Università
di Genova

MSc Computer Engineering
Artificial Intelligence & Human-Centered Computing

Automated Semantic Reconstruction of IT Network Infrastructure: From Raw Logs to Ontology-driven Knowledge Graph

Meschi Maurizio, Trebino Nicolò

Academic Year 2025/2026

Advisor

Prof. Alessandro Armando
Prof. Enrico Russo

Co-advisor

Dr. Gianluca Ceccoli

Abstract

The continuous escalation of global cyber threats, coupled with the rising complexity of modern enterprise networks, has intensified the need for robust situational awareness and automated infrastructure modeling. These capabilities are essential for effectively analyzing potential attack paths and developing defensive strategies to safeguard critical infrastructure and data. However, establishing accurate and scalable models of operational environments remains a significant challenge. This difficulty primarily stems from the fragmented and noisy nature of conventional network discovery methods. Current security operations often rely on isolated architectural views that fail to provide a cohesive representation of the domain, thereby requiring resource-intensive manual intervention to synthesize.

To address these limitations, this thesis introduces a formal semantic framework designed for the automated reconstruction of heterogeneous IT network infrastructures into unified knowledge graphs. Utilizing Web Ontology Language (OWL) semantics and deductive reasoning, the proposed system standardizes multi-source network observations, automatically resolves conflicting evidence, and synthesizes an actionable, traceable infrastructure model. The framework is evaluated within different realistic virtualized scenarios, where automated data extraction and benchmarking against an engineered ground truth demonstrate its capacity to deliver consistent, scalable, and trustworthy environmental models for advanced security assessment and verification.

Table of Contents

Chapter 1

Introduction

Modern IT infrastructures have evolved into highly dynamic, complex, and heterogeneous ecosystems. The widespread adoption of virtualization, containerized microservices, and hybrid cloud environments has drastically increased both the density and structural fluidity of enterprise assets. While these advancements deliver unprecedented scalability and operational agility, they simultaneously introduce profound challenges for cybersecurity operations: as network perimeters dissolve and internal configurations shift continuously, maintaining comprehensive environmental visibility ceases to be a trivial administrative task and becomes, instead, a complex engineering problem in its own right [19]. A foundational principle in cybersecurity holds that complete structural awareness — a precise mapping of all hardware nodes, virtual instances, logical interfaces, and active subnets — is an absolute prerequisite for any meaningful security operation [18]. This architectural baseline is no longer merely a technical best practice; it is increasingly mandated by stringent regulatory and operational resilience frameworks. Standards and directives such as the NIST Cybersecurity Framework (CSF) 2.0 [23], the European Union's NIS2 Directive [16], and the Digital Operational Resilience Act (DORA) [17] explicitly require organizations to maintain comprehensive, continuous visibility over their Information and Communication Technology (ICT) assets and network topologies to ensure effective risk management. Whether performing vulnerability management, modeling attack paths, or conducting post-compromise incident response, security teams must rely on an accurate architectural baseline. Attack path analysis in particular requires a broad and accurate

inventory of systems, identities, permissions, vulnerabilities, and network relationships before any meaningful threat traversal can be computed [13]. Without such a baseline, defensive mechanisms degrade: assets go unpatched, blind spots accumulate, and the true blast radius of any exploit becomes impossible to evaluate.

1.1 Motivations and Objectives

The automated construction of a comprehensive network topology is fundamentally challenged by the inherent fragmentation and incompleteness of architectural data [12]. Standard discovery utilities typically operate within isolated functional silos, capturing specific layers of the infrastructure without providing a holistic context across the Open Systems Interconnection (OSI) reference model. For instance, link-layer (Layer 2) protocols can identify immediate physical neighbor relationships but possess no intrinsic awareness of network-layer (Layer 3) routing topologies or logical subnet structures. This technological separation forces defenders to work with disjointed pieces of telemetry that fail to convey the true end-to-end connectivity of the environment.

This baseline fragmentation is further compounded by vantage point dependency. Firewalls, NAT gateways, and access control lists fundamentally alter the behavior of active probes depending on where the discovery agent is positioned [3], making raw outputs inherently local, transient, and often contradictory rather than objective representations of the infrastructure [12].

The core insight motivating this thesis is that no single tool can reliably capture the ground truth of a modern, multi-segmented network. The proposed solution is therefore semantic data fusion: aggregating multi-perspective observations into a unified representation while preserving data provenance and reliability [2, 15] — a capability that serves as the foundational prerequisite for next-generation security workflows. Knowledge graphs offer a compelling paradigm for this purpose, enabling holistic processing of complex, multi-source data [20]. When grounded in a formal ontology, they further enable the encoding of domain-specific networking constraints and the application of automated logical reasoning — transforming conflict resolution from an ad-hoc procedural task into a principled,

auditable inference process [15, 14].

1.1.1 Thesis objectives

The primary objective of this thesis is to design a formal framework and develop a corresponding Proof of Concept (PoC) for the automated generation of an infrastructural Knowledge Graph (KG).
This primary objective is pursued through three explicit sub-goals: defining the operational scope and the formal constraints that structure the semantic model; designing and implementing a logical inference engine capable of executing data fusion, resolving conflicting facts, and computing data provenance; and formulating a validation methodology to assess the framework's accuracy and scalability across controlled scenarios.

1.2 Contributions

The main contributions of this thesis are the following:

1. **Ontology-Driven Representation Model:** Design of a tailored ontological framework for semantic network data fusion, enabling unified representation of fragmented cross-layer topologies and credibility-aware infrastructure reconciliation.

2. **Logical Data Fusion Strategy:** Design and development of a deterministic reconciliation methodology for ingesting noisy, partial, and heterogeneous discovery outputs into a coherent, conflict-free semantic representation of the target infrastructure.

3. **PoC Implementation:** A working end-to-end prototype that takes raw discovery telemetry as input and produces a structured, queryable Infrastructure KG as output.

4. **Evaluation Analysis:** Systematic experimental assessment of reconstruction fidelity under restricted information conditions.

1.3 Overview of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 establishes the theoretical foundations needed to contextualize the work. Chapter 3 reviews the existing literature on network discovery, data fusion methodologies, and the use of knowledge graphs in cybersecurity. Chapter 4 details the proposed framework, including the ontology formalization and the formal logic driving the inference engine. Chapter 5 describes the technical implementation of the PoC, including the design of the ontology. Chapter 6 presents the empirical evaluation across distinct test scenarios. Chapter 7 offers a critical assessment of future development directions, concluding with a summary of the key findings and their strategic implications for enterprise infrastructure defense.

Chapter 2

Background

This chapter outlines the framework's foundational technologies. Focusing on the Semantic Web, it provides the theoretical grounding necessary to understand the design and implementation choices discussed in subsequent chapters.

