ConfigReco: Network Configuration Recommendation with Graph Neural Networks

Zhenbei Guo, Fuliang Li, *Member, IEEE,* Jiaxing Shen, *Member, IEEE,*
Tangzheng Xie, Shan Jiang, Xingwei Wang, *Member, IEEE*

***Abstract*—Configuration synthesis is a fundamental technology in the context of self-driving networks, aimed at mitigating network outages by intelligently and automatically generating configurations that align with network intents. However, existing tools often fall short in meeting the practical requirements of network operators, particularly in terms of generality and scalability. Moreover, these tools disregard manual configuration which remains the primary method employed for daily network management. To address these challenges, this paper introduces ConfigReco, a novel, versatile, and scalable configuration recommendation tool tailored for manual configuration. ConfigReco facilitates the automatic generation of configuration templates based on the network operator's intent. First, ConfigReco leverages existing configurations as input and models them using a knowledge graph. Second, graph neural networks are employed by ConfigReco to estimate the significance of nodes within the configuration knowledge graph. Lastly, configuration recommendations are made by ConfigReco based on the computed importance scores. A prototype system has been implemented to substantiate the effectiveness of ConfigReco, and its performance has been evaluated using real-world configurations. The experimental results demonstrate that ConfigReco achieves a coverage rate of 93.35% while concurrently maintaining a redundancy rate of 23.07% within a configuration knowledge graph comprising $890,464$ edges and $40,885$ nodes. Furthermore, ConfigReco exhibits high scalability, enabling its applicability to arbitrary datasets, while simultaneously providing efficient recommendations within a response time of 1 second.**

***Index Terms*—network management, configuration synthesis, graph neural network, knowledge graph, configuration recommendation**

I. Introduction

In self-driving networks, one of the key areas of digital twins, network configuration plays a crucial role in the development and operation such as fault detection, network optimization, and network automation [1]–[7]. Network configuration contains distributed network protocols with different parameters (Figure 1), which determine the correct implementation of network functions. However, managing configuration is challenging and brittle [2], leading to severe network outages [6]–[8]. To this end, there is an increasing trend in configuration synthesis [1]–[4] which automatically generates network-wide configurations that satisfy routing policies [1], [2] and network management objectives [3].

Existing configuration synthesis tools effectively address human-induced misconfigurations and intelligent network configuration. However, they face practical limitations for three main reasons. *(i) Poor generality*: Many synthesis tools support only a limited set of routing protocols (e.g., OSPF and BGP [2], [3]), resulting in operators having to manually configure other unsupported network protocols (Figure 1). Manual intervention becomes necessary for handling these uncovered protocols. *(ii) Limited scalability*: Data-driven synthesis tools like Aurora [4] may struggle to adapt to datasets with diverse configurations in terms of size and protocol distribution. SMT-based tools like NetComplete [2] may have efficiency issues in large-scale or realistic networks. *(iii) Lack of interpretability*: Synthetic configurations differ wildly from hand-crafted ones [2]. Synthesis tools provide limited means to interpret them, making it difficult for operators to understand how configurations are generated and rarely deploy them. These limitations of existing tools make manual configurations still the preferred approach in daily network management. For example, operators manually edit configuration templates, add new services or devices, and update existing configurations. However, existing tools lack assistance for manual configuration modifications. Although embedded network operating systems (NOS) offer completion functions for internal configuration commands based on initials, they fail to provide comprehensive suggestions or generate corresponding configuration segments based on user input. For example, in Figure 2a, the built-in completion function can complement command *policy-statement* based on its first letter *p*. However, it cannot recommend the sub-command *POL* of *policy-statement* or the entire configuration segment.

In addition, we observe that devices of the same type, such as routers, have the feature of *configuration similarity*. As reported in a survey, 90% of network operators implement unified configuration management by maintaining similar/identical configurations across devices of the same type [3]. They deploy similar/identical configurations by using uniform templates embedded in scripts, or by replicating configurations verbatim (such as security policy). Configurations from existing devices of the same type are more acceptable to operators when writing new configuration files. Thus, a practical and interesting research question motivated by *configuration similarity* and the aforementioned challenges is: *Can we design a configuration recommendation tool for manual configurations to cover different network protocols and scale to any datasets?*

Z. Guo, F. Li, T. Xie, and X. Wang are with the Northeastern University, Shenyang 110819, China. E-mail: guozb777@163.com, lifuliang@cse.neu.edu.cn, 2201891@stu.neu.edu.cn, wangxw@mail.neu.edu.cn. J. Shen is with the Department of Computing and Decision Sciences, Lingnan University, HKSAR. E-mail: jiaxingshen@ln.edu.hk. S. Jiang is with the Department of Computing, The Hong Kong Polytechnic University, HKSAR. E-mail: cs-shan.jiang@polyu.edu.hk.
(Corresponding authors: Fuliang Li and Xingwei Wang.)

In this paper, we provide an affirmative answer to the question by proposing ConfigReco, a general and scalable configuration synthesis tool that recommends configuration templates based on the operator's intent. A configuration template is a file with multiple segments for different network functions as shown in Figure 2a. Configuration commands form a vast space distributed among segments. ConfigReco explores the existing configuration space to identify templates in the network and recommends the matching template based on *configuration similarity*.

