ConfigTrans: Network Configuration Translation Based on Large Language Models and Constraint Solving

Naigong Zheng[†], Fuliang Li[†✉], Ziming Li[†], Yu Yang[‡], Yimo Hao[†], Chenyang Liu[†], Xingwei Wang[†✉]
[†]Northeastern University, China, [‡]Hong Kong Polytechnic University, China

***Abstract*—Damage or relocation of network devices necessitates the replacement of devices. Additionally, manufacturers of devices may also change due to factors such as price, policies, and functional limitations. This brings great needs of converting existing configurations to fix new devices and ensuring consistent network policy behaviors, which is configuration translation. Manually translating requires network administrators to possess significant configuration expertise and to spend extreme time translating thousands of commands for a core device. Therefore, we propose ConfigTrans, a novel configuration translation method for commands with and without parameters based on constraint solving and large language models (LLMs), respectively. For commands with parameters, we explore constraints about parameters, commands, and format limitations, and design a heuristic algorithm to solve them. For commands without parameters, we utilize an LLM to find the translation. The most likely several commands are selected based on semantic similarity, and the final result and keywords in it are chosen by the LLM. To meet the requirements of the view, the result commands are arranged while supplementing missing higher-view commands. Experimental results show that our method achieves an accuracy rate of 82.47% for translating commands with parameters and 75.5% for commands without parameters.**

***Index Terms*—network configuration, configuration translation, large language models, network management, network protocol**

I. Introduction

In the operation of networks, replacing devices is inevitable due to outages or damages. Network administrators need to replace existing devices with new ones, and configure the new devices to perform the same tasks currently handled by the existing devices. However, the manufacturer of the devices may be changed after replacing. The syntaxes of different manufacturers' configuration commands are different due to the patents [1] to protect manufacturers' interests, resulting in the inability to use configurations from different manufacturers interchangeably.

At this point, the current configuration files need to be converted to the new configuration files conforming to the new manufacturer's syntax. This process is called configuration translation. It's similar to the process of natural language translation. Network experts need to study the product manuals of the new devices, search for new commands having the same functions as the current commands, and finally write new configuration files, which is extremely time-consuming and high labor cost.

Currently, there are very few existing works that accomplish the configuration translation task and none of them are open source. Most of the configuration translations are still performed manually. It consumes a significant amount of time and manpower, leading to low efficiency and waste of excellent human resources. Furthermore, the manual process has poor scalability since every time a new manufacturer appears, the translating knowledge will be relearned.

Therefore, we aim to develop a tool that can assist experts in completing configuration translations, improving efficiency, and achieving scalability. However, it's not easy to achieve the goal. The challenges faced can be summarized as follows.

Translating Commands with Parameters. Existing research, NAssim [2], has achieved the correspondence of configuration parameters from different manufacturers to a unified device model. An intuitive idea is that if the parameters of the existing device and the new device correspond to the same parameter in the unified device model, then these two parameters are considered equivalent. Since parameters exist within commands, the corresponding results of the parameters can be used to achieve the translation of commands with parameters. However, the translation of commands is not a simple "one-to-one" translation. It involves "one-to-many", "many-to-one", and "many-to-many" relationships. Corresponding methods need to be designed to solve this problem.

Translating Commands without Parameters. Due to the lack of parameter constraints on commands, translating commands without parameters becomes more challenging. A single model of a device may contain tens of thousands of configuration commands, including potentially thousands of commands without parameters. Finding the corresponding command for a specific command among such a large number of commands is difficult.

Command View Requirements. Commands only work when under correct views. However, the view structures are different depending on manufacturers, making it difficult to ensure the translated commands under proper views.

In this paper, we propose ConfigTrans, a configuration translation method based on large language models (LLMs) and constraints solving, respectively, to assist network ex-

Corresponding authors: Fuliang Li, Xingwei Wang.

979-8-3503-5171-2/24/$31.00 ©2024 IEEE

perts. It contains three parts: configuration manual parsing by NAssim, configuration command translation, and command arrangement. Firstly, the product manual is parsed by NAssim to obtain the Vendor-specific Device Model, command hierarchy, and parameters correspondence result. Then, commands with and without parameters are translated respectively by the heuristic algorithm and the LLMs. Finally, commands in the preliminary translation result are arranged based on the command hierarchy to ensure the view requirement. It should be emphasized that the LLM and embedding models used in ConfigTrans are accessed directly through APIs or open-source platforms, which do not require fine-tuning so that they can be replaced by modules with better performance easily.

The contributions of this paper can be summarized as follows.

- We propose a translation method for commands with parameters. Based on limitations of parameter correspondence, commands, and format, we establish constraints to describe the "many-to-many" problem of command translation and design a heuristic algorithm to solve these constraints. It enables the incorporation of parameters from existing devices into new devices while ensuring that the new commands maintain the correct format.
- We propose a translation method for commands without parameters. First, the view context is used to narrow down the possible range of result commands. Then, semantic similarity is calculated to filter out several candidate commands having the most similar semantics. Subsequently, an LLM selects one command from the candidate commands as the translated result and completes the selection of keywords within the command.
- We design a command arrangement algorithm to adjust commands' views and supplement lacked commands. The translated commands are constructed into a configuration tree based on the command hierarchy. Additionally, by iteratively checking whether higher-level commands exist and supplementing them when they are missing, the command view requirements are fulfilled.
- All BGP and OSPFv3 configuration file examples in the Huawei NE40E manual are tested. ConfigTrans successfully translates 82.47% of commands with parameters and 75.5% of commands without parameters for the first time. Additionally, we open-source the prototype system of ConfigTrans[1], making it the first open-source method for the configuration translation task.

II. Preliminaries

A. Network Configuration

Parameters, commands, and views are the three most critical components in configuration.

Parameters largely determine how the network operates and are the most crucial part of network protocols. For instance, the "Cost" value in OSPF determines the routing path of the traffic, the "Local preferences" in BGP determines the routing priority, and the source and destination addresses in an ACL determine whether the traffic is allowed through. Correctly configuring the parameters of the original device in the new device is essential for the new device to maintain consistent network policy behaviors

However, parameters alone are insufficient for completing the configuration task; the device needs to know which parameter is being configured. Different parameters can appear identical in form. For example, the dotted decimal IP address `10.0.0.1` cannot, on its own, indicate whether it is configuring a port's IP address or a BGP peer. For most traditional network devices, commands are the only way to distinguish parameters. Non-parameter keywords in the commands identify the type of command and the type of parameters within the command. For instance, configuring the port IP and BGP peer can be distinguished using commands "`ip address 10.0.0.1 255.255.255.0`" and "`peer 10.0.0.1 as-number 65002`", respectively.

Furthermore, commands without parameters themselves also have configuration effects, such as enabling or disabling a specific function, providing context for other commands. For instance, the "`undo shutdown`" command can start an interface, and the "`ipv4-family unicast`" command can display the BGP-IPv4 unicast address family.