2.1 The Resource Description Framework

The Resource Description Framework (RDF) is a W3C [30] standard data model designed to represent information about resources in a machine-readable and interoperable way [7]. Unlike relational databases, which organize data in tables with fixed schemas, RDF represents knowledge as a directed, labeled graph composed of atomic statements called **triples**.

2.1.1 The triple model

Each RDF triple encodes a single atomic fact using the structure:

$$\langle \textit{subject},\ \textit{predicate},\ \textit{object} \rangle \tag{2.1}$$

where the *subject* is the resource being described, the *predicate* expresses a property or relationship, and the *object* is either another resource or a literal value. For example, the fact that a network interface has IP address `192.168.1.1` is encoded as:

```
:interface_eth0 :hasIPAddress "192.168.1.1"
```

Resources are identified by Internationalized Resource Identifiers (IRIs), ensuring global uniqueness across distributed datasets. A collection of RDF triples forms an **RDF graph**, where subjects and objects are nodes and predicates are directed edges. This graph-based structure is inherently flexible: new facts about any resource can be added without altering an existing schema, making RDF particularly suitable for integrating heterogeneous data sources with variable and unpredictable structure.

2.1.2 Triplestores

A **triplestore** is a database management system specifically designed to store, index, and query RDF graphs. Unlike relational database management systems (RDBMS), triplestores are optimized for graph traversal and pattern matching over triple data. They expose query interfaces through the SPARQL protocol (Section 2.2) and, in their more advanced variants, integrate semantic reasoning engines that can derive new facts from existing ones. Triplestores are the standard persistence layer for knowledge graph applications.

2.2 SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) is the W3C standard query language for RDF data [6]. Its syntax is modeled after SQL but operates on graph patterns rather than relational tables.

2.2.1 Graph pattern matching

The fundamental operation in SPARQL is **graph pattern matching**: a query specifies a set of triple patterns containing variables (denoted with a `?` prefix), and the engine returns all variable bindings for which matching triples exist in the graph. For example:

```
SELECT ?device ?ip WHERE {
    ?device :hasIPAddress ?ip .
    ?device a :NetworkDevice .
}
```

This query retrieves all resources classified as `NetworkDevice` along with their associated IP addresses.

2.2.2 SPARQL CONSTRUCT

Beyond data retrieval, SPARQL supports the `CONSTRUCT` query form, which generates new RDF triples from query results. Given a graph pattern and a template, `CONSTRUCT` produces a new RDF graph by instantiating the template with each set of variable bindings found. This makes SPARQL CONSTRUCT a powerful tool for rule-based inference: new facts can be derived and materialized into the graph based on patterns detected in existing data, without requiring a dedicated rule engine.

2.3 OWL and Ontological Reasoning

While RDF provides the data model, the Web Ontology Language (OWL) provides the means to formally define the schema and semantics of an RDF knowledge graph [4]. An OWL ontology is a formal specification of the concepts (classes), relationships (properties), and constraints (axioms) that govern a domain of knowledge.

2.3.1 Description Logic foundation

OWL is grounded in Description Logic (DL), a family of formal knowledge representation languages that are decidable fragments of first-order logic. This formal foundation gives OWL ontologies a precise, unambiguous semantics and enables automated reasoning: given a set of facts, the Assertional Box (A-Box) and a set of axioms, the Terminological Box (T-Box), a *reasoner* can automatically derive all logical consequences that follow from them.

A fundamental principle governing DL reasoning is the Open World Assumption (OWA). Unlike traditional relational databases where missing information is assumed false (Closed World Assumption), OWA dictates that knowledge is inherently incomplete. If a fact is not explicitly stated in the ontology, the reasoner considers it unknown rather than false.

In the context of this work, the ontology encodes the formal rules of network topology. For example, an OWL axiom can assert that a network interface belongs to exactly one physical device (cardinality constraint), or that two IP addresses on the same subnet must be reachable via the same gateway (domain rule). When data ingested from log files violates these axioms, the reasoner automatically detects the inconsistency, elevating conflict detection from manual inspection to formal logical inference.

2.3.2 OWL 2

Following the widespread adoption of the original OWL specification, several limitations regarding its expressivity and syntactic constraints became apparent. To address these shortcomings, the W3C introduced OWL 2, a major revision that provides richer modeling capabilities—such as extended datatypes, property chains, and more flexible annotations—while maintaining backward compatibility with its predecessor.

However, this increased expressive power comes at a cost. Applying logical reasoning across the full OWL 2 specification is computationally expensive. In the worst-case scenario, reasoning tasks are EXPTIME-complete, meaning the execution time can grow exponentially and become unmanageable as the volume of data increases. To solve this scalability issue, the W3C defined a set of OWL 2 profiles [5]. These are simplified, syntactic subsets of the language that restrict certain complex logical features in exchange for guaranteed, highly efficient processing times.

The specific profile adopted in this thesis is OWL 2 RL (Rule Language). This subset is expressly engineered to work seamlessly with rule-based inference engines. By limiting the complexity of allowable axioms, OWL 2 RL guarantees that all logical deductions can be resolved in polynomial time using standard forward-chaining rules. From an operational perspective, this mathematical trade-off is essential: it enables the framework to perform continuous, large-scale, and high-speed reasoning over massive network graphs without encountering computational bottlenecks.

2.4 RDF* and Metadata on Edges

A fundamental limitation of standard RDF is the reification problem: it is not possible to directly attach metadata (e.g., a confidence score, a provenance source, or a timestamp) to a triple without cumbersome workarounds. The standard solution, known as RDF reification, requires replacing a single triple with four additional triples to represent the original statement as a resource, which is verbose and inefficient to query.

2.4.1 The RDF* extension

RDF* (pronounced "RDF star") is an extension to the RDF data model, currently under W3C standardization as RDF 1.2, that addresses this limitation by allowing triples themselves to appear as the subject or object of other triples [9]. This enables concise, first-class annotation of statements. The syntax uses double angle brackets to embed a triple as a resource:

```
<< :interface_eth0 :connectedTo :switch_core >>
    :confidenceWeight  0.91 ;
    :sourceLog         :nmap_scan_2024 .
```

This construct directly annotates the structural edge `:interface_eth0 :connectedTo :switch_core` with a numerical confidence weight and a provenance reference, without generating auxiliary triples.

2.5 Summary

Table 2.1 summarizes the Semantic Web technologies introduced in this chapter and their specific role within the proposed framework.

Table 2.1: Semantic Web technologies and their role in the framework.