Our vision, however, faces three challenges in reality. First, accurately capturing configuration semantics and syntax is essential to avoid invalid configurations in the formal language of networks. Second, certain configuration commands are unique to specific devices, while conflicting functions can coexist in the same place (e.g., *accept/reject* in Figure 2a). Understanding the context of configuration commands is crucial to prevent errors and security issues. Third, determining the appropriate recommendation range is important. A wide range may (e.g., multiple segments) introduce unexpected configurations, while a narrow scope like a single command may not fulfill operator needs.

To address the challenges, we capture the semantics and syntax of existing configurations by translating them into *configuration knowledge*. We then build a configuration knowledge graph that explores the configuration space by allowing neighbors on the graph to be evaluated. A three-layer graph neural network (GNN) estimates the importance of neighbors based on a specified command as the central node (Figure 3). Configuration segments independently implement network functions (Figure 2a). Therefore, the configuration template's segments serve as minimal recommendation units, with the first command (e.g., *policy-statement*) as the central node. Multiple inputs yield different configuration recommendations for desired network functionality (Figure 2b). Lastly, recommendations consider *configuration similarity* to ensure efficiency and operator acceptance.

To validate the performance of ConfigReco, we randomly recommend $10,000$ times in the real-world configuration from *Internet2* [9]. The experimental results indicate that the overall coverage rate of ConfigReco reaches 93.35% with a redundancy rate of 23.07%. In summary, our contributions could be summarized as three folds:

- We are the first to propose a recommendation tool to recommend configuration templates for manual configuration.
- We present the design and implementation of ConfigReco using a knowledge graph and graph neural network to generate the configuration template for a newly added configuration file.
- We implement a prototype system and verify its performance in real-world configurations.

We first present the background (Section II). Then, we show the design overview (Section III) and details of ConfigReco (Section IV), followed by the experiment results (Section V). Finally, we discuss related works (Section VI) and conclude (Section VII).

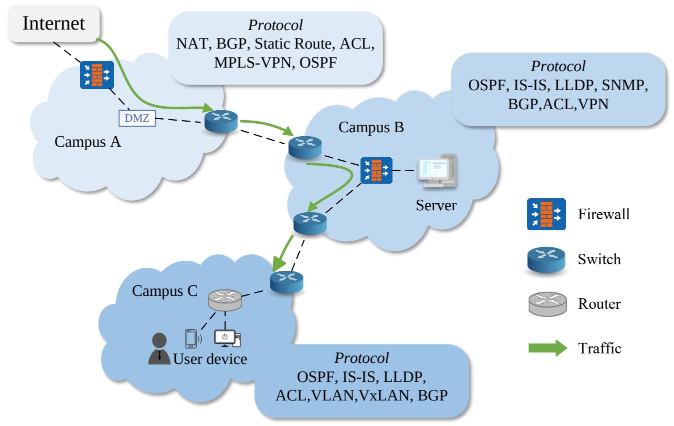

Internet
Protocol
NAT, BGP, Static Route, ACL, MPLS-VPN, OSPF
Protocol
OSPF, IS-IS, LLDP, SNMP, BGP,ACL,VPN
DMZ
Campus A
Campus B
Server
Campus C
User device
Protocol
OSPF, IS-IS, LLDP, ACL,VLAN,VxLAN, BGP
Firewall
Switch
Router
Traffic

Fig. 1. A campus network with different devices running various protocols.

II. Background

This section first introduces a campus network with common protocols and then describes the background of the knowledge graph and GNN.

A. Campus Network

Figure 1 shows a campus network that includes three different autonomous systems (ASes), each running a large number of network protocols. Network operators usually divide ASes (campuses) by network function. For instance, *Campus A* controls access to the external/internal network. Typically, devices in the same autonomous system (AS) communicate with the internal gateway protocol (IGP) like OSPF, while ASes communicate with each other via border gateway protocol (BGP). Existing tools can effectively synthesize partial routing protocols (e.g., OSPF [2], [3] and BGP [1]–[3]), but there are still other protocols like VLAN/VxLAN that require manual intervention.

B. Knowledge graph and GNN

Recently, the knowledge graph and GNN have attracted increasing attention in fields such as relation extraction [10] and importance estimation [11].

Knowledge Graph A knowledge graph (KG) is a multi-relational graph used to represent and store *knowledge*, where nodes correspond to entities, and edges correspond to relations between head and tail entities. The central concepts of a knowledge graph are entities and relationships. Entities represent concrete objects or abstract concepts in the real world (such as people, places, and things), while relationships represent the connections and associations between entities. Entities and their relations constitute *knowledge*. For example, a *configuration knowledge* (Figure 2a) is denoted by <*policy-statement*, *contain*, *POL*>, where *policy-statement* and *POL* are head and tail entities respectively, and *contain* is the relation between them. KG enables semantic reasoning and inference over knowledge. By leveraging the semantic associations between entities and relations in the graph, it can perform inference to discover new knowledge, fill knowledge gaps, or verify knowledge consistency. KGs such as DBpedia [12] have

