ChatGosen: A Network Configuration Synthesis Approach with Semantic-Computation Decoupling

Chunyuan Liu[†*], Zhaokun Tan[†*], Fuliang Li[†✉], Bocheng Liang[†], Jiaxing Shen[‡], Lei Yang[†], Xingwei Wang[†]
[†]School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China
[‡]Lingnan University, Hong Kong, China
Email: liuchunyuan-neu@foxmail.com, tzk1529620423@163.com, lifuliang@cse.neu.edu.cn, 2110666@stu.neu.edu.cn, jiaxingshen@LN.edu.hk, yanglei@mail.neu.edu.cn, wangxw@mail.neu.edu.cn

***Abstract*—Network configuration synthesis aims to translate high-level configuration intent into concrete device-level configurations. Recent research has explored leveraging large language models (LLMs) for intent inference and configuration synthesis. However, configuration synthesis inherently requires deterministic topology analysis and protocol-parameter resolution, such as deriving OSPF link costs, BGP route preferences, or static-route next hops from topology constraints and intended forwarding behavior. These operations require explicit protocol semantics and cannot be reliably guaranteed by the probabilistic generation process of LLMs. We propose ChatGosen, a network configuration synthesis method based on semantic-computation decoupling. This approach separates intent inference from deterministic protocol-parameter resolution through a structured, vendor-independent intermediate representation (IR). The LLM handles intent reasoning, task decomposition, generation of device-level configuration subtasks, and final configuration file generation, while topology parsing and protocol-parameter resolution are processed by a deterministic protocol-parameter resolution module. We implemented a prototype system supporting static routing, OSPF, and BGP protocols. Testing on the Topology Zoo real-world network topology dataset demonstrated ChatGosen achieves fully correct routing forwarding configurations in single-intent scenarios across 64-router topologies, with accuracy exceeding 85% in multi-intent scenarios. The synthesis process completes in mere minutes, proving that semantic-computation decoupling provides a practical and scalable foundation for network-wide configuration synthesis.**

***Index Terms*—Network Configuration Synthesis, Large Language Models, Intermediate Representation**

I. Introduction

Network configuration synthesis is the process of translating high-level configuration intents into device-level configuration files that satisfy protocol semantics, topology constraints, and vendor-specific syntax. As network scale and heterogeneity grow, manual configuration becomes increasingly error-prone and labor-intensive [1]–[3].

Formalized approaches require explicit encoding of protocol behavior and guarantee the correctness of synthesized configurations in predefined scenarios [4]–[14]. However, their synthesis logic is often tightly coupled with specific protocols and deployment scenarios. Extending these methods to support new protocols, evolving intentions, and dynamic network environments typically requires restructuring domain-specific languages (DSLs) and modifying underlying configuration solvers, limiting their scalability and practicality in real-world networks.

The rapid advancement of LLM has reignited research interest in automated network configuration synthesis [15], [16]. Leveraging their robust natural language understanding and generalization capabilities [17]–[21], LLMs have been applied to tasks such as configuration intent parsing, configuration generation, and configuration transformation [15], [16], [22]. LLMs can parse intents, generate configuration fragments, and perform semantic reasoning, allowing operators to specify intents in natural language without learning specialized DSLs.

However, reliable configuration synthesis requires *deterministic topology analysis and protocol-parameter resolution*, such as calculating OSPF link costs, BGP preference values, route-map priorities, or static-route next hops based on network topology and intents. These operations must strictly adhere to protocol semantics and cannot be reliably performed solely by the probabilistic generation of LLMs [23]. Existing LLM-based approaches often rely on prompt engineering [24] or syntax verification, but they cannot fully guarantee network behavior matches the intended policy, especially under complex topologies or multiple concurrent intents.

In this paper, we present **ChatGosen**, a network configuration synthesis framework that separates semantic reasoning from deterministic protocol-parameter resolution. The LLM handles parsing natural language intents and decomposing them into IR subtasks. A deterministic module then resolves the protocol parameters required for the configuration. This design mitigates errors due to LLM hallucinations while ensuring that configuration complies with protocol semantics.

While ChatGosen separates semantic reasoning from deterministic protocol computation, multi-intent scenarios introduce additional challenges. In networks with multiple concurrent configuration intents, semantic interactions among intents can cause LLM-based task decomposition to omit subtasks or produce redundant configurations, potentially reducing forwarding correctness if not carefully managed.

To mitigate these issues, ChatGosen employs a structured IR that organizes subtasks and constrains LLM outputs. The deterministic computation module ensures reliable resolution of protocol parameters, alleviating the risk of hallucinations or unstable generation from the LLM. This design maintains correctness even in complex topologies and multi-intent conditions.

From a practical deployment perspective, operators bene-

[*]These authors contributed equally to this work.
979-8-3315-4940-4/25/$31.00

fit from pre-defined IR templates and modular computation units. Such measures reduce the operational burden and learning curve for network administrators, facilitating real-world adoption without requiring deep expertise in domain-specific languages (DSLs) or low-level protocol details.

Importantly, ChatGosen synthesizes configuration *parameters that guide standard routing protocols to converge to the intended forwarding paths*, rather than directly selecting routes. This ensures that network behavior is enforced through protocol-compliant parameters and respects the inherent decision processes of OSPF, BGP, and other routing protocols.

The key contributions of this paper are as follows:

- We propose **ChatGosen**, a semantic-computation decoupled framework for network configuration synthesis that leverages the strengths of LLMs for intent parsing while delegating deterministic protocol parameter resolution to a dedicated module.
- We design a structured, vendor-independent **IR** that serves as a clear interface between semantic reasoning and protocol parameter computation, enabling modularity and cross-vendor support.
- We implement a prototype of ChatGosen supporting static routing, OSPF, and BGP, and evaluate it on real-world network topologies. The results demonstrate superior forwarding correctness compared to both LLM-only and computation-only baselines.
- We clarify the distinction between semantic intent parsing and protocol parameter resolution, and emphasize that configuration synthesis is performed *through parameter instantiation that induces intended routing behavior*, ensuring protocol-compliant outcomes.

Overall, ChatGosen provides a practical, scalable, and robust foundation for automated network configuration synthesis by combining LLM-based semantic reasoning with deterministic protocol-parameter resolution.