Technology	**Standard**	**Role in Framework**
RDF	W3C Rec.	Core data model for topology representation
Triplestore	—	Persistence and graph query layer
SPARQL	W3C Rec.	Graph querying and rule-based inference
OWL 2 RL	W3C Rec.	Ontological constraints and automated reasoning
RDF*	W3C Draft	Confidence metadata on structural edges

Chapter 3

State of the Art

The automated reconstruction of network topology from disparate, heterogeneous log files—such as operating system network configurations, network device CLI outputs, and active scanner reports—introduces a fundamental challenge in data integration: resolving structural and semantic conflicts. When multiple independent logs describe the same underlying infrastructure, they frequently yield incomplete, overlapping, or directly contradictory information.

This chapter surveys the state of the art in topology reconstruction, focusing specifically on how existing methodologies handle multi-source data fusion and conflict resolution. It demonstrates the limitations of procedural and probabilistic approaches and validates the use of formal ontologies and KGs as the modern standard for resolving semantic conflicts in IT infrastructure modeling.

3.1 Topology Reconstruction and Data Fusion

This section examines the two dominant paradigms for integrating heterogeneous network data into topology representations: rule-based procedural approaches and machine learning-based inference, highlighting the fundamental limitations that motivate the adoption of a semantic framework.

3.1.1 Ad-hoc and procedural data integration

Historically, the integration of network data into CMDBs or topology graphs has relied on procedural scripting and heuristic rules. Traditional IT systems handle data collisions using prioritization rules (e.g., assigning a higher "trust score" to an SNMP trap over a syslog message). While functional for small-scale operations, procedural integration lacks a formal foundation. When integrating arbitrary offline logs, hard-coded priority rules fail to capture the complex interdependencies of network topology. The data integration literature [2, 8] shows that procedural fusion across heterogeneous sources with variable quality leads to fragmented and inconsistent representations, as the conflict resolution logic cannot adapt to unexpected data variations.

3.1.2 Machine learning and probabilistic approaches

Recent advancements have attempted to resolve topological ambiguities using Machine Learning (ML). Probabilistic and learning-based models have been employed to infer missing connections or resolve conflicting edges in network graphs [11].
However, ML-based topology fusion suffers from a major drawback in the context of infrastructure engineering: the lack of explainability and exactness. Network routing and switching operate on strict, deterministic rules. Probabilistic ML models cannot guarantee that the resolved topology adheres to fundamental networking constraints, making them unsuitable for exact infrastructure reconstruction where logical conflicts must be definitively resolved, not merely approximated [20].

3.2 Information Extraction from Heterogeneous Logs

Before resolving conflicts, raw log text must be parsed into structured data. The evolution of Large Language Models (LLMs) has revolutionized log parsing. A recent exploratory study by Ma et al. [22] introduced LLMParser, demonstrating that modern LLMs can extract structured entities from highly unstructured software logs with near-perfect accuracy, eliminating the need for rigid regular expressions. Wang et al. [21] further show that this capability extends to the networking domain, where LLMs can parse technical device

documentation and CLI outputs to extract key configuration data. Despite this, information extraction alone does not solve the topological problem. The parser provides the raw material for the conflict but lacks the domain-specific reasoning capabilities to determine the ground truth. This semantic gap necessitates a higher-level knowledge representation framework [15].

3.3 Ontologies for Semantic Conflict Resolution

The core justification for employing ontologies in topology reconstruction lies in their ability to formalize domain knowledge and apply logical reasoning to resolve inconsistencies. KGs backed by OWL are the state-of-the-art paradigm for complex data integration in cybersecurity and networking [20].

3.3.1 Formal modeling of networking constraints

Ontologies allow for the explicit definition of network infrastructure rules using DL. Corcho et al. [14] presented a high-level ontology network for ICT infrastructures that successfully models these relationships at an industrial scale. By defining strict constraints (e.g., disjoint classes, cardinality limits), the ontology acts as a mathematical model of a valid network. More recently, Tailhardat et al. [24] introduced NORIA-O, an ontology specifically designed for anomaly detection and incident management, proving the viability of semantic models for complex network topologies.

3.3.2 Automated reasoning and inconsistency detection

When parsed data from conflicting logs is ingested into the KG as RDF triples, a semantic reasoner can automatically evaluate the graph against the ontology's rules. In the modern state of the art, the W3C standard SHACL (Shapes Constraint Language) [10] is heavily utilized to validate RDF graphs. Instead of writing procedural code to cross-check IP addresses and VLANs, the validation logic is abstracted into shape graphs, instantly flagging topological contradictions imported from disparate log files. In the modern state of the art, SHACL is heavily utilized to validate RDF graphs against ontological constraints. Re-

cent work on ICT network ontologies, such as NORIA-O, employs SHACL specifically for post-deployment data quality validation on network infrastructure graphs, demonstrating its applicability in operational networking contexts.

3.3.3 Conflict resolution strategies in knowledge graphs

When an inconsistency is detected, rule-based systems based on Semantic Web Rule Language (SWRL) [1] can be triggered to resolve the conflict based on topological context. For example, a semantic rule can infer that an active MAC address table log overrides a static configuration log regarding a port connection. This approach elevates conflict resolution from arbitrary data filtering to formal, logical inference [15].

3.4 Synthesis and Positioning of This Thesis

The review of the literature reveals a clear trajectory: while modern parsers excel at extracting data from arbitrary logs [22], traditional data structures are inadequate for resolving the multi-dimensional conflicts that arise when fusing independent sources.
This thesis positions itself precisely at this gap. It assumes the availability of arbitrary network logs and utilizes semantic web technologies not merely as a storage format, but as an active computational tool for conflict resolution.
By encoding the rules of networking into an ontology [14, 24], this work leverages automated reasoning to logically resolve contradictions between log files, producing a logically consistent infrastructure graph

Chapter 4

Methodology

This chapter delineates the formal methodology and structural design of the proposed semantic reconstruction framework. Moving from the high-level objectives established in the introduction, the following sections formalize the process of converting raw, multi-source infrastructure telemetry into a cohesive, deterministic, and queryable knowledge graph. The architecture is engineered as a multi-layered synthesis pipeline designed to systematically eliminate observation noise, resolve spatial inconsistencies, and establish structural truth across disparate network layers.

The chapter is organized as follows: first, a concrete example is presented to illustrate the intrinsic limitations of isolated, host-centric network scans. Second, the structural boundaries and formal constraints of the domain are defined. Finally, the four-layer architectural pipeline is detailed, with a specific focus on the deductive reasoning engine and constraint validation mechanisms that enable automated topological reconciliation.

4.1 Running Example

To ground the technical challenges of infrastructure discovery in a realistic scenario, consider the corporate network topology illustrated in Figure 4.1. This baseline infrastructure represents a standard segmented network architecture featuring an edge router, specialized subnets, client workstations, and enterprise servers.