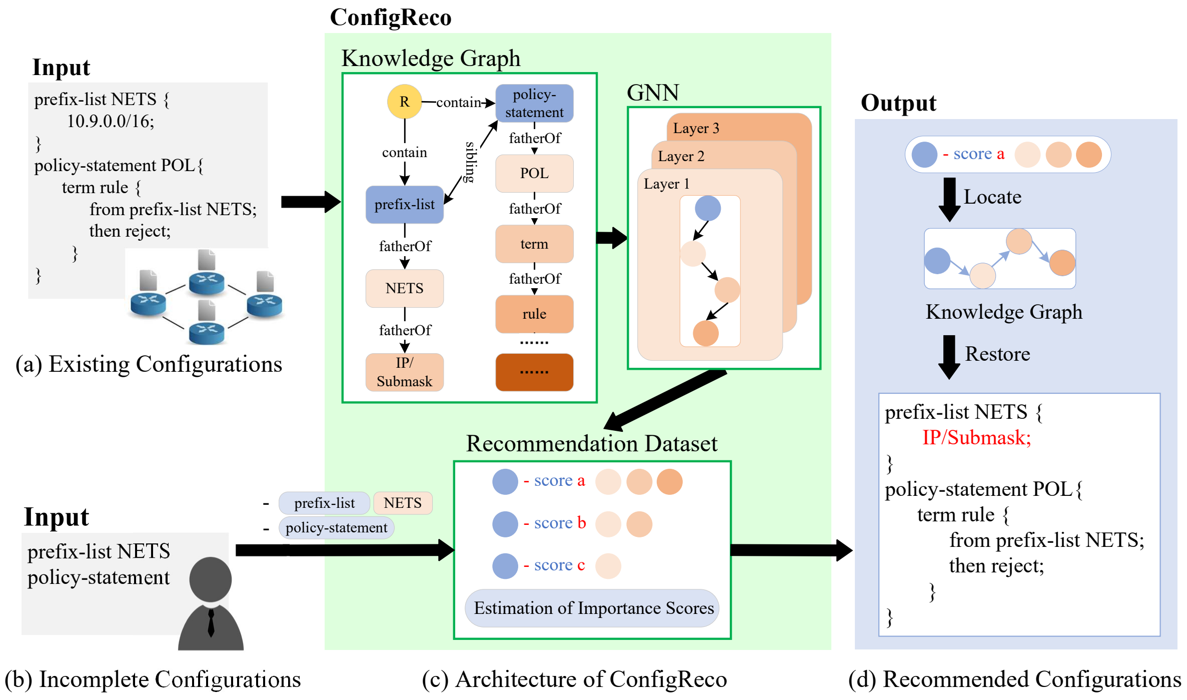


Fig. 2. The design overview of ConfigReco. Fig 2a shows the existing configurations used to construct the knowledge graph. Fig 2b shows the incomplete configurations inputted by operators. Fig 2c shows the architecture of ConfigReco. Fig 2d shows the output based on the incomplete configurations shown in Fig 2b.

proven to be highly practical resources and have been applied in different areas, such as explainable recommendation [13].

GNN Graph Neural Networks, as a sub-field of artificial intelligence (AI), are a class of deep learning methods designed to perform learning and inference tasks on graphs. Graph Neural Networks can learn node-level, edge-level, and graph-level representations by aggregating information from neighboring nodes and edges, providing an easy way to exploit the structural information to enhance learning and inference tasks [10], [11], [14]. Generally, in a GNN with L layers, an operator can specify L based on the characteristics of the current graph-structured data. l denotes the current layer, and each node i in l-th layer ($l \geq 1$) receives the new feature vectors from (l-1)-th layer. Then each node i updates itself by aggregating those feature vectors from their neighbors in the l-th layer. The feature vectors of the 0-th layer can be given flexibly by the operators (*Scoring Network* shown in Figure 3). In addition, we can assign labels to specific nodes, designating them as central nodes to better accomplish the training task [11]. It allows the model to focus on these nodes, making them more influential in the GNN learning process.

III. Design Overview

In this section, we first present the design overview of ConfigReco shown in Figure 2, including input, model architecture, and output. The detailed implementation is shown in Section IV.

Input The input is divided into *(a) Existing Configurations* and *(b) Incomplete Configurations*. ConfigReco constructs the knowledge graph and trains the recommendation dataset based on existing configurations. Figure 2a shows partial configurations in Juniper format, with curly braces and indentations to indicate the configuration hierarchy, while other formats like Cisco use indentation. We can quickly split each configuration segment based on these curly braces and indentations. For incomplete configurations, operators can customize them according to their intent, and then ConfigReco recommends configurations based on them.

Model Architecture ConfigReco consists of three components *(i) Knowledge Graph*, *(ii) Three-layer GNN*, and *(iii) Recommendation Dataset*. ConfigReco first constructs a configuration knowledge graph based on existing configurations, transforming the configurations into graph-structured data. Second, a three-layer GNN is applied to the configuration knowledge graph to estimate the importance scores of the central nodes (e.g., *prefix-list*). Finally, the scores of all central nodes and their children constitute the recommendation dataset.

Output ConfigReco matches the input incomplete configurations in the dataset and outputs the configuration template with the highest score. Note that some existing parameters are replaced by pre-defined keywords, such as 10.9.0.0/16 is replaced by the keyword *IP/Submask*. These parameters are usually meaningless in a new configuration file. For example, the network interface names of the current device differ. More importantly, it prevents the matching from failing due to different parameters during recommendations.

IV. Design Detail

We present the design details of ConfigReco in this section. We first present the knowledge graph construction, followed by the GNN implementation. Finally, we show the configuration recommendation.