II. Background and Motivation

A. Network Configuration Synthesis

Generating network device configurations is a fundamental task in network operations and maintenance. As networks expand in scale and increase in heterogeneity, manually translating high-level configuration intent into device-level configurations becomes increasingly difficult and error-prone. Network configuration synthesis aims to alleviate this burden by automatically generating end-to-end configurations that satisfy expected network-wide behavior, comply with protocol semantics, and adhere to vendor-specific syntax.

Previous research [4]–[12] have made significant progress in network configuration synthesis using formal methods. These systems typically rely on DSLs and configuration templates. Configuration parameters, computed through constraint solving or formal reasoning, are then instantiated into predefined templates to generate device-level configurations. For example, NetComplete uses SMT solvers to compute OSPF and BGP parameters that satisfy user-specified routing intentions, automatically populating them into pre-designed configuration templates to achieve full network OSPF and BGP configuration synthesis.

Although these methods provide correctness guarantees in predefined scenarios, they also face practical limitations. First, network operations personnel must learn and use DSLs to express intent and design configuration templates for specific deployment scenarios, resulting in a steep learning curve and requiring significant manual effort. Second, DSL design and templates are often tightly coupled with underlying system modeling assumptions. Extending these systems typically requires DSL refactoring or solver modifications. These limitations constrain their applicability in real-world networks, particularly when network operations personnel must handle diverse configuration scenarios. This may necessitate using multiple synthesis tools, and in most cases, these tools cannot be deployed concurrently within the same network.

B. Why LLMs Are Appealing but Insufficient

LLMs have demonstrated formidable capabilities across domains including natural language understanding, intent inference, technical document parsing, and multi-step generation [15], [16], [22], [25]–[29]. These capabilities align closely with configuration synthesis requirements, particularly excelling in parsing natural language configuration intent and generalizing from historical configuration examples. Additionally, employing LLMs for network management enables operations personnel to specify configuration intent using natural language rather than DSLs, substantially reducing operational complexity for maintenance staff.

However, LLMs alone are insufficient to support reliable network-wide configuration synthesis. While they excel at semantic interpretation and structural generalization, they lack explicit enforcement of network topology constraints and protocol semantics. Configuration synthesis requires enforcing these constraints to ensure network behavior aligns with expectations. Relying solely on LLMs for these inferences may lead to network anomalies, security vulnerabilities in configurations, and other issues, particularly in large-scale networks or complex multi-intent scenarios.

Some approaches attempt to combine LLMs with formal solvers to mitigate these limitations. For example, CEGS integrates LLM-based intent translation with existing formal synthesis tools, converting natural language intents into DSL descriptions and configuration templates required by formal tools. Subsequently, the formal synthesis tools perform deterministic protocol-parameter resolution and instantiate configurations. While this approach improves usability compared to manually authoring DSLs, it remains tightly coupled with DSL design and the modeling assumptions of synthesis tools. Supporting new protocols or intent patterns still requires extending existing synthesis tools. Within this framework, LLMs primarily replace manual translation of configuration intent into DSL, rather than achieving a clear separation between semantic reasoning and deterministic protocol-parameter resolution.

C. Motivation

These observations indicate that formal methods and LLMs exhibit complementary strengths at different stages of the synthesis process. Formal methods excel at deterministic topological analysis and protocol-parameter resolution, while LLMs excel at intent reasoning and task structuring. However, existing systems either tightly couple synthesis logic with specific formal abstractions or rely on LLMs to implicitly perform protocol reasoning. This prevents clear separation between stages, thereby limiting the synergistic benefits achievable through combining both approaches.

This motivated our design of a configuration synthesis architecture that explicitly decouples semantic reasoning from protocol-parameter resolution. LLMs parse natural language configuration intent and decompose it into a series of structured configuration subtasks. All topology parsing and protocol-parameter resolution tasks are handled by a dedicated deterministic protocol-parameter resolution module, rather than implicitly relying on LLM inference. By enforcing this separation, the synthesis process preserves the LLM's usability advantages across diverse configuration scenarios while ensuring correctness of protocol-parameter resolutions.

These observations highlight key considerations for designing an effective network configuration synthesis system. First, while LLMs excel at interpreting high-level intents and reasoning about task structure, they do not inherently enforce protocol semantics or topology constraints. Solely relying on LLMs risks inconsistent or unsafe network behavior, particularly in large-scale or multi-intent deployments.

Second, formal methods guarantee deterministic computation of protocol parameters but are tightly coupled to specific DSLs and templates, limiting adaptability to evolving intents or heterogeneous network environments. Extending such systems often requires significant re-engineering, which is impractical in dynamic real-world networks.

Combining these insights, the design of ChatGosen is motivated by the need to leverage LLM reasoning for semantic understanding while ensuring correctness through explicit deterministic computation. Importantly, this approach allows for modular and vendor-independent IRs, facilitating scalability, extensibility, and parallel handling of multiple intents. By clearly articulating these design principles in the motivation section, we provide a strong theoretical justification for the architectural choices and experimental evaluation presented in subsequent sections.

III. Semantic-Computation Decoupled Architecture

A. Architectural Overview

Figure 1 illustrates the overall architecture of ChatGosen. This system synthesizes network-wide configurations by processing high-level configuration intent expressed through network topology and natural language. Unlike end-to-end generative approaches that directly map intent to configuration text, ChatGosen explicitly separates semantic interpretation from protocol-level computation. This architectural decision ensures that language understanding and deterministic reasoning are executed in distinct and well-bounded stages, preventing LLM hallucinations from affecting protocol parameter computation.

ChatGosen comprises three core modules: the task decomposition module, the deterministic protocol-parameter resolution module, and the configuration generation module. These modules are interconnected via a structured, vendor-independent IR, which serves as the boundary between semantic interpretation and deterministic computation. The IR encodes device-level configuration relationships and preserves sufficient structural information to enable subsequent protocol-level reasoning without requiring reinterpretation of natural language input.

The task decomposition module parses configuration intent and converts it into a series of configuration subtasks represented using IR. Each subtask contains only one unknown or known parameter, enforcing minimal semantic granularity. This design reduces ambiguity in later deterministic resolution and prevents implicit coupling between multiple configuration decisions.