Views are contexts that indicate where a command takes effect. For instance, "`shutdown`" performs different functions depending on the view: in the BGP view, it terminates all sessions between the device and its BGP peers; in the port view, it shuts down the specified port. Views also help distinguish between different elements of the same type. For example, if there are ports GE1 and GE2 in the configuration, and a "`shutdown`" command is issued, it is the view that determines whether port GE1 or GE2 is shut down.

In summary, if a tool can translate commands with parameters, translate commands without parameters, and ensure that the result commands are placed in the correct views, then it can be said that this tool is capable of performing the configuration translation task.

B. NAssim

For the correspondence of parameters across different devices, which can be considered as a part of configuration translation, NAssim [2] has conducted an exploration. NAssim achieves the correspondence of configuration parameters from different manufacturers to the parameters possessing the same configuration functionality in the Unified Device Model (UDM). NAssim uses OpenConfig [3] as the UDM, which is one kind of YANG model [4]. NAssim first parses the command reference pages from manufacturers' device manuals and extracts the information, such as command descriptions and command formats, into a JSON file, called the Vendor-specific Device Model (VDM). By leveraging the information from manuals and empirical data, the relationship between views and the commands enabling these views is inferred, and a tree-like model hierarchy is built based on this relationship. Finally,

[1]https://anonymous.4open.science/r/ConfigTrans

NAssim constructs contextual vectors for VDM and UDM parameters based on the command description, parent view, and parameter description. The Sentence-BERT [5] is utilized to calculate the semantic similarity between the contexts of parameters from VDM and UDM. The top-k VDM parameters with the highest similarity are corresponded to the UDM parameter obtaining the VDM-UDM parameter correspondence.

C. Scenarios of Translating Commands with Parameters

As mentioned before, it is a straightforward idea to translate commands based on the parameter correspondence results provided by NAssim. However, when we try to do so, we realize that it is not a simple problem due to the need to address various scenarios. Examples of four different command translating cases are illustrated in Fig. 1. Fig. 1(a) demonstrates the most basic "one-to-one" translation between commands, i.e. one command is translated to one command. But there is not always an absolute "one-to-one" corresponding between commands and parameters. A command typically consists of multiple parameters, and in the new device, multiple commands may be required to cover the parameters present in this single command. This leads to the "one-to-many" translation between commands, as shown in Fig. 1(b). Swapping the models of the existing device and the new device in Fig. 1(b) results in the "many-to-one" translation, as shown in Fig. 1(c). Combining the above scenarios can give rise to the "many-to-many" translation, as illustrated in Fig. 1(d). The occurrence of the above scenarios makes the translation of commands with parameters more complex.

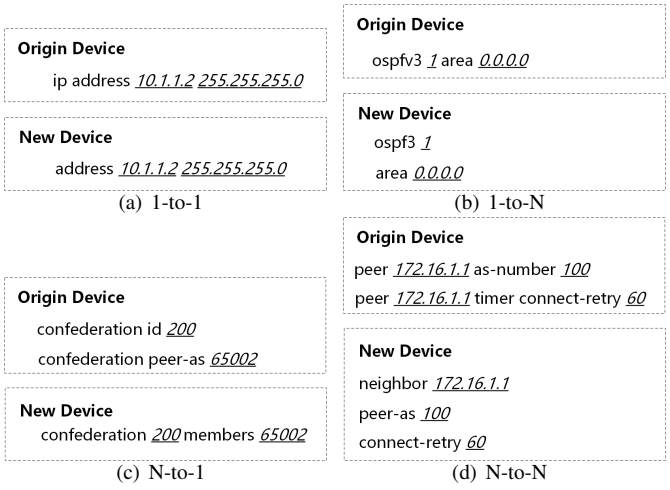

Fig. 1. Four scenarios of translating commands with parameters.

III. Methodology

A. ConfigTrans Overview

Based on the analysis above, ConfigTrans is designed. ConfigTrans' architecture and process is illustrated in Fig. 2. The included modules are as follows:

Inputs: The input consists of two parts: product manuals, including manuals of the existing device and the new device, and the configuration file to be translated.

NAssim: NAssim parses the product manuals to obtain VDMs, command hierarchy, and VDM-UDM corresponding parameters for both existing and new devices.

Parameter Corresponder: The parameter corresponder corresponds the parameters from existing and new devices based on the two VDM-UDM corresponding parameters.

Parser: The command parser builds a CLI graph model, using the methodology proposed in NAssim, based on the command format in VDM to obtain the parameters in the command and to determine the original command of the CLI collaborating with the model hierarchy. Then, the parser converts the configuration file to the configuration tree.

Command Translator: The command translator translates commands of the tree nodes. For commands with parameters, it obtains the translated commands and the parameters to be filled in these commands. For commands without parameters, only translated commands are obtained.

Filler: The filler attempts to fill the parameters into the new commands and provides feedback on additional parameters that are required but missing for the format correctness. The command translator then searches for the missing parameters in other nodes of the configuration tree and hands them over to the filler to re-fill, getting a preliminary translation result.

Command View Arranger: The command arranging module arranges the commands in the current translation result, adjusts the order and hierarchical relationship between commands, and adds missing higher-level parent commands to achieve the view requirements and to obtain the final translation result.

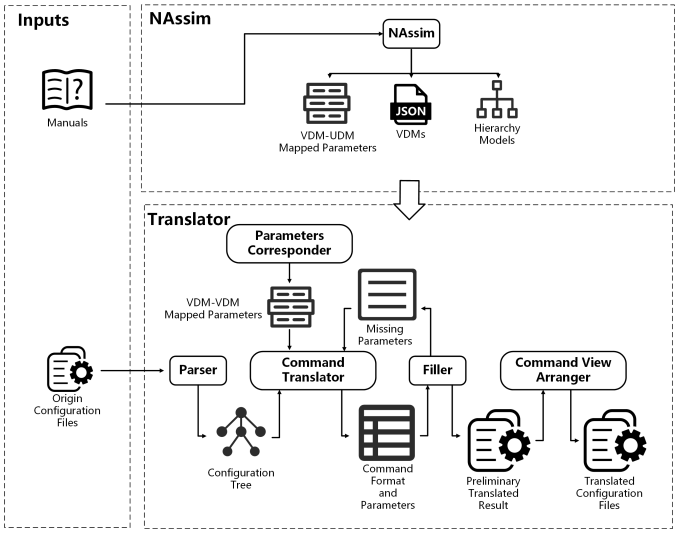

Fig. 2. ConfigTrans overview.

B. Parameter Corresponding

Since NAssim completed the corresponding of VDM-UDM parameters, our method considers UDM as the intermediate layer and associates the VDM parameters corresponding to the same UDM parameters to accomplish parameter corresponding. We illustrate this method with a simple example.