A. Knowledge Graph Construction

We first transform the existing configurations into configuration knowledge in the form of triples. Configuration commands serve as head and tail entities, and they are linked by custom keywords (e.g., *contain*) to form knowledge triples. We accurately capture configuration semantics through knowledge in atomic form, and ensure the accuracy of configuration syntax through the graph structure formed by multiple relevant knowledge. Since we only need the entities and the relations between them to build a knowledge graph, we can generalize to arbitrary configuration datasets. The partial configurations shown in Figure 2a are converted into knowledge as follows.

```
<R1, contain, prefix-list>
<prefix-list, fatherOf, NETS>
```

where $R1$ denotes the current device/file name. We define auxiliary keywords to help build triples, which will not be recommended by ConfigReco as configurations to network operators. For example, we use the keyword *contain* to indicate which central nodes are included by the current device and keyword *fatherOf* to denote the sub-commands.

Based on the configuration knowledge, ConfigReco creates a configuration knowledge graph managed by the graph database management system Neo4j. For example, *<R1*, *contain*, *prefix-list>* are added into the graph database as follows:

```
MATCH (a:Keyword {name:'R1'}),
      (b:Keyword {name:'prefix-list'})
MERGE (a)-[:contain]->(b)
```

where ConfigReco creates head node *R1* and tail node *prefix-list*, linking them by the relation *contain*. Finally, a configuration knowledge graph example is shown in Figure 2. To reiterate, ConfigReco only needs to capture the semantics and syntax of configurations, rather than simulating network protocols. Thus, ConfigReco can support arbitrary network protocols.

B. GNN implementation

Figure 3 shows the three-layer GNN model employed by ConfigReco to estimate importance scores. In general, GNN propagates information among neighbors through node embedding, causing entities and their neighbors to influence one another. Therefore, an entity can combine its current neighbors for better representation. For example, *policy-statement* can be combined with *POL* to better represent its position in the current configuration space. In node importance estimation, the importance of a node is jointly determined by its multi-layer neighbors, such that *rule* is the three-layer neighbor of *policy-statement*.

A single GNN explores the direct neighbors (Layer 1) of a node and directly aggregates the importance scores of the neighbors (Figure 3b [14]). GNN, however, with multiple layers can explore more neighbors, making the importance score more reliable [11]. In addition, most configuration commands have no more than three layers of neighbor depth, such as prefix-list has only two-layer neighbors. Therefore, the three-layer neighbors of the central node are sufficient to cover a large number of configuration contents, and we design a three-layer GNN to estimate the importance score. In addition, we adopt multiple score aggregation (SA) heads in each SA layer as multiple SA heads to be helpful for the model performance and the stability of optimization procedure [11].

Scoring Network GNN naturally allows us to train a model with flexible adaptation by scoring the network using pre-defined importance scores. We design two different scoring networks to better capture the influence of the central node and its neighbors on each other in the configuration space.

- *Scoring Network A*: We calculate the dependence of the central node i on its neighbor j by $W_{ij}^{l} = \frac{num(i,j)}{num(i)}$. $num(i,j)$ denotes the number of $< i, j >$, where i and j are configured together. $num(i)$ denotes the total number of i in configurations.
- *Scoring Network B*: We calculate the dependence of the neighbor j on the central node i by $W_{ji}^{l} = \frac{num(j,i)}{num(j)}$.

The weight W is inputted into the corresponding layer of GNN as the initialization importance score such as $s_1^0(j_1)$. Note that the initial score of the central node i is set to 0 in this paper. In addition, W is used to calculate the attention coefficient α.

Score Aggregation The score aggregation of the central node i at the l layer is denoted by follows:

$$s_h^l(i) = \begin{cases} Scoring\ Network, & l = 0 \\ s_h^{(l-1)}(i) + \sum_{j=1}^{N} \alpha_{ij}^{h} \times s_h^l(j), & l \geq 1 \end{cases} \tag{1}$$

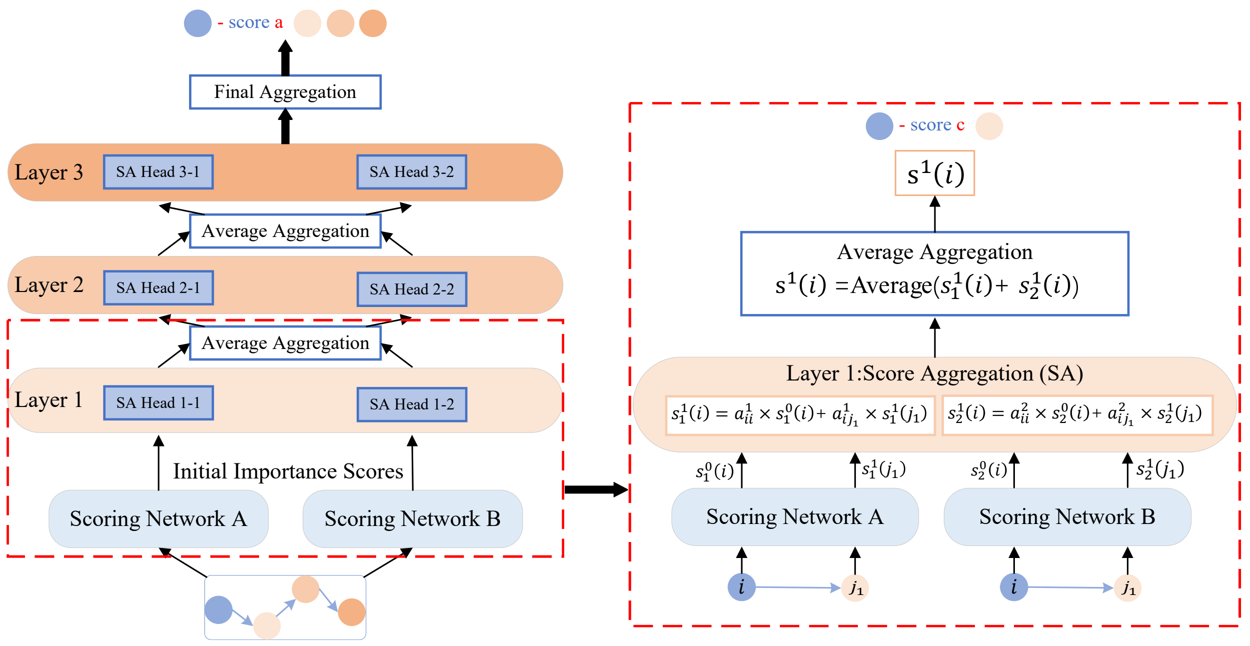
where h denotes an index of an SA head, N is the number of i's neighbours in the l layer, and α_{ij}^h is the attention coefficient of i to j [14] calculated as follows:

$$\alpha_{ij} = \frac{exp(LeakyReLU(\vec{a}^T[W^l(\vec{f_i})||W^l(\vec{f_j})]))}{\sum_{k\in N} exp(LeakyReLU(\vec{a}^T[W^l(\vec{f_i})||W^l(\vec{f_k})]))} \tag{2}$$

where *LeakyReLU* is an activation function commonly used in artificial neural networks, $\vec{a}$ is a weight vector, $\vec{f_i}$ is the feature vector of node i in the current graph dataset, $\cdot^T$ represents transposition, and $||$ denotes the concatenation operation. In this paper, we replace $\vec{a}^T[W^l(\vec{f_i})||W^l(\vec{f_j})]$ with $W_{ij}^l = \frac{num(i,j)}{num(i)}$ and $W_{ji}^l = \frac{num(j,i)}{num(j)}$ in the scoring networks *A* and *B*, respectively. The attention coefficient α determination of which neighbors should be concerned by assigning weights that indicate their importance to the central node. We use the attention coefficient to focus on configuration commands that are frequently configured with the central node, avoiding rare configurations being ignored.

When all the SA heads are computed, we calculate their average value (*Average and Final Aggregation*) and transform it to the next layer. After the final aggregation, the importance score of the central node i and its neighbors are stored in the recommendation dataset by a pre-defined data structure D_m^i. We make configuration recommendations based on the recommendation dataset.

Configuration Recommendation Let D_m^i be the data structure to store the score of the central node i and its neighbors, where m denotes the record index. Based on the



(a) Three-layer GNN (b) Single GNN

Fig. 3. An illustration of node importance estimation. Fig 3a: The three-layer GNN model employed by ConfigReco to estimate importance scores. Fig 3b: Estimation of the importance of node i based on the neighbor j at a single score aggregation layer.

input (Figure 2b), We recommend configurations based on the following two rules:

1. Prefer longer matching sequences;
2. Prefer higher importance score.

For example, we first locate the input *"prefix-list NETS"* in the $D^{\textit{prefix-list}}$. We then search the records in $D^{\textit{prefix-list}}$ that contain neighbor *NETS* and subsequently select the one with the highest score from among them. If matching *NETS* fails, we preferentially select the record with the highest score.

V. Evaluation

A. Experiment Setup

We implement ConfigReco in Java and Python and create a prototype system through Neo4j that manages the constructed knowledge graph and visualizes the specified configurations. We compare the recommendation performance of ConfigReco and GAT [14]. GAT is a graph attention network with a single-layer neural network that operates on graph-structured data. ConfigReco and GAT perform on a server with three 2.30 GHz Inter(R) Xeon(R) Gold 5218 CPU and $16GB$ RAM.

Dataset We run extensive experiments on real-word network configurations from the *Internet2*. There are 34 devices, of which 24 devices have an average configuration statement of 1260 and 10 devices have an average configuration statement of 8236. In addition, we divide four datasets ($CS1$ to $CS4$) according to the *configuration similarity*, they range in configuration size from 18,388 to 112,600 lines shown in Table I, and each dataset contains at least two protocols (such as BGP and OSPF). We run $10,000$ configuration recommendations randomly on each dataset.

Recommendation For each recommendation, we randomly intercept existing configurations as input. We set three different input lengths, $Length1$, $Length2$, and $Length3$. $Length1$ only contains the central node, while $Length2$ and $Length3$ contain one and two neighbours, respectively. For example, the length of input *prefix-list NETS* (Figure 2b) belongs to $Length2$. Note that in practice, an operator might enter a completely new configuration command as the central node. Since it does not exist in the dataset, ConfigReco would return *NULL*. ConfigReco does not update this new command for two important reasons. First, it could have been an error command entered by the operator. Second, a new configuration command needs to be treated with caution, as it can cause extensive configuration changes and network outages [2], [3], [6].

Performance Let Y be the original configurations, and $X = \{x_{rec}, x_{red}\}$ denotes the recommendation configurations. x_{rec} denotes the configurations that exist in Y and x_{red} denotes the redundancy configurations that do not exist in Y. We evaluate the performance of ConfigReco from the following three aspects.

