TopoIntent: Compiling Security Intent into Executable, Compliance-Checked Network Topologies

Xiaokang Qu*, Jianliang Ma†, Zao Fan†, Tianshu Chu†, Tianlong Fan*, Linyuan Lü*
* *School of Cyber Science and Technology, University of Science and Technology of China*
Hefei, China
xkqu@mail.ustc.edu.cn, tianlong.fan@ustc.edu.cn, linyuan.lv@ustc.edu.cn
† *Topsec Technologies Group Inc.*
Beijing, China
ma_jianliang@topsec.com.cn, fan_zao@topsec.com.cn, chu_tianshu@topsec.com.cn

***Abstract*—Enterprise security topology design begins before device configuration. Architects must translate business intent, regulatory requirements, and risk assumptions into zones, boundary devices, inter-zone paths, and access-control policies. Existing NetOps automation tools mostly operate after this design has been fixed: they translate policies into configurations or interpret existing diagrams, but provide limited support for producing a structured security topology from an underspecified natural-language request.**

We describe TopoIntent, a system that compiles security intent into executable, compliance-checked network topologies. TopoIntent uses a schema contract to constrain topology generation, retrieves reference architectures from a curated template library through dense-vector search, and applies staged fusion to separate intent-template alignment from IG-level security completion. It then checks the generated topology against CIS Controls v8.1.2 safeguards whose evidence is visible at the topology layer, records cases requiring manual review, and repairs structural gaps through additive schema-preserving edits. The final topology is exported to Mininet scripts with kernel-level `iptables` ACLs, so that reachability and allow/deny behavior can be exercised rather than inspected only as a static diagram.

We build an evaluation set from reference security architecture diagrams because no public benchmark exists for this requirement-to-topology task. The retrieval reference set contains 22 templates and 44 synthetic intents across five scenarios; the held-out evaluation set contains 7 templates and 14 intents from finance and government scenarios excluded from the retrieval index. We evaluate TopoIntent with structural coverage, topology-visible CIS satisfaction, Mininet policy tests, and diagnostic feedback from Stage 5 to Stage 4. On the held-out set, additive repair raises first-pass topology-visible CIS satisfaction from 0.78/0.78 (IG2/IG3) to 1.00 in fewer than 1.5 rounds on average. In the end-to-end ablation, one feedback round raises the post-ACL policy pass rate from 0.78 to 0.88.

***Index Terms*—Intent Networking, Topology Generation, Compliance, RAG, Network Security**

1. Introduction

Enterprise network security design is still largely a manual engineering task. Before a network is deployed, security architects must turn organizational requirements—for example, regulatory exposure, business functions, and risk tolerance—into a topology with explicit zones, boundary devices, access-control paths, and monitoring points. The design also has to be checked against security frameworks such as CIS Controls [1]. In practice, engineers draft diagrams, compare them with compliance checklists, identify missing controls, and revise the design over multiple iterations. This work is slow, expensive, and difficult to scale as enterprise networks become larger and compliance obligations become more specific.

LLMs have recently been used for network management tasks, including translating natural-language intents into device configurations [2]–[5], generating NETCONF/YANG-based configurations [6], and supporting network operations through multi-agent workflows [7]. These systems mainly address the implementation stage: the architecture is assumed to exist, and the model helps produce or modify device-level artifacts. The earlier design stage remains less explored. Given only a business-level requirement, there is still no standard way to synthesize a complete security-zone topology, check its structural compliance obligations, and test whether the resulting connectivity and policy artifacts behave as intended.

This distinction matters. Existing intent-driven networking tools work within a known architecture [2]–[4]. Formal synthesis systems such as Propane [8], Propane/AT [9], and NetComplete [10] take formal specifications as input and synthesize routing configurations, not security architectures. GeNet [11] accepts topology diagrams and assists engineers in modifying them, but it does not start from a free-form requirement and does not include compliance checking. We study a different problem: how to generate a security topology from natural language, preserve the user intent, check the topology against structural compliance constraints, and validate the generated reachability and access-control behavior in an executable environment.

The difficulty comes from two sources. First, natural-language requirements are often incomplete. A request may specify a sector and a few critical services, but omit the DMZ, management plane, monitoring points, or boundary enforcement devices that a secure design would normally require. Second, the necessary design knowledge is distributed across reference architectures and compliance documents. A useful system must therefore combine intent understanding, design priors, compliance constraints, and executable validation without reducing the task to a single unconstrained LLM generation step.

We present **TopoIntent**, an LLM-assisted system for intent-based security topology generation and compliance checking. TopoIntent follows a compiler-like workflow: it parses user intent into a schema-governed topology contract, retrieves relevant reference architectures from a curated template library, fuses retrieved templates with the user intent, checks the result against topology-visible CIS Controls v8.1.2 requirements, repairs structural gaps through diagnostic feedback, and exports the topology to Mininet [12] for reachability and ACL validation. The output is not only a diagram-like topology, but a set of inspectable artifacts: JSON/XML topology records, generated policies, compliance decisions, repair history, visualization, and executable tests.

We make the following contributions:

- **A security-design compilation pipeline.** TopoIntent maps natural-language requirements into a schema-constrained topology representation that can be retrieved against, fused, checked, repaired, and emulated through a single machine-readable contract.
- **Template-grounded topology synthesis.** We build a curated multi-industry template library and use dense retrieval to ground topology generation in reference security architectures. The ablation shows that retrieval-augmented fusion improves structural coverage over direct generation and over using retrieved templates without fusion.
- **CIS-guided structural checking and repair.** We integrate CIS Controls v8.1.2 as a scoped set of topology-visible safeguards, distinguish structural evidence from management-layer evidence, and use additive repair to address unsatisfied or manual-review findings while preserving the original topology elements.
- **Executable validation and diagnostic feedback.** We construct a held-out benchmark for the new task and evaluate the system with structural metrics, scoped CIS satisfaction, and Mininet/`iptables` tests. Stage 5 reports separate topology and ACL failure categories and feeds repairable diagnostics back to Stage 4.

The evaluation reports both stage-level behavior and end-to-end ablations. A key point is that Stage 5 does not merely report whether tests pass; it separates missing paths from ACL-enforcement errors, which allows Stage 4 to repair the relevant part of the generated design rather than blindly changing the whole topology.

The remainder of this paper is organized as follows. Section 2 introduces background concepts and preliminaries. Section 3 presents the system overview. Section 4 details the design of each TopoIntent module. Section 5 describes our dataset construction, evaluation methodology, and experimental results. Section 6 discusses limitations and threats to validity. Section **??** reviews related work. Section 7 concludes the paper.

2. Preliminaries

2.1. Security Zone-Based Network Architecture

A security zone (also referred to as a security domain or network segment) is a logical partition of a network in which devices and services share a common security policy, trust level, and access control posture [13]. Zone-based network design is the foundational approach recommended by major security frameworks for structuring enterprise and institutional networks: rather than applying flat, perimeter-only defenses, the network is partitioned into multiple zones—such as an internet-facing demilitarized zone (DMZ), an internal office zone, a data center zone, and a management zone—each separated by boundary enforcement devices such as firewalls and intrusion prevention systems (IPS).

A *security topology* specifies (i) the set of security zones and their logical boundaries, (ii) the types and placement of security devices at zone boundaries, (iii) the inter-zone access control policies governing which traffic is permitted or denied, and (iv) the nodes (servers, endpoints, security appliances) assigned to each zone. Designing a compliant security topology requires translating organizational requirements and regulatory mandates into all four of these structural dimensions simultaneously—a task that demands both deep domain expertise and systematic compliance knowledge.

2.2. CIS Controls v8.1.2 and Implementation Groups

The CIS Controls [1] (Center for Internet Security Controls, version 8.1.2) are a prioritized set of 18 security control domains and 153 specific safeguards that provide actionable guidance for defending against the most prevalent cyber threats. Each safeguard is assigned to one of three *Implementation Groups* (IGs) based on the risk profile and resource capacity of the organization:

- **IG1** covers the 56 foundational safeguards essential for every organization, regardless of size or sector. These address basic cyber hygiene such as asset inventory, access control, and data recovery.
- **IG2** extends IG1 with 74 additional safeguards targeting organizations that manage sensitive data or

support critical services. Controls in this group address network monitoring, incident response, and security configuration management.
- **IG3** encompasses all 153 safeguards and is designed for organizations facing sophisticated, targeted attacks. It adds advanced capabilities such as penetration testing, application layer filtering, and security awareness training programs.

Only part of CIS can be judged from a topology. We therefore define a safeguard as *topology-visible* when its evidence can be expressed through zones, boundary devices, remote-access paths, monitoring placement, exposed assets, or inter-zone filtering. This screening yields 22 safeguards, mainly from controls related to network infrastructure management, monitoring, access control, and traffic filtering. Safeguards that require asset inventories, operational procedures, vulnerability records, user training, or runtime configuration audits are outside the automated topology layer. Accordingly, TopoIntent does not claim organizational CIS certification; it checks the scoped set of CIS-derived structural requirements that can be represented in a security topology.

