Explainable Network Verification via Localized Subspecification

Yongzheng Zhang
ShanghaiTech University
Shanghai, China
yongzheng@shanghaitech.edu.cn

Yaxuan Lin
ShanghaiTech University
Shanghai, China
linyx2025@shanghaitech.edu.cn

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

Ruize Ma
ShanghaiTech University
Shanghai, China
marz2024@shanghaitech.edu.cn

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

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

Peng Zhang
Xi'an Jiaotong University
Xi'an, China
p-zhang@xjtu.edu.cn

Abstract

Network verification, synthesis, and repair tools help enforce high-level operational intent, but their limited explainability makes configuration maintenance costly in practice, as operators must still manually reason about large, low-level configurations. We propose *localized subspecifications*, which explain how individual configuration elements preserve a given network property by constraining their admissible behaviors. A user study with 15 professional network operators and 8 graduate students shows 52% higher accuracy and 23% time savings, and 70% of participants reported that they would like to use subspecifications in daily operations, demonstrating practical benefits. To support real deployments, we develop SpecLens, an explainable network verification system that generates localized subspecifications using a scalable algorithm with soundness guarantees. SpecLens computes line-level and field-level subspecifications in 10 minutes on the real-world Internet2 configuration and 25 minutes on FatTree networks with up to 1,280 routers.

CCS Concepts

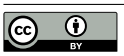
• **Networks** → **Network manageability**; **Formal specifications**.

Keywords

Network Verification, Explainability, Modular Reasoning

ACM Reference Format:
Yongzheng Zhang, Yaxuan Lin, Haoxian Chen, Ruize Ma, Amirmohammad Nazari, Mukund Raghothaman, and Peng Zhang. 2026. Explainable Network Verification via Localized Subspecification. In *ACM SIGCOMM 2026 Conference (SIGCOMM '26), August 17–21, 2026, Denver, CO, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3789240.3829155

*Corresponding author



SIGCOMM '26, Denver, CO, USA

ACM ISBN 979-8-4007-2467-1/26/08
https://doi.org/10.1145/3789240.3829155

1 Introduction

Network operators manage a network by enforcing a set of *high-level* properties, or specifications, *e.g.,* reachability, isolation, and routing path preference [1, 5, 16, 25], with *low-level* configurations, *e.g.,* route maps, prefix lists, and attribute manipulations. The control plane (*e.g.,* BGP, OSPF) interprets these configurations and computes a network state that realizes them. Such a *semantic gap* between low-level configuration and high-level specification is a long-standing source of configuration mistakes and operational overhead, especially during network maintenance and updates [6, 8, 10].

A broad range of automated tools address this semantic gap across different operational settings. Network verifiers [2, 5, 25] check whether a given configuration C satisfies a specification φ. Synthesis tools [6, 7, 14, 15] construct brand new configurations that satisfy a given specification, while repair tools [21] generate patches to a configuration that violates the specification.

Despite significant progress in automated tools, their adoption in production networks is still low. We argue a key reason is the lack of *explainability*, a topic largely underexplored in the community. Modern network configurations are large and complex, making it difficult for operators to fully understand the intricate connections between configurations and specifications. Without such an understanding, and considering the critical nature of network operation, it is hard for operators to trust and apply the results of verification, synthesis, or repair tools.

```
permit 10.0.0.0/16 ge 16
```
(a) Exact.

```
permit 10.0.0.0/8 ge 8
```
(b) Overly restrictive.

Figure 1: Two prefix-lists for private IPs in `10.0.0.0/16`.

For example, consider a policy requiring that no private IPv4 prefixes within the 10.0.0.0/16 block are advertised externally. Fig. 1 shows two policy implementations using prefix-list. Both could pass the verification against the intent, but the second is overly restrictive: it blocks all prefixes under `10.0.0.0/8` rather than just those under `10.0.0.0/16`, unintentionally suppressing legitimate internal routes. This motivates explainable verification methods that not only validate correctness but also reveal which configuration choices are *actually required* to enforce the intended semantics.

Therefore, we envision that a tool providing fine-grained explanation for configurations would be quite useful for operators to connect high-level specification with low-level configuration, thereby paving the way for adoption of automated tools.

A natural approach is to leverage LLM-powered code explanation. However, LLM-based code explanation does not guarantee correctness. A misleading explanation can cause operators to adopt an erroneous configuration. Secondly, current LLMs still have a limited context, and fail to capture the intricate dependencies among a huge number of configuration lines on thousands of devices. For example, in our user study (Sec. 6), a professional operator used an AI-based assistant to interpret a route-map line. Although it correctly explained the line's local semantics, it failed to expose a downstream dependency, leading the participant to approve an unsafe update.

Key idea: precise configuration explainability via subspecification. We approach explainability via *localized subspecification*: the precise requirement that each configuration element must satisfy for the global specification to hold, assuming the rest of the configuration remains *fixed*. This localization reduces reasoning about large, complex networks to focused, per-element analysis, making explicit how each configuration choice contributes to high-level operational intent. Returning to the example in Fig. 1, the line h: `permit [`p`] ge [`l`]` should have the following subspecification:

$$\psi(h) : \texttt{10.0.0.0/16} \subseteq \texttt{Addr}(p) \ \wedge \ l \leq 16 \tag{1}$$

That is, as long as the prefix parameter p covers all addresses in `10.0.0.0/16` and the `ge` parameter $l \leq 16$, the policy remains satisfied. It reveals that the line in Fig. 1b is overly restrictive, as it enforces conditions stronger than required.

Conceptually, localized subspecifications (subspecs for short) are analogous to *pre/post conditions* in program verification: precisely capturing the behavioral contract at each configuration location. This contract-based view enables modular reasoning, clearer explanations, and safer configuration evolution.

Compared to LLM-based code explanation, subspecifications have a number of merits: (1) they specify what a line *must satisfy* to preserve correctness, rather than its current behavior; (2) they provide precise and sound explanations without hallucinations; and (3) they can be computed efficiently (Sec. 7).

Use cases. We realize this approach in SpecLens, an explainable network verification system that generates and presents localized subspecifications for verified network configurations. Localized subspecifications form an *explanation layer* for existing verification, repair, and synthesis tools, helping operators understand and safely modify configurations during routine network operation.

Explaining large-scale configurations. Localized subspecs identify the few locations critical to a property, *e.g.,* ≈ 2% of locations in Internet2 (Sec. 7). This guides attention to route-relevant locations, supporting scalable debugging, maintenance, and refactoring.

Auditing repairs and updates. When configurations change through manual or automated repair, operators need to know not just whether a patch or update is correct, but why. Localized subspecifications expose the admissible behavior at each modified location, making repair rationales explicit and enabling confident auditing.

Understanding synthesized configuration. Synthesis tools may generate correct yet unintuitive configurations or programs [22, 23]. Localized subspecifications expose how each element is constrained and how the configuration satisfies the global specification.

This work focuses on explaining *correct* configurations, *i.e.,* they have been verified to satisfy a given specification. Explaining failed verification is complementary and is discussed in Sec. 8.

Challenges. Deriving precise subspecifications, *i.e.,* without fixing the surrounding configuration, is challenging because configuration elements interact through routing protocols, and many global configurations may satisfy the same specification. Isolating the exact contribution of a single element requires network-wide reasoning, and remains a non-trivial manual effort in practice [3, 28].

By fixing the surrounding configuration and analyzing selected locations locally, we avoid exploring alternative implementations elsewhere, making automatic reasoning tractable. Localization also implies non-compositionality, *i.e.,* individually safe updates may interact when combined (Proposition 5), but localized subspecifications remain highly effective for configuration maintenance.

Contributions and organization. This paper tackles these challenges after reviewing network verification (Sec. 2) and presenting an illustrative example (Sec. 3).

Subspecification formulation. (Sec. 4) We define valid configuration locations at multiple granularities and formalize the syntax and semantics of localized subspecifications.

Subspecification generation algorithm. (Sec. 5) We develop SpecLens, which uses a scalable algorithm to decompose global reasoning into router-local analysis based on control-plane stable state. We establish its correctness, conciseness, and parallel scalability.

User study. (Sec. 6) In our task-based user study [36] with professional network operators and graduate students, subspecifications improve accuracy by 52% and reduce completion time by 23%, and 70% of participants would like to use our tool frequently.

Evaluation. (Sec. 7) On the real-world and synthetic configurations, subspec generation is efficient and scalable: 10 minutes on Internet2, and 25 minutes on FatTree networks with up to 1,280 routers.

We conclude by discussing the limitations and fundamental tradeoffs of localized subspecifications.

Ethics statement. This work includes a user study with informed consent and no sensitive data. It followed institutional and community guidelines and received approval from the lead author's institution. Participants were allowed to withdraw at any time, and were debriefed after the study.

2 Background and motivation

We begin by outlining why network configurations are difficult to interpret, discuss limitations of existing approaches, and then motivate our design of localized subspecifications.

Network configurations are hard to read. Configurations are hard to reason about for several reasons. In particular, we highlight: (1) *semantic subtlety*: similar commands can have subtle differences across vendors; (2) *scale*: long prefix-lists make IP prefix matching hard to compute mentally; (3) *interdependencies*: interactions

Approach	Configuration	Routes
Verification	Fixed	Computed
Synthesis	Computed (global, single)	Fixed
Localized Subspec	Computed (local, all)	Fixed

Table 1: Conceptual contrast with existing work.

across protocols and routers are complex; and (4) *global reasoning*: alternative converged routing states are hard to analyze.

Automated reasoning. To cope with this complexity, various automatic reasoning tools have been developed.

Verification. Verification tools check if a configuration satisfies desired properties. Symbolic methods [5] encode protocol semantics and policies as logical constraints over routes and explore feasible stable states via constraint solving, providing strong guarantees but limited scalability. Simulation-based methods [16, 25, 33] instead compute a converged stable state (*e.g.,* a forwarding graph) and check properties directly on it. They are typically more scalable and form the basis of our stable-state–driven decomposition.

Synthesis and repair. Similarly, synthesis and repair techniques treat parts of the configuration as unknowns and solve for assignments that satisfy routing constraints [1, 6, 7, 14, 15, 26], often relying on verifiers to validate candidate solutions.

Explainability: a missing part in network verification. As participants reported in our user study (Sec. 6), despite advances in automatic network reasoning, manual configuration inspection remains unavoidable. This is partially due to the high-stakes nature of networks and imperfect modeling in these tools. The other reason, we argue, is the lack of explainability of the verification results.

Localized subspecification for explainable verification. We use *localized subspecifications* as an explainability mechanism, which associates each configuration element with constraints on the edits that preserve a verified property under the current stable state. Tab. 1 highlights the contrast: unlike verification (configs fixed, routes computed) and synthesis (single config computed globally), localized subspecifications fix the stable routes and compute *all admissible local configuration implementations*, *e.g.,* Eq. 1, turning global correctness into localized explanations.