As shown in Fig. 3, NAssim is used to construct VDM1 and VDM2, which are two VDMs obtained from devices of the existing and new models. They are respectively associated with UDM, resulting in the correspondence of VDM1-UDM and VDM2-UDM. In VDM1, "ParamV1_1" corresponds to "ParamU_1" in UDM. Similarly, "ParamV2_1" in VDM2 corresponds to "ParamU_1" in UDM. This indicates that "ParamV1_1" and "ParamU_1" have similar semantics, and "ParamV2_1" and "ParamU_1" also have similar semantics. Therefore, we can infer that "ParamV1_1" and "ParamV2_1" are semantically similar, meaning they are corresponding. In a similar way, "ParamV1_2" corresponds to "ParamV2_3", and "ParamV1_3" corresponds to "ParamV2_2". By identifying all the corresponding parameters in VDM1 with respect to VDM2, the parameter corresponding is completed. Moreover, NAssim allows network configuration experts to participate in modifying the parameter correspondence results, ensuring relatively accurate parameter correspondence.

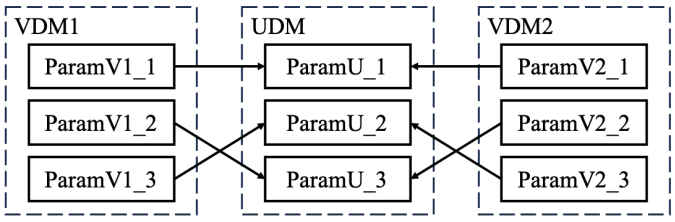

Fig. 3. Example of parameter corresponding.

C. Translating Commands with Parameters

After obtaining the corresponding parameter pairs, the parameter pairs can be considered as known information. And, the commands that the parameters belong to can be identified, since parameters are attached to commands. Therefore, constraints can be established to cover the translation scenarios mentioned earlier, and algorithms can be designed for their resolution.

Constraints. The translation problem of commands with parameters can be regarded as a Constraint Satisfaction Problem (CSP), where constraints such as the correspondence between parameters from existing and new devices and the correctness requirements of command formats are inherent. Identifying these constraints aids in the analysis and resolution of the problem.

CSP is formalized as a triplet (X, D, C), where $X = \{X_1, X_2, \ldots, X_n\}$ represents the set of variables, $D = \{D_1, D_2, \ldots, D_n\}$ denotes the set of domains for the variables, and $C = \{C_1, C_2, \ldots, C_m\}$ represents the set of constraints defined over the variable set X.

In the translation problem with commands with parameters, let the command to be matched with the current existing device be denoted as A. The set of parameters contained in command c is represented as $HaveParam(c)$, and the corresponding parameter representation of parameter p in the new device from the existing device is denoted as $CorParam(p)$. Then, the CSP is concretely defined as the following elements:

X: The variable set X includes command variables and parameter variables. The command variable c_i represents a command of the new device, while the parameter variable p_j represents a parameter in a command of the new device. X is composed of all commands and parameters of the new devices, that is, $X = \{c_1, c_2, \ldots, c_n, p_1, p_2, \ldots, p_m\}$.

D: The domain $D(x)$ of each variable $x \in X$ is $\{0, 1\}$, indicating whether the command or parameter is selected (1) or not (0) in this particular instance.

C: $C = \{C1, C2, C3, C4\}$, consisting of four constraints:

Constraint $C1$ can be represented by (1): If parameter p in the command A of the original device is selected, then the corresponding parameter p_i in the new device must also be selected.

$$\forall p \in HaveParam(A), p_i = CorParam(p) \Rightarrow p_i = 1 \quad (1)$$

Constraint $C2$ can be expressed by (2): In each selected command c_i, at least one parameter in it must correspond to the parameter from command A. This ensures that the selected commands are associated with the current command A while aiming to select the minimum number of commands.

$$\begin{aligned} \forall c_i \in X, c_i = 1 \Rightarrow \exists p \in HaveParam(A), \\ CorParam(p) \in HaveParam(c_i) \end{aligned} \quad (2)$$

Constraint $C3$ can be expressed by (3): All selected parameters must appear in at least one selected command.

$$\forall p_j \in X, p_j = 1 \Rightarrow \exists c_i \in X, c_i = 1, p_j \in HaveParam(c_i) \quad (3)$$

Constraint $C4$ can be expressed by (4): Each selected command c_i, after being filled with all or part of selected parameters, must meet the correct formatting requirements. Here, $Format$ is a function used to check whether the combination of commands and parameters satisfies the format.

$$\forall c_i \in X, c_i = 1 \Rightarrow Format(c_i, \{p_i \in X | p_i = 1\}) = True \quad (4)$$

Heuristic Algorithm. To solve the constraints, we propose a heuristic algorithm. Firstly, based on the parameter correspondence results, we identify the parameters in the new device that correspond to the existing device parameters. Then, we locate the command associated with these parameters and attempt to fill in all the parameters of the existing device into the new device, while ensuring the grammatical correctness of the new commands. By fill the parameters into the respective commands, we obtain the corresponding results and achieve a primary translation, thereby addressing the "many-to-many" problem, that other scenarios can be considered as special cases of the "many-to-many".

Building the Configuration Tree. To distinguish the configuration of the same parameter for different entities, it is necessary to establish the relationship between commands and their higher-level commands. A tree structure can effectively

describe this type of information since the nested and parallel structures among views can be described by parent-child node and sibling nodes in the tree. Moreover, since blanks in front of the commands can indicate whether a new view is entered and the depth of views entered, the configuration tree can be constructed. Fig. 4 shows an example of converting a configuration file into a configuration tree. It has a similar structure to the command hierarchy but does not merge identical commands of the same type to represent different configuration entities, such as the two "interface" nodes in the example. The construction process of the configuration tree is explained below:

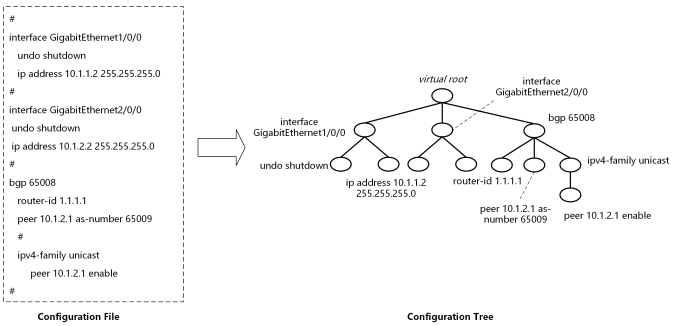

Fig. 4. Example of converting configuration file to configuration tree.

First, create an empty node as the root node of the configuration tree and set the depth of the current view to 0. Then, iterate through the commands in the configuration file to construct nodes corresponding to each command. Compare the number of preceding indentation blanks of the command with the depth of the previous command's view, and there are three results:

1. If the number of blanks is greater than the depth, it indicates that the previous command has enabled a new view, the new node should be the child node of the previous command's node.

