Localized Explanations for Automatically Synthesized Network Configurations

Amirmohammad Nazari*
University of Southern California
Los Angeles, California, USA
nazaria@usc.edu

Yongzheng Zhang*
ShanghaiTech University
Shanghai, China
yongzheng2024@outlook.com

Mukund Raghothaman
University of Southern California
Los Angeles, California, USA
raghotha@usc.edu

Haoxian Chen
ShanghaiTech University
Shanghai, China
hxchen@shanghaitech.edu.cn

ABSTRACT

Network synthesis simplifies network management by automatically generating distributed configurations that fulfill high-level intents. However, typical network synthesizers operate as monolithic algorithms, obscuring the internal workings of the synthesis process and showing no clear connection between the generated configurations and the global intents. Given the critical role of networks as infrastructure, it is crucial for network operators to understand the synthesized configurations to establish trust in these automatic tools. To address this challenge, we propose using subspecifications localized to each component in the network topology to enhance the interpretability of network synthesis. These subspecifications provide insights into the workings of synthesizers by connecting each component's functionalities with the global configuration intents.

We propose a potential solution based on constraint simplification techniques to make the explanation of network configurations manageable. Preliminary results confirm the feasibility of simplifying constraints to a manageable size, though generating high-level subspecifications that fully capture global intents remains future work. This work highlights the importance of interpretability in network synthesis and sets the stage for future research to develop more robust and trustworthy network synthesis tools.

1 INTRODUCTION

Managing networks through manual configurations has long been a complex and error-prone task. Intent-Based Networking (IBN) aims to address this issue by automatically generating network configurations that enforce high-level intents. IBN provides a user-friendly interface for network management, allowing operators to specify desired outcomes rather than intricate configuration details. This approach not only simplifies the management process but also enables automatic verification and synthesis of network configurations [2–4, 13, 20–22], significantly enhancing network correctness and robustness.

Despite these promising advancements, ensuring the trust and interpretability of network synthesis remains a challenge. Operators often find the synthesized configurations to be complex and opaque [12, 17], which hampers broader adoption. Given the intricate nature of network protocols and the extensive scale of configurations, it is difficult for operators to understand how these automatically generated configurations meet their specified intents. This gap in interpretability can lead to mistrust and reluctance in relying on automated network synthesis tools.

Although synthesis outputs are generally verified against user-specified intents, they are not without issues. The inherent complexity of network systems and their protocols means that both verifiers and synthesizers can contain bugs [7, 9]. Additionally, intent specifications themselves can sometimes be ambiguous [9, 18]. Existing tools aimed at identifying verifier bugs operate in a black-box mode, leaving users without insight into why the generated configuration fulfills the specified policies.

NetComplete [12] enhances understandability by allowing users to work with familiar templates. However, it assumes users have a comprehensive understanding of the original network configuration and can accurately identify which parts need changes to support new policies. This approach does not address the fundamental interpretability problem of network synthesis output, leaving a gap in making synthesized configurations easily understandable and trustworthy for network operators.

In contrast to the end-to-end specification and synthesis flow, when network operators manually configure the network, they typically configure network devices one by one. This method naturally breaks down the global policy into smaller, local functionalities, allowing operators to reason about how each local setting contributes to the overall policy.

*Both authors contributed equally to this research.

Modular network verification research [3, 20] has formalized this concept into a semi-automatic process. In this process, operators explicitly define local invariants for different network components. Similar to modular reasoning in software engineering, this modularization greatly improves verification efficiency and simplifies bug localization when verification fails. These studies highlight the potential for decomposing global intent specifications into localized specifications for individual network devices or components.

Interpretability to foster trust. Inspired by these ideas, we aim to utilize subspecifications to foster trust in network synthesis. Given a high-level network intent and the synthesized network configuration, our goal is to generate subspecifications for each configuration component. These subspecifications collectively establish the validity of the global specifications. Similar to function comments that improve software readability, subspecifications establish connections between each part of the network configurations and the global intents, revealing insights on why the generated configurations can fulfill the global intents.