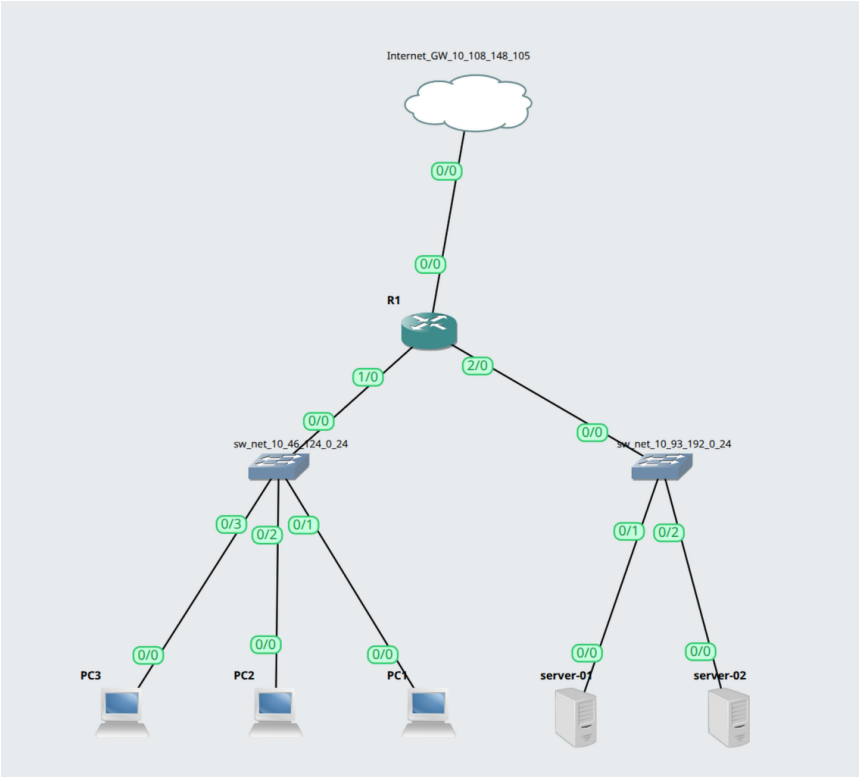

Figure 4.1: Network infrastructure.

The physical and logical structure of this target network is composed of the following entities:

- **Central Gateway:** A core Layer 3 routing entity (`R1`) serving as the central gateway for the environment. It maintains an upstream link via interface `0/0` to an external gateway network.
- **Client Subnet:** Router interface `1/0` is logically configured to govern the client broadcasting domain, mapped through a local Layer 2 switch. Three distinct client

endpoints (`PC1`, `PC2`, and `PC3`) are interconnected via physical ports `0/1`, `0/2`, and `0/3` respectively.

- **Server Subnet:** Router interface `2/0` isolates the enterprise asset domain, managed via a secondary Layer 2 switch. This switch interconnects two critical server nodes (`server-01` and `server-02`) through physical ports `0/1` and `0/2`.

At first glance, this architecture appears straightforward, elementary, and almost purely academic in nature. Its linear layout and minimal node density suggest that mapping its active assets should be a straightforward, deterministic exercise. However, this apparent structural simplicity is fundamentally deceptive.

In live operational environments, topological minimalism does not equate to observational clarity. Even within an elemental layout consisting of a single routing entity and two isolated broadcasting domains, decentralized discovery mechanisms remain highly vulnerable to cross-layer ambiguities caused by firewall filtering policies, blocked probing traffic, stale or corrupted ARP tables, and inconsistent neighbor discovery information. The localized view obtained by a probing agent at any single vantage point is therefore inherently decoupled from the global architectural truth. As a result, even a trivial network skeleton can conceal significant semantic conflicts, demonstrating that structural visibility depends on logical data reconciliation rather than simple topological scale.

4.2 Problem Formalization and Operational Boundaries

To ensure a well-defined and computationally tractable reconstruction process, the operational scope of this framework is strictly constrained to the lower layers of the network architecture. Specifically, the system focuses exclusively on Layer 2 (Data Link) and Layer 3 (Network) of the OSI reference model. All higher-layer abstractions—such as transport-layer ports, application services, and software configurations—are explicitly excluded from the current scope of this work.

The framework is designed to ingest, process, and reconcile data related strictly to the foundational structural skeleton of the infrastructure, which encompasses:

- **Physical Devices:** The hardware nodes acting as hosts, switches, or routers within the environment.
- **Link-Layer Components (Layer 2):** Hardware MAC addresses and local neighbor relationships within immediate broadcasting domains.
- **Network-Layer Components (Layer 3):** Logical IP addresses, subnet configurations, default gateways, and routing paths.

By defining these boundaries, the framework eliminates application-layer noise and focuses the inference engine entirely on resolving the physical and logical connectivity of the network infrastructure.

4.3 The Semantic Reconstruction Framework

The core of the proposed methodology lies in a modular, multi-layered synthesis architecture designed to transform heterogeneous, multi-source network discovery inputs into a globally consistent knowledge base. As illustrated in Figure 4.2, the framework is structured across four sequential layers, establishing a deterministic, unidirectional data pipeline that progresses from raw telemetry collection to formal infrastructure knowledge representation.

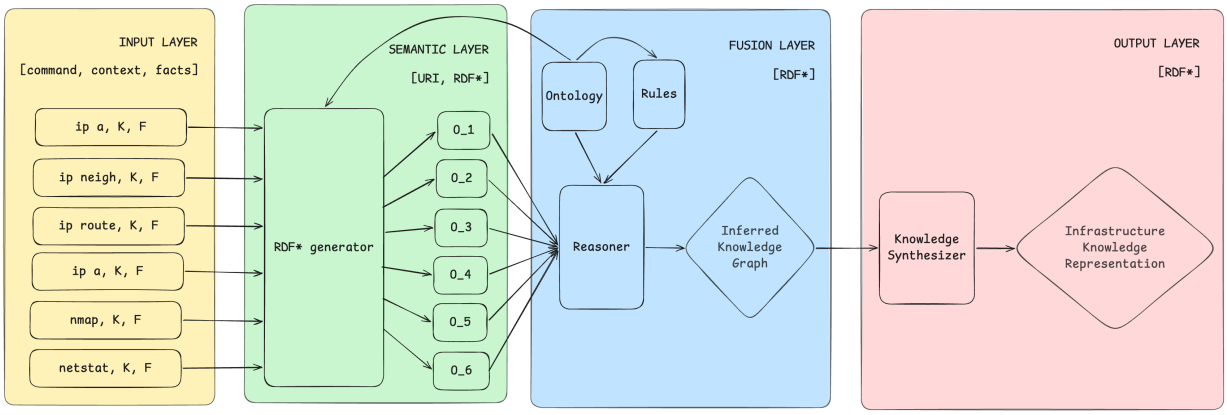

Figure 4.2: The multi-layered semantic reconstruction framework architecture.