2. If the number of blanks is equal to the depth, it means that the current command and the previous command belong to the same view. Connect the current node to the parent node of the previous command, making them sibling nodes.

3. If the number of blanks is less than the depth, it indicates that the current command is not directly related to the previous command but belongs to a higher-level view. In this case, based on the difference between the number of spaces and the depth, search for ancestor nodes starting from the parent node of the previous command. Connect the current node to the corresponding ancestor node, making it a child node.

Translating Commands. Afterward, ConfigTrans translates commands with parameters based on the constraints established earlier and the configuration tree. The method uses a post-order traversal of the tree structure, prioritizing commands from leaf nodes and then translating other internal nodes of the configuration tree. For each node, the command is first parsed to obtain the located VDM and form the parameter list including the parameters used in the command. The order of the parameters needs to be maintained in the list to ensure the correct parameter order after translation. Then, ConfigTrans traverses the parameter list and finds the command format in VDM where the parameter is located by string matching.

When the traversal is complete, all corresponding parameters are corresponded to one or multiple commands, satisfying constraint $C1$ and $C2$. The command filler fills the parameters into their corresponding command format, and a list of additional missing parameters needed to complete the simplest command is provided. Then, based on the missing parameter list, ConfigTrans searches the original device's configuration tree for the nodes that contain these parameters. The search is performed in the order of sibling nodes with different commands, parent nodes, sibling nodes of the parent node with different commands, and so on. Then, the selected parameters are filled into the corresponding command, ensuring the correctness of the command format and satisfying constraint $C4$. In the above steps, all selected parameters are filled into selected commands, satisfying constraint $C3$.

Finally, a virtual empty parent node is created to connect the generated commands to it as child nodes. When the traversal of the current layer nodes is complete, it returns to the higher layer node. Then all corresponding nodes of the current layer are connected to the corresponding nodes generated by the higher layer node.

An Example of Command Translating Process. As shown in Fig. 5, a simple example is given to illustrate the general process. And Fig. 6 shows the parameter corresponding result that Fig. 5 uses. We focus on explaining the process when there exist missing parameters, specifically steps 3-7. First, the CLI "`confederation peer-as 65002`" is parsed, and it is determined that the corresponding command used is "`confederation peer-as { <as-number> } &<1-32>`", with the parameter list "`[as-number: 65002]`". Based on the corresponding results of the parameters, it is found that the parameter used in the new device is "`as-number`" in Nokia VDM "`confederation_0`". It is determined that the command format to be filled in should be "`confederation <confed-as-num> [ members <as-number> [ <as-number> ] ]`" (step 3). The parameter list is filled into the command format using the command filling method (step 4). It is found that the command is missing the parameter "`confed-as-num`". Based on the corresponding results of the command, the required parameter is "`as-number`", which can be found in Huawei VDM "`confederation-id (BGPOM)`". It is searched in the configuration tree and the corresponding parameter is found in the command "`confederation-id 200`" (step 5). The parameter found is returned to the command filling part for supplementation (step 6), resulting in the final command (step 7).

Explanation for Traversal Order. The reason for prioritizing leaf nodes is that leaf nodes correspond to the lowest and most fundamental commands in the configuration, and they mostly perform the actual configuration tasks. Other internal nodes are responsible for creating and declaring work areas

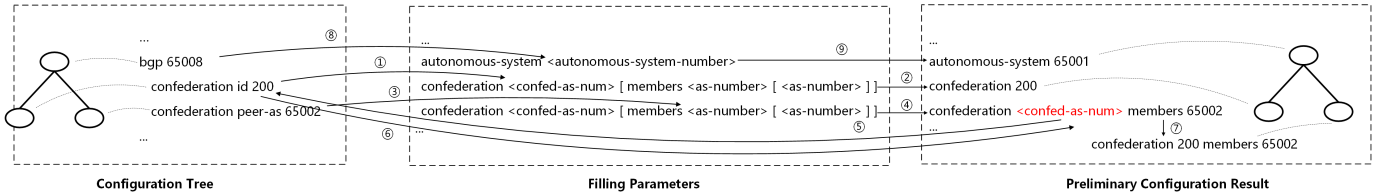

Fig. 5. Example of command translating process.

Huawei param	Huawei VDM	Nokia param	Nokia VDM
as-number	bgp(BGPOM)	autonomous-system-number	autonomous-system_0
as-number	confederation-id(BGPOM)	confed-as-num	confederation_0
as-number	confederation-peer-as	as-number	confederation_0
ipv4-address	peer-as(BGPOM)	ip-address	neighbor_0
as-number	peer-as(BGPOM)	as-number	peer-as_0

Fig. 6. Example of parameter corresponding result.

(i.e., views). The detailed objects of the configuration are further explained in the leaf nodes, and internal nodes cannot contain all the configuration content.

D. Translating Commands without Parameters

For translating commands with commands, a straightforward approach involves identifying commands on new devices that share similar semantic functionality, which can be regarded as a form of natural language semantic understanding. Given the excellent performance of LLMs in traditional natural language processing (NLP) tasks such as translation [6], summarization [7], and named entity recognition [8], as well as their unique contributions in network operations [9], [10], particularly in network configuration generation [11], [12], we have thus attempted to employ LLMs to accomplish this task.

However, LLMs have limitations on input text length [13], and their performance noticeably declines as the input text becomes longer [14]. This makes it impractical to input all commands without parameters from new devices into the LLM for selection, necessitating a reduction in the amount of input text.

The view in which commands without parameters are used on new devices can be roughly determined. Most commands without parameters primarily serve to enable or disable certain functions, or to enter specific views, which are functions within the current view. Thus, their correspondence on new devices should also be located within the equivalent views. Consequently, the search for commands can be restricted to the corresponding views in the new device. Specifically, it is the view enabled by the corresponding command that enables the current view.

At this point, there are still numerous commands under each view. To mitigate this issue, we draw inspiration from the Retrieval-Augmented Generation (RAG) [15] framework. We compute the semantic similarity between candidates (all commands without parameters under the corresponding view) and queries (existing commands to be translated), and select the top K candidates with the highest semantic similarity for processing by the LLM, i.e. reranking. In this task, the command format, command function description, and keyword description of the commands without parameters are the most relevant descriptors. Therefore, we aggregate the above information into the following format for computing semantic similarity for each command: "*<command>: <command_function_description>. The keywords in the commands are: <parameter_i>: <parameter_definition_i>*".