Faster specification refinement iteration. Network synthesis, like many program synthesis tasks, is an iterative process where network operators refine the specifications based on the synthesizer output. Network specifications in general are hard to write correctly and completely at the first time [8, 9], as there are many corner cases in the network, and the precise semantics of the specification language can be hard for the network operators to grasp, especially for ones who are new to formal methods. Therefore, the interpretability of the synthesizer output is crucial for network operators to efficiently identify and refine problematic parts of the specification in an interactive manner.

Localized subspecifications. We propose and investigate the use of localized subspecifications to interpret network synthesis outputs. Subspecifications are local specifications that describe the behavior requirements of individual network components. In contrast to code explanation, which primarily focus on clarifying the code logic or structure given a concrete implementation [15, 19], subspecifications mandate the minimal conditions for a device to satisfy the global specifications, which is more general and flexible.

In addition, subspecifications are tailored to the global intents in question. Essentially, they project all device configurations onto the specific global intent, isolating the local behaviors irrelevant to the global specifications. This is crucial because networks typically have to satisfy multiple global specifications simultaneously, and configurations are often massive in practice. In fact, we believe that code explanations and subspecifications are synergistic. Code explanations can assist network administrators in validating the concrete configuration lines against the subspecifications, which is a more feasible task than directly validating against the global specifications.

Challenges. Generating and validating subspecifications is inherently difficult [3, 17, 20]. Real-world networks involve complex interactions between numerous components and protocols, making it challenging to ensure that synthesized configurations meet the specified global intents. The sheer volume of configurations can be overwhelming, making it difficult for network operators to maintain consistency and coherence. Additionally, determining what constitutes a useful explanation for network operators is complex and requires careful validation to ensure it effectively aids in network management tasks.

Contributions. We propose a framework for generating localized subspecifications to enhance the interpretability of network synthesis. The subspecification form provides a new way to describe the behavior requirements of individual network components in relation to global intents. Our approach is based on constraint-based configuration synthesizers, and uses constraint simplification techniques to create manageable subspecifications that offer clear insights into how the synthesized configurations meet high-level requirements.

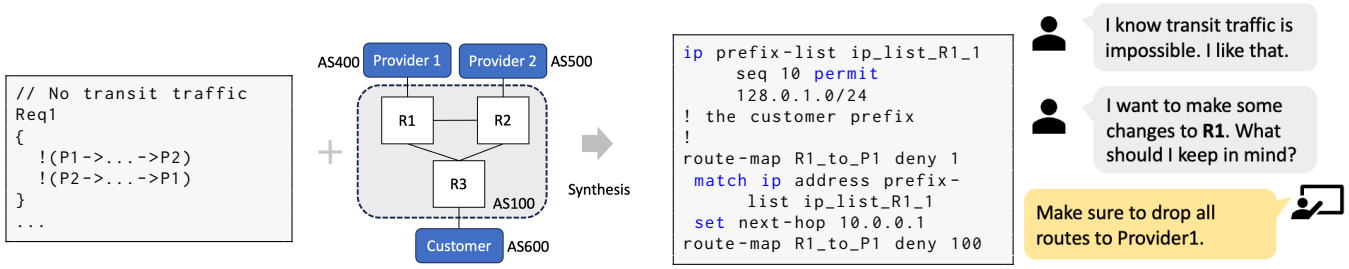
We highlight the need for interpretable network synthesis through three motivating examples, showing how subspecifications help pinpoint misconfigurations, clarify intended behaviors, and isolate relevant configuration lines to simplify large-scale network management.

We validate our approach with a prototype implementation and a case study. Preliminary results show that it is possible to simplify constraints to a manageable size, facilitating easier inspection and validation of synthesized configurations. However, generating high-level subspecifications that fully capture the connection between individual components and the global specification remains a challenge for future research.

This work emphasizes the importance of interpretable designs in network synthesis tools, aiming to bridge the gap between complex automated systems and human operators. By focusing on modularized approaches and exposing internal reasoning, we hope to foster trust and improve the manageability of network infrastructures.