The architectural pipeline orchestrates the data flow through the following macro-phases:

- **Input Layer:** Responsible for gathering raw structural telemetry, command execution outputs, and spatial execution contexts from decentralized network sources.
- **Semantic Layer:** Ingests the raw, heterogeneous telemetry and utilizes a parsing and generation engine to map the disparate text formats into standardized, IRI-based RDF triple segments.
- **Fusion Layer:** Serves as the logical core of the framework. It ingests the individual semantic sub-graphs into an inference engine governed by a domain-specific ontology and deductive rules, deploying a reasoner and a constraint validator to resolve topological conflicts and infer missing connectivity elements.
- **Output Layer:** Acts as the final consolidation phase where a knowledge synthesizer aggregates the validated semantic facts to produce a unified, queryable knowledge graph representing the original infrastructure representation.

Each of these operational layers and their underlying logical mechanics will be examined in detail within the following subsections.

4.3.1 Input layer: raw data and context acquisition

The operational cycle of the framework begins at the Input Layer, which models the ingestion of decentralized, unstructured, or semi-structured diagnostic telemetry extracted from the infrastructure. To programmatically reconcile observations from varying vantage points, the framework cannot treat discovery outputs as isolated text streams; instead, it must formalize each ingestion instance by binding the raw structural evidence to its specific operational metadata.

Let I be the chronological sequence of input payloads ingested by the system during an operational window, denoted as:

$$I = \langle \omega_1, \omega_2, \ldots, \omega_m \rangle \tag{4.1}$$

Each individual input payload $\omega \in \Omega$, where Ω represents the universe of all possible system inputs, is formally structured as a 3-tuple:

$$\omega = (\pi, k, \mathbf{f}) \tag{4.2}$$

The components of the tuple are defined as follows:

- **Command (π):** Let Π be the set of all supported diagnostic and network commands (e.g., `ip route`, `nmap`, `dig`). Then, $\pi \in \Pi$ denotes the specific command executed to generate the telemetry.
- **Context (k):** The operational environment in which the command c was executed is defined as a 5-tuple:

  $$k = (h, r, o, u, \tau) \tag{4.3}$$

  Where:

  - $h \in H$: The Fully Qualified Domain Name (FQDN) of the executing host.
  - $r \in R$: The functional role of the machine within the infrastructure (e.g., *Gateway*, *Database Server*).
  - $o \in O$: The Operating System identifier and version.
  - $u \in U$: The executing user account or privilege context.
  - $\tau \in \mathbb{T}$: The exact timestamp of the execution, mapping the event to a continuous temporal domain.
- **Facts ($\mathbf{f}$):** A finite sequence of structured records systematically extracted and normalized from the raw standard output of command π. Let n be the total number of extracted records; then $\mathbf{f}$ is an n-tuple:

  $$\mathbf{f} = \langle \phi_1, \phi_2, \ldots, \phi_n \rangle \tag{4.4}$$

  To rigorously capture the nested structure of these records, we model them as sets of attribute-value pairs. Let A be the finite set of all domain attributes, and let P be the set of primitive values (e.g., strings, integers, booleans). We define the universal set of values $\mathcal{V}$ inductively as the smallest set such that:

1. $P \subseteq \mathcal{V}$ (it contains all primitive values);
2. if $v_i \in \mathcal{V}$ for $i = 1, \ldots, k$, then the finite sequence $\langle v_1, \ldots, v_k \rangle \in \mathcal{V}$.
3. if $f : A \rightharpoonup \mathcal{V}$ is a partial function mapping attributes to values, then $f \in \mathcal{V}$.

A single fact ϕ_i in the sequence $\mathbf{f}$ is formally defined as one such partial function:

$$\phi_i : \mathcal{A} \rightharpoonup \mathcal{V} \tag{4.5}$$

For instance, a simple normalized fact extracted from an `ip a` command output can be explicitly represented as a mapping to primitive values:

$$\begin{aligned}\phi_i = \{ & \\ & (\texttt{"ifname"},\ \texttt{"eth0"}), \\ & (\texttt{"address"},\ \texttt{"44:42:c3:b0:68:56"}), \\ & (\texttt{"mtu"},\ 1500) \\ \}\end{aligned}$$

Because $\mathcal{V}$ is defined inductively to include finite sequences and partial functions, nested structures such as arrays of sub-records are naturally supported. For example, representing the `addr_info` attribute:

$$\begin{aligned}\phi_j = \{ & \\ & (\texttt{"ifname"},\ \texttt{"eth0"}), \\ & (\texttt{"addr\_info"},\ \langle \psi_1 \rangle) \\ \}\end{aligned}$$

where $\psi_1 \in (A \rightharpoonup \mathcal{V})$ is itself a partial function representing the nested IPv4 address record:

$$\begin{aligned}\psi_1 = \{ & \\ & (\texttt{"family"},\ \texttt{"inet"}), \\ & (\texttt{"local"},\ \texttt{"10.46.124.2"}), \\ & (\texttt{"prefixlen"},\ 24) \\ \}\end{aligned}$$

4.3.1.1 Concrete instantiation: `ip neigh` on `PC1`

To ground the formal definitions above in a concrete operational instance, consider the execution of the `ip neigh` command on endpoint `PC1` of the motivational network introduced in Section 4.1. This command queries the ARP table of the host, exposing the Layer 2 adjacencies observed from its local broadcasting domain. This single observation yields a well-defined input payload $\omega_{\texttt{neigh\_PC1}} = (\pi, k, f)$, where the executed command is $c = \texttt{ip neigh}$, and the operational context is fully qualified as:

$$k = (\texttt{PC1},\ \textit{client},\ \texttt{alpine},\ \texttt{root},\ \texttt{2026-05-21T17:36:35Z}) \tag{4.6}$$

The extracted fact sequence $\mathbf{f} = \langle \phi_1, \phi_2, \phi_3 \rangle$ comprises three records, each encoding a distinct ARP entry observed by `PC1` on interface `eth0`.
The first fact ϕ_1 encodes the neighbor entry for destination `10.46.124.1`:

$$\begin{aligned} \phi_1 = \{ & \\ & (\texttt{"dst"},\ \texttt{"10.46.124.1"}), \\ & (\texttt{"dev"},\ \texttt{"eth0"}), \\ & (\texttt{"lladdr"},\ \texttt{"02:df:7f:e2:2d:b0"}), \\ & (\texttt{"state"},\ \langle \texttt{"PERMANENT"} \rangle) \\ \} & \end{aligned} \tag{4.7}$$