ChatGosen employs Retrieval-Augmented Generation (RAG) [30] to retrieve similar configuration intent examples and IR decomposition examples from a knowledge base. The retrieved examples assist the LLM in decomposing the current configuration intent into subtasks, enhancing coverage and structural consistency while reducing illusory or semantically inconsistent decompositions. Subtasks are divided into two categories:

- **Computation Subtasks**, requiring topology analysis and protocol parameter calculation;
- **Non-computation Subtasks**, which can be directly converted into configuration commands without additional computation.

This classification determines whether further deterministic reasoning is required. Computation is confined to computation subtasks, preventing unnecessary protocol-parameter resolution for other tasks.

The deterministic protocol-parameter resolution module handles all computation subtasks. These subtasks require explicit topological analysis and protocol-level numerical computations, such as metric derivation, preference ordering, or constraint enforcement. The module derives parameters for each task and populates the corresponding IR instance. Importantly, it operates purely on structured inputs and does not reinterpret natural language intent, ensuring deterministic derivation.

After computation subtasks are resolved, the configuration generation module instantiates all subtasks—both computation and non-computation—and converts them into vendor-specific configuration fragments. Since all required parameters are determined, this stage focuses on structured instantiation rather than additional inference.

The configuration generation module also uses RAG to ensure fragments adhere to vendor hierarchy and command syntax. Retrieved knowledge provides syntactic constraints, and

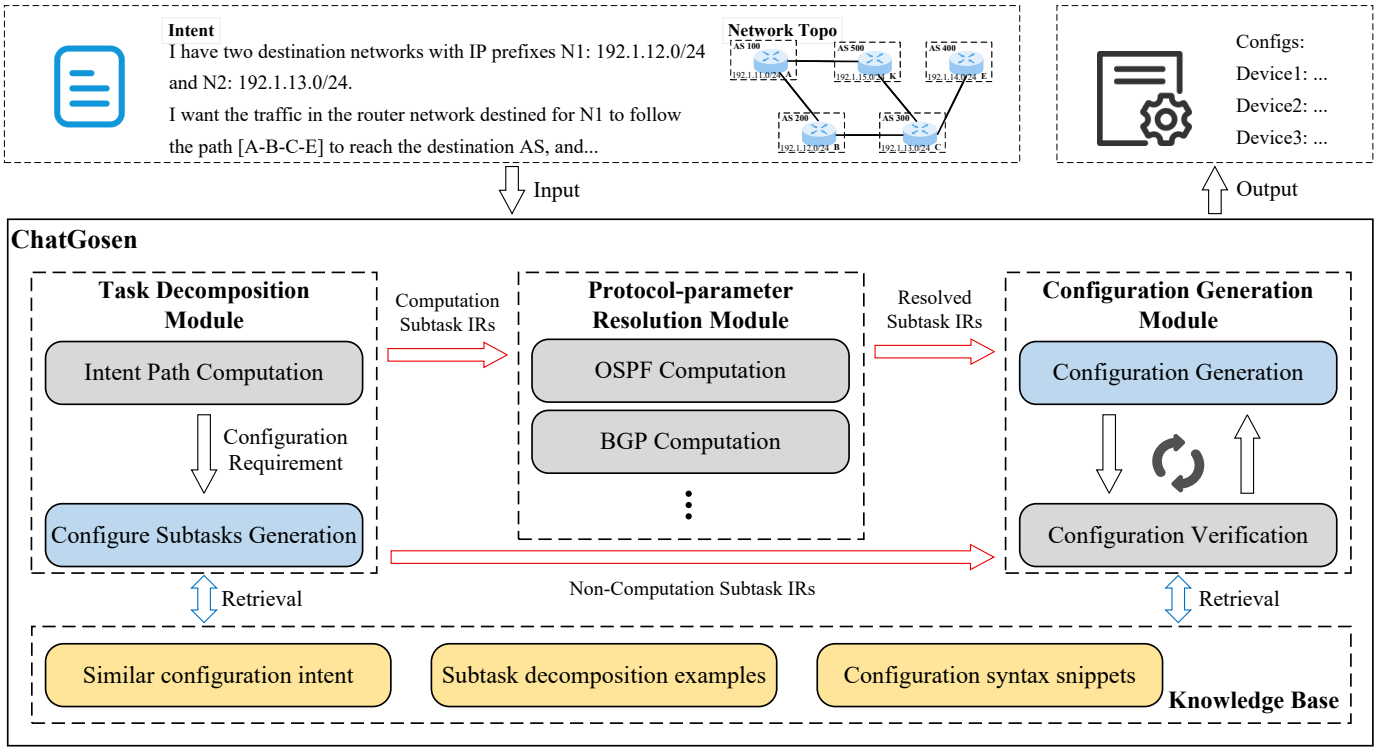


Fig. 1: ChatGosen Workflow.

configuration validation ensures generated fragments remain syntactically valid. Since parameters are already resolved, this stage does not perform further protocol-parameter inference.

The output is configuration files for all devices, reflecting deterministic resolution of previously decomposed subtasks.

This decoupled design enhances modularity and scalability. New protocols and vendors can be supported by extending the deterministic module and knowledge base without modifying the LLM-based task decomposition. Similarly, LLM performance improvements can be integrated without altering the deterministic module. The IR boundary ensures enhancements in one module do not introduce semantic inconsistencies in another, preserving stability while supporting incremental evolution.

This architecture inherently supports robustness in multi-intent scenarios. By enforcing minimal semantic granularity through IR-encoded subtasks, semantic interactions among concurrent intents are contained within distinct subtasks, reducing the likelihood of task decomposition errors.

Moreover, separating computation from semantic reasoning offers predictable computational complexity for protocol-parameter resolution, as deterministic module operations scale linearly with the number of computation subtasks. This allows network operators to anticipate resource requirements and ensures consistent performance under increasing network size or intent concurrency.

Finally, the modular separation and well-defined IR boundaries facilitate system evolution. Enhancements to task decomposition heuristics, LLM capabilities, or knowledge base content can be integrated independently, while deterministic resolution guarantees correctness.