2 MOTIVATING EXAMPLES

We motivate the need for explainable network synthesis through three scenarios where understanding the synthesis output is critical for identifying potential issues in the specification or configuration. These scenarios further demonstrate how subspecifications can facilitate the understanding of network configurations. We follow the formulation of NetComplete [12] for routing policy specification, which uses a



(a) Specification **(b) Topology** **(c) Configuration (partial)** **(d) Explanation**

Figure 1: Overview. Given (a) global specifications, (b) network topology, and (c) synthesized configurations, we generate (d) subspecifications to help network admins interpret the synthesized configurations.

```
R1 {
    !(R1->P1)
}
```

Figure 2: Subspecification at $R1$ for no-transit requirement.

```
...
// For D1, prefer routes through P1 over routes
    through P2
Req2 {
    (C->R3->R1->P1->...->D1)
    >> (C->R3->R2->P2->...->D1)
}
```

Figure 3: Path preference for customer to $D1$.

domain-specific language to describe path preferences and certain prohibited path.

Scenario 1: identifying underspecified paths. Consider the network in Figure 1b, which connects a customer Autonomous System (AS) with two provider ASes.

Suppose initially the network only needs to prevent transit traffic and maintain regular connectivity for the rest of the nodes. This goal is specified in Figure 1a: all paths between Provider 1 (P1) and Provider 2 (P2) are prohibited. The !P syntax means forbidding the following path P.

With these specifications and the network topology, a synthesizer generates concrete network configurations that fulfill these requirements. Figure 1c shows part of the generated configurations for router R1.

At first it may be puzzling to the network administrator why the configuration lines are blocking routes from R1 to Provider1 with IP prefix 128.0.1.0/24 and reset the next-hop attribute for these matched routes.

In order to understand what R1 does to prevent transit traffic, the network administrator asks: "I know there is no transit traffic between the two peering providers. This is good. I like this. Now if I want to make changes to R1, what should I keep in mind?" The answer is: "Make sure to block all routes going to Provider1."

Figure 2 shows the subspecification for R1. In essence, the generated configuration in Figure 1c prevents the R1 from exporting any routes to Provider1, and the set next-hop line is redundant. It is generated because a template is provided.

Here, the network administrator realizes that the configuration lines prevents transit traffic by simply blocking all routes to Provider 1. Although this satisfies the no transit traffic requirement, and is very simple and effective, it is clearly not the behavior the network administrator intended, as it effectively blocks all traffic from Provider 1 to the local network. In order to resolve this issue, the network administrator adds another specification to allow routes from Provider 1 to the customer network.

Scenario 2: resolving ambiguous specifications. Continuing with the network in Figure 1b, the network administrator wants to further specify the preference for routes to a destination network $D1$. Specifically, the administrator wants to ensure that the path through P1 is preferred over the path through P2, as shown in Figure 3.

The path preference requirement in Figure 3 may be ambiguous. It states that for the destination prefix, the path through P1 is preferred over the path through P2. However, the preference for other available paths, such as $Customer \rightarrow R3 \rightarrow R2 \rightarrow R1 \rightarrow P1$, is not specified. There are two possible interpretations:

(1) All unspecified paths are blocked.
(2) All unspecified paths can be chosen when none of the specified paths are available.

One of the authors encountered this issue while using the NetComplete [12] synthesizer. The author intended for the second interpretation, but the tool interpreted the path preference specification according to the first interpretation. As a result, the synthesized configurations had less path redundancy than the user expected.

```
R3 {
    preference {
        (R3->R1->P1->...->D1)
        >> (R3->R2->P2->...->D1)
    }
    !(R3->R1->R2->P2->...->D1)
    !(R3->R2->R1->P1->...->D1)
}
```

Figure 4: Subspecification at $R3$ for path preference requirement.

Such ambiguity could have gone unnoticed because the synthesized configuration is too large to validate easily.

The subspecification shown in Figure 4 clarifies what $R3$ *should do* to meet the path preference requirement in Figure 3: (1) prefer routes through $P1$ over the ones through from $P2$; (2) at import interface to $R1$, drop route $R1 -> R2 -> P2 -> D1$; (3) at import interface to $R2$, drop route $R2 -> R1 -> P1 -> D1$;

Upon seeing this subspecification, the network administrator realizes that the the network configuration is actually trying to block path that are not explicitly specified, contradicting the original intend. And therefore adds additional specifications to allow other available paths as the last resort when none of the specified paths are available.