2.3. Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) [14] augments the inference of a large language model by dynamically retrieving relevant documents from an external knowledge base and injecting them into the model's context at query time. This approach addresses two fundamental limitations of parametric LLMs: knowledge staleness and hallucination. A comprehensive survey of RAG variants and applications is provided by Gao et al. [15].

Standard RAG systems encode a document corpus into dense vector representations using a text embedding model, index the vectors in a dedicated vector database, and at inference time retrieve the top-k most semantically similar documents to the input query via approximate nearest-neighbor search. The retrieved documents are concatenated with the user query as context before being passed to the generative model.

TopoIntent uses retrieval in two places. In Stage 2, each reference template is encoded as a single vector over its concatenated fields and retrieved by cosine similarity; Section 5.4 shows that this simple strategy performs better than the tested multi-field variants on the current template library. In Stage 4, CIS safeguards are not retrieved semantically. The scoped structural safeguards are stored in a control database and selected by IG level, which avoids omitting a required check because of embedding similarity.

3. System Overview

3.1. Architecture Overview

TopoIntent transforms free-form security requirements into structured, compliance-checked security zone topologies. As illustrated in Fig. 1, the system comprises five sequential stages: Intent Analysis, Template Retrieval, Intent Fusion, CIS-Aware Compliance Check & Repair, and Emulation-Based Validation. All inter-stage data exchange is governed by a unified structured data contract (SCHEMACONTRACT), which defines the canonical JSON representation of a security topology and is described in Section 4.

Stage 1 — Intent Analysis. The user's natural language input is processed by an LLM-based intent parser that extracts an initial topology template conforming to SCHEMACONTRACT. The parser normalizes scenario, zone, node, and link fields into the schema enumerations and produces a machine-readable artifact for retrieval and fusion.

Stage 2 — Template Retrieval. The structured intent JSON is used to query the TopoIntent Template Library, a curated collection of reference security topologies sourced from authoritative industry publications and vendor reference architectures. Each template is encoded as a single dense vector over its concatenated fields using BGE-M3 [16] and indexed in Qdrant. At query time, the Stage 1 output is encoded using the same procedure and the top-k most similar templates are retrieved via cosine similarity.

Stage 3 — Intent Fusion. The user intent JSON and the top-k retrieved templates (default $k = 3$) are fused through a two-stage LLM pipeline: (1) *Semantic Fusion*, which aligns and merges the two JSON structures without inference, preserving all user-specified fields verbatim; and (2) *Intelligent Completion*, which fills in missing fields in accordance with the user's target CIS Controls Implementation Group (IG) level. If no IG level is specified, the stage produces two independent topology instances calibrated to IG2 and IG3 assurance levels respectively. The output is a structurally complete topology JSON ready for compliance verification.

Stage 4 — CIS-Aware Compliance Check & Repair. The generated topology is checked against the scoped CIS Controls v8.1.2 safeguards. The checker evaluates the topology JSON and labels each safeguard as *satisfied*, *unsatisfied*, or *manual_review*, using topology evidence rather than general product assumptions. Unsatisfied or uncertain safeguards are sent to an additive repair step that proposes the smallest schema-valid changes needed to provide the missing structural evidence. After repair, the system generates connectivity tests covering permitted flows, blocked flows across security boundaries, and paths through boundary enforcement devices. The repaired topology is exported as a structured XML artifact and rendered as an interactive HTML visualization.

Stage 5 — Emulation-Based Validation. The XML topology is converted into a Mininet emulation script that instantiates topology nodes as Linux network namespaces and realizes only node-level links as point-to-point links. Security zones remain logical metadata rather than emulated devices. The script first checks raw node-link reachability, then executes policy tests before and after installing `iptables` ACL rules derived from the generated policy set. The resulting JSON report records reachability, latency, ACL verdicts, and diagnostic failure categories. If validation failures indicate missing topology paths or inconsistent ACL

behavior, the report can be fed back to Stage 4 to trigger targeted topology repair, forming a closed-loop refinement cycle.

3.2. Design Principles

TopoIntent follows four design principles.

Determinism First. LLMs are not used for tasks that can be handled by rules. Node attribute inference, data-flow classification, schema validation, name normalization, and simulation-script generation are implemented as deterministic procedures so that identical inputs produce identical outputs.

Contract-Driven Modularity. All modules exchange data through SCHEMACONTRACT, a JSON schema that defines field semantics, type constraints, and referential-integrity rules. A module may enrich the topology only through fields admitted by the contract, which keeps intermediate results inspectable and prevents silent schema drift.

Artifact Persistence. Each stage writes structured logs and artifacts, including topology XML, interactive HTML, simulation scripts, repair records, and test reports. These files make it possible to inspect how a generated topology changed from intent parsing to executable validation.

Graceful Degradation. Local failures are handled explicitly. Malformed JSON is normalized when possible; unparseable configuration lines are logged as `unknown_lines`; nodes without usable IP information are skipped during script generation with a warning rather than causing the entire run to fail.

4. Design of TopoIntent

4.1. Data Contract Design

All inter-stage data exchange in TopoIntent is governed by SCHEMACONTRACT (v2.1), a formally defined JSON schema that serves as the canonical representation of a security topology throughout the pipeline. The schema comprises three field groups. The *identity* group captures the scenario category (7 types), a free-text description, and a structured topology summary. The *spatial* group defines zones (each with a `zone_id` and one of 11 `zone_type` categories) and nodes (each with a `node_id`, a `zone_id` reference, and one of 20 `normalized_type` device categories). The *connectivity* group encodes zone-level and node-level directed links, each annotated with a `link_type` (9 categories) and an optional label.

Three structural invariants are enforced: (i) ID-based referential integrity between nodes, zones, and links; (ii) enumerated type constraints on `zone_type`, `normalized_type`, and `link_type` to prevent semantic drift; and (iii) separation between the base topology and later artifacts such as policies, compliance verdicts, and test cases. This separation lets each stage be inspected independently instead of collapsing topology synthesis, compliance checking, and executable validation into one opaque output.

4.2. Intent Analysis

The intent analysis stage transforms a free-form natural language description of network security requirements into an initial topology template conforming to SCHEMACONTRACT v2.1. The extraction is performed by Qwen2.5-72B-Instruct [17] in a zero-shot setting. The full SCHEMACONTRACT definition is embedded in the system prompt as a structural constraint, and the model is instructed to emit a single valid JSON object with no additional commentary.

The model is expected to populate all three field groups of the schema: the *identity* group (scenario category, description, topology summary), the *spatial* group (zones with `zone_type` assignments and nodes with `normalized_type` and `zone_id` references), and the *connectivity* group (node-level and zone-level links with `link_type` annotations).

A post-processing validation step handles imperfect model outputs: JSON is extracted from markdown fences if present; missing required fields are populated with typed defaults; out-of-vocabulary `scenario`, `zone_type`, and `normalized_type` values are remapped to their closest valid enumeration entry or normalized to `other`; and `zone_id` references in `nodes` and `links` that do not correspond to any declared zone are corrected by assignment to the first available zone. If the model output cannot be parsed after two attempts, the stage raises a structured error and logs the raw outputs for inspection, while returning a minimal valid template to allow downstream stages to proceed.

Stage 1 outputs are evaluated against ground-truth templates on three metrics: *scenario accuracy* (exact match), *zone type recall* (fraction of ground-truth `zone_type` values covered by the predicted zone set), and *node type recall* (fraction of ground-truth `normalized_type` values covered by the predicted node set).

4.3. Template Retrieval

The Template Library stores reference security topologies indexed in Qdrant as dense vector representations. Each template is encoded as a single dense vector by concatenating five textual fields—*title*, *description*, *scenario*, *topology_summary*, and *security*—and encoding the result with BGE-M3 [16] for approximate nearest-neighbor search [18]. The *security* field is constructed by concatenating the template's zone type sequence and the deduplicated set of node types, providing a compact structural fingerprint of the topology's security architecture.