Unlike provenance-style explanations [12, 37, 38] (why behavior occurred) or global intent inference [8, 20] (what the network as a whole enforces), subspecs give *local, normative explanations*: what each element must satisfy to remain correct. They address explainability along two dimensions:

Exposing modeling assumptions. Subspecs explicitly reveal the assumptions made by the underlying verifier, clarifying subtle vendor-dependent semantics, *e.g.,* whether `le 32` is interpreted as inclusive, thereby improving transparency.

Reducing reasoning burden. (1) *Semantic clarification*: subspecs make config semantics explicit, reducing ambiguity; (2) *Attention focus*: only a small fraction of configuration locations are relevant to routes for a given prefix, *e.g.,*≈ 2% in Internet2 (Sec. 7); (3) *Intuition validation*: subspecs expose overlooked cross-router dependencies when operators' intuitive judgments conflict with the subspecification (Sec. 6); (4) *Local reasoning*: operators can check edits against local constraints without mentally simulating global convergence.

Symbol	Meaning
r	A network router.
C	A network configuration.
P	A target network prefix.
φ	A global network specification or correctness property.
S(C)	Stable control-plane state (routes), induced by C.
h	A specific location in the configuration.
g	A replacement value for a specific configuration location.
$\psi(h)$	Localized subspecification at location h.
$\mathsf{C}[h \leftarrow g]$	Configuration C with location h replaced by g.
$\mathsf{S}_r^{\mathsf{P}}$	Router-level functional subspecification for r and P.
C_r	Router-local configuration slice for r.
$\textsc{Encode}(\mathsf{C}_r)$	Router-local symbolic encoding of C_r.
$\textsc{StateEq}(\mathsf{S}_r^{\mathsf{P}})$	Stable-state equality constraints induced by $\mathsf{S}_r^{\mathsf{P}}$.
$\textsc{Simplify}(F)$	Equivalence-preserving simplification of formula F.

Table 2: Summary of notations.

3 Overview

Fig. 2 shows the generation flow of *localized subspecifications*.

Goal. Given a configuration C that satisfies specification φ ($\mathsf{C} \models \varphi$), derive for each location h (*e.g.,* a prefix-list line or parameter) a constraint $\psi(h)$ capturing the minimal requirement on h to preserve φ. Equivalently, $\psi(h)$ characterizes all replacements g at h such that $\mathsf{C}[h \leftarrow g] \models \varphi$. Tab. 2 summarizes the notations.

Weaker guarantee for tractability and interpretability. Computing a subspecification for a single configuration location can be as hard as verifying the entire network because its effect depends on network-wide routing convergence (Sec. 5.1). This analysis must then be repeated for many locations.

To make the problem tractable, we relax the goal. Rather than explaining why a configuration C satisfies a specification φ under *all* possible converged states, we focus on a more concrete question: why does C converge to the *observed stable control-plane state* S(C) that satisfies φ? Thus, subspecifications explain the correctness of the observed stable state, not all hypothetical alternatives.

This relaxation intentionally trades completeness for (1) *efficiency*: it enables a two-phase, router-local analysis, yielding a scalable algorithm with a soundness guarantee and parallel scalability (Sec. 5.4); and (2) *interpretability*: operators reason about concrete, realized routing behavior, rather than mentally simulating alternative convergence outcomes.

The subspecification generation proceeds in two phases.

(1) Router-level functional subspecification (Fig. 2, middle). From the simulated stable control-plane state we extract for R1:

- `PEER-BEST`: the overall best routes at directly connected peers R2 and R3 for prefix `10.1.0.0/16`;
- `DECISION`: the best-path selection at R1, selecting the preferred route via R2 for prefix `10.1.0.0/16`;
- `LOCAL-BEST`: the overall best route at R1 for prefix `10.1.0.0/16`.

Together, these conditions form R1's router-level functional subspecification $\mathsf{S}_{\mathsf{R1}}^{\mathsf{P}}$: `PEER-BEST` provides the assumptions on neighboring routers, while `DECISION` and `LOCAL-BEST` specify the *local behavior* that R1 must preserve (Sec. 5.2). Because this stable state has already been validated by verification, preserving it is sufficient to preserve the verified global network property φ.

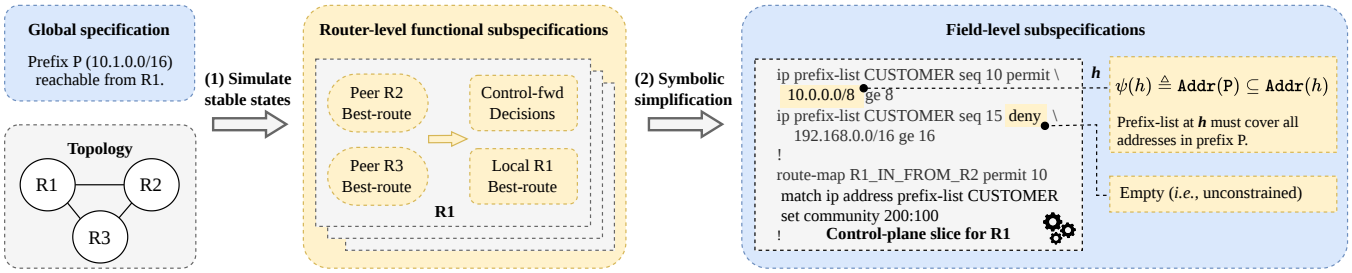


Figure 2: Subspecification generation flow, using stable control-plane behavior as router-level functional subspecification.

In this example, the router-level functional subspecification requires that, given the overall best routes at R2 and R3, R1 continues to select the route via R2 as its overall best route for `10.1.0.0/16`, with the same route attributes as in the observed stable state.

This router-level functional subspecification localizes subsequent reasoning: field-level subspecifications need only characterize the constraints on each field that preserve this *local behavior*, without considering the rest of the network.

(2) Field-level subspecification (Fig. 2, right). We extract R1's router-local configuration slice (or router-local slice) C_{R1} from C:

- export routing policies to R1 at all directly connected peers;
- all import routing policies at R1;
- the local best-path selection process at R1.

We obtain router-local symbolic encoding (or *router-local encoding*) $\textsc{Encode}(C_{R1})$ of C_{R1} and replace a configuration location h with a fresh symbolic variable v, where all other configuration locations remain fixed. Conjoining the resulting encoding with the constraints induced by R1's router-level functional subspecification S^{P}_{R1} yields a *seed subspecification*. The seed subspecification characterizes all value combinations at the target location that preserve the router-level behavior. We then simplify it using SMT-based equivalence-preserving tactics to obtain the final field-level subspecification.

Continuing with the example, after simplification, the subspecification for the prefix-list CUSTOMER seq 10 `10.0.0.0/8` field requires that it match all addresses in the target prefix `10.1.0.0/16`. For the seq 15 deny field, the subspecification is empty (*i.e.,* unconstrained), indicating that the field may take any value without violating the router-level functional subspecification. Sec. 5.3 provides details and the subspecification generalization to larger blocks.

In this process, field-level subspecifications directly connect the *concrete configuration* with the *abstract router-level functional subspecification*. Each $\psi(h)$ captures the minimal condition that a specific configuration field must satisfy, explaining precisely *why* that field matters and filtering out irrelevant details.

Scope. This formulation explains properties expressible as constraints on the network's stable control-plane state, *e.g.,* reachability and path preference, and therefore assumes control-plane convergence [5]. Its precision is defined with respect to the underlying verifier: edits that satisfy a subspecification are guaranteed to preserve the verifier's conformance result, while inaccuracies in the verifier's model may lead to imprecise explanations. It does not cover transient behaviors [27] or performance [4].

Limitation. Because subspecifications preserve the observed stable state rather than all possible correct stable states, they are sound but not complete: an edit that violates a subspecification may still satisfy the global property by inducing a different valid stable state.

Deployment workflow. Despite these limitations, subspecifications are highly useful in practice: they serve as post-hoc correctness contracts that complement verification, synthesis, repair, and LLM-based configuration tools. Concretely, (1) *offline precomputation*: after a configuration has been verified, SpecLens computes the router-level functional subspecifications, all line-level joint subspecs and all field-level subspecs. The line-level formulas are reused to derive the corresponding field-level subspecs. (2) *online update checking*: for simultaneous edits spanning multiple locations (*e.g.,* multiple lines), a joint subspec (Sec. 5.3) is generated on demand.

4 Subspecification

We formalize localized subspecifications by first identifying valid configuration locations at different granularities, defining the corresponding subspecification formats for each level, and then illustrating them with concrete examples.

Location in a network configuration. A *location* h denotes a structural unit in a router configuration. As summarized in Tab. 3, locations range from entire routers and route-maps down to individual lines and fields.

Subspecification forms. Subspecifications must align with the semantics of the configuration elements they explain. High-level constructs such as routers and route-maps define *functional behavior* over routes, while low-level configuration elements encode *parameter choices* within a given structure. Accordingly, we adopt two complementary subspecification forms.

(1) Assume–guarantee contracts. For routers and route-maps, subspecifications use functional assume–guarantee contracts. A router contract relates peers' overall best routes to its best-path decisions and resulting overall best route, while a route-map contract relates an input route to a permit or deny decision and an updated route.

(2) Value constraints. For lines and fields, subspecifications are predicate-style constraints that restrict the admissible parameter values, such as prefix ranges, community set, or attribute bounds.

Semantics. Given a configuration C that satisfies a global network property φ (*i.e.,* $C \models \varphi$) and a location h, the *localized subspecification* at h is a constraint $\psi(h)$ such that $\forall g.\ g \models \psi(h) \iff C[h \leftarrow g] \models \varphi$. That is, $\psi(h)$ precisely characterizes the set of local instantiations at h that preserve φ, assuming the remainder of the configuration $C \setminus h$ is fixed.

Location	Signature	Interpretation
Assume–guarantee functional subspecifications		
Router	(IF → Route) ⇒ ((IF → Boolean), Route)	Peers' overall best routes to the router's best-path decisions and resulting overall best route.
Route-map	Route ⇒ (Boolean, Route)	Input route, to permit/deny decision and updated route.
Predicate (range) subspecifications		
Line	Field∗ ⇒ Boolean	Constraint on admissible combinations of its fields.
Field	Field ⇒ Boolean	Constraint on the field's admissible values.

Table 3: Subspec forms across config granularities.

Intuitively, any update at location h that satisfies $\psi(h)$ preserves correctness, while violating $\psi(h)$ may break the global property. By explicitly characterizing this *viable range* of safe edits, localized subspecifications explain why a configuration element must take its current form and how it contributes to the global intent.

Relaxation. Reasoning over all possible control-plane steady states is expensive and often produces large, hard-to-interpret subspecifications (Sec. 3). We therefore relax the problem to preserve only the same stable state (Sec. 5). This yields shorter, more readable subspecifications and enables scalable generation (Sec. 7).

Example 1: [Subspec at different granularities] Fig. 3 illustrates how a global network property is explained through localized subspecification at different granularities. The specification φ requires prefix P to remain reachable from router R1. For R1, its router-local slice contains the export policies of its peers toward R1, the local import policies, and best-path selection process.