Note that subspecification here is different from just explaining the configuration lines, as it uses a more general constraint language to describe the behavior of the router. In other words, there can be multiple ways to configure the router to meet the subspecification. For instance, to drop the route $R1 -> R2 -> P2 -> D1$ at the import interface $R1 - R3$, $R3$ can either: (1) match the destination prefix $D1$; or (2) match the community tag $100 : 2$. And many other possible implementations. The subspecification captures all these possibilities, and summarizes the local behavior of the router in a concise way that is easier to understand.

This scenario highlights the importance of clear and unambiguous specifications to ensure that the synthesized configurations meet user expectations.

Scenario 3: taming complexity. Continuing with the previous scenarios, the network administrator adds a few additional requirements to the network. With the help of the synthesizer, a set of configurations is generated that meets all the requirements. However, the volume of configurations can easily overwhelm the network administrator.

To manage this complexity, the network administrator can ask questions about each requirement individually. For instance, when asked about the no transit traffic requirement, the subspecifications reveal that $R3$ can do anything to meet this requirement (empty subspecification).

```
R2 to P2 {
    !(P1->R1->R2->P2)
    !(P1->R1->R3->R2->P2)
}
```

Figure 5: Subspecification at $R2$ for no-transit requirement.

Therefore, the network administrator can focus on validating the configurations for $R1$ and $R2$, the routers relevant to the no transit traffic requirement. As shown in Figure 5, the subspecification for $R2$ is to drop all routes from P1 to P2. Similarly, the subspecification for $R1$ is to drop all routes from P2 to P1.

As demonstrated in this scenario, subspecifications can isolate relevant configuration lines, helping administrators understand the synthesized configurations. This approach simplifies validation by breaking down complex configurations into manageable parts aligned with specific global requirements.

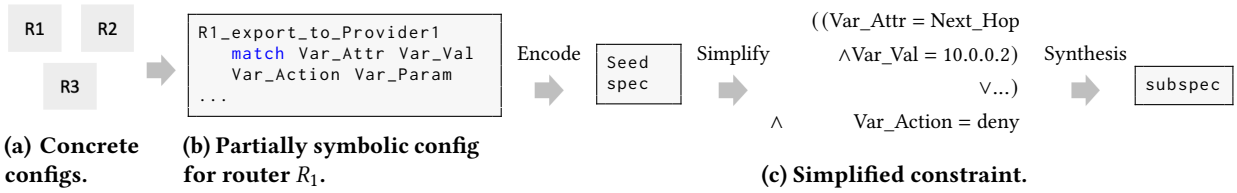
3 GENERATING SUBSPECIFICATION

The synthesis problem. We formulate the synthesis problem based on NetComplete [12], where the specification is a set of path requirements. Specifically, in this case study we only concern two kinds of requirements: (1) Path preference: traffic from source to destination should follow the most preferred and available path; (2) Forbidden path: certain paths cannot be taken by any traffic (e.g., no-transit traffic rule).

Given the specification, network topology, and an optional network configuration sketch, the synthesis problem is to generate a concrete network configuration that satisfies the specification.

The Explanation Problem. We use the same language for subspecifications as for the global specification. The difference is that subspecifications concern only the local behavior of a particular router. The rationale for choosing the same specification language is twofold. First, the explanation problem can be viewed as a reverse engineering problem, where the goal is to derive the specification for individual devices from the concrete configuration. Thus, it is natural to use the same language as the global specification. Second, using the same language as the global specification, rather than developing a new explanation language, reduces the cognitive load on network administrators.

The explanation problem, therefore, is to generate a subspecification for each individual device, given the input and output of the synthesis problem. The subspecification should be in the same language as the global specification and should be validated against the concrete configuration.



(a) Concrete configs. **(b) Partially symbolic config for router R_1.** **(c) Simplified constraint.**

Figure 6: The subspecification generation flow

Generating subspecification. The basic idea is to search within the explanation language space, which is the same as the global specification language, and find a subspecification consistent with the local configuration and the rest of the network configuration encoding. However, this approach is computationally expensive due to the complexity of the explanation space and the network configuration.

The key insight is that, although the encoding of network configurations and routing policies is typically complex, consisting of more than 1000 constraints even in the simple scenario in Section 2, these constraints can be significantly reduced when the configurations for most devices are given concrete values and only a few variables in the device in question are left symbolic. In our experimental scenario, this reduction resulted in only a few constraints.