Note that the `"state"` attribute is formally represented as a sequence $\langle \dots \rangle$; this explicitly leverages the inductive sequence rule of our universal value set $\mathcal{V}$, allowing for multiple state flags.
The second and third facts, ϕ_2 and ϕ_3, encode dynamically resolved neighbor entries cor-

responding to peer endpoints within the same broadcasting domain:

$$\begin{aligned}
\phi_2 = \{ & \\
& (\texttt{"dst"},\ \texttt{"10.46.124.4"}), \\
& (\texttt{"dev"},\ \texttt{"eth0"}), \\
& (\texttt{"lladdr"},\ \texttt{"64:46:ee:6c:ad:ae"}), \\
& (\texttt{"state"},\ \langle \texttt{"REACHABLE"} \rangle) \\
\} &
\end{aligned} \tag{4.8}$$

$$\begin{aligned}
\phi_3 = \{ & \\
& (\texttt{"dst"},\ \texttt{"10.46.124.3"}), \\
& (\texttt{"dev"},\ \texttt{"eth0"}), \\
& (\texttt{"lladdr"},\ \texttt{"88:d3:d6:0b:69:5e"}), \\
& (\texttt{"state"},\ \langle \texttt{"REACHABLE"} \rangle) \\
\} &
\end{aligned} \tag{4.9}$$

4.3.2 Semantic Layer: RDF generation

Formalization of the reference ontology

To provide a formal, unambiguous semantic foundation, we introduce a domain-specific reference ontology. In the context of DL, the ontology acts as T-Box, establishing the foundational vocabulary and constraints.

Let $\mathcal{U}_{IRI}$ denote the universal, infinite set of valid IRIs. The ontology, denoted as Σ, is defined as a 4-tuple:

$$\Sigma = (\mathcal{C}, \mathcal{D}, \mathcal{P}, \mathcal{A}) \tag{4.10}$$

Where:

- $\mathcal{C} \subset \mathcal{U}_{IRI}$ is a finite set of taxonomic Concepts (classes) representing infrastructure entities (e.g., `Host`, `NetworkInterface`, `IPAddress`).
- $\mathcal{D} \subset \mathcal{U}_{IRI}$ is a finite set of supported Datatypes defining the data value space (e.g., `xsd:string`, `xsd:integer`, `xsd:dateTime`).

- $\mathcal{P} \subset \mathcal{U}_{IRI}$ is a finite set of Properties (roles), strictly partitioned into two disjoint subsets: object properties $\mathcal{P}_{obj}$ (linking instances to instances) and datatype properties $\mathcal{P}_{data}$ (linking instances to literal values). Thus, $\mathcal{P} = \mathcal{P}_{obj} \cup \mathcal{P}_{data}$ and $\mathcal{P}_{obj} \cap \mathcal{P}_{data} = \emptyset$.
- $\mathcal{A}$ is a finite set of structural Axioms defining logical constraints, including concept subsumption ($C_1 \sqsubseteq C_2$), role domain constraints ($\exists p.\top \sqsubseteq C$), and role range constraints ($\top \sqsubseteq \forall p.C$ for object properties, or $\top \sqsubseteq \forall p.D$ for datatype properties, where $p \in \mathcal{P}$ and $D \in \mathcal{D}$).

Semantic lifting and structural validation

The system performs a *Semantic Lifting* process, translating the isolated input payloads (ω) into formalized knowledge assertions. These assertions constitute the A-Box, which populates the ontology.

- **Ontology-bound graph space:** Let $\mathcal{G}$ be the universal space of all constructible RDF graphs, and let $\mathcal{L}_{\mathcal{D}}$ be the universal set of all valid literal values typed strictly by the datatypes in $\mathcal{D}$. We define $\mathcal{G}_{\Sigma}^{syntactic} \subseteq \mathcal{G}$ as the subspace of structurally compliant graphs. A graph $G \in \mathcal{G}_{\Sigma}^{syntactic}$ strictly adheres to the structural rules defined in the T-Box Σ.

  Specifically, for every RDF triple $t = (s, p, o) \in G$, the following structural assertions must hold true:

  1. The predicate must be a defined property: $p \in \mathcal{P}$.
  2. The subject must be an identified resource: $s \in \mathcal{U}_{IRI}$.
  3. If $p \in \mathcal{P}_{obj}$, the object must also be a resource: $o \in \mathcal{U}_{IRI}$.
  4. If $p \in \mathcal{P}_{data}$, the object must be a properly typed literal: $o \in \mathcal{L}_{\mathcal{D}}$.

  Furthermore, the instances s and o (when an IRI) must be typed as belonging to a class $C \in \mathcal{C}$ via the `rdf:type` predicate.

- **The observation tuple:** An individual semantic observation O_i is generated for each payload and formalized as an ordered pair:

$$O_i = (id_i, G_i) \in \mathcal{U}_{IRI} \times \mathcal{G}_{\Sigma}^{syntactic} \tag{4.11}$$

  Where id_i is a uniquely minted IRI identifying the observation event (acting as a metadata anchor for provenance), and G_i is the corresponding A-Box RDF graph containing the topological and state data inferred from the payload.

- **Semantic Lifting Operator ($\mathcal{M}$):** The mapping from an input payload to a semantic observation is governed by the operator $\mathcal{M}$, operating as a mapping function:

$$\mathcal{M} : \Omega \to \mathcal{U}_{IRI} \times \mathcal{G}_{\Sigma}^{syntactic} \tag{4.12}$$

  Such that $\mathcal{M}(\omega_i) = O_i$. This operator guarantees correctness-by-construction concerning the syntactic constraints of the output graph.

Final observation set: The set of structurally valid observations $\mathcal{O}$ generated during an ingestion cycle is defined as the image of the input sequence I under the operator $\mathcal{M}$:

$$\mathcal{O} = \{\mathcal{M}(\omega_i) \mid 1 \leq i \leq m\} \tag{4.13}$$

4.3.2.1 Concrete instantiation: semantic lifting

Relevant T-Box fragment for ARP observations

Before applying the operator $\mathcal{M}$, we introduce the fragment of the reference ontology Σ exercised when lifting an `ip neigh` payload. Table 4.1 summarises the relevant concepts and properties; all identifiers belong to the `EX` namespace (`:` $\equiv$ `http://www.example.com/EX/`).

Table 4.1: Formal T-Box specification schema.