Command Selection Guide
Your task is to find a command from the candidate commands that has the most similar functionality to the given command. …
Given command
<given_command>
Candidate commands
<candidate_commands>
Instructions
Thoroughly analyze a given command to understand its purpose and functionality. Each candidate command is evaluated to find commands that are very similar in functionality to the given command. …
Output format
```json{ "reason": your reason,
"Command number": the number of the command,
"Give command": the command you select}```
**Let's think step by step.**

Fig. 7. An excerpt of prompt used to select commands.

After reranking, the top-K commands with the highest semantic similarity are selected as candidate commands for further processing by LLMs. An excerpt of the prompt used is illustrated in Fig. 7. The prompt includes the following components: (1) Guide, informing the LLM of the current task, i.e. identifying the command with the most similar function from the candidate commands. (2) Given command, indicating the command to be translated. (3) Candidate commands, representing the K commands obtained through reranking. (4) Instructions, guiding the LLM to complete the task process based on these instructions. (5) Output format, specifying the format in which the LLM should output the results that can be easily parsed. (6) "Let's think step by step", according to literature [16], incorporating this sentence can enhance the effectiveness of LLM in completing tasks.

To translate commands without parameters, besides selecting the correct command, it is also crucial to choose the appropriate keywords. For instance, given command "`ipv6-family vpnv6`" and the command selected by the LLM as command "`family [ ipv4 ] [ vpn-ipv4 ] [ ipv6 ] [ vpn-ipv6 ]`", the keyword "`ipv4`", "`vpn-ipv4`", "`ipv6`", "`vpn-ipv6`" within the command must be accurately chosen to enable the correct view.
```

Keyword Selection Guide
Your task is to give a command according to the command format that has the most similar functionality to the given command. …
Given command
<given_command>
Command format
<candidate_command>
Function of command format
<candidate_command_function>
Keywords and descriptions in command format
<keywords_and_ descriptions>
Supplementary instructions
Each optional part in the selection structure is separated by |. In [ ], 0 or 1 optional parts can be selected. In {}, 1 optional section must be selecte. …
Steps
Thoroughly analyze a given command to understand its purpose and functionality. …
Output format
First, answer the following four questions in sequence.
1. What are the keywords in the command format?
2. Which keyword/keywords did you choose? …
```json{ "reason": your reason,
"Determine command": The command determined according to the command format.}```
**Let's think step by step.**

Fig. 8. An excerpt of prompt used to select keywords.

Upon revisiting the task using the capabilities of the LLM, an excerpt of the prompt employed is depicted in Fig. 8, similar to the prompt utilized for command selection. It comprises the following components: (1) Guide, guiding the LLM to comprehend that the current task involves selecting keywords from the commands. (2) Given command, the command to be translated from the original device. (3) Command format, the format of the command selected in the previous stage, including optional structures enclosed in "[]" or "". (4) Function of the command format, describing the function of the new command. (5) Keywords in the command and their descriptions. (6) Instructions, specifying the rules for selecting elements within "[]" and "" structures. (7) Steps, outlining the steps to be followed to complete the task. (8) Output format, directing the LLM to sequentially answer questions regarding keyword selection, thereby reducing hallucination, particularly the issue of outputting commands in an incorrect format, and ultimately outputting in a specific format. (9) "Let's think step by step", serving the same function as in the previous stage.

### *E. Command Arranging*

After translating commands, the system obtains preliminary translation results, which are saved in the form of a configuration tree. However, the internal sentence structure of the current configuration tree is roughly constructed based on the hierarchical structure of the existing device configuration file. It cannot guarantee that the translated commands are in the correct views of the new device. Therefore, the configuration file at this stage cannot be correctly read by the device. It is necessary to arrange the command positions to meet the view requirements.

Command arrangement algorithms use hierarchical structures generated by NAssim as the basis for arrangement and employ recursive algorithms based on the characteristics of the configuration tree and the model hierarchy's tree structure. The command arrangement algorithm prioritizes processing lower-level configuration commands through post-order traversal, merging lower-level arranged command nodes into higher-level nodes. Ultimately, the algorithm completes the command arrangement for the entire configuration tree, ensuring that the arranged commands conform to the specified view relationships within the hierarchical structure. The specific method is as follows.

First, use a hash algorithm to index each node in the model hierarchy, so that the corresponding node can be quickly accessed by command name. Then, use a recursive algorithm to traverse the configuration tree nodes, giving priority to processing deeper nodes, i.e., more detailed configuration commands. For a leaf node $L$, directly connect it to the upper-level empty node. For an internal node $I$, for each node $O$ in the subtree rooted at $I$, attempt to connect the child node $C$ of $I$ to node $O$ as its child node. The successful connection condition is that the command-enabled view of node $O$ in the model hierarchy is the parent view of the command of node $C$. After processing all the nodes, the result is an empty node with the highest-level commands as its child nodes. However, the parent view of these nodes may not be the root view of the system. Therefore, it is necessary to sequentially supplement higher-level commands for incorrect commands until the command's parent view is the root view.

## IV. Evaluation and Analysis

We implemented the prototype system of this method using Python in approximately 1400 lines of code. The evaluation was completed on a cloud server consisting of 32 vCPU Intel(R) Xeon(R) Platinum 8352V CPU @ 2.10GHz, 180GB memory, and 2 24GB NVIDIA RTX 4090 GPU. We tested the prototype system using publicly available datasets as well as datasets we constructed ourselves. The main focus of the tests was to evaluate the correctness of the translation method, including commands with and without parameters and full configuration files.

### *A. Datasets*

The publicly available datasets used include the VDM data of Huawei and Nokia provided by NAssim [17], as well as the command hierarchy data. We constructed the configuration file dataset by extracting 126 configuration files from configuration examples in the NE40E Product Documentation [18], including 34 files of the OSPFv3 configuration and 92 files of BGP configuration. Since the dataset provided by NAssim does not include the correspondence for most parameters, and the method proposed by NAssim achieves an 89% accuracy in the top 10 recommendations on the Huawei dataset, we directly constructed a set of corresponding parameter data for the parameters included in the configuration files used in the evaluation. This set contains a total of 138 types of parameters, including parameter names and the corresponding VDMs. Additionally, during the evaluations, we found that many commands in Nokia's hierarchy model in the NAssim dataset
```

are not correctly connected to their render views. Therefore, we manually reconstructed the corresponding hierarchy model for Nokia commands involved in the test dataset.

B. Selection of Embedding Models and LLMs

The embedding technology has evolved over a long period [19], with new embedding models continuously emerging. As of April 2024, Hugging Face hosts over 4000 models tagged with Sentence Similarity [20]. Therefore, this paper references the Massive Text Embedding Benchmark (MTEB) leaderboard [21] to select suitable models, which evaluates over 200 embedding models across more than 150 datasets. Consequently, this paper tests the effectiveness of the top three models from the Reranking task leaderboard of MTEB, SFR-Embedding-Mistral [22], GritLM-7B [23], and E5-mistral-7b-instruct [24].