Based on this insight, we propose a solution to the explainable network synthesis problem that utilizes constraint simplification techniques. Figure 6 illustrates this process. We start by constructing a specification as constraints on variables in the configuration to be explained, as the "seed specification". These constraints are then simplified through a set of rewrite rules [17]. The simplified constraints are typically small, making it tractable for a synthesizer to find a subspecification in the explanation language space that is consistent with the simplified constraints.

Note that this approach relies on the assumption that the underlying synthesizer is constraint-based. The constraint simplification technique requires a "seed specification" that encodes the global requirements. It is essential to use the same encoding process as the synthesizer to generate a "seed specification" consistent with the synthesizer's interpretation. Notably, many existing network synthesizers fall within this category [1, 6, 11, 12, 21].

Continuing with scenario 3, we walk through the subspecification generation process. (1) For the device in question (R1), it replaces the concrete configuration lines with symbolic variables, resulting in a partially symbolic configuration. As shown in Figure 6b, concrete configuration lines are replaced by symbolic variables representing the matching attribute (`Var_Attr`), action (`Var_Action`), and the corresponding parameters (`Var_Val`, `Var_Param`).

(2) The partially symbolic configuration is then encoded with the concrete configurations of the other devices, the network topology, and the protocol mechanics, into a set of constraints on the symbolic variables in the configuration lines. The encoding process follow the same process as the NetComplete [12] synthesizer. The encoding is referred to as the "seed specification" as it will be used to generate the subspecification.

(3) Given the seed specification, we then simplify it by applying simplification procedures in prior work [17], where a set of rewriting rules are applied to the seed subspecification iteratively to achieve the minimal form. There are 15 simlification rules, and here are two examples:

$$False \longrightarrow a = True$$

$$a \vee !a = True$$

Part of the simplified constraints are shown in Figure 6c, which represents the minimal requirement for the device in question (R1) in order to make the whole network satisfy the global specification.

(4) Finally, we search for a subspecification in the specification language space that is consistent with the simplified constraints, as shown in Figure 2. Although we have outlined the basic steps for this approach, the specific methods for efficiently searching the specification language space remain an open question. Developing effective techniques for this search process will be a key focus of our future work.

4 PRELIMINARY RESULTS

We implement a prototype to simplify the extracted seed specifications. Using NetComplete [12] as the configuration synthesizer, we generate configurations for the topology shown in Figure 1b using a configuration sketch and specifications for three scenarios. Seed specifications for router R1 are created by replacing parameters in the configuration lines with symbolic variables. The synthesizer's encoding procedure is then invoked to generate constraints on the symbolic configuration of R1, combined with concrete configurations of the other devices, forming the seed specification.

Observations. The preliminary results reveal several key insights. (1) By manually examining the simplified constraints, we can infer why certain configurations were generated in

a specific manner. This process provides hints about the decision-making behind the synthesized configurations. (2) Such inference is possible because, given the configurations of the rest of the network are fixed, the configuration options for a particular router are significantly limited, making the reasoning process more manageable. (3) However, there are still missing elements in our approach. Specifically, we need precise encoding information for the network configuration, such as the explicit dependencies among the generated intermediate variables. Such information is essential to further simplify the constraints for useful insights.

5 DISCUSSION

Limitations of generic explainable program synthesis approaches. As we demonstrate in Section 4, Directly applying current explainable program synthesis tools [17] to network synthesis problems does not adequately address these challenges. While these tools can simplify SMT constraints, the resulting subspecifications remain expressed in terms of low-level variables and constraints that are difficult for network administrators to interpret.

The fundamental issue with these generic tools is the lack of the contextual understanding needed to map these constraints to meaningful network specifications. Nevertheless, having the simplified low-level constraints are still useful as it reduces the amount of work in the following subspecifiation generation process. The next interesting research question then is how to lift the low-level specifications to the high-level specification language.

High-level summary of the global behaviors. Similar to the modular verification approach [3, 20], reasoning about local network behaviors requires assumptions about global behaviors. For instance, in the no-transit specification in Section 2, when inspecting the local subspecification for router $R1$, which denies routes with community $100:2$ from $R1$ to P1, it is essential to ensure a route is tagged with community $100:2$ if received from P2. A possible solution is to view the rest of the network as a single component and determine the necessary actions of other devices to fulfill the global specification, given the concrete configurations of a particular router. However, as the number of network devices increases, so do the low-level variables and constraints. Addressing this combinatorial explosion is a promising area for future research.