In the observed stable state, R2 and R3 have overall best routes a and b for prefix P, respectively. Given these peer routes, R1 selects the route via R2, and its resulting overall best route carries community c. The router-level functional subspecification S^{P}_{R1} therefore requires R1 to preserve this best-path decision and the required best route attributes under the same peer-route assumptions.

At finer granularities, this reduces to parameter constraints. The line-level subspec requires the action to be `permit`, the prefix to cover all addresses in P, and the ge value to be at most 16, while the field-level subspec retains only the prefix constraint. □

5 Subspecification generation

We formulate the subspecification generation problem, present a two-phase algorithm that balances completeness and scalability, and establish key properties of the algorithm.

5.1 Problem Formulation and Challenges

We first revisit the challenge of deriving localized subspecification in its original definition in Sec. 4.

Challenge. Deriving an *iff* (logical equivalence) localized subspecification *w.r.t.* the global property φ is as hard as full network verification. A local configuration change can alter control-plane convergence and route propagation across the entire network, requiring global reasoning to determine whether φ still holds. Since

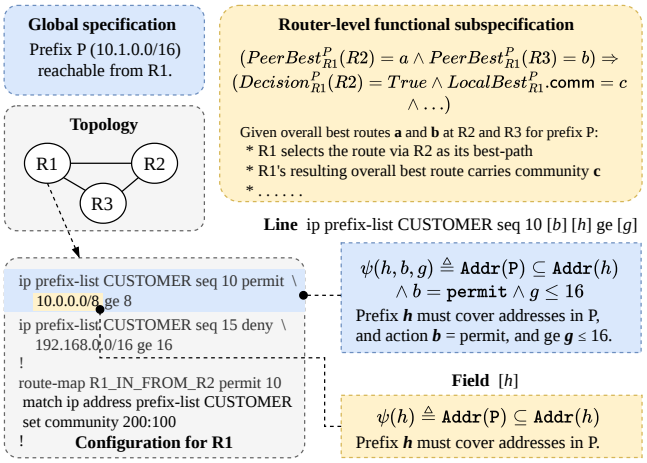


Figure 3: Subspecification at different granularities.

full verification is computationally expensive for large networks, repeating it independently for each of N configuration locations is prohibitively costly (see experimental comparisons in Sec. 7).

Example 2: [Update-induced global reasoning] Router R1 in Fig. 3 reaches prefix P via neighbor R2 (primary path) and has a backup path via R3. A route-map at R1 is modified to add a community filter that causes the route from R2 to be dropped. As a result, R1 selects the backup route via R3, changing the stable control-plane state (different next hop and AS-PATH), while end-to-end reachability to P still holds. Hence determining whether reachability still holds requires re-analyzing a new control-plane convergence. □

We therefore adopt a relaxed formulation that enables router-local decomposition and scalable generation.

Relaxed formulation. We relax the notion of subspecification based on the *observed stable control-plane state.*

Input. (1) A global network specification φ; and (2) a configuration C that has been verified to satisfy the property φ;

Output. A set of subspecifications: $\Psi = \{\psi(h) \mid h \in C\}$, where $h \in C$ means that h is a configuration location in C and each $\psi(h)$ characterizes the set of configuration values at location h that preserve the same stable state S(C), assuming all other configuration elements remain unchanged. Preserving S(C) in turn guarantees the configuration continues to satisfy the property φ.

Based on this relaxed formulation, we use a two-phase approach: first extract router-level functional subspecifications via control-plane simulation, then derive field-level subspecifications from these summaries through symbolic simplification.

5.2 High-level Functional Subspecification

For each router r and prefix P, we derive a router-level functional subspecification S^{P}_{r} from the simulated stable control-plane state:

$$S^{P}_{r} \triangleq PeerBest^{P}_{r} \Rightarrow \left(Decision^{P}_{r} \wedge LocalBest^{P}_{r}\right) \tag{2}$$

$PeerBest^{P}_{r}$ records the stable overall best routes at r's directly connected peers and forms the assumptions under which r is analyzed.

Input: router subspec S_r^P, config slice C_r, target location h.
Output: field-level subspecification $\psi(h)$.

1: **procedure** GenSubspec (S_r^P, C_r, h)
2: $\psi(h) \leftarrow \top$ ▷ initialize subspec
3: $v \leftarrow$ fresh symbolic variable for h ▷ symbolize h
4: **if** $\textsc{Encode}(C_r[h \leftarrow v]) \wedge \neg\textsc{StateEq}(S_r^P)$ is UNSAT **then**
5: $\psi(h) \leftarrow \top$ ▷ empty subspec
6: **else** ▷ non-empty subspec
7: $\text{seed}_r^P(v) \leftarrow \textsc{Encode}(C_r[h \leftarrow v]) \wedge \textsc{StateEq}(S_r^P)$
8: $\psi(h) \leftarrow \textsc{Simplify}(\text{seed}_r^P(v))$ ▷ simplify subspec
9: **if** v does not occur in $\psi(h)$ **then**
10: $\psi(h) \leftarrow (v = C_r[h])$ ▷ retain original config
11: **end if**
12: **end if**
13: **return** $\psi(h)$
14: **end procedure**

Figure 4: Field-level subspecification derivation.

$Decision_r^P$ constrains r to preserve its stable-state best-path selection outcome, while $LocalBest_r^P$ constrains the components of its resulting overall best route attributes required to preserve the local behavior and satisfy neighboring peer-route assumptions.

Thus, S_r^P is an assume–guarantee contract: given the stable-state overall best routes at neighboring routers, r must preserve its stable-state selection decision and the required route attributes of its resulting overall best route. We write $C_r \models S_r^P$ when the router-local slice C_r satisfies this contract.

Our decomposition uses overall best routes as interfaces between router-local slices. Each router is analyzed independently under the peer-route assumptions in $PeerBest_r^P$, without encoding the entire network. Its resulting overall best route in turn forms part of the assumptions under which neighboring routers are analyzed. If the guarantees of all router modules satisfy the corresponding peer-route assumptions, their local behaviors are mutually consistent and reproduce the original stable control-plane state.

Appendix A.1 details the slicing design, including non-transitive attributes, operational-state compatibility, multi-protocol support, shared locations, and BGP loop prevention.

Example 3: [Router-level functional subspecification] Consider a stable propagation path through R1, R2, and R3 for prefix P under a no-transit property. In the observed stable state, R1's overall best route carries community `NO_TRANSIT`. The export and import policies along the path preserve this community, and R3 matches the community to block further export.

Preserving R3's filtering decision requires R2's resulting overall best route to carry `NO_TRANSIT`. Because the policies between R1 and R2 preserve rather than introduce it, R1's resulting overall best route must carry it as well. Thus, S_{R1}^P includes the constraint

$$\texttt{NO\_TRANSIT} \in LocalBest_{R1}^P.\text{community}.$$

More generally, $LocalBest_r^P$ constrains all encoded attributes of the resulting overall best route to their values in the simulated stable state. Thus, S_r^P preserves both the best-path decision and the resulting route attributes, including auxiliary attributes such as the `MONITOR` community. □

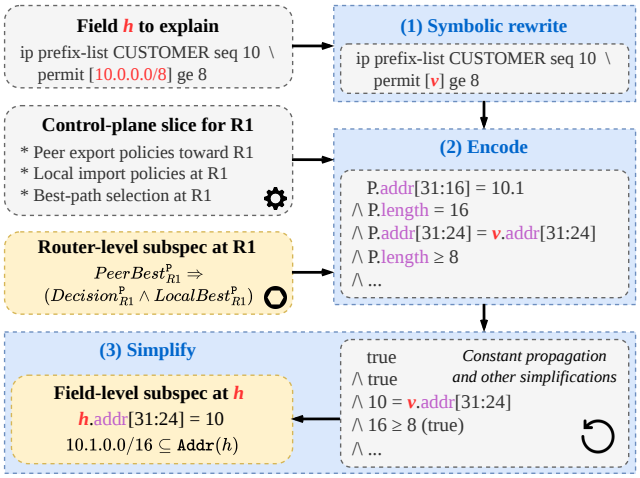


Figure 5: Example of field-level subspec derivation.

Proposition 1: *[Correctness of router-level functional subspecifications] Suppose (1) every router r satisfies S_r^P, and (2) for every pair of adjacent routers a and b, the best route selected by b satisfies a's assumption on the incoming route from b. Then the original stable control-plane state for* P *is preserved.* □

Proof sketch. The peer-route assumptions are instantiated with the routes from the original stable state. Condition (2) requires that each router's best route matches these assumed values, so every router sees the same peer inputs as in the original stable state. Under identical inputs, condition (1) guarantees each router reproduces its original best-path decision and outgoing route. The network state therefore equals the original stable state for P, which has been verified to satisfy the property. □

5.3 Low-level Parameter Constraints

We next derive field-level and line-level subspecifications that capture how individual parameters preserve the verified stable states.

Field-level subspecification problem. Given a router-level functional subspecification S_r^P defined in Eq. 2, and the corresponding router-local slice C_r, for each configuration location $h \in C_r$, we replace the target location h with a fresh symbolic variable v and define the *seed subspecification*:

$$\text{seed}_r^P(v) \triangleq \textsc{Encode}(C_r[h \leftarrow v]) \wedge \textsc{StateEq}(S_r^P) \quad (3)$$

The seed subspecification characterizes the values of v that preserve the router-level behavior. In the general case, we simplify it to derive field-level subspecification at location h:

$$\psi(h) \triangleq \textsc{Simplify}(seed_r^P(v)) \quad (4)$$

That is, an update at location h satisfies $\psi(h)$ *iff* the modified configuration preserves the router-level functional subspecification.

Algorithm. GenSubspec (Fig. 4) takes as input a router-level functional subspecification S_r^P, the corresponding router-local slice C_r, and a target location $h \in C_r$, and outputs a field-level subspecification for h. It first replaces h with a fresh symbolic variable v (line 3), and preliminarily checks whether every value of v preserves router-level subspec S_r^P (line 4). If so, it sets $\psi(h) \leftarrow \top$ (line 5), indicating

that the location is unconstrained (*i.e.,* an empty subspec). Otherwise, it constructs and simplifies the seed subspecification $\text{seed}_r^{\mathsf{P}}(v)$ using SMT-based equivalence-preserving tactics (lines 7–8). If v remains in the simplified formula, it directly uses $\textsc{Simplify}(\text{seed}_r^{\mathsf{P}}(v))$ as $\psi(h)$ (line 8). If simplification eliminates v, it instead constrains v to the location's original configuration value, $v = \mathsf{C}_r[h]$ (line 10). The resulting constraint is interpreted as the field-level subspecification $\psi(h)$, with v renamed to h for presentation.