At query time, the Stage 1 output template is encoded using the same concatenation procedure to produce a single query vector. The top-k most similar templates are retrieved via cosine similarity and returned to the fusion stage. We compare this single-vector strategy against two multi-field variants in Section 5: a uniform-weight variant that encodes each field independently and aggregates scores with equal weights, and a weighted variant with tuned per-field weights. Empirical results on the reference set show that single-vector

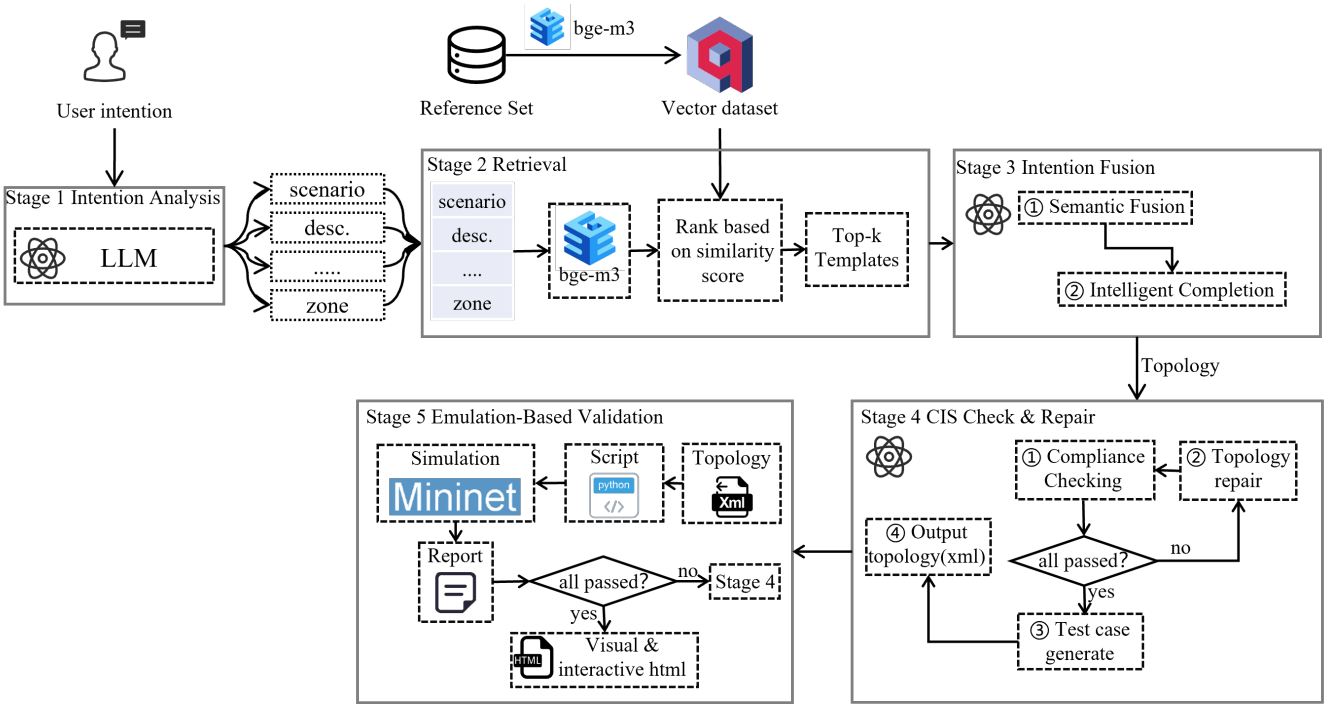

Figure 1. System architecture of TopoIntent. The pipeline comprises five stages: intent analysis, template retrieval, intent fusion, CIS-aware compliance check and repair, and emulation-based validation.

retrieval achieves the highest Top-1 accuracy and MRR among the three variants (Section 5.4), which we attribute to the relatively small template library size (22 templates) where a concatenated embedding captures sufficient cross-field signal without requiring field decomposition. Retrieval quality is measured by Top-k accuracy ($k \in \{1, 3, 5\}$) and Mean Reciprocal Rank (MRR).

4.4. Intent Fusion

The intent fusion stage combines the extracted user intent JSON with the top-ranked retrieved template through two sequential LLM calls.

4.4.1. Stage 3.1: Semantic Fusion. The first call receives both the user intent JSON and the template JSON as a joint input. Its objective is structural alignment without inference: fields present in the user intent take precedence and are preserved verbatim; fields absent from the user intent but present in the template are used to fill the corresponding positions in the output. The model is explicitly prohibited from introducing fields not present in either input or from making semantic inferences beyond direct structural merging. The output is a structurally complete but not yet fully specified topology JSON.

4.4.2. Stage 3.2: Intelligent Completion. The second call receives the Stage 3.1 output and performs IG-level-aware completion. If the user has specified a target Implementation Group level (IG1, IG2, or IG3), the model completes the topology according to the safeguard requirements of the specified IG level while preserving the simplified SCHEMA-CONTRACT. Completion focuses on adding missing zones, nodes, node links, and zone-level logical relations using the allowed schema fields; access-control policies, compliance verdicts, and executable test cases are generated later by Stage 4 as separate artifacts rather than embedded in the base topology schema. If no IG level is specified, the stage produces two independent topology instances: one completed for IG2 and one completed for IG3, providing the user with topologies calibrated to two distinct IG levels. Throughout completion, the model is constrained to maintain ID-based referential integrity across zones, nodes, and links.

4.5. CIS-Aware Compliance Check and Repair

4.5.1. IG-Level Parameter and Control Selection. Stage 4 receives the Stage 3 topology and a target IG level. If the user does not specify an IG level, the stage runs separately for IG2 and IG3 and produces two checked topology instances. The control database first filters CIS Controls v8.1.2 to the 22 safeguards with topology-level evidence. It then selects the IG-applicable subset: IG2 includes the structural safeguards assigned to IG1 or IG2, and IG3 includes the full structural subset.

4.5.2. Compliance Verification. The selected safeguards are submitted to Qwen2.5-72B-Instruct together with the topology JSON. Each safeguard record contains its identifier, title, IG level, topology evidence fields, and a short

predicate-style checking instruction. The checker returns a JSON object in which every safeguard has a `status` field (*satisfied*, *unsatisfied*, or *manual_review*), a confidence score in $[0, 1]$, and evidence text pointing to the topology elements used in the decision. A normalization step inserts missing safeguards as *manual_review*, removes malformed fields, and clamps confidence values. The checker is therefore used as a topology-evidence assessor, not as an oracle for full CIS certification.

4.5.3. Iterative Additive Repair. Controls classified as *unsatisfied* or *manual_review* are forwarded to the repair step. Unlike full-topology rewriting, the repair call is explicitly constrained to return only *additive* topology changes: new zones, new nodes, and new links that should be inserted to satisfy the identified findings. The additions are validated and merged into the current topology without modifying any existing elements. Following each repair round, the topology is re-verified against the full required control set. The loop runs for at most `max_repair_rounds` iterations (default: 2), stopping early when all findings are resolved. To limit over-generation, the repair prompt asks for the smallest addition set that addresses the current findings; the merger rejects duplicate IDs, dangling references, and schema-invalid elements. Each round also writes a diff record listing the safeguards addressed, the additions applied, and the before/after counts for zones, nodes, zone links, and node links.

4.5.4. Stage 5 Diagnostic Feedback. If Stage 5 emulation reveals policy failures or unexecutable test cases, the validation report is fed back to Stage 4 via a dedicated diagnostic repair interface. The interface converts the four repair-actionable Stage 5 failure categories—topology failures, ACL failures, ACL conflicts, and node-link failures—into structured repair findings and submits them to the additive repair call; unresolvable (*skipped*) test endpoints are surfaced for manual review rather than fed into automated repair, since they typically reflect intent inconsistencies that topology edits cannot resolve. A subsequent CIS re-verification pass is then executed to ensure that Stage 5-driven topology modifications do not introduce new compliance gaps.

4.5.5. Test Case Generation. After all repair rounds complete, connectivity test cases are generated from the finalized topology. The generator produces a structured set of node-to-node verification cases covering permitted flows across explicit zone and node links, blocked flows across security boundaries, and traffic paths traversing boundary enforcement devices (firewalls, VPN gateways, IDS/IPS, WAF). Each test case specifies `source_zone`, `target_zone`, `source_node`, `target_node`, `protocol`, `port`, expected verdict (*allow* or *block*), and a natural language rationale. A post-generation normalization step discards any test case whose source or target node identifier does not exist in the topology node list, ensuring all generated cases are executable by Stage 5.

4.5.6. XML Export. The finalized topology, compliance verification results, repair history, and test cases are serialized into a structured XML artifact. The XML root element carries the scenario category and IG level as attributes and contains sections for `Domains` (zone definitions), `Nodes` (node definitions with zone references), `Links` (zone- and node-level links), `Policies` (test cases as `Allow`/`Deny` elements with protocol and port attributes), `Compliance` (per-control verdicts with evidence, missing reason, and remediation text), and `RepairHistory` (per-round repair diffs). This artifact is the handoff document consumed by Stage 5 for emulation-based validation.

Stage 4 is evaluated on three metrics: *compliance satisfaction rate* (fraction of required safeguards classified as satisfied, averaged across samples), *average repair rounds*, and *average test cases generated per topology*.

4.6. Emulation-Based Validation

4.6.1. XML Parsing and Name Normalization. Stage 5 reads the XML artifact produced by Stage 4 and reconstructs the topology into a generator-friendly representation. Node identifiers are normalized to valid Linux interface names (maximum 9 usable characters, to accommodate the `-ethN` suffix appended by Mininet) using a two-step procedure: non-alphanumeric characters are stripped and the result is lowercased; names exceeding the length budget are truncated and disambiguated by a 4-character MD5 suffix; subsequent collisions among truncated names are resolved by appending an incrementing numeric suffix.