Symbol	Ontology IRI	Domain	Range / Type
CONCEPTS ($\mathcal{C}$)			
C_{dev}	`:PhysicalDevice`	—	`owl:Class`
C_{iface}	`:NetworkInterfaceL2`	—	`owl:Class`
C_{addr}	`:NetworkAddress`	—	`owl:Class`
C_{net}	`:GlobalNetworkL3`	—	`owl:Class`
OBJECT PROPERTIES ($\mathcal{P}_{\text{obj}}$)			
p_{hasL2NWI}	`:hasL2NWI`	C_{dev}	C_{iface}
p_{netprop}	`:hasNetworkProperty`	C_{iface}	C_{addr}
p_{connects}	`:connectsTo`	C_{addr}	C_{net}
DATATYPE PROPERTIES ($\mathcal{P}_{\text{data}}$)			
p_{iface}	`:hasIface`	C_{iface}	`xsd:string`
p_{mac}	`:hasMAC`	C_{iface}	`xsd:string`
p_{addr}	`:hasAddress`	C_{addr}	`xsd:string`
p_{cidr}	`:hasCIDR`	C_{net}	`xsd:string`

Application on `ip neigh` on `PC1`

We now apply $\mathcal{M}$ to $\omega_{\texttt{neigh\_PC1}}$ (Section 4.3.1.1), producing the observation $O_{\texttt{neigh\_PC1}} = (id_{\texttt{neigh\_PC1}}, G_{\texttt{neigh\_PC1}})$.

- **Observation identifier:**

$$id_{\texttt{neigh\_PC1}} \;=\; \texttt{:obs\_PC1\_ip\_neigh} \tag{4.14}$$

- **A-Box graph $\mathbf{G}_{\texttt{neigh\_PC1}}$:**

  The operator $\mathcal{M}$ processes each ARP fact ϕ_i in $\mathbf{f} = \langle \phi_1, \phi_2, \phi_3 \rangle$ (Section 4.3.1.1) translating its fields into typed RDF triples grounded in the T-Box fragment presented above.

  The host device node, inferred from the execution context metadata, and the local interface node, derived from the `dev` field shared across all three facts, are instantiated once within the assertion space:

(:dev_PC1, rdf:type, :PhysicalDevice)

(:Iface_PC1_eth0, rdf:type, :NetworkInterfaceL2)

(:Iface_PC1_eth0, :hasIface, "eth0"^^xsd:string) (4.15)

(:dev_PC1, :hasL2NWI, :Iface_PC1_eth0)

The `lladdr` field yields a `:NetworkInterfaceL2` individual carrying the observed MAC address, while the `dst` field yields a `:NetworkAddress` individual. The two are linked via `:hasNetworkProperty`, and the address is further connected to a `:GlobalNetworkL3` individual reconstructed from the destination IP. Instantiating this pattern for ϕ_1 (4.7):

(:NeighIface_02, :hasMAC, "02:df:7f:e2:2d:b0"^^xsd:string) (4.16)

(:Addr_10.46.124.1, :hasAddress, "10.46.124.1"^^xsd:string) (4.17)

(:NeighIface_02, :hasNetworkProperty, :Addr_10.46.124.1)

(:Net_10.46.124.0_24, rdf:type, :GlobalNetworkL3)

(:Net_10.46.124.0_24, :hasCIDR, "10.46.124.0/24"^^xsd:string) (4.18)

(:Addr_10.46.124.1, :connectsTo, :Net_10.46.124.0_24)

The same pattern applies to ϕ_2 (4.8) and ϕ_3(4.9).

- **Structural Validity:**
  Every triple $(s, p, o) \in G_{\texttt{neigh\_PC1}}$ satisfies the four constraints of $\mathcal{G}_{\Sigma}^{syntactic}$: all predicates belong to $\mathcal{P}$, subjects and object IRIs carry an explicit `rdf:type` to a class in $\mathcal{C}$, and all literals are typed by a datatype in $\mathcal{D}$. Therefore $\mathcal{M}(\omega_{\texttt{neigh\_PC1}}) = O_{\texttt{neigh\_PC1}} \in \mathcal{U}_{IRI} \times \mathcal{G}_{\Sigma}^{syntactic}$, and $O_{\texttt{neigh\_PC1}}$ is admitted into $\mathcal{O}$.

Credibility Manager formalization

To manage the inherent uncertainty and varying degrees of reliability in infrastructural telemetry, the methodology introduces a formal epistemic credibility model. Since standard

RDF 1.0 does not natively support metadata annotations on relationships, we extend the structurally compliant graph space $\mathcal{G}_{\Sigma}^{syntactic}$ using the RDF* specification. This extension permits individual triples to act as the subject of subsequent metadata assertions.
During the semantic lifting phase, every structural triple $t = (s, p, o) \in G_i$ generated from an input payload $\omega = (\pi, k, \mathbf{f})$ is assigned a deterministic credibility score $\chi(t) \in (0, 1]$.
The credibility of an assertion is mathematically formalized as a bipartite function dependent on two orthogonal dimensions: the intrinsic reliability of the source tool and the empiric proximity of the observation.
1. Trust in source (w_{src}): The trust in source quantifies the inherent reliability of the specific diagnostic command $\pi \in \Pi$ that originated the fact. We define a mapping function $w_{src} : \pi \to (0, 1]$ which assigns a static weight to each tool based on its operational nature. For example, a local configuration command yielding a deterministic system state acts as a highly trusted source, whereas an active remote scanner may be subject to partial network visibility or active spoofing defenses, resulting in a lower trust factor.
2. Observation modality (w_{mod}): The observation modality assesses the epistemic distance between the executing context and the observed entity at the exact moment of telemetry generation. Let $\Gamma = \{\texttt{Local}, \texttt{Remote}, \texttt{Inferred}\}$ be the finite set of valid observation modalities. We define an evaluation function $w_{mod} : \Gamma \to (0, 1]$ mapped as follows:

- `Local`: Direct observation of a physical or locally configured property on the executing host.
- `Remote`: Indirect observation of an external entity over the network (e.g., dynamic ARP caches or remote port scanning).
- `Inferred`: A topological entity or property that has not been empirically observed but generated through heuristic deduction based on infrastructural constraints (e.g., an L3 subnet generated from a local IP).