Example 4: [Field-level subspec generation] Fig. 5 illustrates the derivation process of a field-level subspecification. Consider the prefix-list entry and a stable route for prefix $\mathsf{P} = \texttt{10.1.0.0/16}$.

```
ip prefix-list CUSTOMER seq 10 permit 10.0.0.0/8 ge 8
```

Let h denote the prefix field `10.0.0.0/8`. We replace h with a fresh symbolic variable v, and construct a seed subspecification $\text{seed}_r^{\mathsf{P}}$ defined in Eq. 3. Applying simplification (Fig. 5, bottom) removes redundant constraints and produces the field-level subspecification

$$\psi(h) \triangleq h.\text{addr}[31{:}24] = \texttt{10}.$$

This constraint precisely characterizes all updates to the prefix field that preserve the stable routing behavior. □

Irrelevant vs. constant field. SMT simplification may eliminate the symbolic variable v. We distinguish constant and irrelevant fields using the preliminary check in GenSubspec (Fig. 4, line 4). If every value of v preserves the router-level behavior, the field is irrelevant and its subspecification is $\psi(h) \leftarrow \top$ (line 5). Otherwise, if simplification eliminates v, we conservatively constrain it to its original value $\psi(h) \leftarrow (v = \mathsf{C}_r[h])$ (line 10).

Example 5: [Irrelevant vs. constant field] For a route-map line:

```
set local-preference 200
```

Suppose the router-level behavior is preserved for every local-preference value. The preliminary check in GenSubspec (Fig. 4) finds that no value can violate the router-level behavior, so the field is irrelevant and $\psi(h) = \top$. In contrast, if only the original value (*i.e.,* 200) preserves the router-level functional subspecification and simplification eliminates v, the algorithm constrains it to its original value, $\psi(h) \leftarrow (v = 200)$. □

Line-level and multi-location joint subspecifications. We preserve a line's structure while treating its fields symbolically, deriving a joint constraint over them. A line-level subspecification is thus a special case of a joint subspecification over multiple selected fields within one router-local slice C_r.

More generally, for multiple locations $\vec{h} = \langle h_1, \ldots, h_k \rangle$ within the same router-local slice C_r, we replace all locations simultaneously with fresh symbolic variables $\vec{v} = \langle v_1, \ldots, v_k \rangle$ and derive

$$\mathit{seed}_r^{\mathsf{P}}(\vec{v}) \triangleq \textsc{Encode}\big(\mathsf{C}_r[\vec{h} \leftarrow \vec{v}]\big) \wedge \textsc{StateEq}(\mathsf{S}_r^{\mathsf{P}}) \tag{5}$$

Simplifying this formula yields the joint subspecification

$$\psi(\vec{h}) \triangleq \textsc{Simplify}(\mathit{seed}_r^{\mathsf{P}}(\vec{v})) \tag{6}$$

It characterizes safe value combinations for locations and captures interactions among edits within the same router-local slice C_r.

Remarks. When locations span multiple router-local slices, we do not construct a network-wide joint encoding. Instead, for each affected slice C_r, we jointly symbolize the locations within that slice and derive a joint subspecification with respect to router-level functional subspecification $\mathsf{S}_r^{\mathsf{P}}$. Cross-slice interactions are constrained by these router-level functional subspecifications, and correctness follows by composition via Proposition 1.

Optimization: prefix-match scope analysis. To scale to large prefix-lists, we statically analyze, for a target prefix P, which rule l in the configuration it matches. Rules after l are irrelevant (empty subspec), while preceding rules only need to preserve that they do not override l's decision. This prunes semantically irrelevant lines: instead of generating subspecs for every field in the list, we only compute them for the matching rule and one of the preceding rules.

Example 6: [Prefix-list analysis] Consider a prefix-list:

```
ip prefix-list PL seq  5 deny   11.0.0.0/8 le 8
ip prefix-list PL seq 10 permit 12.0.0.0/8 ge 8
ip prefix-list PL seq 15 permit 10.0.0.0/8 ge 8 <- P
ip prefix-list PL seq 20 permit  0.0.0.0/0 ge 0
```

The target prefix $\mathsf{P} = 10.1.0.0/16$ matches rule l at seq 15. Rules after l (seq 20) are never reached and receive empty subspecs. Rules before l (seq 5, 10) share the constraint: they must not match P with conflicting action, so one representative subspec suffices. Only rule (seq 15) and first rule (seq 5) require a full subspec derivation. □

5.4 Properties

We establish correctness, characterize the completeness tradeoff, and analyze compositionality and performance.

Lemma 2: *[Local equivalence] For any router-local slice C_r, assume the router-local encoding $\textsc{Encode}(\mathsf{C}_r)$ in* GenSubspec *(Fig. 4) correctly captures router-local semantics*

$$\mathsf{C}_r \models \mathsf{S}_r^{\mathsf{P}} \iff \textsc{Encode}(\mathsf{C}_r) \wedge \textsc{StateEq}(\mathsf{S}_r^{\mathsf{P}}) \text{ is satisfiable} \tag{7}$$

For any router r, prefix P*, location $h \in \mathsf{C}_r$, and any replacement g at h, let $\psi(h)$ be the subspecification produced by* GenSubspec*. Then*

$$g \models \psi(h) \iff \mathsf{C}_r[h \leftarrow g] \models \mathsf{S}_r^{\mathsf{P}} \tag{8}$$

In other words, any edit at h satisfying its subspecification preserves the stable-state behavior of router r for prefix P*.* □

Proof sketch. By Eq. 3, the seed subspecification characterizes exactly the replacements of h that preserve the router-level subspecification $\mathsf{S}_r^{\mathsf{P}}$. Since Simplify preserves logical equivalence, the resulting $\psi(h)$ characterizes the same set of replacements. Therefore, Eq. 8 holds for any g. □

Proposition 3: *[Soundness] Let $\psi(h)$ be the subspec at location h and φ the verified global property. If $g \models \psi(h)$ and other config elements remain unchanged, then $C[h \leftarrow g] \models \varphi$.* □

Proof sketch. By Lemma 2, $g \models \psi(h)$ preserves the router-level stable behavior. By Proposition 1, router-level preservation implies global stable-state preservation. Since φ holds on that state, $\mathsf{C}[h \leftarrow g] \models \varphi$. □

Example 7: [Soundness] Suppose a property requires reachability to `10.1.0.0/16`. In the verified stable state, router r forwards traffic destined for this prefix to peer R2, as captured by the router-level

functional subspecification S_r^P. For a location h in a prefix-list at r, GenSubspec (Fig. 4) derives a subspecification $\psi(h)$ that preserves this forwarding decision, ensuring that traffic still exits r via R2.

If an update violates $\psi(h)$, router r may choose a different next hop, yet reachability to `10.1.0.0/16` can still hold via another path. Thus, subspecifications are *sound*: satisfying $\psi(h)$ preserves the verified behavior, while violating $\psi(h)$ does not necessarily violate the global specification φ. □

Proposition 4: *[Non-completeness] There may exist an update g such that $\neg(g \models \psi(h))$ yet $C[h \leftarrow g] \models \varphi$ still holds.* □

Proof sketch. Since $\psi(h)$ is derived to preserve a specific stable state S, it excludes alternative configurations that lead to a different but still correct stable state S′ satisfying φ. Therefore, $\psi(h)$ guarantees safety but not completeness. □

Proposition 5: *[Non-compositionality] Let h_1 and h_2 be two distinct configuration locations with corresponding subspecifications $\psi(h_1)$ and $\psi(h_2)$, derived independently with respect to the same router-level functional subspecification S_r^P.*

There is no general guarantee that simultaneous updates g_1 at h_1 and g_2 at h_2 satisfying $g_1 \models \psi(h_1)$ and $g_2 \models \psi(h_2)$ will jointly preserve S_r^P, nor the global specification. Such interactions can instead be captured by a joint subspecification (Sec. 5.3). □

Proof sketch. Each $\psi(h_i)$ assumes all other locations are fixed. Updating h_1 and h_2 together breaks these assumptions, so their combined effect may alter routing behavior. Thus, satisfying $\psi(h_1)$ and $\psi(h_2)$ individually does not guarantee preservation of S_r^P. □

Example 8: [Non-compositionality] Consider a prefix-list:

```
ip prefix-list PL seq  5 permit 10.0.0.0/8
ip prefix-list PL seq 10 permit 10.1.0.0/16
```

The target prefix `10.1.0.0/16` is accepted by either line. Individually, each line's subspec allows arbitrary changes, since the other line still matches the prefix. However, modifying both lines simultaneously can leave the prefix unmatched, changing the router-local behavior. A joint subspec captures this interaction and requires that at least one line continue to accept the target prefix. □

Multi-element changes are frequent in practice [21, 34]. To mitigate non-compositionality, the SpecLens UI warns operators when editing multiple locations within the same router configuration, and computes a *joint subspecification* for all edits as described above.

Normalization. Assuming the underlying solver's simplification tactics are equivalence-preserving and eliminate tautologies and implied conjuncts, we apply them to each seed constraint. Under this assumption, the resulting subspecification is in normalized form with no syntactic redundancy: one cannot remove or simplify any remaining conjunct without changing its logical meaning.

Example 9: [Normalization] Consider a prefix-list entry at location h with field-level subspecification $\psi(h) \triangleq (h.\text{addr}[31{:}24] = 10) \wedge (h.\text{addr}[31{:}16] = 10.1)$, constraining h to `10.0.0.0/8` and `10.1.0.0/16`, respectively. Since the second prefix is covered by the first, the first constraint is redundant and simplified away, yielding a normal form $\psi(h) \triangleq h.addr[31{:}16] = 10.1$. □

Group	#	Experience	Background
Grad. students	8	0.5–6.5 years	Networking research
Net. operators	15	1.5–11 years	ISPs / enterprise networks

Table 4: Participant demographics.

ID	Level	#Prefix	#Line	Task	Specification
1	Easy	8	117	Update	ECMP Reachability.
2				Cleanup	Block private IPs.
3	Hard	89	289	Cleanup	Reachability.
4				Update	No transit routes.

Table 5: User-study tasks and configuration scale.

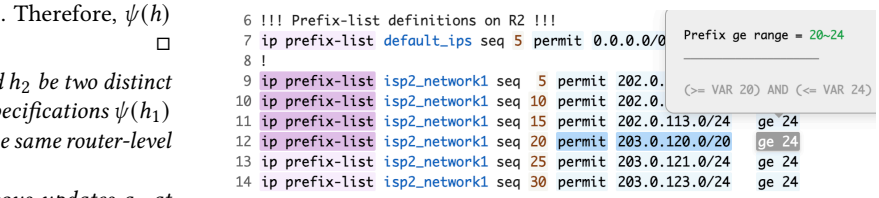


Figure 6: SpecLens UI with subspec shown as hover pop-up. Top: contextual interpretation; bottom: raw symbolic constraint. Here ge parameter field is constrained within [20, 24].

Remark. The properties above assume semantic consistency between the control-plane simulator (Batfish [16]) and the symbolic encoder (Minesweeper [5]). We discovered and resolved several modeling inconsistencies; details appear in Appendix A.2.