The development of LLMs has accelerated rapidly over the past years, with the release of ChatGPT [25] significantly boosting public usage, attention, and recognition of LLMs. It also promotes the development of various open-source LLMs, resulting in thousands of options available on Hugging Face [26]. Therefore, similarly, we reference the OpsEval leaderboard [27], which tests the performance of existing LLMs across various tasks in operational scenarios, including "Wired Network Operations" scenario which is similar the tasks addressed in this paper. This paper selects the Qwen1.5-14B-chat model [28] as a representative of open-source LLMs for testing due to hardware constraints. Additionally, given the significant advantages of closed-source LLMs, this paper aims to test their effectiveness on the tasks as well. Consequently, the Qwen-max model [29], which belongs to the same Qwen series ,is also selected for testing.

C. Translating Commands with Parameters

We used a prototype system to translate Huawei configuration file datasets into Nokia switch configurations, collecting commands with parameters and their translation results. In total, we collected 1751 commands comprising 128 types of commands from Huawei devices. After evaluating the correctness of the translation results and analyzing the reasons for any errors, the statistical results are shown in Table I.

A correct translation is defined as a Huawei device command that, once translated, is semantically and syntactically accurate. Out of the total number of commands, 1444 commands were correctly translated, accounting for 82.47%. And, we have analyzed four scenarios of correct translation, namely "one-to-one", "one-to-many", "many-to-one", and "many-to-many", illustrated in section II.C. The distribution of occurrence times is 193, 44, 26, and 174 times, respectively. It can be seen that the "one-to-one" situation is the most common, which means that the functionality of most individual commands between different manufacturers is still similar, and the functionality of one command from the original manufacturer can be achieved through one command. Moreover, the situation of "many-to-many" is more common than the remaining two cases combined, indicating that there is a significant connection between configuration and commands, and it is necessary to design methods to address these scenarios.

For the commands that were not correctly translated, the errors were categorized into four types:

TABLE I
RESULTS OF TRANSLATING COMMANDS WITH PARAMETERS

Result categories	Lines of commands	Proportion
Correct translation	1444	82.47%
Command category change	43	2.46%
Parameters transformation	44	2.51%
Non-unique result	12	0.68%
Missing function	208	11.88%

Command Category Change. The current method primarily focuses on correspondence origin parameters to new commands, resulting in the translation of commands with parameters only to commands with parameters, and the translation of commands without parameters only to commands without parameters. However, in practice, there are cases where a command with parameters should be translated into both a command with parameters and a command without parameters. For example, "`peer 10.1.1.2 next-hop-local`" should be translated into two commands: "`neighbor 10.1.1.2`" and "`next-hop-self`". It is difficult to solve this problem because not all translations of commands with parameters will yield additional commands without parameters, and the number of commands without parameters cannot be determined.

Parameters Transformation. In most cases, parameters correspond to each other one-to-one and do not require modification. However, there are still scenarios where there is a one-to-many or many-to-one relationship between parameters, as well as cases where modifications are needed according to certain rules. This can be considered as the merging, splitting, and converting of parameters. For example, "`ip address 192.168.0.1 255.255.255.0`" and "`address 192.168.0.1/24`" are both correct translations of each other. Converting is needed between the subnet mask and the mask length, as well as for merging and splitting "`192.168.0.1/24`". The current method is unable to solve this problem, and addressing it is also challenging because the system needs to learn the rules for merging, splitting, and converting, and determine the scenarios where these operations are required and successfully perform them.

Non-unique Result. If we only ensure the grammatical correctness of the translation, then there can be multiple cases. In the current method, to prevent the fusion of different commands with the same format under the same view, commands that can be merged are not combined. Alternatively, multiple commands are used as translation results as much as possible, but this may lead to translation errors. For example, when translating "`ospfv3 1, router-id 10.0.0.1`", there can be two translation results. One is "`ospf3 1 10.0.0.1`" and another one is translated into two commands "`ospf3 1`" and "`ospf3 10.0.0.1`". Both cases have correct syntax, but the first result is the correct

translation because the first one creates one ospfv3 process, while the second one creates two processes. It is challenging to determine when to perform the merge.

Missing Function. The functionality supported by the two manufacturers' devices is not entirely the same, and not all functions can be implemented on devices from other manufacturers. Alternatively, if the corresponding parameters are unfound, it is classified as this issue since not finding it and being unable to find it yield the same result, which also troubles experts.

D. Translating Commands without Parameters

In the constructed configuration profile dataset, 52 types of commands without parameters are included, among which 20 commands have corresponding counterparts. The effectiveness of the command correspondence method based on semantic similarity is tested using these 20 pairs of commands. The experiment utilizes three different embedding models to calculate semantic similarity and employs recall at top K (K=1, 3, 5, 10) to represent the percentage of cases within the top K test scenarios. Each experimental configuration is repeated five times, and the average recall rate is taken.

The command used in the embedding process is: "*Given a router configuration command and its relevant information, select one command that has a similar function to the given command.*".The experimental results of the three embedding models selected are specifically presented in Table II.

TABLE II
RECALL RATES OBTAINED WITH DIFFERENT EMBEDDING MODELS AND K.

Embedding models	k in recall@top k (%)			
	1	3	5	10
SFR-Embedding-Mistra	50	**85**	**85**	**85**
GritLM-7B	**55**	**85**	**85**	**85**
E5-Mistral-b-instruct	20	40	45	55

The SFR-Embedding-Mistral and GritLM-7B models achieved an 85% recall rate at k=3, and the recall rate did not increase when k was increased to 10. Upon observing the specific experimental results, it was found that the remaining 15%, 3 types of commands, did not have their correct results within the search view, leading to these results not being correctly retrieved. Therefore, for the aforementioned two models, all correct commands can be recalled at the top 3. As for E5-Mistral-b-instruct, it only achieved a 55% recall rate at k=10, which is significantly lower than the SFR-Embedding-Mistral and GritLM-7B models.

The experiment also utilized the translation dataset of configuration commands, consisting of 20 pairs of corresponding commands without parameters. The experiment selected the SFR-Embedding-Mistral and GritLM-7B, which performed well in the previous section, as well as Qwen-max and Qwen1.5-14B-chat for the complete method.

In this experiment, based on the given command without parameters, the corresponding command in the new device needs to be provided, along with the selection of keywords in the command. For cases where only semantic similarity calculation is used without LLM, only the command ranked first is considered. If the correct command is selected and there are no keywords to choose from, the result is deemed correct. However, if the correct command is selected but there are keywords to choose from, the result is considered incorrect.

For the two cases using LLM, based on the test results from the previous section, the correct command is always included in the Top 3. Therefore, input includes the top three ranked commands. The result is considered correct only if the output of the LLM is identical to the correct answer. Each command is tested 10 times. The experimental results of accuracy are shown in Table III.