4.6.2. Topology Instantiation. The emulation script instantiates all nodes as `LinuxRouter` hosts—Mininet hosts with IP forwarding enabled—connected by direct links using the per-link point-to-point addresses. After the network starts, each node's loopback address is configured on its `lo` interface, IP forwarding and reverse-path filtering settings are applied, and shortest-path routes to all peer loopback addresses are installed using BFS over the node-link graph. This per-node routing setup enables multi-hop end-to-end reachability without requiring a dedicated router node.

4.6.3. ACL Deployment. Policies are compiled into an ACL table and installed as `iptables` rules on the Mininet namespaces corresponding to the relevant endpoints or boundary nodes available in the topology. For each tested flow, Stage 5 installs protocol- and port-aware `ACCEPT` or `DROP` rules and then adds fallback `DROP` rules for the same endpoint pair. The goal is to test the executable allow/deny semantics of the generated policy table. It is not intended to emulate vendor-specific firewall behavior.

4.6.4. Test Execution. Two categories of tests are executed. *Node-link connectivity tests* verify that each point-to-point link in the topology is operationally reachable by pinging the peer interface address; results are recorded with pass/fail verdict and round-trip latency. *Policy tests* execute each test case from the `Policies` section twice: once before ACL

deployment to test whether the underlying topology path is executable, and once after ACL deployment to test whether the generated ACL table enforces the expected verdict. For TCP test cases with a specified port, reachability is probed using a short-lived Python socket server on the destination host and a connection attempt from the source; the server accepts one connection and exits. For all other test cases, three ICMP ping packets are issued from the source to the destination loopback address. In both cases, the observed outcome (*allow* or *deny*) is compared against the expected verdict from the test case, and a pass/fail judgment is recorded along with latency. Test cases whose source or destination cannot be resolved to an instantiated host are marked as *skipped* and counted separately in the summary.

4.6.5. Report Generation. All results are serialized to a structured JSON report containing aggregate summaries, per-link reachability and latency records, pre-ACL and post-ACL policy-test records, and a diagnostic section. The diagnostic section explicitly separates topology failures, ACL failures, ACL conflicts, node-link failures, and unresolvable (*skipped*) test endpoints, allowing Stage 4 to repair structural connectivity issues without mistaking them for policy generation errors.

5. Evaluation

5.1. Dataset Construction

Constructing a high-quality ground-truth topology dataset from authoritative reference architecture diagrams requires resolving two independent challenges: accurate structural extraction from visual diagrams, and reliable CIS Controls compliance annotation. Both tasks are subjective and error-prone when performed by a single model, so we adopt a multi-model consensus pipeline with a judge-arbitration step, as illustrated in Fig. 2.

5.1.1. Stage D1: Dual-Model Topology Extraction. Each reference architecture diagram is independently processed by two vision-language models (VLMs): InternVL3.5-38B [19] and Qwen2.5-VL-72B [20], both served locally via vLLM. Each model is prompted to extract only the visually observable topology structure—zones, nodes, and links—into a SCHEMACONTRACT-conforming JSON, without inferring hidden devices, IP addresses, or implicit security functions. A conservative extraction protocol is enforced: grouped nodes represent repeated identical devices; zone-level connections are captured as `zone_links` rather than forced into `node_links`; and all `zone_type`, `normalized_type`, and `link_type` values are constrained to the predefined schema enumerations. Each output is validated against referential integrity rules before being accepted.

5.1.2. Stage D2: Dual-Model CIS Compliance Annotation. In parallel, each reference architecture diagram is independently evaluated by two large language models via CLI: GPT-5.5 (reasoning effort = xhigh) and Claude Sonnet 4.6. Each model assesses which topology-visible CIS Controls v8.1.2 safeguards are satisfied, rejected, or require manual review, following conservative matching rules that prohibit inferring compliance from product names alone and require explicit visual evidence for each safeguard.

5.1.3. Diff Generation and Sanity Check. Before judge arbitration, pairwise diffs are computed between the two Stage D1 topology outputs and between the two Stage D2 CIS outputs. The topology diff computes signature-based agreement and disagreement on zones, nodes, and links, and flags schema violations in either output. The CIS diff classifies each safeguard into consensus-matched, consensus-rejected, hard-conflict (one matched, one rejected), or soft-conflict (involving manual review) categories, and surfaces safeguards with large confidence divergence. These diffs serve as structured focus documents for the judge.

5.1.4. Stage D3: Judge Arbitration. Gemini 2.5 Pro [21] acts as the final judge, receiving the original topology image, both Stage D1 topology candidates, both Stage D2 CIS candidates, and the precomputed diffs as joint input. The judge is instructed to use the image as the primary source of truth, resolve conflicts by visible evidence rather than product capability assumptions, and produce a single final SCHEMACONTRACT-conforming topology template and a complete CIS compliance decision in which every safeguard from Stage D2 is classified exactly once into matched, rejected, or manual-review. Outputs are validated against full referential integrity and schema enum constraints before acceptance; failed outputs trigger up to two retries.

5.1.5. Stage D4: Synthetic Intent Generation and Dataset Split. For each finalized ground-truth topology, Gemini 2.5 Pro generates two synthetic user intent descriptions per template—one business-oriented and one architecture-oriented—by conditioning on the topology image, the final topology template, and the CIS compliance result. Generation constraints prohibit leaking internal schema identifiers, CIS safeguard IDs, or template IDs into the generated text, and enforce a word count range of 25–110 words per intent.

The finalized templates are split at the template level into a *reference set* and a *held-out evaluation set*. The reference set is indexed by the Template Library and used by Stage 2. The evaluation set is never indexed. We assign the finance and government scenarios to the holdout split because they carry stronger compliance expectations and make retrieval leakage especially problematic. This choice is stricter than a random split: during evaluation, the system must adapt design patterns from other sectors instead of retrieving near-duplicates from the same scenario family.

5.2. Experimental Setup

5.2.1. Hardware and Runtime Environment. All experiments are run on a local server with 8× NVIDIA RTX 4090 GPUs. LLMs and embedding models are served through

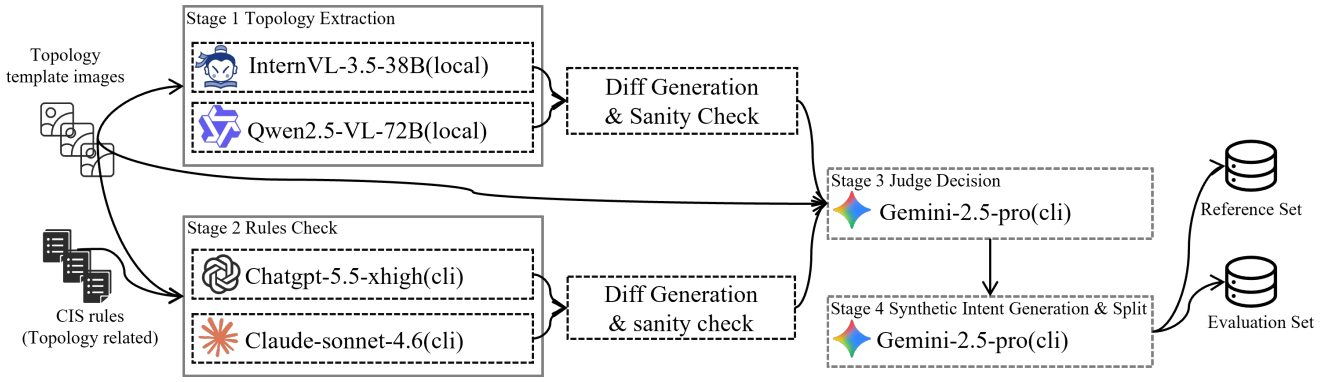

Figure 2. Dataset construction pipeline. Stage D1 extracts topology structures from reference architecture images using two VLMs in parallel; Stage D2 annotates topology-visible CIS Controls evidence using two LLMs; Stage D1 and Stage D2 outputs are reconciled via diff generation and sanity check; Stage D3 uses Gemini-2.5-Pro as a judge to produce the final ground-truth template; Stage D4 generates synthetic user intents and splits the dataset into reference and held-out evaluation sets.

vLLM [22]. Mininet scripts are executed on the same host with `iptables` enabled.

5.2.2. Model and System Configuration. Qwen2.5-72B-Instruct [17] serves as the primary instruction model across all LLM-driven stages: intent analysis, intent fusion, CIS verification, repair, and test-case generation. BGE-M3 [16] is used as the dense encoder for single-vector template retrieval, and Qdrant is used as the vector database for the Template Library. The SCHEMACONTRACT and all post-processing validators are shared across stages, ensuring that intermediate artifacts remain machine-checkable throughout the pipeline.

5.2.3. Baselines. We compare the full system with four ablations, each removing one part of the pipeline. **Direct**: the user's natural language input is passed directly to the LLM with the SCHEMACONTRACT definition as the only constraint; no intent analysis, retrieval, fusion, compliance verification, or emulation is performed. **RAG w/o Fusion**: Stage 1 intent analysis and Stage 2 template retrieval are performed, but the retrieved template is used directly as output without Stage 3 intent-template merging. **RAG w/o Repair**: the full Stage 1–3 pipeline is executed, followed by Stage 4 CIS verification; however, the iterative repair loop is skipped and the first-pass topology is exported directly without compliance-driven correction. **RAG w/o Feedback**: the complete Stage 1–4 pipeline is executed including iterative repair, but Stage 5 emulation results are not fed back to Stage 4 for further repair. **Full TopoIntent**: all five stages are executed, including Stage 5 diagnostic feedback to Stage 4.