Parallel scalability. The overall subspecification generation procedure is parallelizable. Specifically: (1) router-level functional subspecifications are obtained via standard control-plane simulation, with scalability comparable to existing tools [16]; and (2) field-level subspecifications are derived independently per configuration location, enabling full parallelism; selected joint subspecifications are generated on demand. Empirical scalability results are in Sec. 7.

6 User Study

To evaluate SpecLens and the usefulness of localized subspecifications, we conducted a small user study [36] to answer the following:

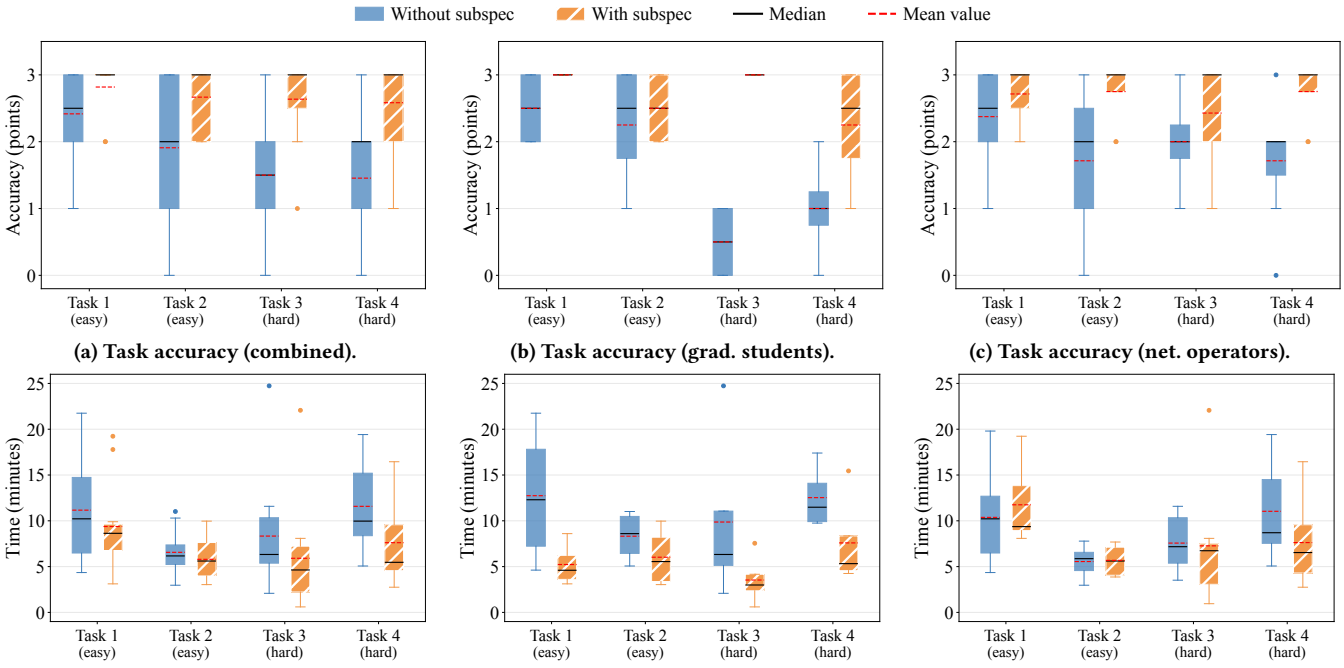
RQ1. Do localized subspecifications help users improve accuracy and efficiency in different network maintenance tasks?

RQ2. Do users find SpecLens and its subspecifications intuitive and easy to use?

6.1 Tasks and Study Structure

Participants. We recruited 15 professional network operators from ISPs and enterprise networks, and 8 graduate students in networking research, through industry and academic contacts. Participant demographics are summarized in Tab. 4.

Configuration and tasks. Tab. 5 summarizes four maintenance tasks and their configuration complexities. The configurations are designed to (1) reflect realistic operational challenges, while (2) remain solvable within the one-hour study budget. We prepare two complexity levels, each with one refactoring task and one update task.



(a) Task accuracy (combined). (b) Task accuracy (grad. students). (c) Task accuracy (net. operators).

(d) Task completion time (combined). (e) Task completion time (grad. students). (f) Task completion time (net. operators).

Figure 7: User-study task accuracy and completion time.

For each task, three candidate edits are provided, and participants select those that preserve the specified properties.

Properties are enforced using three common constructs: prefix-lists for filtering, community tags for classification, and AS-PATH length for ECMP. This follows the engineering practice of using a small set of protocol attributes to simplify maintenance, as confirmed by our industry participants.

User interface. As shown in Fig. 6, SpecLens presents localized subspecifications as hover pop-ups attached to configuration blocks. Each pop-up shows an interpreted constraint at the top, where symbolic variables are instantiated with their configuration meaning, and the raw form below for exact semantics. Lines or fields with non-empty subspecifications are highlighted in a darker color (*e.g.,* lines 9–12) to help users quickly identify semantically relevant elements. In this example, the target prefix matches line 12, while preceding lines are only required not to override line 12's decision.

Control and treatment groups. We employ a within-subject design to neutralize individual ability variation. Specifically, each participant performs four tasks: two under the *control* condition and two under the *treatment* condition, and each condition contains one easy task and one hard task. In the control condition, participants are given only the plain configuration; in the treatment condition, they additionally receive localized subspecifications. All other task information is identical across the two conditions.

To reflect realistic operational settings, participants in *both* groups are allowed to use *any* tool they typically use in practice, *e.g.,* IP prefix conversion utilities or LLM-based assistants for clarifying configuration syntax and semantics (see further discussion below). This ensures that any observed improvements stem from subspecification-based explanations rather than tool restrictions.

Metrics. We consider two objective metrics: (1) *accuracy*, whether an option is correct, and (2) *completion time*, time spent per task.

Workflow. The study begins with a short tutorial on using SpecLens, followed by several interactive warm-up questions to familiarize participants with localized subspecifications and the tasks. Participants then complete the tasks at their own pace. We record the accuracy and completion time for each user and task. After completing the tasks, participants fill out a System Usability Scale (SUS) questionnaire [9], a widely used 10-item Likert-scale instrument for measuring perceived usability and learnability of interactive systems. See Appendix B.1 for the full questionnaire.

6.2 RQ1. Accuracy and Time Improvement

Accuracy. As shown in Fig. 7a, (1) subspecs lead to more accurate configuration decisions overall, with the treatment group achieving an average 52% improvement over the control group; (2) gains are largest on complex tasks 3–4, where long prefix-lists and richer interactions make manual reasoning error-prone; (3) without subspec, accuracy decreases with task complexity, as expected; and (4) with subspec, accuracy remains stable, showing robust performance. The Mann–Whitney U tests confirm significant accuracy improvements on the harder Tasks 3 and 4 ($p = 0.0068$ and $p = 0.0046$, respectively), but not on Tasks 1 and 2 ($p = 0.11$ and $p = 0.06$).

Time. As shown in Fig. 7d, (1) participants with subspec are on average 1.3× faster than those without (corresponding to a 23% reduction in completion time); (2) the largest speedup appears in Task 4 (1.5×), due to its complexity; (3) participants spend more time on Task 1 for initial familiarization; (4) without subspec, completion time increases notably on harder tasks 3–4; and (5) with subspec,

time remains largely stable, with a slight increase on Task 4. The Mann–Whitney U tests find a significant time reduction only for Task 4 ($p = 0.0289$); the differences for Tasks 1–3 are not statistically significant ($p = 0.44$, $p = 0.55$, and $p = 0.17$, respectively).

Impact of experience. Comparing graduate students and network operators (Figs. 7b-7c, 7e-7f), (1) operators show smaller accuracy gains than students (39% vs. 164%), mainly due to a stronger baseline in the control group; and (2) operators did not save time on simple tasks due to a more careful review of the configurations regardless of subspec, while graduate students improved consistently.

Impact of complexity. Network operators show little improvement on simple tasks (Tasks 1–2) but benefit substantially on harder tasks (Tasks 3–4). Mann–Whitney U tests confirm this pattern: accuracy gains on Tasks 3 and 4 are significant ($p < 0.01$), while those on Tasks 1–2 are not. Since this is a small-scale study with simplified configurations, we expect the benefits to be larger in practice, where larger configurations impose greater cognitive load on operators.

Anecdotes. Several anecdotes reveal interesting insights.

Subspec usage patterns. We observe three frequent patterns. (1) Many first jump to lines with non-empty subspecs using the visual cues (Fig. 6), rather than scanning long prefix-lists as in the control group, which is time-consuming. (2) When encountering unfamiliar vendor syntax, they use subspec to clarify the effective semantics assumed by the verifier, *e.g.,* interpreting `le 32` as mask length ≤ 32, resolving confusion without external lookup. (3) Some rely directly on subspecifications to judge edits without reading the full configuration, while others use them as a validator: when an intuitive judgment conflicts with a subspec, they revisit the relevant lines and often uncover overlooked dependencies.

Common cross-block errors. In the control group, errors often arose from large configurations and missed cross-element dependencies. For example, Task 3 includes long prefix-lists that require mentally tracing which entry a target IP matches; misjudgments here led to incorrect route classification and flawed downstream reasoning. With subspecifications, the matched and downstream-relevant lines are highlighted by non-empty subspecifications, directing attention to them more directly.

Benefits beyond existing tools. Even with IP prefix utilities or AI-based assistants, subspecs still led to measurable improvements. In our study, 7 participants used AI tools including Gemini [19], DeepSeek [13], and Doubao [11], supplying the full configuration, and when available, the corresponding subspecs as their queries. Without subspecifications, AI assistants sometimes returned incorrect answers for the same tasks, whereas with subspecifications as additional context, the answers became correct. This indicates that subspecifications provide localized semantic constraints that are difficult to infer from configuration syntax alone. Detailed AI-assistant usage is reported in Appendix B.3.

Practitioners are more cautious. With subspecification, network operators spent 1.4× more time reviewing the configuration on average, compared to graduate students. In post-study interviews, several of them noted that they preferred to fully understand the configuration before making any critical update, due to the high-stakes nature of network operations and habits reinforced through training and practice. Even so, subspecifications served as quick validators of their intuition, especially for unfamiliar syntax, thereby reducing the completion time.

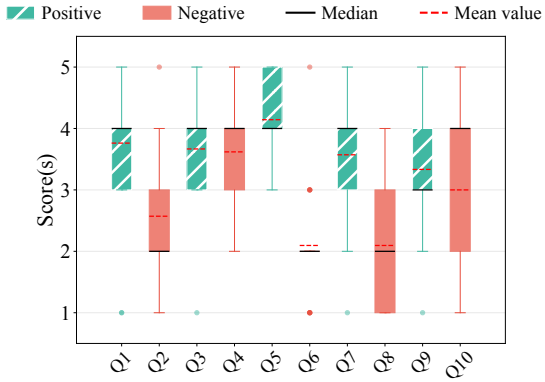


Figure 8: System usability scale (combined).