IV. Task Decomposition Module and Intermediate Representation

A. Task Decomposition Framework

Based on natural language configuration intent and network topology, the task decomposition module transforms high-level configuration intent into structured IR instances—a series of device-level configuration subtasks. These subtasks describe device configuration operations. Instead of directly generating configuration commands, the module first converts the intent into an intermediate structured form, ensuring that subsequent processing operates on explicitly defined configuration operations rather than free-form natural language descriptions.

Before executing configuration subtask generation, the configuration intent undergoes preprocessing. The LLM identifies path-related content within the intent, invokes the intent path calculation component, and substitutes the specific path into the intent description. This ensures that any path-dependent constraints embedded in the original intent are concretely resolved before decomposition. As a result, the configuration subtask generation process produces a complete and coherent sequence of subtasks, since ChatGosen must generate configuration subtasks for each device involved in every intent. By explicitly materializing path information in advance, the decomposition stage avoids generating incomplete or structurally inconsistent subtasks.

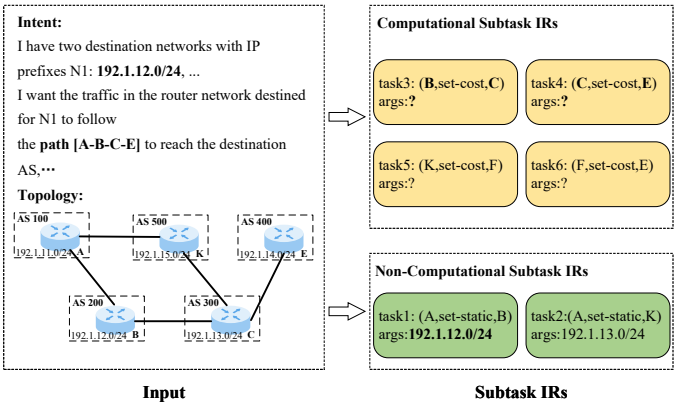


Fig. 2: Example of Configuring Subtasks

Configuration subtask generation is performed by an LLM guided by a knowledge base containing configuration intent examples and subtask decomposition examples. Semantic similarity retrieval locates analogous examples, guiding the LLM to generate IR instances conforming to predefined structures. Aligning the current intent with decomposition methods in the knowledge base ensures structural consistency with the device-centric IR format, enhancing stability and reducing errors in subtask decomposition.

B. Device-Centric IR Design

The IR in ChatGosen adopts a device-centric relational structure. Each configured subtask is represented as a tuple:

$$(d_i,\ op,\ d_j,\ arg)$$

Here, d_i denotes the device requiring the current configuration operation, op specifies the type of configuration operation, d_j represents the associated device, and arg indicates the parameters corresponding to this operation. This four-element structure ensures each subtask explicitly captures the device performing the operation, the operation type, the associated context, and the relevant parameter values.

Figure 2 illustrates an example of converting an intent into configuration subtasks, including computation and non-computation subtasks:

- $(B,\ \textit{set-cost},\ C,\ ?)$ is a computation subtask that calculates the routing cost from device B to device C. Its specific parameter $?$ must be determined in the subsequent deterministic protocol-parameter resolution module.
- $(A,\ \textit{set-static},\ B,\ 192.1.12.0/24)$ is a non-computation subtask. It represents configuring a static route on device B pointing to device E. The parameter is explicitly specified and does not require further deterministic computation.

In summary, IR instances may contain fully specified parameters or unresolved placeholders, depending on whether deterministic computation is required.

C. IR Design Principles

Explicit parameterization. Network configuration subtasks often involve numerical or attribute-specific computations (e.g., link costs, metrics, policy priorities, access control parameters). Embedding arg into the IR ensures that all configuration-related numerical decisions are preserved explicitly throughout the synthesis process, preventing implicit inference during configuration generation and maintaining determinism.

Atomic Operation. The tuple design satisfies device-level configuration operation requirements with minimal complexity. Each tuple corresponds to a single configuration command, avoiding over- or under-expression.

Vendor-independent. Unlike DSLs in formal synthesis tools, the IR is a vendor-independent operational representation, capturing only operations and parameters. Conversion to vendor-specific commands is deferred to the generation stage.

Composability. Configuration intents can be decomposed into independent or partially ordered sets of IR instances, supporting complex intents. Dependencies among parameters are resolved during deterministic protocol-parameter resolution, not configuration generation.

Overall, the IR design provides a structured and operationally complete representation that bridges natural language intent with device-level configuration, preserving parameter determinism and enabling vendor independence.

D. Computation Role of IR Instances

Non-computation IR instances. These tuples can be directly instantiated as configuration commands without additional topology parsing or protocol-parameter resolution. For example, $(A,\ \textit{set-static},\ B,\ 192.1.12.0/24)$ can proceed directly to configuration generation.

Computation IR instances. These tuples require topology or protocol value computation before configuration instantiation. For example, $(B,\ \textit{set-cost},\ C,\ ?)$ requires the deterministic module to compute appropriate parameters based on topology, path constraints, or protocol semantics. Placeholders indicate that deterministic resolution is required prior to generating final configuration.

V. Deterministic Protocol-Parameter Resolution

A. Architectural Role and Computation Scope

Network configuration synthesis requires deterministic protocol-parameter resolution. Many protocol configuration decisions depend on explicit topology parsing and precise numerical derivation. Unlike semantic interpretation based on natural language, protocol parameters must be derived from clearly defined constraints and numerical rules. Deterministic computation ensures that specific configuration items—such as routing decisions, metric assignment, and policy enforcement—align with both protocol semantics and configuration intent.

The deterministic protocol-parameter resolution module models involved protocols, solves protocol parameters using formal methods, and operates on predefined interfaces of

configuration subtasks. Rather than generating configuration commands directly, it operates on structured IR instances and derives concrete parameter values. All computations are based on explicit topology information and protocol-defined constraints.

The module treats each IR tuple as a placeholder for specific protocol parameters. It explicitly reads topology data such as node connectivity, link metrics, and interface attributes, and applies formal protocol rules to compute valid parameter values. The computation follows a deterministic procedure to ensure that each IR instance corresponds to a feasible and consistent network configuration. Typical computation tasks include:

- **Topology-based path selection**, where candidate paths between devices must be itemized and evaluated according to protocol rules.
- **Metric and cost derivation**, as required by intra-domain routing protocols such as OSPF. Numerical parameters must satisfy ordering or optimization constraints defined by the protocol.
- **Preference and policy resolution**, particularly in inter-domain routing protocols such as BGP. Attributes and preference relationships among alternative paths are evaluated deterministically.