5.3. Evaluation Metrics

We define metrics at four evaluation points along the pipeline.

Stage 1 intent analysis is evaluated on the reference set with scenario accuracy (exact match against ground-truth scenario label), zone-type recall (fraction of ground-truth `zone_type` values present in the generated topology), and node-type recall (fraction of ground-truth `normalized_type` values covered).

Stage 2 retrieval is evaluated on the reference set using Top-k accuracy ($k \in \{1, 3, 5\}$) and mean reciprocal rank (MRR). Since evaluation templates are excluded from the retrieval database, exact-template Top-k accuracy is not reported on the evaluation set.

Stage 4 compliance is evaluated on both IG2 and IG3 runs using topology-visible CIS satisfaction rate before and after the repair loop, average repair rounds, and average number of generated test cases.

Stage 5 executable validation is evaluated on initial and post-feedback runs for both IG2 and IG3 using node-link reachability, pre-ACL path-executability pass rate, post-ACL policy pass rate, average latency, and count of diagnostic failures.

Security topology generation is not a one-to-one reconstruction task: several designs may satisfy the same intent. We therefore use recall-oriented structural metrics to check whether required zone and device classes are covered, and rely on scoped compliance and executable validation metrics to test whether the generated structure supports the intended security behavior. Repair-history diffs report any additive expansion introduced by the compliance loop.

5.4. Stage-Level Results

Table 1 reports intent analysis results on the reference set. *Scenario Accuracy* measures exact match between the predicted scenario label and the ground-truth label. *Zone Type Recall* measures the fraction of ground-truth `zone_type` values that appear in the predicted topology's zone set, reflecting how well the model captures the required security zone categories. *Node Type Recall* measures the fraction of ground-truth `normalized_type` values covered by the predicted node set, reflecting device-class coverage relative to the reference design. The high scenario

TABLE 1. STAGE 1 INTENT ANALYSIS RESULTS (REFERENCE SET, $n = 44$)

Metric	Value
Scenario Accuracy	0.98
Zone Type Recall	0.68
Node Type Recall	0.53

TABLE 2. STAGE 2 RETRIEVAL ABLATION (REFERENCE SET, $n = 44$)

Retrieval Strategy	Top-1	Top-3	Top-5	MRR
Multi-field uniform	0.60	0.93	1.0	0.75
Multi-field weighted	0.57	0.93	0.98	0.75
Single-vector	0.75	0.91	1.0	0.84

accuracy (0.98) confirms that the model reliably identifies the deployment context; the lower node type recall (0.53) is expected, as natural language descriptions typically do not enumerate all device classes explicitly.

Table 2 compares three retrieval strategies on the reference set. *Top-k accuracy* measures whether the ground-truth template appears within the top-k retrieved results; *MRR* rewards strategies that rank the ground-truth template higher. Single-vector retrieval, which encodes each template as a single concatenated embedding, achieves the highest Top-1 accuracy (0.75) and MRR (0.84), outperforming both multi-field variants on the current 22-template library. We attribute this to the small library size, where a concatenated embedding captures sufficient cross-field signal without requiring field decomposition. All three strategies converge at Top-5, suggesting that the correct template is consistently retrievable within the top-5 candidates regardless of encoding strategy.

Table 3 reports scoped CIS results before and after iterative repair. *CIS Sat. (Before Repair)* is the fraction of required structural safeguards classified as satisfied on the first pass; *CIS Sat. (After Repair)* is the same fraction after repair. *Avg. Repair Rounds* records the number of verify-repair iterations per sample, and *Avg. Test Cases* records the number of executable connectivity tests generated from the final topology. The repair loop raises satisfaction from 78% to 100% for both IG2 and IG3 in fewer than 1.5 rounds on average. This result concerns the selected topology-visible safeguards only; executable behavior is evaluated separately in Stage 5.

TABLE 3. STAGE 4 CIS COMPLIANCE VERIFICATION AND REPAIR RESULTS (EVALUATION SET, $n = 14$)

Metric	IG2	IG3
CIS Sat. (Before Repair)	0.78	0.78
CIS Sat. (After Repair)	1.00	1.00
Avg. Repair Rounds	1.43	1.36
Avg. Test Cases	41.14	41.14

5.5. Executable Validation and Feedback

Table 4 summarizes the Mininet validation results. *Node-Link Reach.* measures whether declared `node_links` become reachable point-to-point links. *Path Exec. (Pre-ACL)* measures whether each policy test has an underlying path before any `iptables` rule is installed; in this phase every expected verdict is treated as *allow*, so block-flow tests are used only to check path availability. *Post-ACL Pass* measures whether the observed result after ACL installation matches the intended *allow* or *block* verdict. For this reason, post-ACL pass can exceed pre-ACL path executability: a blocked flow is a failure before policy enforcement but a success after the corresponding ACL is installed. The 100% node-link reachability indicates that Stage 4 produces Mininet-instantiable node-level connectivity. Remaining post-ACL failures come from policy-table behavior rather than missing physical links, which is why Stage 5 diagnostics are useful for targeted repair.

5.6. End-to-End Ablation

Table 5 reports end-to-end results across five system variants on the evaluation set. *Zone Type Recall* and *Node Type Recall* measure structural faithfulness to the ground-truth topology, using the same definitions as in Stage 1 evaluation. *CIS Sat. Rate* measures the fraction of required IG2 and IG3 safeguards satisfied in the final output topology, averaged across both IG levels; '–' indicates that the variant does not include CIS verification. *Post-ACL Pass Rate* measures the fraction of generated policy test cases that enforce the expected verdict in Mininet after ACL deployment; '–' indicates that the variant does not include Stage 5 emulation. Each variant incrementally adds one pipeline component, allowing the contribution of each stage to be isolated: Direct establishes the LLM generation baseline without any structured pipeline; RAG w/o Fusion adds template retrieval but skips intent-template merging; RAG w/o Repair adds the full fusion pipeline but skips compliance repair; RAG w/o Feedback adds repair but disables the Stage 5 diagnostic loop; Full TopoIntent enables the complete pipeline. Notably, RAG w/o Fusion achieves lower zone-type recall than Direct (0.54 vs. 0.63), which we attribute to the scenario mismatch between the reference set and the evaluation set: retrieved templates are drawn from five non-overlapping scenarios, and using them directly without fusion propagates structural patterns misaligned with the finance and government evaluation topologies. The ablation clarifies the role of each component. Fusion improves structural recall over directly using retrieved templates. Repair raises scoped CIS satisfaction from 0.70 to 1.00. Adding Stage 5 feedback then increases the post-ACL pass rate from 0.78 to 0.88, showing that executable diagnostics catch policy-level problems that the structural checker does not see. This ablation-level gain should not be confused with the per-IG before/after rows in Table 4, where feedback mainly reduces diagnostic failures and changes the aggregate post-ACL rate only slightly. Node-type recall is the only non-monotonic metric (0.82

TABLE 4. STAGE 5 EXECUTABLE VALIDATION RESULTS WITH ONE FEEDBACK ROUND (EVALUATION SET, n=14).

Setting	Node-Link Reach.	Path Exec.	Post-ACL Pass	Avg. Latency (ms)	Diag. Failures
IG2 Initial	1.00	0.64	0.87	0.06	214
IG2 After Feedback	1.00	0.74	0.86	0.07	165
IG3 Initial	1.00	0.67	0.87	0.06	189
IG3 After Feedback	1.00	0.77	0.89	0.07	147

for RAG w/o Feedback and 0.77 for Full TopoIntent), likely because a feedback repair can substitute a structurally equivalent device class while preserving the policy behavior measured by Stage 5.

5.7. Qualitative Case Study

We examine one held-out sample, `Financial-4_req1`, to illustrate TopoIntent's generation process. This sample was excluded from the reference template library. The user intent requires a new branch office with high availability, voice service continuity (PSTN failover on primary link failure), SIP-based voice, centralized authentication, and security logging.

As shown in Figure 3, TopoIntent produces an IG3 topology with clear zone-based segmentation, expanding from 6 zones (25 nodes) in the baseline IG2 design to 11 zones (32 nodes). The expansion introduces dedicated Security, Management, MFA, Asset Management, and External Testing zones.

The diagnostic feedback loop proves effective: the initial IG2 topology achieved a post-ACL pass rate of 0.72. After one feedback round, the pass rate reached 1.00. For IG3, the pass rate improved from 0.95 to 1.00. Figure 3 visualizes the final topology, with green regions highlighting newly introduced compliance and management zones.

6. Discussion and Limitations

TopoIntent shows that natural-language security requirements can be turned into schema-governed topologies, checked against a scoped set of CIS-derived structural safeguards, and exercised in an executable network emulator. The contribution is the decomposition of the design task into inspectable steps: contract-based topology synthesis, template grounding, structural repair, and policy validation. Several boundaries of the current system should be kept in view.