6.3 RQ2. Usability Assessment

SUS scores. We use the System Usability Scale (SUS) to assess participants' perceived usability of SpecLens; its definition and questionnaire are provided in Appendix B.1.

As shown in Fig. 8, based on 23 responses: (a) 70% (16) would use SpecLens in daily practice, while 9% (2) would not. (b) 52% (12) find it easy to use, whereas only 4% (1) find it difficult. (c) Views on support are split: 52% (12) expect to need technical support, vs. 48% (11) who do not. These responses show that subspecs provide practical explanatory value beyond task-performance gains. For space reasons, we summarize the main findings here and report the detailed results and analysis in Appendix B.2.

The system achieved an average SUS score of 62, below the commonly reported benchmark of 68. Two factors likely contributed to this result. First, subspecification introduces a new operational paradigm that requires initial learning and adaptation. Nevertheless, several participants noted that the tool could become an integral part of their workflow once they became familiar with it and developed sufficient trust in its results. Second, experienced operators were cautious about automatically generated subspecifications, often evaluating them against established operational practices and placing greater confidence in their own experience and judgment.

Qualitative feedback. Our post-study survey finds:

Positive themes. (1) Several participants saw explainability as a real workflow challenge and viewed subspecs as a promising way to understand configurations; (2) many said subspecs make it easy to sanity-check small updates without rereading everything; (3) the visual design, especially highlighting non-empty subspecs (Fig. 6), was often described as intuitive.

Limitations and concerns. (1) Some operators were concerned about limited support for vendor-specific syntax. (2) Several expected more compositional explanations; for example, when a line has an empty subspec due to redundancy, they wanted to know which other line causes it. (3) The notion of subspecification was not immediately clear to a few participants, with some initially seeing it as a generic code explainer, though most understood it after the warm-up task.

These concerns suggest engineering and research directions. Broader vendor coverage and clearer onboarding can be addressed through better modeling and UI design. Compositional explanations and richer dependency tracing instead motivate research on relating subspecifications and exposing cross-location interactions.

Summary. The user-study results show that subspecifications improve task accuracy by 52% and reduce completion time by 23% on average. The gains are larger on complex tasks and among graduate students than among network operators. Moreover, 70% of participants would use a subspec-based explainer in practice.

7 Evaluation

We aim to answer the following research questions:

RQ3. Can subspecification be generated efficiently?

RQ4. Does it scale with network size and configuration complexity?

Implementation and baselines. We implemented our explanation system SpecLens with a subspecification-generation backend, denoted SubSpec in this evaluation, in 7769 lines of Python and extended Batfish [16] and Minesweeper [5] in 3419 lines of Java. We compare it against two baselines: (1) FullSym, which directly simplifies full network encoding to obtain localized subspecifications, without router decomposition; and (2) $\text{SubSpec}_{\text{NS}}$, a variant of SubSpec without prefix-match scope analysis that analyzes all configuration fields. Our current implementation supports eBGP, iBGP, OSPF, static routes, and connected routes. Extending to ACL and firewall rules is straightforward given their sequential-match semantics. MPLS/SR policies are left as future work.

Configurations. To evaluate RQ3, we use real-world Internet2 configurations [1], and three synthetic configurations (Bics, Columbus, and USCarrier) from Config2Spec [24], which are generated from real-world network topologies [2]. Tab. 6 summarizes the configuration characteristics. For each configuration, we randomly select ten prefixes and use their reachability as the target specifications.

To evaluate RQ4, we use large-scale synthetic FatTree configurations from NV [18] and our own synthesized configurations, varying the number of configuration lines (prefixes) on a single device to assess worst-case per-device complexity.

Setup. For each configuration and target prefix, we first simulate the stable state with Batfish [16] and cross-check it using Minesweeper's symbolic encoding [5] to calibrate modeling accuracy (discussed in Appendix A.2). We then generate subspecifications for all configuration locations (fields and lines) using SubSpec and the two baselines. Figs. 9-10 and Appendix C report end-to-end offline precomputation, including line-level joint and field-level subspecification generation. We set a 4-hour timeout. Experiments ran on a 10-core (20-thread) 3.7 GHz machine with 256 GB of memory.

Results. We summarize the main findings below.

RQ3: Efficiency. As shown in Fig. 9, (1) on Internet2, SubSpec takes 10 minutes on average (7 ms per location), showing practical effectiveness; (2) on three synthetic configs, SubSpec takes

[1]We remove the IGP configurations unsupported by Minesweeper and add multiple external devices to establish external BGP sessions and originate routes.
[2]OSPF-only configurations are omitted because they contain no routing policies.

Network	Type	#Routers	#Locations	Avg. %Subspec
Internet2	real-world	10	89586	1.8%
Bics	synthetic	49	1130	8.3%
Columbus	synthetic	86	1932	9.9%
USCarrier	synthetic	174	2470	11.2%

Table 6: Dataset characteristics.

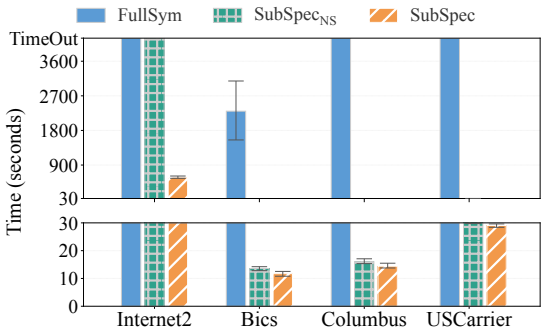


Figure 9: Average runtime over ten target prefixes.

12-30 seconds on average (11 ms per location); and (3) across all configurations, SubSpec consistently outperforms two baselines. FullSym times out on all but Bics; on Bics, SubSpec is 1.2× faster than $\text{SubSpec}_{\text{NS}}$ and 193× faster than FullSym.

Ratio of non-empty subspecification. As shown in Tab. 6, only 2–11% of configuration locations have non-empty subspecifications. This low subspec-to-location ratio means only a small fraction of the configuration requires attention, indicating that subspecifications help reduce cognitive load.

Optimization correctness. For all configuration locations, SubSpec and $\text{SubSpec}_{\text{NS}}$ produce equivalent subspecifications, validating the correctness of prefix-scope optimization.

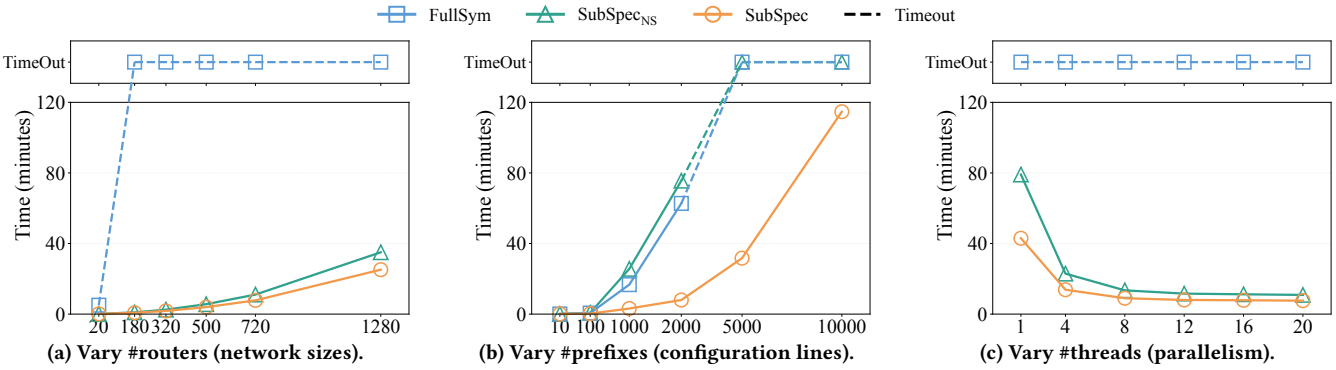
RQ4: Scalability. We assess the scalability of SubSpec by varying the number of routers, prefixes, and threads.

Vary #routers (network sizes). As shown in Fig. 10a, when the number of network devices increases from 20 to 1,280, (1) the runtime of SubSpec increases from 6 seconds to 25 minutes; (2) FullSym times out beyond 20 devices; and (3) SubSpec is 1.3× faster than $\text{SubSpec}_{\text{NS}}$ on average. A detailed runtime breakdown is provided in Tab. 8 in Appendix C.1.

Vary #prefixes (configuration lines). Fig. 10b shows that as prefixes grow from 10 to 10,000, SubSpec runs in 4 seconds to 115 minutes, while $\text{SubSpec}_{\text{NS}}$ and FullSym time out beyond 5,000, highlighting the benefit of prefix-scope analysis.

Vary #threads (Parallelism). As shown in Fig. 10c, on FatTree (720 routers), when the number of threads increases from 1 to 20, (1) SubSpec and $\text{SubSpec}_{\text{NS}}$ are 6× and 7× faster, respectively; (2) the main bottleneck is preprocessing. Subspecification generation scales well with parallelism, achieving an 8× speedup with 8 threads and an 11× speedup with 20 threads over single-threaded execution.

Summary. Our evaluation shows that SubSpec is efficient (averaging 10 minutes on Internet2), scales to large networks (25 minutes on FatTree with 1,280 routers), handles router-local slices with up to 10,000 configuration lines (115 minutes), and scales well in parallel (8× faster with 8 threads than with a single thread).



(a) Vary #routers (network sizes). (b) Vary #prefixes (configuration lines). (c) Vary #threads (parallelism).

Figure 10: Scalability of subspecification generation under different dimensions.

8 Related work

A workshop paper [23] introduced localized subspecifications for explaining synthesized network configurations. In contrast, we provide (1) a formal subspecification formulation, (2) an efficient generation algorithm with a soundness guarantee, and (3) a user study and large-scale evaluation.

Network verification. Minesweeper [5] introduced symbolic encodings for control-plane verification, offering generality but facing scalability challenges. Subsequent systems such as Plankton [25] and Tiramisu [2] improve scalability via control-plane simulation, model checking, and domain-specific optimizations. Lightyear [28] and Timepiece [3] further scale through modular decomposition.

Our work builds on these verification encodings and simulation techniques, but uses them to generate explanations. Modular verification [3, 28] decomposes global reasoning by summarizing behavior over *all* possible inputs, typically requiring manually specified module interfaces. In contrast, localized subspecifications fix the observed stable state and surrounding configuration, enabling automatic generation of local constraints.

Network synthesis and repair. Propane [6, 7], NetComplete [15], and subsequent synthesis frameworks [1, 14, 26] automatically generate or repair configurations from high-level intents. These approaches typically operate in a *black-box* manner, offering little insight into why each configuration element is generated as is.

Differential analysis. Prior work analyzes network differences upon configuration updates. DNA [34] detects property changes via data-plane simulation. Rela [31] checks that intended path or flow changes occur without affecting others. These approaches validate *end-to-end behavioral differences*, whereas our work explains a configuration at a *fine-grained, per-element* level, with explanations expressed as the set of admissible alternative configurations.