All computation subtasks share a common characteristic: their parameter values cannot be directly inferred from natural language intent alone but must be derived through deterministic analysis of topology and protocol semantics.

The module ensures that for each IR instance, all dependent parameters are resolved in accordance with protocol constraints. This includes calculating cumulative link metrics, selecting feasible paths, and determining route preferences. The procedure guarantees consistency across all IR tuples representing the same intent.

B. Case Study: OSPF Preference Enforcement

We illustrate deterministic protocol-parameter resolution using an OSPF routing priority example. Unresolved parameters in IR instances are concretely determined through formal constraint satisfaction.

Suppose an intent requires that traffic between a source node and destination node prioritize path P_1 over paths P_2 and P_3. After task decomposition, the LLM produces IR instances representing device relationships. IR tuples indicate which devices require metric adjustments, but the exact values remain unspecified, highlighting the need for deterministic computation.

Let the network topology be a weighted graph $G = (V, E)$ with link weights $w(e)$. Candidate paths are P_1, P_2, P_3. The deterministic module itemizes feasible paths and evaluates cumulative weights under OSPF semantics.

The desired ordering constraint is:

$$\sum_{e \in P_1} w(e) < \sum_{e \in P_2} w(e) < \sum_{e \in P_3} w(e). \tag{1}$$

This formalizes path preference. The deterministic module searches for feasible weight assignments within valid OSPF domains (e.g., non-negative integers), ensuring compliance with ordering and protocol constraints.

To derive a consistent and minimal solution:

$$\min \left\{ \sum_{e \in P_1} w(e) \right\} \tag{2}$$

$$\sum_{e \in P_1} w(e) < \sum_{e \in P_2} w(e) < \sum_{e \in P_3} w(e) \tag{3}$$

The optimization ensures P_1 receives the minimal feasible cumulative cost while preserving strict ordering. All solutions respect OSPF non-negativity and integer constraints.

The output is a consistent set of link weights $\{w(e)\}$ attached to corresponding IR instances. For example: $(B,$ *set-cost*, $C,$ $1)$, $(C,$ *set-cost*, $E,$ $1)$, $(B,$ *set-cost*, $D,$ $2)$, $(D,$ *set-cost*, $E,$ $2)$.

After parameter instantiation, IR instances become fully determined configuration subtasks, ready for configuration generation without further numerical inference. This example demonstrates how deterministic protocol-parameter resolution connects unsolved IR placeholders to concrete configuration parameters while strictly adhering to protocol rules. The resolved IR tuples directly drive configuration generation, ensuring that the generated configuration faithfully implements the high-level intent specified by the user.

VI. Configuration Generation and Verification

A. IR-Grounded Configuration Generation

ChatGosen designs configuration generation as a text-generation process grounded on resolved IR instances. Each configuration subtask is translated into vendor-specific configuration fragments. For example, a resolved IR instance $(B,$ *set-cost*, $C, 1)$ carries the OSPF cost value computed by the deterministic protocol-parameter resolution module. The generation module maps this IR instance to vendor-specific commands for device B. The generated output strictly adheres to resolved IR instances. This phase does not introduce new protocol parameters, alter computed values, or modify deterministic decisions.

To support multiple vendors, ChatGosen maintains a knowledge base with configuration command examples for each vendor. These examples guide the LLM in generating configuration fragments by providing similar configuration subtask examples, instantiated commands, format specifications, and hierarchical structures.

For similarity retrieval of configuration subtasks, a retrieval strategy based on configuration operation matching is employed. If the knowledge base contains a matching example for the vendor and operation, the corresponding instantiated commands and specifications are used as prompts for the LLM. If no matching example exists, the LLM extracts information from the official vendor configuration manual to determine commands, format, and hierarchy.

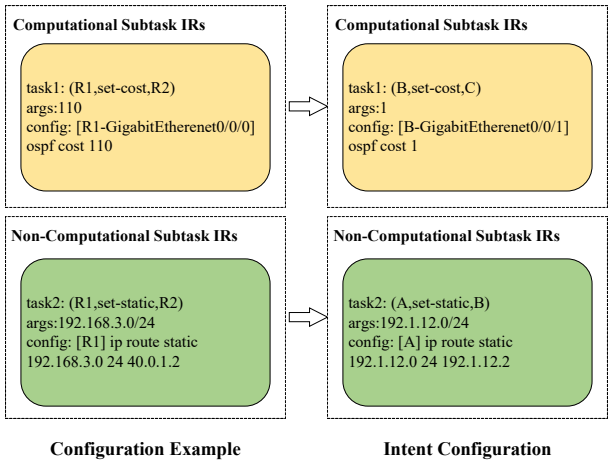


Fig. 3: IR-grounded Configuration Generation

Finally, as shown in Figure 3, the LLM maps the current subtask to a target configuration fragment and updates the knowledge base with any new subtask examples.

B. Configuration-Level Verification

A vendor-specific configuration syntax validator is built based on the syntax hierarchy tree of Nassim [31]. Starting from the root of the tree, the validator matches commands layer by layer. If a command matches, traversal proceeds to the next level. If no match exists at the current level, backtracking occurs. Commands without matching items are flagged.

If full validation passes, the final network-wide configuration file is output. If validation fails, the exception is fed back to the LLM for regeneration. The LLM iteratively regenerates the configuration until validation succeeds or the maximum iteration limit is reached.

VII. Evaluation

A. Experimental Setup

We implemented the ChatGosen system for Cisco device configuration orchestration using approximately 1,000 lines of Python code. The system was deployed on a server equipped with an AMD EPYC 7352 CPU, NVIDIA A10 GPU, and 128 GB of memory. GPT-4o is used for task decomposition, configuration generation, and knowledge base retrieval.

Topologies. We extracted ten real network topologies from the Topology Zoo dataset [32], ranging from 16 to 64 routers. These topologies include metropolitan, regional, national, and intercontinental networks. Each topology contains multiple autonomous systems (ASes) and exhibits both intra-domain and inter-domain routing relationships.