Scope of compliance claims. CIS Controls v8.1.2 is used as a normative source, but TopoIntent does not certify full organizational CIS compliance. Controls depending on inventories, procedures, vulnerability-management records, training, or live configuration audits are outside the topology layer. The reported CIS rates therefore refer only to the selected safeguards whose evidence can be represented in the generated topology.

LLM judgments. The pipeline constrains LLM outputs with schemas, retries, normalization, deterministic post-processing, multi-model dataset construction, and persistent evidence records. Still, the checker and repair stages can make judgment errors, especially when a safeguard is only weakly implied by a topology. A natural extension is to replace more of these checks with deterministic graph and policy predicates once the structural interpretation of each safeguard is fixed.

Emulation fidelity. Mininet validation gives executable evidence for paths and allow/deny policy behavior, but it is not a production deployment. The emulator represents nodes as Linux namespaces and policies as `iptables` rules. This is sufficient for checking consistency between the generated topology and the generated policy table, but it does not model stateful firewall semantics, cloud control planes, proprietary appliances, or application-layer inspection.

Dataset scale. The current dataset is intentionally held out at the scenario level to reduce retrieval leakage, but it remains modest. The experiments should be read as an auditable first benchmark for requirement-to-security-topology generation, not as full coverage of enterprise architectures. Larger evaluations should include more sectors, larger diagrams, real practitioner-written requirements, and independent expert review of the compliance labels.

Additive repair. Additive repair preserves provenance because user-specified topology elements are not silently deleted. The tradeoff is that the system may add redundant devices or links when the checker is conservative. TopoIntent limits this by requesting minimal additions, rejecting invalid elements, and recording diffs, but future versions should add explicit repair costs or graph-edit penalties.

Feedback convergence. Stage 5 feedback supplies useful diagnostics, but it does not guarantee that one round will make every policy test pass. Some failures come from contradictory test cases, underspecified intent, or design choices that require human judgment. In these cases the system should expose the conflict rather than force an unsafe automatic repair.

7. Conclusion

We presented TopoIntent, a system for compiling natural-language security intent into executable, compliance-checked network topologies. The system converts free-form requirements into a schema-governed topology contract, grounds generation with retrieved reference architectures, repairs structural gaps against a scoped set of CIS Controls v8.1.2 safeguards, and validates reachability and ACL behavior in Mininet/`iptables`. This design links static topology generation with executable evidence while keeping topology, compliance decisions, repairs, and

TABLE 5. END-TO-END ABLATION RESULTS (EVALUATION SET, n=14). A CHECK MARK INDICATES THE COMPONENT IS ENABLED IN THAT VARIANT. **PIPELINE COMPONENTS. S1/S2/S3** ARE STAGES 1–3 OF THE PIPELINE; **VERIFY** AND **REPAIR** ARE STAGE 4 CIS VERIFICATION AND THE ITERATIVE ADDITIVE-REPAIR LOOP; **EMUL** IS STAGE 5 EXECUTABLE EMULATION; **FB** IS THE STAGE 5→STAGE 4 DIAGNOSTIC FEEDBACK LOOP. **METRICS. ZONE** AND **NODE** ARE ZONE-TYPE AND NODE-TYPE RECALL AGAINST THE GROUND-TRUTH TOPOLOGY; **CIS** IS THE CIS SATISFACTION RATE (FRACTION OF REQUIRED IG2/IG3 SAFEGUARDS SATISFIED, AVERAGED ACROSS BOTH IG LEVELS); **POST-ACL** IS THE POST-ACL POLICY PASS RATE MEASURED IN MININET EMULATION. A DASH (–) MARKS METRICS NOT APPLICABLE TO A VARIANT.

	Pipeline Components							Metrics			
Variant	**S1**	**S2**	**S3**	**Verify**	**Repair**	**Emul**	**FB**	**Zone**	**Node**	**CIS**	**Post-ACL**
Direct								0.63	0.61	–	–
RAG w/o Fusion	✓	✓						0.54	0.63	–	–
RAG w/o Repair	✓	✓	✓	✓				0.75	0.73	0.70	–
RAG w/o Feedback	✓	✓	✓	✓	✓	✓		0.74	0.82	1.00	0.78
Full TopoIntent	✓	✓	✓	✓	✓	✓	✓	0.76	0.77	1.00	0.88

policy tests as separate inspectable artifacts. On the held-out evaluation set, additive repair raises topology-visible CIS satisfaction from 0.78/0.78 (IG2/IG3) to 1.00, and the end-to-end ablation shows that one Stage 5 feedback round raises post-ACL pass rate from 0.78 to 0.88. The results suggest that requirement-to-security-topology generation can be treated as an end-to-end design workflow, while also making clear where structural checking ends and production-grade deployment validation must begin.

8. LLM Usage Statement

Large language models were used in two ways. First, they are part of the TopoIntent system: Qwen2.5-72B-Instruct is used for intent analysis, topology fusion and completion, CIS-oriented checking and repair, test-case generation, and diagnostic-feedback-based refinement; BGE-M3 is used for embedding-based template retrieval. These uses are described in the methodology sections. Generated artifacts were checked through schema validation, scoped CIS verification, Mininet-based tests, and manual inspection of representative outputs.

Second, LLMs were used as editorial aids during manuscript preparation, including wording and organization. The authors inspected the resulting text and take full responsibility for the correctness, originality, and integrity of all claims, tables, figures, experiments, and conclusions. The data used in this work consists of synthetic network-intent samples and topology templates constructed for this study; no private user data was used. We report model configuration, prompts, schema contracts, evaluation scripts, and generated artifacts to reduce reproducibility risks from model updates, decoding randomness, prompt sensitivity, and deployment differences.

References

[1] Center for Internet Security, "CIS Critical Security Controls Version 8.1," Center for Internet Security (CIS), Technical Report, 2024, accessed: May 2026. [Online]. Available: https://www.cisecurity.org/controls/v8-1

[2] C. Wang, M. Scazzariello, A. Farshin, S. Ferlin, D. Kostić, and M. Chiesa, "Netconfeval: Can llms facilitate network configuration?" *Proceedings of the ACM on Networking*, vol. 2, no. CoNEXT2, pp. 1–25, 2024.

[3] O. G. Lira, O. M. Caicedo, and N. L. da Fonseca, "Large language models for zero touch network configuration management," *IEEE Communications Magazine*, vol. 63, no. 7, pp. 146–153, 2024.

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

[5] N. Tu, S. Nam, and J. W.-K. Hong, "Intent-based network configuration using large language models," *International Journal of Network Management*, vol. 35, no. 1, p. e2313, 2025.

[6] G. Hollósi, D. Ficzere, and P. Varga, "Generative ai for low-level netconf configuration in network management based on yang models," in *2024 20th International Conference on Network and Service Management (CNSM)*. IEEE, 2024, pp. 1–7.

[7] Z. Wang, S. Lin, G. Yan, S. Ghorbani, M. Yu, J. Zhou, N. Hu, L. Baruah, S. Peters, S. Kamath, J.-L. Yang, and Y. Zhang, "Intent-driven network management with multi-agent llms: The confucius framework," *Proceedings of the ACM SIGCOMM 2025 Conference*, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusID:280953986

[8] R. Beckett, R. Mahajan, T. D. Millstein, J. Padhye, and D. Walker, "Don't mind the gap: Bridging network-wide objectives and device-level configurations," *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:263266303

[9] ——, "Network configuration synthesis with abstract topologies," *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:26265051

[10] A. El-Hassany, P. Tsankov, L. Vanbever, and M. T. Vechev, "Netcomplete: Practical network-wide configuration synthesis with autocompletion," in *Symposium on Networked Systems Design and Implementation*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:4787777

[11] B. Ifland, E. Duani, R. Krief, M. Ohana, A. Zilberman, A. Murillo, O. Manor, O. Lavi, H. Kenji, A. Shabtai, Y. Elovici, and R. Puzis, "Genet: A multimodal llm-based co-pilot for network topology and configuration," *2025 IEEE 45th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 117–122, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:271097938