Campion [29] also adopts localization to identify deviating subcomponents between two configurations, *e.g.,* router backups or vendor changes. We instead explain a single configuration against its stable state, complementing such analyses by clarifying deviating components and providing semantic guidance for fixes.

Intent inference. Config2Spec [8] and Anime [20] infer *global* specifications from configurations. In contrast, we derive fine-grained *localized subspecifications* and compute them efficiently via router-local decomposition.

Network debugging. NetCov [30] finds config lines exercised by tests. Network provenance [12, 37, 38] explains *why* behavior occurs by tracing derivations (positive explanations). In contrast, localized subspecifications give *normative, counterfactual* explanations: what each configuration element must satisfy to preserve a property, supporting safe edits.

CEL [17] targets *failed* configurations, using MCS enumeration to locate statements that cause a violation (*where* the error is), but not *what* they should satisfy. Localized subspecification can be extended to cover this case by generating repair constraints at CEL-identified locations, using a reference forwarding graph [32] as the decomposition basis. We leave this as future work.

Explainability for programs. Explainable program analysis and synthesis aim to make automated reasoning more interpretable. Interpretable Program Synthesis [35] exposes internal synthesis steps, while S^3 [22] localizes specifications to program structures. Our work shares the *localization principle*, but handles distributed protocol convergence and *cross-device dependencies* absent in single-program settings.

9 Conclusion

We presented SpecLens, an explainable network verification system based on localized subspecifications, which precisely characterize the local conditions that preserve the verified stable state and thereby guarantee the global property. Using router-local decomposition and localized symbolic simplification, subspecifications can be generated efficiently and significantly improve maintenance accuracy and efficiency in a user study. Localization and decomposition involve tradeoffs: localized subspecifications are not compositional, and explain configuration elements with respect to the converged stable state, rather than all alternative or transient states. This focus enables scalability and clear explanations that are difficult to obtain with global reasoning.

Acknowledgments

We sincerely thank our anonymous shepherd and reviewers for their valuable feedback on this paper, and all participants from industry and academia who took part in our user study and pilot testing. This work is supported by the ShanghaiTech Startup Fund, the US National Science Foundation under grant CCF-2146518, and the National Natural Science Foundation of China (No. 62272382).

References

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

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

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

[4] Mina Tahmasbi Arashloo, Ryan Beckett, and Rachit Agarwal. 2023. Formal methods for network performance analysis. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 645–661.

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

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

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

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

[9] John Brooke. 1996. SUS: a "quick and dirty" usability scale. In P. W. Jordan, B. Thomas, B. A. Weerdmeester and A. L. McClelland (eds.), *Usability Evaluation in Industry*. London: Taylor & Francis, 1996. https://en.wikipedia.org/wiki/System_usability_scale Retrieved from Wikipedia page "System usability scale" on 3 Sep 2025.

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

[11] ByteDance. 2025. Doubao. https://www.doubao.com.

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

[13] DeepSeek. 2025. DeepSeek. https://chat.deepseek.com.

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

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

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

[17] Aaron Gember-Jacobson, Ruchit Shrestha, and Xiaolin Sun. 2022. Localizing router configuration errors using minimal correction sets. *arXiv preprint arXiv:2204.10785* (2022).

[18] Nick Giannarakis, Devon Loehr, Ryan Beckett, and David Walker. 2020. NV: An intermediate language for verification of network control planes. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 958–973.

[19] Google. 2025. Google Gemini. https://gemini.google.com.

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

[21] Xu Liu, Peng Zhang, Anubhavnidhi Abhashkumar, Jiawei Chen, and Weirong Jiang. 2024. Automatic Configuration Repair. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks*. 213–220.

[22] Amirmohammad Nazari, Yifei Huang, Roopsha Samanta, Arjun Radhakrishna, and Mukund Raghothaman. 2023. Explainable Program Synthesis by Localizing Specifications. *Proceedings of the ACM on Programming Languages* 7, OOPSLA2 (2023), 2171–2195.

[23] Amirmohammad Nazari, Yongzheng Zhang, Mukund Raghothaman, and Haoxian Chen. 2024. Localized Explanations for Automatically Synthesized Network Configurations. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks*. 52–59.

[24] ETH Zürich Networked Systems Group. [n. d.]. Config2Spec: Mining Network Specifications from Network Configurations. https://github.com/nsg-ethz/config2spec. Accessed: 2026-01-25.

[25] Santhosh Prabhu, Kuan Yen Chou, Ali Kheradmand, Brighten Godfrey, and Matthew Caesar. 2020. Plankton: Scalable network configuration verification through model checking. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 953–967.

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

[27] Roland Schmid, Tibor Schneider, Georgia Fragkouli, and Laurent Vanbever. 2025. Transient Forwarding Anomalies and How to Find Them. *Proceedings of the ACM on Networking* 3, CoNEXT2 (2025), 1–23.

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

[29] Alan Tang, Siva Kesava Reddy Kakarla, Ryan Beckett, Ennan Zhai, Matt Brown, Todd Millstein, Yuval Tamir, and George Varghese. 2021. Campion: debugging router configuration differences. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 748–761.

[30] Xieyang Xu, Weixin Deng, Ryan Beckett, Ratul Mahajan, and David Walker. 2023. Test Coverage for Network Configurations. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1717–1732.

[31] Xieyang Xu, Yifei Yuan, Zachary Kincaid, Arvind Krishnamurthy, Ratul Mahajan, David Walker, and Ennan Zhai. 2024. Relational Network Verification. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 213–227.

[32] Rulan Yang, Gao Han, Hanyang Shao, Xiaoqiang Zheng, Xing Fang, Ziyi Wang, Lizhao You, Ruiting Zhou, Linghe Kong, Ennan Zhai, et al. 2026. Diagnosing and repairing distributed routing configurations using selective symbolic simulation. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*. 1635–1652.

[33] Fangdan Ye, Da Yu, Ennan Zhai, Hongqiang Harry Liu, Bingchuan Tian, Qiaobo Ye, Chunsheng Wang, Xin Wu, Tianchen Guo, Cheng Jin, et al. 2020. 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*. 599–614.

[34] Peng Zhang, Aaron Gember-Jacobson, Yueshang Zuo, Yuhao Huang, Xu Liu, and Hao Li. 2022. Differential network analysis. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 601–615.

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

[36] Yongzheng Zhang, Yaxuan Lin, Haoxian Chen, Ruize Ma, Amirmohammad Nazari, Mukund Raghothaman, and Peng Zhang. 2026. SpecLens: User-study interface and open-source code. https://declarative-systems-lab.github.io/SpecLens.

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

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

Appendices are supporting material that has not been peer-reviewed.

A Implementation details

A.1 Router-local Configuration Slice

For each router r in the network topology, we construct a router-local slice C_r. It includes:

- the export routing policies to r at all directly connected peers;
- all import routing policies at r;
- the local best-path selection process at r.

This slicing scheme can be viewed as a set of interlocking gears ⚙. The protruding teeth of each gear correspond to the export routing policies at directly connected peers toward r, which are included in C_r. The center of the gear contains the import routing policies at r and the local best-path selection process. The complementary gaps correspond to the export routing policies at r toward its neighbors; these policies are not part of C_r, but instead belong to the neighboring router-local slices. Thus, adjacent slices interlock through their export-policy boundaries, while each slice independently captures local import processing and best-path selection.

The resulting best-route-based slicing design has five properties.

Non-transitive attributes. Starting from peers' overall best routes, each slice applies peer export policies and r's import policies before best-path selection. In eBGP, this captures how non-transitive attributes (*e.g.,* local preference and MED) affect route selection without fixing intermediate values, avoiding an overly restrictive router-level functional subspecification.

Operational-state compatibility. The only external input required is each router's overall best route in the stable state, which is readily available from control-plane simulation or operational state, without modifying existing tools. This abstraction aligns well with simulated and real deployments.

Multi-protocol support. Using overall best routes as slice interfaces makes the encoding independent of how routes are produced. Routes learned through BGP, OSPF, or static routing are resolved by the local route-selection process before the resulting overall best route is used for subspec generation. The slice therefore supports routers running multiple routing protocols simultaneously.

Shared locations across slices. Because each router-local slice includes neighboring export policies toward the local router, a configuration location may appear in multiple slices, *e.g.,* when a shared route-map or prefix-list is referenced by several policies. We derive its subspecification in each containing slice and require updates to satisfy all of them. This preserves all affected router-level behaviors; correctness follows by composition via Proposition 1.

Loop prevention. While BGP route loop prevention is already enforced by the underlying symbolic encoding, which models AS-path–based semantics, when we construct router-level functional subspecs, we summarize the stable state using only best routes and abstract route-propagation relationships. If this summary is used naively, it may admit combinations of routes that are individually valid but collectively inconsistent with loop-prevention semantics.

To preserve consistency with the stable state, we also record, for each best route, the neighbor from which it is learned (*i.e.,* the route-propagation direction). Fig. 11 illustrates this: R3 's best route is learned from R2; for R2 router-level functional subspec, we enable control-forwarding from R3 to R2 (overriding a false value if necessary). This prevents R2 from re-accepting a route that would contain its own AS in the AS-path and thus avoids a routing loop.

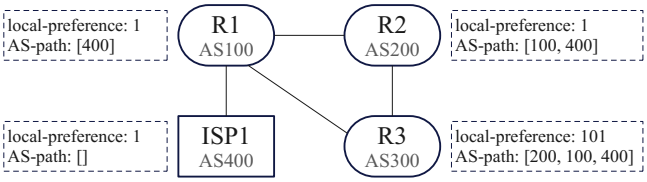


Figure 11: Example of BGP routing loop prevention.

We therefore do not re-implement loop prevention; rather, we ensure that the router-level abstraction respects the protocol semantics already captured by the verifier, keeping subspecification generation aligned with the original stable routing behavior.

A.2 Aligning Simulation and Encoding

During implementation, we identified and resolved several modeling inconsistencies between the Batfish simulator [16] (version: commit `6128b04`, March 2021) and the Minesweeper verifier [5] (version: commit `6128b04`, March 2021).

We cross-validated the two models as follows. For each router and prefix, we take stable best routes computed by Batfish and constrain Minesweeper's symbolic encoding with these routes. If these constraints are satisfiable, we treat the two tools as behaviorally consistent for that scenario and proceed with subspec generation.