- The coverage rate. We calculate the coverage rate of ConfigReco by $\frac{x_{rec}}{Y}$. The coverage rate reflects the effectiveness of ConfigReco recommendation. The higher the coverage rate, the more accurate the recommendation.
- The redundancy rate. We calculate the redundancy rate of ConfigReco by $\frac{x_{red}}{X}$. The redundancy rate is an important metric, which reflects the configuration composition with the highest importance score based on the input. Thus, redundant configurations can provide an important reference for operators to complement the configuration more completely.
- *Runtime*. We record the time spent on knowledge graph construction and configuration recommendation. The runtime can give us more insight into ConfigReco's perfor-

TABLE I
THE CONSTRUCTED KNOWLEDGE GRAPH BASED ON THE DATASETS DIVIDED BY *configuration similarity*.

CS	CL	Edge	Node	CT (s)	RT (s)
CS1	18388	695535	23702	589.5	0.22
CS2	31831	890464	40885	992.3	0.28
CS3	46313	1033325	59136	1470.4	0.49
CS4	112600	3792997	146081	3621.4	0.82

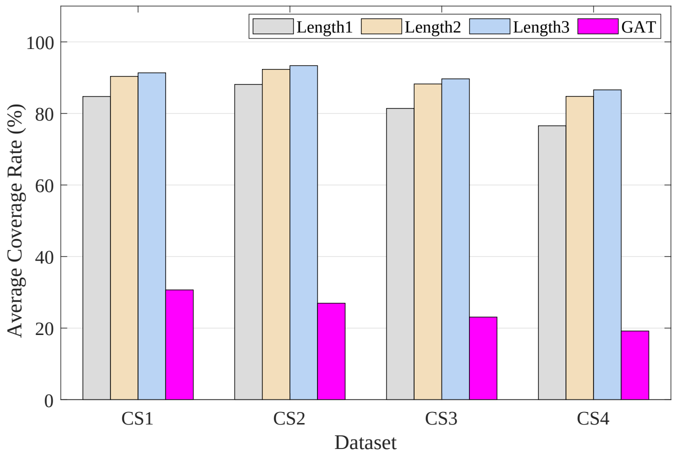


Fig. 4. The average coverage rate of ConfigReco and GAT.

mance.

B. Numerical Analysis

Knowledge graph construction Table I shows the knowledge graph constructed by ConfigReco, including the four different datasets $CS1$ to $CS4$, CL (configuration line), edge, node, CT (construct time), and RT (recommendation time). The construction time of the knowledge graph is equivalent to the extraction time of configuration knowledge. Both CT and RT are positively related to the number of configuration lines and configuration complexity. ConfigReco can parse $18,388$ line configurations in 10 minutes based on our server hardware. Or it can resolve $112,600$ line configurations in 1 hour and build a knowledge graph with $3792,997$ edges and $146,081$ nodes. Finally, ConfigReco can recommend configurations within 1s in all datasets.

Coverage Rate Figure 4 shows the average coverage rate of ConfigReco and GAT, where ConfigReco is higher than GAT. First, the coverage rate and input length are positively correlated, resulting in $Length3$ obtaining the highest coverage rate. Such as in $CS4$, ConfigReco achieves the lowest coverage rate of 76.55% under $Length1$, and it grows to 86.59% at $Length3$. Second, the coverage rate decreases with the size of the dataset. The coverage rate of $Length2$ and $Length3$ in datasets $CS1$ and $CS2$ both exceed 90%, with the highest reaching 92.32% and 93.35%, respectively. They are reduced to 84.77% and 86.59% respectively in $CS4$. With the increase in database size, such as from $18,388$ lines ($CS1$) to $112,600$ lines ($CS4$), more configuration templates with complex structures can be matched leading to a decrease in coverage rate. However, ConfigReco still achieves at least a coverage rate of 76.55% after $10,000$ recommendations, and the coverage rate can be optimized by increasing the input length or adopting more layers to explore further neighbors. Finally, the coverage rate of ConfigReco is higher than that of GAT, which has a maximum coverage rate of only 30.68%. This result suggests that a GNN with more layers that can aggregate scores from more neighbors makes the results more accurate and reliable.

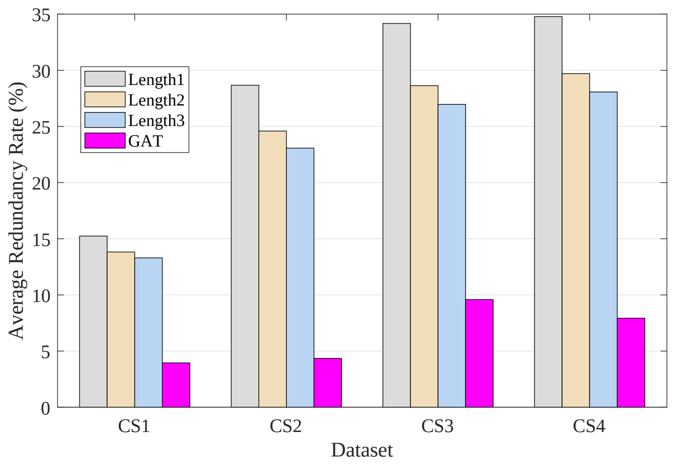


Fig. 5. The average redundancy rate of ConfigReco and GAT.

