang received his B.S. degree in computer science from the Shenyang Agricultural University in 2021. He is currently studying for a master's degree with the School of Computer Science and Engineering, Northeastern University, China. His research interests include network configuration translation and neural machine translation.

Jiaxing Shen is an Assistant Professor with the School of Data Science at Lingnan University. He received the B.E. degree in Software Engineering from Jilin University in 2014, and the Ph.D. degree in Computer Science from the Hong Kong Polytechnic University in 2019. He was a visiting scholar at the Media Lab, Massachusetts Institute of Technology in 2017. His research interests include human-centric computing, mobile computing, and data mining. His research has been published in top-tier journals such as IEEE TMC, ACM TOIS, ACM IMWUT, and IEEE TKDE. He was awarded conference best paper twice including one from IEEE INFOCOM 2020.

Chengxi Gao (Member, IEEE) received his B.S. and M.S. degrees from the Department of Computer Science, Northeastern University, China, and his PhD degree from the Department of Computer Science, City University of Hong Kong. He is now an assistant professor at Shenzhen Institutes of Advanced Technology (SIAT), Chinese Academy of Sciences (CAS). He has published more than 30 Journal/conference papers. His research interests include networking system and resource allocation.

Xingwei Wang (Member, IEEE) received the BS, MS, and PhD degrees in computer science from Northeastern University, Shenyang, China, in 1989, 1992, and 1998, respectively. He is currently a professor with the College of Computer Science and Engineering, Northeastern University, Shenyang, China.

Authorized licensed use limited to: Lingnan University Library. Downloaded on February 11,2026 at 02:30:35 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.