User study. A comprehensive user study is needed to validate whether the proposed subspecifications can assist network administrators in understanding synthesized configurations. The study would involve network administrators with varying experience levels, providing them with synthesized configurations with and without localized subspecifications. Participants would perform tasks such as identifying misconfigurations, validating policy compliance, and making adjustments. This goal is to assess the effectiveness and usability of localized subspecifications, and provide recommendations for their integration into network management workflows.

Explanation beyond constraint-based synthesizers. Future research should focus on developing methodologies that extend beyond constraint-based techniques to accommodate various synthesis approaches. In this work, we assume the underlying synthesizer uses constraint-based synthesis techniques. However, there are synthesizers that use custom algorithms [5, 18] and other general techniques like LLMs (Large Language Models) [16]. A more general solutions is needed to ensure that the generated explanations are comprehensible and relevant across different contexts and methodologies, enhancing the versatility and applicability of explainable network synthesis tools.

Implication for explainable network verification. We observed significant synergy between explaining network configuration synthesis outputs and network configuration verification. Traditionally, both tasks have operated in a black-box mode, providing users with only a yes or no answer without revealing why the configuration satisfies the given property. We believe the idea of localized subspecifications can also be generalized to assist in explaining network verification.

6 RELATED WORK

Modularized network verification. Lightyear [20] and Timepiece [3] uses modularized approach to scale configuration verification. Our work is inspired by this modularization concept, but focus on using it for explaining configuration synthesis output.

Network configuration synthesis. Existing work on network configuration synthesis [1, 5, 6, 11, 12, 18] typically operates in a black-box mode. We aim to enhance transparency and trust in network synthesis by addressing the explainable configuration synthesis problem.

Intent inference. Similar to our work, specification mining from existing network configuration has been another promising approach for network explainability. Notably, Config2Spec [8] and Anime [14] mine global intents from network configurations. Unlike these work, we focus on generating localized subspecification such that the synthesis output configuration at each device can be validated easily.

Network provenance. Network provenance [10, 24, 25] provides positive explanations, i.e., elucidating why certain events occur by showing the chain of derivations leading to the observed events. In contrast, network configuration

explanation requires counterfactual analysis, i.e., understanding why certain traffic is impossible. Such explanations can be more effectively captured using a specification language.

Explainable program synthesis. Zhang et al. [23] reveal the internal workings of program synthesis to improve interpretability. Nazari et al. [17] explain synthesis outputs by localizing specifications to each syntactic structure. As shown in our case study, these tools need networking context to provide useful explanations.

7 CONCLUSION

This paper highlights the importance of interpretability in network synthesis, emphasizing its role in fostering trust, identifying misconfigurations and ambiguities, and accelerating the iteration process for network operators. We propose a framework for generating localized subspecifications using constraint simplification techniques to assist administrators in understanding and validating synthesized configurations. Preliminary results demonstrate the feasibility of simplifying constraints to a manageable size, and reveals useful insights for understanding the synthesis output. However, generating high-level subspecifications remains a challenge for future research. By emphasizing the need for interpretable designs in verification and synthesis tools, we aim to enhance transparency in intent-based networking.

REFERENCES

[1] Anubhavnidhi Abhashkumar, Aaron Gember-Jacobson, and Aditya Akella. Aed: Incrementally synthesizing policy-compliant and manageable configurations. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, pages 482–495, 2020.

[2] Anubhavnidhi Abhashkumar, Aaron Gember-Jacobson, and Aditya Akella. Tiramisu: Fast multilayer network verification. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 201–219, 2020.

[3] Timothy Alberdingk Thijm, Ryan Beckett, Aarti Gupta, and David Walker. Modular control plane verification via temporal invariants. *Proceedings of the ACM on Programming Languages*, 7(PLDI):50–75, 2023.

[4] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. A general approach to network configuration verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 155–168, 2017.

[5] Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitendra Padhye, and David Walker. Don't mind the gap: Bridging network-wide objectives and device-level configurations. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 328–341, 2016.