Redundancy Rate Figure 5 shows the redundancy rate of ConfigReco and GAT, where GAT achieves the smallest redundancy rate. Contrary to the coverage rate, the redundancy rate decreases with input length and is positively related to the dataset. ConfigReco obtains the smallest redundancy rate of 13.30% ($Length3$ and $CS1$), at which point it has a coverage rate of 91.35%. When ConfigReco achieves the lowest coverage rate of 76.55%, it achieves the highest redundancy of 34.78%. Although GAT has the lowest redundancy rate (3.95% to 9.258%), it also has the lowest coverage rate.

To sum up, ConfigReco has a redundancy rate of 23.07% while achieving the highest coverage rate of 93.35% ($Length3$ and $CS2$). Even with a six-fold increase in dataset size ($18,388$ to $112,600$ lines), ConfigReco still achieves a coverage rate of 86.59% after $10,000$ recommendations, which can be optimized by modifying model parameters. It demonstrates that ConfigReco can implement effective configuration recommendations for any size dataset. Moreover, ConfigReco only needs the configuration context to build the knowledge graph and does not need to simulate network protocols. From this, ConfigReco can be scaled to arbitrary datasets.

VI. RELATED WORK

Configuration Synthesis Synthesizers aim to generate low-level network configurations from high-level network intents by using their own DSL (Domain-Specific Language) or existing techniques. Aurora [4] focuses on tuning configuration parameters for LTE/5G cellular networks. Aurora [4] identifies important parameters from existing/historical configurations through a supervised machine-learning approach and performs performance-based filtering to make conformity-based recommendations. However, recommending existing parameters for network protocols (such as BGP and OSPF) may lead to network errors, such as routing loops and black holes. Propane [1] introduces RIR (Regular Intermediate Representation) to express constraints on intents. However, Propane [1] is limited to BGP protocol and cannot handle dynamic configuration changes. NetComplete [2] and AED [3] support multiple routing protocols (static route, OSPF, and BGP), and they can synthesize configurations incrementally based on existing configurations. Although their practicability improves, NetComplete [2] may be very slow or fail to complete for large networks, and AED [3] is difficult to support other protocols. The state-of-the-art synthesizer Aura [15] that a production-level synthesis system for datacenter routing policies, supporting BGP and OpenR routing protocols. Aura [15] enables network operators to express high-level intents to be automatically configured into the switch policy implementation with minimal reconfiguration.

Configuration Verification Verifiers play an important role in the field of networking by ensuring the correctness and consistency of configurations and network intents. Network configuration verifiers can be divided into control plane [5], [7] and data plane [6], which differ in terms of the scope of verification and focus. Control plane verifiers [5], [7] focus on the control plane of network devices, primarily validating the configurations and intents to ensure their behavior in the network aligns with expectations. Control plane verifiers typically simulate or model network devices and their configurations to validate and check for errors, conflicts, and inconsistencies. This helps prevent network failures and security vulnerabilities caused by configuration errors before actual deployment. Control plane verifier Tiramisu [5] models configurations by abstract interpretation based on the graph, and it supports multiple routing protocols such as OSPF, BGP, and VLANs. Tiramisu [5] can verify if policies hold under failures for various real-world and synthetic configurations within 2.2s. However, it is not optimized for configuration changes and needs to restart the simulation from scratch when configuration changes. The incremental verifier DNA [7] has been proposed to address this challenge, which uses the differential network analysis to identify differences in end-to-end forwarding behaviors arising from control plane changes. For data plane verifiers, they directly check the forwarding information like FIB (Forwarding Information Base) to verify network invariants like blackhole-freedom and loop-freedom. APKeep [6] can achieve sub-millisecond verification time, which is much faster than the control plane verifiers. In addition, it has addressed the problem of EC (equivalence class) explosion problem by incrementally maintaining the minimum number of ECs via a modular network model PPM (Port-Predicate Map). Compared to the control plane verifiers, the data plane verifiers are closer to the true forwarding behavior of network devices, which helps to detect potential issues such as packet loss when devices handle real traffic.

VII. Conclusion

This paper introduces ConfigReco, a novel configuration recommendation tool. ConfigReco leverages a knowledge graph to model network configurations instead of relying on simulations, enabling support for arbitrary network protocols. ConfigReco considers the configuration segment as the minimum recommended unit, capable of independently implementing a network function. The central node is defined as the first configuration command within each segment. By employing a three-layer GNN, ConfigReco explores the neighbors of the central node and computes their importance scores. Consequently, ConfigReco matches operator intents and provides configuration recommendations based on these scores. Experimental results demonstrate that ConfigReco achieves a coverage rate of 93.35% with a redundancy rate of 23.07%, delivering recommendations within 1 second.

Acknowledgment

We would like to thank the Editors and Reviewers of IEEE Network Magazine in advance for their review efforts and helpful feedback. This work is supported by the National Natural Science Foundation of China under Grant Nos. U22B2005, 62072091, 62032013, and 92267206, and the financial support of Lingnan University (LU) (DB23A9) and Lam Woo Research Fund at LU (871236).