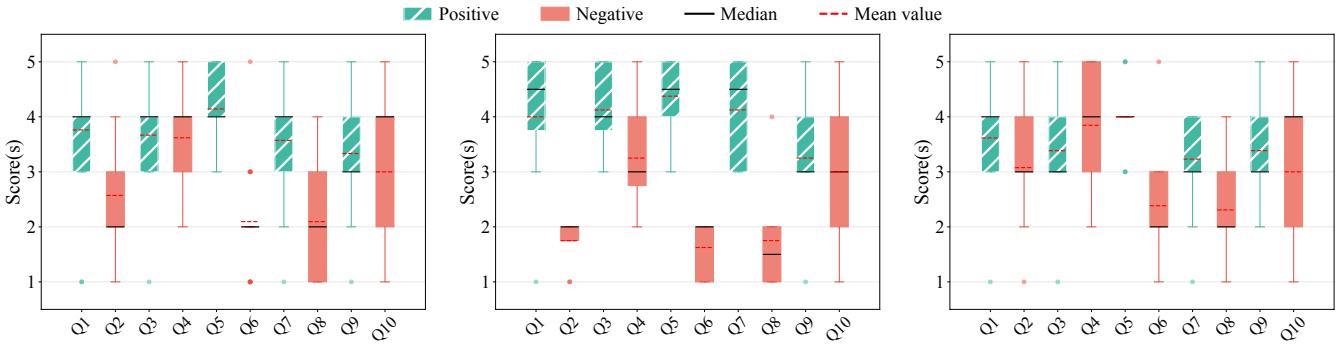
Using this approach, we identified the following inconsistencies.

BGP best route selection: AS-path comparison. Batfish performs AS-path string comparison as a final step among routes with identical key attributes (local preference, AS-path length, MED, etc.). Minesweeper's best-route selection process does not include this step. To align the behaviors of Batfish and Minesweeper, we enabled multipath-relax in all Cisco configurations, effectively disabling AS-path string comparison in Batfish. With this adjustment, both tools produce consistent best-route selection results.

BGP best route selection: tiebreaker. When multiple equally preferred candidate routes are available, and ECMP is disabled, Batfish breaks tie by earliest route arrival times, then by lowest BGP router ID. In contrast, Minesweeper directly selects the route with the lowest router ID. To align the two tools, we disabled the arrival-time comparison in Batfish.

BGP best route selection: withdraw and resend. In BGP, when a neighbor's best route changes, it may first withdraw the old route and then advertise a new one, temporarily replacing routes with the same next hop. We identified an issue in Batfish: when receiving a new route with the same next hop and similar priority as the current best route, Batfish retained the old route and ignored the new one. If the old route was later withdrawn, the router might fail to install the new route and temporarily lose valid routing information. To address this issue, we modified the route selection logic to always prefer a new route from the same next hop, regardless of priority. This ensures correct route installation during withdraw-and-resend.

BGP routing policies: set community. In BGP, communities are optional transitive attributes used to tag routes and influence



(a) System usability scale (combined). (b) System usability scale (grad. students). (c) System usability scale (net. operators).

Figure 12: User study system usability scale (SUS).

routing decisions across autonomous systems. The `set community` command assigns a set of community values to a route and removes all attached communities unless otherwise specified. We identified an issue in Minesweeper: for configurations using `set community`, Minesweeper did not clear existing communities. We corrected this issue to ensure that all non-selected communities are removed.

BGP route transfer process: non-transitive attributes. In eBGP, modifications to non-transitive attributes made by peer export policies and local import routing policies (e.g., local preference and MED) only affect local best-path selection. These attributes are reset to their default values before routes enter export routing policies. We identified an issue in Minesweeper: it did not reset such attributes before route export in eBGP. We corrected this issue to ensure that these attributes are properly reset during route transfer.

B User-study details

B.1 System Usability Scale (SUS) Questionnaire

We include the standard System Usability Scale (SUS) questionnaire [9] used in our user study. Participants respond on a 5-point Likert scale (1 = Strongly disagree, 5 = Strongly agree).

(1) I think that I would like to use this system frequently.
(2) I found the system unnecessarily complex.
(3) I thought the system was easy to use.
(4) I think that I would need the support of a technical person to use this system.
(5) I found the various functions in this system were well integrated.
(6) I thought there was too much inconsistency in this system.
(7) I would imagine that most people would learn to use this system very quickly.
(8) I found the system very cumbersome to use.
(9) I felt very confident using the system.
(10) I needed to learn a lot of things before I could get going with this system.

Following the standard SUS scoring procedure [9], for positive items (question 1, 3, 5, 7, and 9), we subtract 1 from each response, and for negative items (question 2, 4, 6, 8, and 10), we subtract each response from 5. The adjusted scores are summed and multiplied by 2.5, yielding a final score ranging from 0 to 100.

Part.	#W subspec	#W/O subspec	Usage type
Grad. 1	0	2	clarification, verification, task solving
Grad. 2	0	1	clarification
Grad. 3	0	2	clarification
Grad. 4	1	2	clarification, task solving
Oper. 1	0	1	clarification
Oper. 2	0	1	clarification, verification
Oper. 3	2	2	verification, task solving

Table 7: Observed AI-assistant usage conditions and types by participant. The two middle columns report the number of tasks in which each participant used an AI assistant.

B.2 System Usability Scale (SUS) Results

Fig. 12a shows the per-question System Usability Scale (SUS) distribution and the breakdown by graduate students and network operators (Figs. 12b-12c). Overall, participants gave a generally favorable usability assessment, suggesting that subspecification annotations fit into their workflow without substantial cognitive overhead. The system achieved an average SUS score of 62 with lower ratings on perceived usage barriers (Q4) and learning effort (Q10) contributing most to the overall score.

We next show detailed analysis for each question group.

Adoption intention and perceived complexity (Q1–Q2). Most participants (6/8 graduate students and 10/15 network operators) reported that they would be willing to use our system for tasks such as network configuration changes. Only two would not use it, citing the subspec hints as overly complex. Complexity perceptions differed between the groups. No graduate students viewed the system as complex, whereas only 4/15 network operators considered it not complex, suggesting that frontline operators may perceive subspec hints as requiring effort to understand or adding information overhead. This difference likely reflects the conservative, risk-aware nature of operational work and helps explain why adoption intention remains strong despite the moderate SUS score.

Ease of use and support needs (Q3–Q4). Among graduate students, 6/8 found the system easy to use, compared with 6/15 network operators, indicating differences in intuitive understanding and operational proficiency. Q4 supports this: 9/15 network operators reported needing technical support, versus 3/8 graduate students.

FatTree	Shared Preprocessing Stages			SubSpec Subspec Gen.		$\text{SubSpec}_{\text{NS}}$ Subspec Gen.	
	Router-level subspec	Router-local slice	Consistency check	Line-level	Field-level	Line-level	Field-level
FT-4 (20)	3.85	0.60	0.16	0.60	0.29	0.59	0.68
FT-12 (180)	10.82	14.13	1.49	15.38	2.66	15.24	17.66
FT-16 (320)	22.77	34.61	3.01	42.30	5.26	42.00	48.46
FT-20 (500)	54.35	71.92	5.58	95.94	9.06	95.67	108.95
FT-24 (720)	122.51	132.32	9.56	187.66	14.73	187.27	211.88
FT-32 (1280)	535.95	353.77	26.92	559.43	33.95	558.16	626.72

Table 8: Runtime breakdown for the FatTree evaluation (Fig. 10a). The first three preprocessing stages are shared by both SubSpec and $\text{SubSpec}_{\text{NS}}$. The remaining columns report only the line- and field-level subspecification generation time for each algorithm. All experiments use 20 parallel threads, and all times are in seconds.

These results suggest that the main barrier may lie in understanding the subspec concept and its application in operational workflows.

Integration and consistency (Q5–Q6). Responses in this dimension were the most stable and positive. 18/23 participants agreed that the system's functions were well integrated, and Q6 responses were largely positive, with only one participant reporting inconsistency. These results indicate broad recognition of the system's coherence and consistency of information presentation.

Learnability and perceived effort (Q7–Q8). Graduate students reported higher learnability than network operators (5/8 vs. 7/15), suggesting conceptual understanding, rather than interaction burden, is the primary learning challenge. Consistently, 7/8 graduate students found the system not cumbersome, compared with 9/15 network operators, indicating that conceptual overhead disproportionately affects network operators.

User confidence and learning cost (Q9–Q10). Confidence levels were similar between the groups (3/8 graduate students, 6/15 network operators). Both groups reported that learning was required (4/8 graduate students, 8/15 network operators), with network operators reporting a slightly higher learning burden, consistent with Q4, reflecting the cost of understanding the subspecs.

Summary. The main strengths of the system lie in well-integrated functions, a consistent interface, and a low operational burden. Limitations in overall usability are primarily due to the learning cost introduced by the subspec concept and network operators' heightened sensitivity to complexity and support needs.

In particular, network operators' cautious evaluations likely reflect production risk constraints: configuration changes carry high failure costs, and operational practice emphasizes explainability, controllability, and rollback, making them more conservative toward adopting new concepts and workflows. These findings explain why, despite moderately positive SUS scores, a relatively high proportion of participants remain willing to use the system in practice.

B.3 AI-assistant Usage

Participants were allowed to use tools that they normally rely on in practice, including LLM-based assistants. Among the seven AI users, two used Gemini [19], one DeepSeek [13], three Doubao [11], and one an unidentified assistant. We manually categorized the recorded AI interactions into three types for transparency.

Clarification. The participant asked the assistant to explain or help interpret configuration syntax, command semantics, prefix matching, subspecifications, network behavior, or the IP range affected by a modification.

Verification. The participant first formed an initial judgment and then consulted the assistant to confirm or challenge that judgment.

Task solving. The participant provided substantial task context, such as the topology, configuration, target property, subspecifications, or candidate modifications, and asked the assistant to determine the answer.

As summarized in Tab. 7, 7/23 participants used an AI assistant in at least one task, including 4/8 graduate students and 3/15 network operators. AI use was concentrated in tasks without subspecifications: all 7 AI users consulted an assistant in at least one such task, compared with only 2 in tasks with subspecifications. Most used AI for clarification, while a smaller subset used it to verify an existing judgment or directly solve a task. This pattern does not suggest that AI assistance systematically favored the subspecification condition.

C Evaluation details

C.1 Runtime Breakdown for Router Scalability

Table 8 decomposes the runtime of SubSpec and $\text{SubSpec}_{\text{NS}}$ into three shared preprocessing stages and the subsequent line-level and field-level subspecification generation. *Router-level subspec* includes control-plane simulation using Batfish and the extraction of routing-table entries into router-level subspecifications. *Router-local slice* measures the time required to slice the monolithic SMT encoding generated by Minesweeper into router-local encodings. *Consistency check* verifies that each router-level subspecification is consistent with its corresponding router-local configuration slice. These three stages are shared by both SubSpec and $\text{SubSpec}_{\text{NS}}$ algorithms.

SubSpec further applies several optimizations during low-level subspecification generation. In particular, all field-level subspecifications are derived from the corresponding line-level subspecification, rather than independently reconstructed from the router-local control-plane encoding. Consequently, its field-level generation time is substantially lower than that of $\text{SubSpec}_{\text{NS}}$, especially on larger FatTree instances.

This breakdown also reveals the limit to parallel scaling: router-level subspecification generation and router-local slicing are currently sequential, while line- and field-level subspecification generation can be parallelized across configuration locations. Therefore, as shown in Fig. 10c, increasing the number of worker threads mainly reduces the latter costs, while the two sequential preprocessing stages increasingly dominate the end-to-end runtime.