Intents. We evaluate three routing scenarios: static routing, OSPF, and BGP. Based on real-world configuration cases, we constructed ten representative intent types, including OSPF ECMP, arbitrary path routing, BGP transit blocking, and cross-AS policy control. Each intent is expressed in natural language and may require coordinated multi-device configuration.

Baselines. To evaluate the necessity of semantic–computation decoupling, we compare against:

- **LLM-Only:** Directly generates end-to-end configurations from natural language without task decomposition or deterministic protocol-parameter resolution, while retaining syntax validation.
- **Protocol-Computation-Only:** The LLM selects the computation functions to be invoked from the deterministic protocol-parameter resolution module and generates configurations based on manual knowledge, eliminating the need for explicit structured task decomposition.
- **ChatGosen:** Full architecture including task decomposition, deterministic protocol-parameter resolution, and IR-grounded configuration generation.

B. Overall Forwarding Correctness

To evaluate whether synthesized configurations preserve intended routing behavior, we deploy the generated configurations in a network simulator and observe forwarding paths.

Let $F(\text{target})$ denote the forwarding path contained in the expected routing policy, and $F(\text{actual})$ denote the observed forwarding path. Forwarding accuracy is defined as:

$$\frac{|F(\text{actual}) \cap F(\text{target})|}{|F(\text{target})|}$$

Table I reports results under representative scenarios.

Topology	Intents	Method	Forwarding Accuracy
16	1	LLM-Only	23%
16	1	Protocol-Computation-Only	87%
16	1	ChatGosen	**100%**
32	2	LLM-Only	14%
32	2	Protocol-Computation-Only	72%
32	2	ChatGosen	**95%**
64	3	LLM-Only	11%
64	3	Protocol-Computation-Only	46%
64	3	ChatGosen	**89%**

TABLE I: Forwarding behavior comparison under different methods and multi-intent scenarios.

In small-scale scenarios (16 routers, one intent), ChatGosen achieves 100% forwarding accuracy, significantly outperforming LLM-Only (23%) and Protocol-Computation-Only (87%). In medium-scale scenarios (32 routers, two intents), accuracy decreases for all methods due to multi-intent complexity, with Protocol-Computation-Only at 72% and ChatGosen at 95%. In large-scale scenarios (64 routers, three intents), LLM-Only drops to 11%, Protocol-Computation-Only to 46%, while ChatGosen maintains 89%, demonstrating robustness against semantic and structural complexity.

C. Component-Level Analysis: IR Decomposition Accuracy

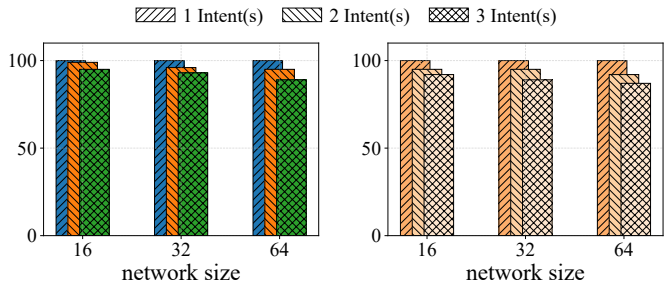
Let $C(\text{target})$ denote the required configuration subtask set. Generated subtasks include correctly matched subtasks $O(\text{coverage})$ and redundant or erroneous subtasks $O(\text{redundancy})$.

Coverage rate:

$$\frac{O_{\text{coverage}}}{C_{\text{target}}}$$

Precision rate:

$$\frac{O_{\text{coverage}}}{O_{\text{coverage}} + O_{\text{redundancy}}}$$



(a) Coverage (b) Precision

Fig. 4: IR decomposition coverage and precision under varying network sizes and intent counts.

Single-intent scenarios achieve 100% coverage and precision across all topologies. As intent count increases, both metrics degrade, showing that semantic interactions among multiple intents are the main source of decomposition errors.

D. Efficiency Analysis: Synthesis Time

Table II reports end-to-end synthesis time under varying network sizes and intent counts.

Network size	1 Intent	2 Intents	3 Intents
16	1m12s	2m35s	3m55s
32	1m37s	3m30s	4m55s
64	2m02s	4m50s	7m47s

TABLE II: End-to-end synthesis time for different network sizes and intent counts.

LLM inference dominates runtime in semantic decomposition and configuration generation. Even for 64-router topologies, synthesis completes within several minutes.

E. Failure Analysis

Errors primarily arise in complex multi-intent scenarios:

- **Missing Subtask.** Interactions among intents may cause IR subtasks to be omitted, leading to deviations from expected network behavior.
- **Redundant Subtask.** Redundant subtasks can be generated, causing configuration redundancy or incorrect network behavior.
- **Parameter Misbinding.** Incorrect associations among devices, interfaces, or prefixes lead to inconsistencies.

Most failures originate from IR construction rather than deterministic protocol-parameter resolution or configuration generation, highlighting that semantic decomposition remains the primary source of errors.

VIII. Discussion

A. Current Limitations

The task decomposition module relies on LLM to convert natural language configuration intents into a series of device-level configuration subtasks. In scenarios involving multiple concurrent intents, semantic interactions between intents can induce reasoning errors in the LLM, leading to missing subtasks, redundant subtasks, or misbound parameters. As shown in Figure 4, coverage and precision degrade as intent count increases, highlighting this primary source of errors in complex network scenarios.

The configuration generation module maps subtasks to vendor-specific configuration fragments using IR-grounded examples stored in the knowledge base. While examples verified in the knowledge base produce correct outputs, handling previously unseen subtasks relies on LLM-guided semantic parsing from vendor manuals. Consequently, generated configurations for novel subtasks remain partially unverified, and the LLM may select incorrect commands. Syntax validation (Section VI) detects structural errors but cannot identify semantic misconfigurations arising from incorrect command selection.

ChatGosen synthesizes configurations using a full-intent approach. In operational networks, configuration updates are typically incremental and continuous. Supporting incremental intent synthesis while maintaining deterministic protocol-parameter consistency introduces additional complexity: newly added intents may conflict with existing configurations, and previously resolved subtasks may require reevaluation. Addressing this challenge requires a conflict detection and merge mechanism to ensure that merged configuration subtasks satisfy all intents while preserving protocol correctness.