TABLE III
ACCURACY OF TRANSLATING COMMANDS WITHOUT PARAMETERS

Embedding models	No LLM	Qwen1.5-14B-chat	Qwen-max
SFR-Embedding-Mistral	20%	**43.5%**	74.0%
GritLM-7B	**40%**	34.0%	**75.5%**

The combination of GritLM-7B+Qwen-max achieved the best results of 75.7%, while the combination of SFR-Embedding-Mistral+Qwen-max achieved similar performance of 74.0%, both are significant improvements compared to using semantic similarity calculation alone. However, the two combinations involving Qwen1.5-15b-chat yielded very poor results, showing little to no improvement. This might be due to that this task is significantly different from common tasks, and it is more challenging, making it difficult for the smaller model to comprehend and complete the task effectively. On the other hand, the Qwen-max model with billions of parameters exhibits better adaptability to uncommon tasks, enabling it to achieve better performance in this task.

E. Translating Full Configuration Files

Firstly, employing the prototype system and the hierarchical structure of Nokia commands from the NAssim dataset, an attempt was made to translate each file in the dataset. However, it was found that the results contained numerous view errors. Upon re-evaluating the command hierarchy of Nokia, it was discovered that the provided hierarchy in the dataset was incorrect.

As shown in Fig 9, the "bgp" command is used to enter the BGP view, and in the renderview, it should include the configuration path "config>router>bgp" information. However, this information was missing, leading to the failure of translation. Therefore, this evaluation selected ten configuration files containing 165 commands in total, representing 25 types of commands. After translating these commands, 180 commands representing 18 types were generated. A correct command hierarchy was created for these 18 commands, and the testing was conducted again.

The test results revealed that all translated commands could correctly complement higher-level commands and be placed

```
{  "cli": "bgp##config>router",
   "type": "cli",
   "renderview": [],
   "path": "nokia\\sr_160R5_public-Issue2.898026\\cmd_corpus\\bgp-cli\\bgp_0.json",
   "name": "bgp##config>router"}
```


Fig. 9. Wrong command hierarchy in NAssim dataset.

Configuration File Excerpt	Corresponding Translation Result
interface GigabitEthernet1/0/0 ospfv3 1 area 0.0.0.0 ospfv3 dr-priority 100 ospfv3 1	ospf3 1 area 0.0.0.0 --ospf3 [<ospf-instance>] [<router-id>]-- --area <area-id>-- --interface <ip-int-name> [secondary]-- priority 100

Fig. 10. Example of configuration logic differences.

under the correct views, but the following shortcomings are still identified.

Configuration Logic Differences. As shown in Fig. 10, in the original device, the configuration view sequence involves configuring OSPFv3 within the interface. However, in the new device, it's the reverse: configuring the interface within OSPFv3. This difference leads to some parameters not being located effectively. However, the supplementary view operation provides the superior views that need to be supplemented, ensuring the views are correct.

One Command Requiring Multiple Executions. As shown in Fig. 11, in the original device commands, both groups A and B belong to the IPv4 address group. However, only group A declares the IPv4 address group in the result. While in group B, there is no declaration, resulting in a lack of semantics.

V. Limitations

Scope of command coverage. All the files in the BGP and OSPFv3 configuration examples in the Huawei NE40E manual were tested in the evaluation because we believe that the two protocols are frequently discussed as important protocols in the network field, and that these examples are written in the manual means that these configurations are representative and commonly used. The results are shown in section IV. But for commands beyond these, it is challenging to theoretically determine which cases are unsupported. They may need to be discovered through actual use or testing, making it a project of ongoing modification and improvement.

Limitations of current methods. In addition to the shortcomings discovered in the section IV of the experiment, our current method does not support identifying or correcting such cases where two commands share the same semantics (similar functional descriptions) but differ in their underlying implementation details discussed in Campion. Although it proposes methods to detect this situation, it still requires a lot of effort to translate them correctly. This will be our future work.

Configuration File Excerpt	Corresponding Translation Result
bgp 65008 ipv4-family unicast peer a enable peer b enable	autonomous-system 65008 bgp group a family ipv4 group b

Fig. 11. Example of command requiring multiple executions.

VI. Related Work

A. Configuration management

In configuration management, configuration synthesis is an important task, which transforms high-level user requirements into low-level network device configurations. Currently, configuration synthesis research is mainly applied to traditional device networks, using formal methods, with SMT as the primary solving tool [30]–[33]. With the development of programmable networks and cellular network research and applications, configuration synthesis is also applied to P4 program [34] and cellular network configuration [35].

Configuration validation, another important task, checks whether the network behavior will conform to the user's intent. It involves formal methods for modeling and solving [36]–[40], similar to synthesis, using Datalog, BDD, and others.

B. LLM applications

Some research efforts about LLMs have focused on establishing general-purpose domain agents, including single-agent systems, such as AutoGPT [41] and BabyAGI [42], and multi-agent systems, such as AutoGen [43] and MetaGPT [44].

Additionally, some studies have demonstrated the effectiveness of LLMs in addressing specific domain problems. In the network operations domain, Nissist [45] and RCACopilot [9] utilizes LLMs to offer engineers recommendations based on historical knowledge for troubleshooting. In terms of network configuration generation, [11] explores using LLMs to generate configurations based on natural language instructions. NetConfEval [12] employs LLMs to produce P4 configurations.

VII. Conclusion

We present a configuration translation method based on the LLM and constraints called ConfigTrans. ConfigTrans includes three parts: configuration manual parsing, configuration command translation, and command arrangement, which can facilitate configuration experts in completing translation tasks more conveniently. Evaluation results show that ConfigTrans achieves an accuracy rate of 82.47% for translating commands with parameters and 75.5% for commands without parameters.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant Nos. 62072091, 62032013, 92267206 and U22B2005.

REFERENCES

[1] N. W. Jim Duffy. (2003, January) Cisco sues huawei over intellectual property. [Online]. Available: https://www.computerworld.com/article/2578617/cisco-sues-huawei-over-intellectual-property.html

[2] H. Chen, Y. Miao, L. Chen, H. Sun, H. Xu, L. Liu, G. Zhang, and W. Wang, "Software-defined network assimilation: bridging the last mile towards centralized network configuration management with nassim," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 281–297.

[3] OpenConfig. (2023) Openconfig. [Online]. Available: https://github.com/openconfig/public

[4] E. M. Bjorklund, "The yang 1.1 data modeling language," Internet Engineering Task Force (IETF), Proposed standard RFC7950, 2016.

[5] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: https://arxiv.org/abs/1908.10084

[6] L. Wang, C. Lyu, T. Ji, Z. Zhang, D. Yu, S. Shi, and Z. Tu, "Document-level machine translation with large language models," *arXiv preprint arXiv:2304.02210*, 2023.

[7] T. Zhang, F. Ladhak, E. Durmus, P. Liang, K. McKeown, and T. B. Hashimoto, "Benchmarking large language models for news summarization," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 39–57, 2024.