[6] Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitendra Padhye, and David Walker. Network configuration synthesis with abstract topologies. In *Proceedings of the 38th ACM SIGPLAN conference on programming language design and implementation*, pages 437–451, 2017.

[7] Rüdiger Birkner, Tobias Brodmann, Petar Tsankov, Laurent Vanbever, and Martin Vechev. Metha: Network verifiers need to be correct too! In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 99–113, 2021.

[8] Rüdiger Birkner, Dana Drachsler-Cohen, Laurent Vanbever, and Martin Vechev. {Config2Spec}: Mining network specifications from network configurations. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 969–984, 2020.

[9] Matt Brown, Ari Fogel, Daniel Halperin, Victor Heorhiadi, Ratul Mahajan, and Todd Millstein. Lessons from the evolution of the batfish configuration analysis tool. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 122–135, 2023.

[10] Ang Chen, Yang Wu, Andreas Haeberlen, Wenchao Zhou, and Boon Thau Loo. The good, the bad, and the differences: Better network diagnostics with differential provenance. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 115–128, 2016.

[11] Ahmed El-Hassany, Petar Tsankov, Laurent Vanbever, and Martin Vechev. Network-wide configuration synthesis. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II 30*, pages 261–281. Springer, 2017.

[12] Ahmed El-Hassany, Petar Tsankov, Laurent Vanbever, and Martin Vechev. {NetComplete}: Practical {Network-Wide} configuration synthesis with autocompletion. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 579–594, 2018.

[13] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. A general approach to network configuration analysis. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 469–483, 2015.

[14] Ali Kheradmand. Automatic inference of high-level network intents by mining forwarding patterns. In *Proceedings of the Symposium on SDN Research*, pages 27–33, 2020.

[15] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. Experiences from using code explanations generated by large language models in a web software development e-book. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 931–937, 2023.

[16] Rajdeep Mondal, Alan Tang, Ryan Beckett, Todd Millstein, and George Varghese. What do llms need to synthesize correct router configurations? In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, pages 189–195, 2023.

[17] Amirmohammad Nazari, Yifei Huang, Roopsha Samanta, Arjun Radhakrishna, and Mukund Raghothaman. Explainable program synthesis by localizing specifications. *Proceedings of the ACM on Programming Languages*, 7(OOPSLA2):2171–2195, 2023.

[18] Sivaramakrishnan Ramanathan, Ying Zhang, Mohab Gawish, Yogesh Mundada, Zhaodong Wang, Sangki Yun, Eric Lippert, Walid Taha, Minlan Yu, and Jelena Mirkovic. Practical intent-driven routing configuration synthesis. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 629–644, 2023.

[19] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*, pages 27–43, 2022.

[20] Alan Tang, Ryan Beckett, Steven Benaloh, Karthick Jayaraman, Tejas Patil, Todd Millstein, and George Varghese. Lightyear: Using modularity to scale bgp control plane verification. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 94–107, 2023.

[21] Fangdan Ye, Da Yu, Ennan Zhai, Hongqiang Harry Liu, Bingchuan Tian, Qiaobo Ye, Chunsheng Wang, Xin Wu, Tianchen Guo, Cheng Jin, et al. Accuracy, scalability, coverage: A practical configuration verifier on a global wan. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications,*

technologies, architectures, and protocols for computer communication, pages 599–614, 2020.

[22] Peng Zhang, Dan Wang, and Aaron Gember-Jacobson. Symbolic router execution. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 336–349, 2022.

[23] Tianyi Zhang, Zhiyang Chen, Yuanli Zhu, Priyan Vaithilingam, Xinyu Wang, and Elena L Glassman. Interpretable program synthesis. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–16, 2021.

[24] Wenchao Zhou, Qiong Fei, Arjun Narayan, Andreas Haeberlen, Boon Thau Loo, and Micah Sherr. Secure network provenance. In *Proceedings of the twenty-third ACM symposium on operating systems principles*, pages 295–310, 2011.

[25] Wenchao Zhou, Micah Sherr, Tao Tao, Xiaozhou Li, Boon Thau Loo, and Yun Mao. Efficient querying and maintenance of network provenance at internet-scale. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 615–626, 2010.