B. Extensibility to New Protocols

Extending ChatGosen to support additional network protocols involves augmentation at both semantic inference and deterministic protocol-parameter resolution levels:

- Expand the knowledge base with new intent examples, IR decomposition examples, and vendor-specific configuration generation examples corresponding to the new protocol.
- Extend the set of available configuration operations to include operations and parameters introduced by the new protocol.
- Implement a protocol-parameter resolution module for the new protocol, enabling topology-aware computation and protocol constraint enforcement to derive valid configuration parameters for all added operations.

While the semantic-computation decoupling of ChatGosen reduces structural expansion complexity, supporting new protocols still requires domain expertise to encode operational semantics and parameter constraints within the deterministic computation module. This ensures that newly added protocol operations maintain correctness under explicit topology and protocol rules.

C. Future Directions

Building on current limitations, we identify three directions for future improvement:

- **Improved multi-intent decomposition:** Incorporating structured prompting, retrieval-augmented LLMs, or hybrid reasoning with formal constraints may reduce missing and redundant subtasks in multi-intent scenarios.
- **Incremental configuration synthesis:** Designing a merge-and-reconcile mechanism for incremental intents can enable continuous updates without violating previously established protocol constraints.
- **Semantic and behavioral verification:** Extending configuration validation beyond syntax to include semantic correctness (e.g., forwarding path verification, loop detection, policy compliance) can further mitigate the risk of undetected misconfigurations.

IX. Related Work

A. Formal Network Configuration Synthesis and Verification

Formal configuration synthesis research investigates how to automatically derive device-level configurations from high-level configuration intents expressed in domain-specific languages (DSLs). Prior work [4]–[12] demonstrates that configuration intents can be compiled into correct configurations using constraint solving, symbolic reasoning, and SMT.

Systems such as SyNET [4], Propane [5], [6], NetComplete [7], and related research [8]–[12] typically assume configuration intents are already formalized. Translating natural language intents into DSL expressions suitable for these tools remains a challenging task that requires domain expertise.

Configuration Verification is also a crucial component of network configuration management. Unlike the syntax validator designed in this paper, existing configuration Verification tools are typically used to determine whether network behavior meets expectations or contains defects. Configuration verification tools [33]–[36] employ formal methods to examine whether configurations satisfy expected behavior. For example, Minesweeper [34] utilize SMT solvers for verification.

B. LLM-Based Network Configuration

Recent work has explored using LLM for configuration generation and intent interpretation. NETBUDDY [37] uses a phased generation strategy to improve controllability. PreConfig [38] improves fragment-level accuracy by fine-tuning the model on configuration segments. CoTNet [16] studies structured prompting techniques to facilitate intermediate reasoning during the synthesis process.

These methods primarily embed protocol reasoning, policy analysis, and configuration synthesis directly into the LLM, enabling end-to-end generation of configuration fragments. Optimization typically relies on prompt engineering, fine-tuning, or stepwise chain-of-thought reasoning.

C. Hybrid LLM–Formal Systems

Some approaches combine the semantic reasoning capabilities of LLMs with formal verification or structured solvers. COSYNTH [39] integrates LLM-based synthesis with iterative formal verification, leveraging feedback to refine outputs. CEGS [15] uses LLMs to translate natural language configuration intents into DSL-based representations and generates corresponding configuration templates. Formal solvers like NetComplete then process the DSL, performing final configuration synthesis.

ChatGosen also falls under a hybrid paradigm, but differs from prior work in that it enforces a strict separation between semantic reasoning and deterministic protocol-parameter computation before configuration implementation. This design ensures that all numerical and protocol-specific computations are handled deterministically, while the LLM focuses solely on high-level semantic decomposition, reducing the risk of misconfigurations caused by probabilistic reasoning.

X. Conclusion

We propose ChatGosen, a network configuration synthesis method that explicitly decouples semantic interpretation from deterministic protocol-parameter resolution. Its core contribution lies in introducing a vendor-independent intermediate representation to express configuration subtasks during synthesis, serving as a structured interface between natural language configuration intent and protocol-parameter resolution. ChatGosen confines LLMs to constructing this intermediate representation while ensuring network behavior aligns with expectations through a deterministic protocol-parameter resolution module. Experimental results demonstrate that this method achieves high-precision IR decomposition and network behavioral correctness in multi-device topologies.

Acknowledgements

This work is supported by the National Natural Science Foundation of China under Grant Nos. 62572105 and U22B2005, as well as the LiaoNing Revitalization Talents Program under Grant No. XLYC2403086.

References

[1] R. Chirgwin, "Google routing blunder sent japan's internet dark on friday," *The Register*, 2017.

[2] G. Corfield, "British airways' latest total inability to support upwardness of planes caused by amadeus system outage," 2018, july 2018.

[3] Meta, "Update about the 4 october outage," October 2021.

[4] A. El-Hassany, P. Tsankov, L. Vanbever, and M. T. Vechev, "Network-wide configuration synthesis," in *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, ser. Lecture Notes in Computer Science, vol. 10427, 2017, pp. 261–281.

[5] 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: brief reflections on abstractions for network programming," *Comput. Commun. Rev.*, vol. 49, no. 5, pp. 104–106, 2019.

[6] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Network configuration synthesis with abstract topologies," *SIGPLAN Not.*, vol. 52, no. 6, p. 437–451, Jun. 2017. [Online]. Available: https://doi.org/10.1145/3140587.3062367

[7] 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*, 2018, pp. 579–594.

[8] B. Tian, X. Zhang, E. Zhai *et al.*, "Safely and automatically updating in-network ACL configurations with intent language," in *Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM 2019, Beijing, China, August 19-23, 2019*. ACM, 2019, pp. 214–226.

[9] K. Subramanian, L. D'Antoni, and A. Akella, "Synthesis of fault-tolerant distributed router configurations," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 2, no. 1, pp. 22:1–22:26, 2018.

[10] S. Ramanathan, Y. Zhang *et al.*, "Practical intent-driven routing configuration synthesis," in *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*, M. Balakrishnan and M. Ghobadi, Eds., 2023, pp. 629–644.

[11] A. Gember-Jacobson, A. Akella, R. Mahajan, and H. H. Liu, "Automatically repairing network control planes using an abstract representation," in *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*. ACM, 2017, pp. 359–373.

[12] 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*. ACM, 2020, pp. 482–495.