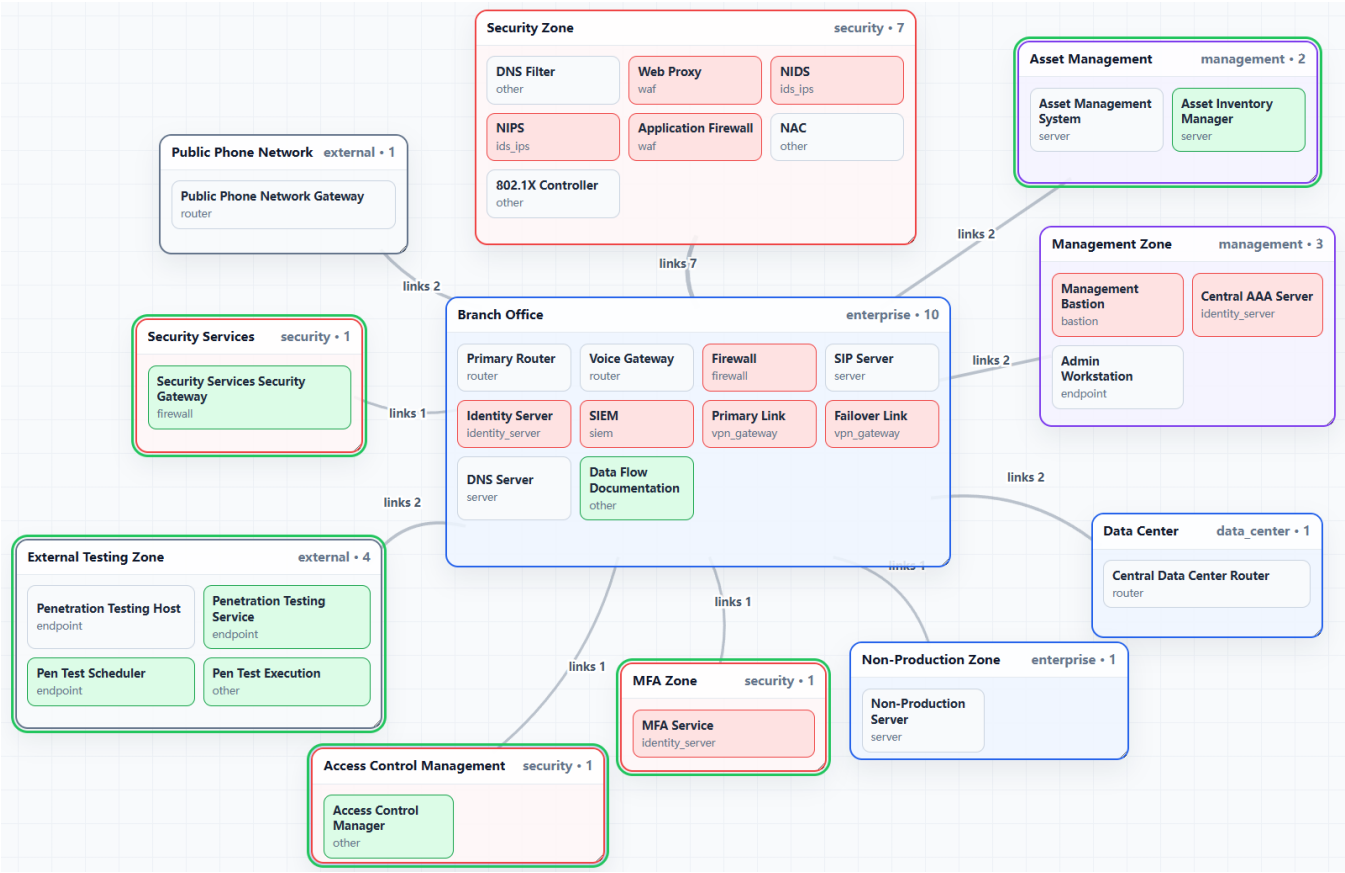

Figure 3. Case study visualization for `Financial-4_req1`. Green regions indicate advanced compliance and management zones introduced in IG3.

[12] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.

[13] K. Stouffer, M. Pease, C. Tang, T. Zimmerman, V. Pillitteri, S. Lightman, A. Hahn, S. Saravia, A. Sherule, and M. Thompson, "Guide to operational technology (OT) security," National Institute of Standards and Technology, NIST Special Publication 800-82r3, Sep. 2023. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-82r3

[14] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Kuttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *ArXiv*, vol. abs/2005.11401, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:218869575

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

[16] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu, "M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation," in *Annual Meeting of the Association for Computational Linguistics*, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:267413218

[17] Q. A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, G. Dong, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y.-C. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, Z. Qiu, S. Quan, and Z. Wang, "Qwen2.5 technical report," *ArXiv*, vol. abs/2412.15115, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:274859421

[18] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2019.

[19] W. Wang, Z. Gao, L. Gu, H. Pu, L. Cui, X. Wei, Z. Liu, L. Jing, S. Ye, J. Shao, Z. Wang, Z. Chen, H. Zhang, G. Yang, H. Wang, Q. Wei, J. Yin, W. Li, E. Cui, G. Chen, Z. Ding, C. Tian, Z. Wu, J. Xie, Z. Li, B. Yang, Y. Duan, X. Wang, H. Hao, S. Li, X. Zhao, H. Duan, N. Deng, B. Fu, Y. He, Y. Wang, C. He, B. Shi, J. He, Y. Xiong, H. Lv, L. Wu, W. Shao, K. Zhang, H. Deng, B. Qi, B. Qi, Q. Guo, W. Zhang, Y. Gu, W. Ouyang, L. Wang, M. Dou, X. Zhu, T. Lu, D. Lin, J. Dai, B. Zhou, W. Su, K. Chen, Y. Qiao, W. Wang, and G. Luo, "Internvl3.5: Advancing open-source multimodal models in versatility, reasoning, and efficiency," *ArXiv*, vol. abs/2508.18265, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusID:280710824

[20] Q. Team, "Qwen2.5-vl," January 2025. [Online]. Available: https://qwenlm.github.io/blog/qwen2.5-vl/

[21] G. Comanici, E. Bieber, M. Schaekermann, I. Pasupat, N. Sachdeva, I. Dhillon, M. Blistein, O. Ram, D. Zhang, E. Rosen *et al.*, "Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities," *arXiv preprint arXiv:2507.06261*, 2025.

[22] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th symposium on operating systems principles*, 2023, pp. 611–626.

Appendix A.
Topology Contract Schema

Our system uses a JSON contract to constrain topology generation. The contract specifies the scenario, zones, nodes, node-level links, and zone-level links. Table 6 summarizes the main fields, followed by a compact schema skeleton.

Table 6. Main fields in the topology contract.

Field	Description
schema_version	Version of the contract schema.
template_id	Identifier of the topology template.
title	Human-readable title of the generated topology.
description	Brief description of the generation task.
scenario	Deployment scenario, e.g., enterprise, cloud, hospital, finance, or government.
topology_summary	Natural-language summary of the generated topology.
zones	List of security or administrative zones.
z.zone_id	Unique identifier of a zone.
z.name	Human-readable zone name.
z.zone_type	Normalized zone category, such as enterprise, DMZ, data center, management, OT, IoT, or external.
nodes	List of network assets assigned to zones.
n.node_id	Unique identifier of a node.
n.name	Human-readable node name.
n.normalized_type	Normalized asset category, such as router, switch, firewall, VPN gateway, server, database, endpoint, or cloud service.
n.zone_id	Identifier of the zone containing the node.
links.node_links	Asset-level links between nodes.
links.zone_links	Zone-level links between zones.
link_id	Unique identifier of a link.
source / target	Source and target endpoints of a link.
link_type	Normalized link category, such as routing, VPN, trunk, management, monitoring, or data flow.
label	Human-readable label for the link.

Listing A.1. Compact JSON skeleton of the topology contract.

```
{
  "schema_version": "2.1",
  "template_id": "...",
  "title": "...",
  "description": "...",
  "scenario": "...",
  "topology_summary": "...",
  "zones": [
    {
      "zone_id": "z1",
      "name": "...",
      "zone_type": "..."
    }
  ],
  "nodes": [
    {
      "node_id": "n1",
      "name": "...",
      "normalized_type": "...",
      "zone_id": "z1"
    }
  ],
  "links": {
    "node_links": [
      {
        "link_id": "nl1",
        "source": "n1",
        "target": "n2",
        "link_type": "...",
        "label": "..."
      }
    ],
    "zone_links": [
      {
        "link_id": "zl1",
        "source": "z1",
        "target": "z2",
        "link_type": "...",
        "label": "..."
      }
    ]
  }
}
```

Appendix B.
Rules-Check Contract Schema

Our system uses a rules-check contract to record how the generated topology is evaluated against topology-visible CIS safeguards. The contract separates matched safeguards, rejected requirements, and items that require manual review. Each safeguard entry in the internal control database includes the safeguard identifier, title, IG applicability, topology evidence fields, and a concise predicate-style checking instruction. Table 7 summarizes the main fields, followed by a compact schema skeleton. For compactness, we use the following abbreviations in Table 7: mcc denotes matched_cis_candidates, rcr denotes rejected_cis_requirements, mrf denotes manual_review_flags, and csi denotes cis_safeguard_id.

Table 7. Main fields in the rules-check contract.

Field	Description
schema_version	Version of the rules-check contract schema.
template_id	Identifier of the topology or checking template.
model	Model used to produce the rules-check output.
mcc	CIS safeguards matched to the generated topology and considered satisfied.
mcc.csi	Identifier of the matched CIS safeguard.
mcc.title	Human-readable title of the matched safeguard.
mcc.status	Matching result, e.g., satisfied.
mcc.confidence	Confidence score assigned to the safeguard match.
rcr	CIS safeguards rejected as not applicable or not supported by the generated topology.
rcr.csi	Identifier of the rejected CIS safeguard.
rcr.title	Human-readable title of the rejected safeguard.
rcr.reject_reason	Explanation for rejecting the safeguard requirement.
mrf	CIS safeguards that require human inspection before final acceptance.
mrf.csi	Identifier of the safeguard requiring manual review.
mrf.reason	Explanation for why manual review is needed.