[8] S. Wang, X. Sun, X. Li, R. Ouyang, F. Wu, T. Zhang, J. Li, and G. Wang, "Gpt-ner: Named entity recognition via large language models," *arXiv preprint arXiv:2304.10428*, 2023.

[9] Y. Chen, H. Xie, M. Ma, Y. Kang, X. Gao, L. Shi, Y. Cao, X. Gao, H. Fan, M. Wen *et al.*, "Automatic root cause analysis via large language models for cloud incidents," in *Proceedings of the Nineteenth European Conference on Computer Systems*, 2024, pp. 674–688.

[10] P. Jin, S. Zhang, M. Ma, H. Li, Y. Kang, L. Li, Y. Liu, B. Qiao, C. Zhang, P. Zhao *et al.*, "Assess and summarize: Improve outage understanding with large language models," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 1657–1668.

[11] M. Bogdanov, "Leveraging advanced large language models to optimize network device configuration," Ph.D. dissertation, Purdue University Graduate School, 2024.

[12] C. WANG, M. SCAZZARIELLO, A. FARSHIN, S. FERLIN, D. KOSTIĆ, and M. CHIESA, "Netconfeval: Can llms facilitate network configuration?" 2024.

[13] OpenAI, "Models," 2024. [Online]. Available: https://platform.openai.com/docs/models

[14] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024.

[15] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, and H. Wang, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2023.

[16] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems*, vol. 35, pp. 22 199–22 213, 2022.

[17] Amy. (2022) Nassim datasets. [Online]. Available: https://github.com/AmyWorkspace/nassim

[18] Huawei. (2023, July) Huawei ne40e product documentation. [Online]. Available: https://support.huawei.com/hedex/hdx.do?docid=EDOC1100299074

[19] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[20] huggingface. Models with the tag of sentence similarity. [Online]. Available: https://huggingface.co/models?pipeline_tag=sentence-similarity&sort=trending

[21] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers, "Mteb: Massive text embedding benchmark," *arXiv preprint arXiv:2210.07316*, 2022. [Online]. Available: https://arxiv.org/abs/2210.07316

[22] M. Rui, L. Ye, J. Shafiq Rayhan, X. Caiming, Z. Yingbo, and Y. Semih, "Sfr-embedding-mistral:enhance text retrieval with transfer learning," Salesforce AI Research Blog, 2024. [Online]. Available: https://blog.salesforceairesearch.com/sfr-embedded-mistral/

[23] N. Muennighoff, H. Su, L. Wang, N. Yang, F. Wei, T. Yu, A. Singh, and D. Kiela, "Generative representational instruction tuning," 2024.

[24] L. Wang, N. Yang, X. Huang, L. Yang, R. Majumder, and F. Wei, "Improving text embeddings with large language models," *arXiv preprint arXiv:2401.00368*, 2023.

[25] OpenAI, "Chatgpt," https://www.openai.com/chatgpt, accessed: 2024-05.

[26] huggingface. open llm leaderboard. [Online]. Available: https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard

[27] Y. Liu, C. Pei, L. Xu, B. Chen, M. Sun, Z. Zhang, Y. Sun, S. Zhang, K. Wang, H. Zhang, J. Li, G. Xie, X. Wen, X. Nie, and D. Pei, "Opseval: A comprehensive task-oriented aiops benchmark for large language models," 2023.

[28] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang, B. Hui, L. Ji, M. Li, J. Lin, R. Lin, D. Liu, G. Liu, C. Lu, K. Lu, J. Ma, R. Men, X. Ren, X. Ren, C. Tan, S. Tan, J. Tu, P. Wang, S. Wang, W. Wang, S. Wu, B. Xu, J. Xu, A. Yang, H. Yang, J. Yang, S. Yang, Y. Yao, B. Yu, H. Yuan, Z. Yuan, J. Zhang, X. Zhang, Y. Zhang, Z. Zhang, C. Zhou, J. Zhou, X. Zhou, and T. Zhu, "Qwen technical report," *arXiv preprint arXiv:2309.16609*, 2023.

[29] Q. Team. Notes on qwen-max-0428. [Online]. Available: https://qwenlm.github.io/blog/qwen-max-0428/

[30] A. El-Hassany, P. Tsankov, L. Vanbever, and M. Vechev, "Network-wide configuration synthesis," in *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II 30*. Springer, 2017, pp. 261–281.

[31] ——, "Netcomplete: Practical network-wide configuration synthesis with autocompletion," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, 2018, pp. 579–594.

[32] B. Tian, X. Zhang, E. Zhai, H. H. Liu, Q. Ye, C. Wang, X. Wu, Z. Ji, Y. Sang, M. Zhang *et al.*, "Safely and automatically updating in-network acl configurations with intent language," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 214–226.

[33] A. Abhashkumar, A. Gember-Jacobson, and 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.

[34] K.-F. Hsu, R. Beckett, A. Chen, J. Rexford, and D. Walker, "Contra: A programmable system for performance-aware routing," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 701–721.

[35] A. Mahimkar, A. Sivakumar, Z. Ge, S. Pathak, and K. Biswas, "Auric: using data-driven recommendation to automatically generate cellular configuration," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 807–820.

[36] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. Millstein, "A general approach to network configuration analysis," in *12th {USENIX} symposium on networked systems design and implementation ({NSDI} 15)*, 2015, pp. 469–483.

[37] A. Gember-Jacobson, R. Viswanathan, A. Akella, and R. Mahajan, "Fast control plane analysis using an abstract representation," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 300–313.

[38] 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, pp. 155–168.

[39] P. Zhang, D. Wang, and A. Gember-Jacobson, "Symbolic router execution," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 336–349.

[40] P. Zhang, A. Gember-Jacobson, Y. Zuo, Y. Huang, X. Liu, and H. Li, "Differential network analysis," in *USENIX NSDI*, 2022.

[41] S. Gravitas, "Auto-gpt," *URL: https://github. com/Significant-Gravitas/AutoGPT*, 2023.

[42] Y. Nakajima, "Babyagi," *URL: https://github. com/yoheinakajima/babyagi*, 2023.

[43] Q. Wu, G. Bansal, J. Zhang, Y. Wu, S. Zhang, E. Zhu, B. Li, L. Jiang, X. Zhang, and C. Wang, "Autogen: Enabling next-gen llm applications via multi-agent conversation framework," *arXiv preprint arXiv:2308.08155*, 2023.

[44] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, C. Zhang, J. Wang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu,

and J. Schmidhuber, "Metagpt: Meta programming for a multi-agent collaborative framework," 2023.

[45] K. An, F. Yang, L. Li, Z. Ren, H. Huang, L. Wang, P. Zhao, Y. Kang, H. Ding, Q. Lin *et al.*, "Nissist: An incident mitigation copilot based on troubleshooting guides," *arXiv preprint arXiv:2402.17531*, 2024.