References

[1] R. Beckett, R. Mahajan, T. D. Millstein, J. Padhye, and D. Walker, "Don't mind the gap: Bridging network-wide objectives and device-level configurations," in *Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016*, M. P. Barcellos, J. Crowcroft, A. Vahdat, and S. Katti, Eds. ACM, 2016, pp. 328–341. [Online]. Available: https://doi.org/10.1145/2934872.2934909

[2] A. El-Hassany, P. Tsankov, L. Vanbever, and M. T. Vechev, "Netcomplete: Practical network-wide configuration synthesis with autocompletion," in *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*, S. Banerjee and S. Seshan, Eds. USENIX Association, 2018, pp. 579–594. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/el-hassany

[3] A. Abhashkumar, A. Gember-Jacobson, and A. Akella, "AED: incrementally synthesizing policy-compliant and manageable configurations," in *CoNEXT '20: The 16th International Conference on emerging Networking EXperiments and Technologies, Barcelona, Spain, December, 2020*, D. Han and A. Feldmann, Eds. ACM, 2020, pp. 482–495. [Online]. Available: https://doi.org/10.1145/3386367.3431304

[4] A. Mahimkar, Z. Ge, X. Liu, Y. Shaqalle, Y. Xiang, J. Yates, S. Pathak, and R. Reichel, "Aurora: conformity-based configuration recommendation to improve LTE/5G service," in *Proceedings of the 22nd ACM Internet Measurement Conference, IMC 2022, Nice, France, October 25-27, 2022*, C. Barakat, C. Pelsser, T. A. Benson, and D. R. Choffnes, Eds. ACM, 2022, pp. 83–97. [Online]. Available: https://doi.org/10.1145/3517745.3561455

[5] A. Abhashkumar, A. Gember-Jacobson, and A. Akella, "Tiramisu: Fast multilayer network verification," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

[6] P. Zhang, X. Liu, H. Yang, N. Kang, Z. Gu, and H. Li, "Apkeep: Realtime verification for real networks," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

[7] P. Zhang, A. Gember-Jacobson, Y. Zuo, Y. Huang, X. Liu, and H. Li, "Differential network analysis," in *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022*, A. Phanishayee and V. Sekar, Eds. USENIX Association, 2022, pp. 601–615. [Online]. Available: https://www.usenix.org/conference/nsdi22/presentation/zhang-peng

[8] H. Zhao, B. Yang, J. Cui, Q. Xing, J. Shen, F. Zhu, and J. Cao, "Effective fault scenario identification for communication networks via knowledge-enhanced graph neural networks," *IEEE Transactions on Mobile Computing*, 2023.

[9] "The internet2 observatory," accessed on 2023/07/13. [Online]. Available: http://www.internet2.edu/research-solutions/research-support/observatory.

[10] A. Bastos, A. Nadgeri, K. Singh, I. O. Mulang', S. Shekarpour, J. Hoffart, and M. Kaul, "RECON: relation extraction using knowledge graph context in a graph neural network," in *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, J. Leskovec, M. Grobelnik, M. Najork, J. Tang, and L. Zia, Eds. ACM / IW3C2, 2021, pp. 1673–1685. [Online]. Available: https://doi.org/10.1145/3442381.3449917

[11] N. Park, A. Kan, X. L. Dong, T. Zhao, and C. Faloutsos, "Estimating node importance in knowledge graphs using graph neural networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis, Eds. ACM, 2019, pp. 596–606. [Online]. Available: https://doi.org/10.1145/3292500.3330855

[12] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer, "Dbpedia - A large-scale, multilingual knowledge base extracted from wikipedia," *Semantic Web*, vol. 6, no. 2, pp. 167–195, 2015. [Online]. Available: https://doi.org/10.3233/SW-140134

[13] S. Geng, Z. Fu, J. Tan, Y. Ge, G. de Melo, and Y. Zhang, "Path language modeling over knowledge graphs for explainable recommendation," in *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, F. Laforest, R. Troncy, E. Simperl, D. Agarwal, A. Gionis, I. Herman, and L. Médini, Eds. ACM, 2022, pp. 946–955. [Online]. Available: https://doi.org/10.1145/3485447.3511937

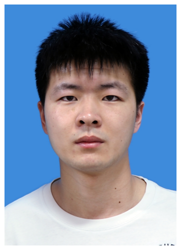
[14] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[15] S. Ramanathan, Y. Zhang, M. Gawish, Y. Mundada, Z. Wang, S. Yun, E. Lippert, W. Taha, M. Yu, and J. Mirkovic, "Practical intent-driven routing configuration synthesis," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 629–644. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/ramanathan

Zhenbei Guo received the BS degree in information and computational science from Northeastern University, Shenyang, China, in 2016 and the 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 network security.

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 an associate professor at the School of Computer Science and Engineering, Northeastern University, China. He has published more than 50 Journal/conference papers. His research interests include network management and measurement, cloud computing, and network security.

Jiaxing Shen (Member, IEEE) 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 mobile computing, data mining, and IoT systems. 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.



Tangzheng Xie received the BS degree in communication engineering from Northeastern University, Shenyang, China, in 2022. He is currently pursuing an MS degree in computer science from Northeastern University, Shenyang, China. His research interests include network management and network testing.

Shan Jiang received the B.Sc. degree in computer science and technology from Sun Yat-sen University, Guangzhou, China, in 2015 and the Ph.D. degree in computer science from The Hong Kong Polytechnic University, Hong Kong SAR, in 2021. He is currently a Research Assistant Professor with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR. Before that, he visited Imperial College London from November 2018 to March 2019. He won the best paper award from BlockSys 2021 International Conference on Blockchain and Trustworthy Systems. His research interests include distributed systems, blockchain, and edge computing.

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. He has authored or coauthored more than 100 journal articles, books and book chapters, and refereed conference papers. His research interests include cloud computing and future Internet. He was the recipient of several best paper awards.