[13] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Don't mind the gap: Bridging network-wide objectives and device-level configurations: brief reflections on abstractions for network programming," *SIGCOMM Comput. Commun. Rev.*, vol. 49, no. 5, p. 104–106, Nov. 2019. [Online]. Available: https://doi.org/10.1145/3371934.3371965

[14] Y. E. Sung, X. Tie, S. H. Y. Wong, and H. Zeng, "Robotron: Top-down network management at facebook scale," in *Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016*. ACM, 2016, pp. 426–439.

[15] J. Liu, L. Chen, D. Li, and Y. Miao, "{CEGS}: Configuration example generalizing synthesizer," in *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, 2025, pp. 1327–1347.

[16] J. Wei, X. Gao, X. Cao, H. Deng, Y. Qian, and P. Zhang, "Let's synthesize step-by-step: Generating network configurations with chain of thought," in *2025 IEEE 33rd International Conference on Network Protocols (ICNP)*. IEEE, 2025, pp. 1–6.

[17] J. Lee, N. Stevens, S. C. Han, and M. Song, "A survey of large language models in finance (finllms)," *CoRR*, vol. abs/2402.02315, 2024.

[18] J. K. Matelsky, F. Parodi, T. Liu, R. D. Lange, and K. P. Kording, "A large language model-assisted education tool to provide feedback on open-ended responses," *CoRR*, vol. abs/2308.02439, 2023.

[19] GitHub, "Github copilot: Your ai pair programmer," 2023.

[20] N. Jain, S. Vaidyanath, A. Iyer, N. Natarajan, S. Parthasarathy, S. Rajamani, and R. Sharma, "Jigsaw: Large language models meet program synthesis," in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, 2022, pp. 1219–1231.

[21] D. M. Manias, A. Chouman, and A. Shami, "Towards intent-based network management: Large language models for intent extraction in 5g core networks," in *2024 20th International Conference on the Design of Reliable Communication Networks (DRCN)*, 2024, pp. 1–6.

[22] Y. Wei, X. Xie, T. Hu, Y. Zuo, X. Chen, K. Chi, and Y. Cui, "Inta: Intent-based translation for network configuration with llm agents," in *2025 IEEE 33rd International Conference on Network Protocols (ICNP)*. IEEE, 2025, pp. 1–16.

[23] Z. Li, Y. Cao, X. Xu, J. Jiang, X. Liu, Y. S. Teo, S.-W. Lin, and Y. Liu, "Llms for relational reasoning: How far are we?" in *2024 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code)*, 2024, pp. 119–126.

[24] D. Cheng, S. Huang, J. Bi, Y. Zhan, J. Liu, Y. Wang, H. Sun, F. Wei, W. Deng, and Q. Zhang, "UPRISE: Universal prompt retrieval for improving zero-shot evaluation," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, H. Bouamor, J. Pino, and K. Bali, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 12 318–12 337. [Online]. Available: https://aclanthology.org/2023.emnlp-main.758/

[25] C. Wang, M. Scazzariello, A. Farshin, S. Ferlin, D. Kostić, and M. Chiesa, "Netconfeval: Can llms facilitate network configuration?" *Proceedings of the ACM on Networking*, vol. 2, pp. 1 – 25, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:270321937

[26] N. Zheng, F. Li, Z. Li, Y. Yang, Y. Hao, C. Liu, and X. Wang, "Configtrans: Network configuration translation based on large language models and constraint solving," *2024 IEEE 32nd International Conference on Network Protocols (ICNP)*, pp. 1–12, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:276117695

[27] R. Mondal, A. Tang, R. Beckett, T. D. Millstein, and G. Varghese, "What do llms need to synthesize correct router configurations?" *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:259766303

[28] A. Angi, A. Sacco, and G. Marchetto, "Llnet: An intent-driven approach to instructing softwarized network devices using a small language model," *IEEE Transactions on Network and Service Management*, vol. 22, pp. 3403–3418, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusID:278643114

[29] W. Ding, J. Li, Z. Niu, H. Chen, and H. Xu, "Poster: Automating network configuration with natural language intents," *Proceedings of the ACM SIGCOMM 2024 Conference: Posters and Demos*, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:271735272

[30] P. Lewis, E. Perez, A. Piktus, F. Petroni *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020.

[31] 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*, ser. SIGCOMM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 281–297. [Online]. Available: https://doi.org/10.1145/3544216.3544244

[32] S. Knight, H. X. Nguyen, N. Falkner, R. A. Bowden, and M. Roughan, "The internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, 2011.

[33] A. Abhashkumar, A. Gember-Jacobson, and A. Akella, "Tiramisu: fast multilayer network verification," in *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation*, ser. NSDI'20. USA: USENIX Association, 2020, p. 201–220.

[34] 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*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 155–168. [Online]. Available: https://doi.org/10.1145/3098822.3098834

[35] A. Tang, R. Beckett, S. Benaloh, K. Jayaraman, T. Patil, T. Millstein, and G. Varghese, "Lightyear: Using modularity to scale bgp control plane verification," in *Proceedings of the ACM SIGCOMM 2023 Conference*, ser. ACM SIGCOMM '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 94–107. [Online]. Available: https://doi.org/10.1145/3603269.3604842

[36] A. Wang, L. Jia, W. Zhou, Y. Ren, B. T. Loo, J. Rexford, V. Nigam, A. Scedrov, and C. Talcott, "Fsr: Formal analysis and implementation toolkit for safe interdomain routing," *IEEE/ACM Transactions on Networking*, vol. 20, no. 6, pp. 1814–1827, 2012.

[37] C. Wang, M. Scazzariello, A. Farshin, D. Kostic, and M. Chiesa, "Making network configuration human friendly," *CoRR*, vol. abs/2309.06342, 2023.

[38] F. Li, H. Lang, J. Zhang, J. Shen, and X. Wang, "Preconfig: A pretrained model for automating network configuration," *CoRR*, vol. abs/2403.09369, 2024.

[39] R. Mondal, A. Tang, R. Beckett, T. D. Millstein, and G. Varghese, "What do llms need to synthesize correct router configurations?" in *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks, HotNets 2023, Cambridge, MA, USA, November 28-29, 2023*. ACM, 2023, pp. 189–195.