Listing A.2. Compact JSON skeleton of the rules-check contract.

```
[
  {
    "schema_version": "2.1",
    "template_id": "...",
    "model": "...",
    "matched_cis_candidates": [
      {
        "cis_safeguard_id": "13.4",
        "title": "...",
        "status": "satisfied",
        "confidence": 0.0
      }
    ],
    "rejected_cis_requirements": [
      {
        "cis_safeguard_id": "12.5",
        "title": "...",
        "reject_reason": "..."
      }
    ],
    "manual_review_flags": [
      {
        "cis_safeguard_id": "...",
        "reason": "..."
      }
    ]
  }
]
```

Appendix C. XML Export Contract Schema

The final stage exports the repaired topology, policies, compliance evidence, repair history, data flows, and residual risks into an XML contract. This XML representation is intended as a stable exchange format after validation and repair. Table 8 summarizes the mapping from the intermediate JSON contract to the XML export format, followed by a compact XML skeleton.

Table 8. Mapping from intermediate JSON fields to XML export elements.

XML Element	Description
Topology	Root element containing schema version, template identifier, title, scenario, and IG level.
Description	Human-readable description of the generated topology.
Summary	Human-readable summary of the generated topology.
Domains/Domain	Security or administrative zones mapped from zones.
Nodes/Node	Network assets mapped from nodes.
Attributes/Attribute	Optional node-level metadata, such as exposure, privilege, target status, or asset level.
Links/ZoneLinks	Zone-level links mapped from links.zone_links.
Links/NodeLinks	Node-level links mapped from links.node_links.
Policies/Allow	Connectivity test cases or policies expected to be allowed.
Policies/Deny	Connectivity test cases or policies expected to be denied.
Compliance/Control	CIS safeguard evaluation result with evidence and optional remediation.
RepairHistory/Repair	Record of topology repair actions performed to satisfy required controls.
DataFlows/Flow	Derived data-flow relationship and trust-boundary annotation.
Risks/Risk	Residual risk description and mitigation.

Listing A.3. Compact XML skeleton of the export contract.

```
<Topology schema_version="2.1" template_id="..." title
    ="..." scenario="cloud" ig_level="IG2">
  <Description>...</Description>
  <Summary>...</Summary>

  <Domains>
    <Domain id="z1" name="Zone Name" type="enterprise"/>
  </Domains>

  <Nodes>
    <Node id="n1" name="Node Name" domain="z1"
        domain_name="Zone Name" type="firewall" ip="">
      <Attributes>
        <Attribute name="asset_level" value=""/>
        <Attribute name="is_target" value="false"/>
        <Attribute name="is_exposed" value="false"/>
        <Attribute name="is_high_priv" value="false"/>
      </Attributes>
    </Node>
  </Nodes>

  <Links>
    <ZoneLinks>
      <Link id="zl1" source="z1" target="z2" type="
          routing" label=""/>
    </ZoneLinks>
    <NodeLinks>
      <Link id="nl1" source="n1" target="n2" type="
          data_flow" label=""/>
    </NodeLinks>
  </Links>

  <Policies>
    <Allow id="tc1" src_zone="z1" dst_zone="z2" src_node
        ="n1" dst_node="n2" protocol="tcp" port="443">
      <Reason>...</Reason>
    </Allow>
    <Deny id="tc2" src_zone="z3" dst_zone="z1" src_node
        ="" dst_node="" protocol="" port="">
      <Reason>...</Reason>
    </Deny>
  </Policies>

  <Compliance ig_level="IG2" all_required_satisfied="true
      ">
    <Control id="13.4" title="Perform Traffic Filtering
        Between Network Segments" status="satisfied"
        confidence="1.0">
      <Evidence>...</Evidence>
      <Remediation></Remediation>
    </Control>
  </Compliance>

  <RepairHistory>
    <Repair iteration="1" control_id="13.4" action="add"
        target_type="node" target_id="n1">
      <Reason>...</Reason>
    </Repair>
  </RepairHistory>

  <DataFlows>
    <Flow id="df1" src="z1" dst="z2" protocol="" port=""
        trust_boundary="true">
      <Reason>...</Reason>
    </Flow>
  </DataFlows>

  <Risks>
    <Risk id="risk1" severity="medium" control_id="13.4">
      <Description>...</Description>
      <Mitigation>...</Mitigation>
    </Risk>
  </Risks>
</Topology>
```

Appendix D. Runtime Prompts

Our pipeline uses stage-specific system prompts to constrain generation, repair, verification, and test-case construction. Rather than using a single open-ended prompt, each stage assigns the model a specific role, restricts the allowed output format, and enforces additive or verification-only behavior. Table 9 summarizes the runtime prompts used in the pipeline.

Table 9. Summary of runtime prompts in the TopoIntent pipeline.

Prompt	Role	Purpose
`Stage1`	Network security topology analyst	Extract structured fields from natural-language intent and produce an initial topology template with scenario, zones, nodes, and empty links.
`Stage3-Fusion`	Network security topology architect	Add minimal missing topology elements using reference templates while preserving user-specified zones, nodes, and intent requirements.
`Stage3-Repair`	Network security compliance engineer	Add minimal zones, nodes, or links needed to support target CIS safeguards without claiming that controls are fully verified.
`Stage4-Verify`	CIS Controls topology compliance auditor	Verify whether the topology satisfies each required CIS safeguard using only explicit or strongly implied topology evidence.
`Stage4-Repair`	Network security topology repair engineer	Propose compact additive repairs for unsatisfied or ambiguous CIS findings while preserving structural integrity.
`Stage4-TestCase`	Network security test engineer	Generate final executable node-to-node connectivity tests after topology verification and repair.

Appendix E. Dataset Construction Prompts

Before running the TopoIntent pipeline, we construct and curate the dataset through four prompt-guided preprocessing steps: topology extraction, CIS topology-relevance screening, judge-based consolidation, and synthetic intent generation. Table 10 summarizes these prompts and their roles.

Table 10. Summary of dataset construction prompts.

Prompt	Role	Purpose
`Topology Extraction`	Extraction engine	Extract only visible zones, nodes, node links, and zone links from topology diagrams. The prompt forbids hidden devices, hidden policies, inferred controls, IP addresses, and CIS mapping.
`CIS Rules Check`	Rules classifier	Classify CIS safeguards as topology-related or not. A safeguard is marked topology-related only when it directly affects zones, boundaries, links, devices, remote access paths, data flows, exposed assets, or network monitoring points.
`As-Judge`	Dataset judge	Consolidate multiple model outputs into a final topology template and final CIS decision. The judge uses the original image as the source of truth and classifies each CIS candidate as matched, rejected, or requiring manual review.
`Intent Generation`	User-intent generator	Generate realistic natural-language user requirements from finalized topology ground truth and the original image. The prompt requires moderately underspecified intents and forbids internal IDs, schema names, CIS IDs, and unsupported additions.

Appendix F.
Topology-Visible CIS Controls v8.1.2 Safeguards

TABLE 11. TOPOLOGY-VISIBLE CIS CONTROLS V8.1.2 SAFEGUARDS

ID	Title	IG1	IG2	IG3
1.1	Establish and Maintain Detailed Enterprise Asset Inventory	✓	✓	✓
3.8	Document Data Flows		✓	✓
3.12	Segment Data Processing and Storage Based on Sensitivity		✓	✓
4.4	Implement and Manage a Firewall on Servers	✓	✓	✓
4.9	Configure Trusted DNS Servers on Enterprise Assets		✓	✓
6.4	Require MFA for Remote Network Access	✓	✓	✓
9.2	Use DNS Filtering Services	✓	✓	✓
9.3	Maintain and Enforce Network-Based URL Filters		✓	✓
12.2	Establish and Maintain a Secure Network Architecture		✓	✓
12.3	Securely Manage Network Infrastructure		✓	✓
12.5	Centralize Network Authentication, Authorization, and Auditing (AAA)		✓	✓
12.7	Ensure Remote Devices Utilize a VPN and are Connecting to an Enterprise's AAA Infrastructure		✓	✓
12.8	Establish and Maintain Dedicated Computing Resources for All Administrative Work			✓
13.3	Deploy a Network Intrusion Detection Solution		✓	✓
13.4	Perform Traffic Filtering Between Network Segments		✓	✓
13.5	Manage Access Control for Remote Assets		✓	✓
13.6	Collect Network Traffic Flow Logs		✓	✓
13.8	Deploy a Network Intrusion Prevention Solution			✓
13.9	Deploy Port-Level Access Control			✓
13.10	Perform Application Layer Filtering			✓
16.8	Separate Production and Non-Production Systems		✓	✓
18.2	Perform Periodic External Penetration Tests		✓	✓