Credibility calculation and RDF* mapping: For every extracted triple t, let $w_{mod}^{t} \in \Gamma$ be its designated observation modality derived from the fact extraction rules. The final

epistemic credibility score $\chi(t)$ is defined as the mathematical product of the source trust and the observation type:

$$\chi(t) = w_{src}(\pi) \times w_{mod}(\gamma)$$

To persist this metadata within the unified Knowledge Graph, the Semantic Lifting operator $\mathcal{M}$ embeds the calculated score into the A-Box using the RDF* nested syntax. Let $p_{cred} \in \mathcal{P}_{data}$ be the designated datatype property within the T-Box Σ established to express epistemic weight. The resulting graph incorporates the reified structural assertion as:

$$\big((s, p, o), p_{cred}, \chi(t)\big)$$

4.3.2.2 Concrete instantiation: credibility application

To operationalize the epistemic credibility model, we establish a baseline matrix of deterministic weights for both w_{src} and w_{mod}, derived from the operational limitations and nature of standard cyber-diagnostic tools. Table 4.2 outlines the reference values adopted in this methodology.

Table 4.2: Reference weights for trust in source (w_{src}) and observation modality (w_{mod})

Trust in Source (w_{src})			**Observation Modality (w_{mod})**	
Source Category	**Example Tool**	**Weight**	**Modality**	**Weight**
Local OS Config	`ip a`, `ifconfig`	1.00	Local	1.00
Network HW State	`show ip arp` (Cisco)	0.95	Remote	0.80
Local OS State	`ip neigh`, `netstat`	0.90	Inferred	0.40
Active Scanner	`nmap`	0.80		

The proposed weight distribution relies on an original heuristic derived from logical deductions regarding network behavior. Local configurations (1.00) represent the unquestionable ground truth of the system. Regarding dynamic states, hardware network devices (e.g., core routers) are assigned a higher credibility (0.95) compared to local endpoints (0.90); core devices actively route traffic across the subnet, strictly maintaining updated state tables, whereas local host caches are populated opportunistically and are more prone to

stale entries. Active scanners exhibit the lowest source credibility (0.80) due to their susceptibility to firewalls, packet loss, or rate-limiting mechanisms.
Furthermore, the observation modality dictates a degradation in trust: direct local reads (1.00) are preferred over remote observations mediated by third-party nodes (0.80). Finally, inferred entities carry a substantial penalty (0.40) because, while structurally logical, their exact parameters (e.g., assuming a specific CIDR mask for an inferred subnet) are inherently uncertain.
To illustrate the Semantic Lifting operator $\mathcal{M}$ in action, we extend the concrete `ip neigh` instantiation of $O_{\texttt{neigh\_PC1}}$ introduced in Section 4.3.2.1, demonstrating how the epistemic credibility score $\chi(t)$ is calculated and subsequently embedded into each RDF triple via RDF* annotations.

- *Example 1: Direct local observation* (4.15)
  The host device node `:dev_PC1` and the local interface node `:Iface_PC1_eth0` are instantiated from the execution context metadata and the `dev` field shared across all facts in $\omega_{\texttt{neigh\_PC1}}$. The interface name `eth0` is a property of the executing host itself, read directly from its kernel network stack: no network exchange is involved and no algebraic deduction is required; the triple is therefore classified as `Local`:

  - w_{src} (`ip neigh` $\rightarrow$ Local OS State) $= 0.90$
  - w_{mod} (`Local` — directly read from the executing host) $= 1.00$
  - $\chi(t) = 0.90 \times 1.00 = 0.90$

$$\Big(\big(\texttt{:Iface\_PC1\_eth0},\ \texttt{:hasIface},\ \texttt{"eth0"}\big),\ p_{cred},\ 0.90\Big) \tag{4.19}$$

  The score of 0.90 reflects almost absolute epistemic certainty: The only remaining source of uncertainty is the tool's classification (`ip neighbor` as *Local OS state*, $w_{src} = 0.90$ instead of 1.00), since the interface name could theoretically result from a virtual or containerized configuration rather than from physical hardware directly managed by the OS. Compared to a static configuration command such as

`ip a`—classified as *Local OS Config* with $w_{src} = 1.00$ —we obtain the semantic distinction between a state dynamically maintained by the kernel and a deterministic configuration property.

- *Example 2: Remote neighbor observation*
  Fact ϕ_1 was extracted from the ARP cache of `PC1`: the local kernel observed a neighbor at `10.46.124.1` reachable through interface `eth0`, with hardware address `02:df:7f:e2:2d:b0`. Although the command itself (`ip neigh`) is classified as a *Local OS State* tool, the observed entity is an *external* host whose reachability was learned dynamically through ARP exchanges on the local segment. The two triples produced for ϕ_1 therefore receive different credibility scores, reflecting their distinct epistemic distances.

  – **MAC address of the neighbor:** (4.16)
    The `lladdr` field reports the hardware address advertised by the remote host in an ARP reply. The value is held in the local ARP cache, but it was obtained indirectly from a network exchange and could in principle be spoofed:

    * w_{src} (`ip neigh` → Local OS State) = 0.90
    * w_{mod} (`Remote` — value sourced from a neighbor ARP reply) = 0.80
    * $\chi(t) = 0.90 \times 0.80 = 0.72$

    $$\Big(\big(\texttt{:NeighIface\_02},\ \texttt{:hasMAC},\ \texttt{"02:df:7f:e2:2d:b0"}\big),\ p_{cred},\ 0.72\Big) \quad (4.20)$$

  – **IP address of the neighbor:** (4.17)
    The `dst` field carries the IP address of the remote host as recorded in the local ARP cache, again the product of a dynamic, network-level observation:

    * w_{src} (`ip neigh` → Local OS State) = 0.90
    * w_{mod} (`Remote` — address resolved via ARP on the wire) = 0.80
    * $\chi(t) = 0.90 \times 0.80 = 0.72$

    $$\Big(\big(\texttt{:Addr\_10.46.124.1},\ \texttt{:hasAddress},\ \texttt{"10.46.124.1"}\big),\ p_{cred},\ 0.72\Big) \quad (4.21)$$

- *Example 3: Inferred network segment* (4.18)

  The `:GlobalNetworkL3` individual `:Net_10.46.124.0_24` and its associated CIDR literal are *not* directly reported by `ip neigh`. They are algebraically deduced by the Semantic Lifting operator $\mathcal{M}$ from the destination address `10.46.124.1` combined with the implicit assumption of a `/24` prefix — a standard inference applied in absence of an explicit netmask in the ARP payload. No packet was ever exchanged that explicitly stated the network boundary; the triple is therefore classified as `Inferred`:

  - w_{src} (`ip neigh` $\rightarrow$ Local OS State) $= 0.90$
  - w_{mod} (`Inferred` — CIDR block derived algebraically, not observed) $= 0.40$
  - $\chi(t) = 0.90 \times 0.40 = 0.36$

  $$\Big(\big(\texttt{:Net\_10.46.124.0\_24},\ \texttt{:hasCIDR},\ \texttt{"10.46.124.0/24"}\big),\ p_{cred},\ 0.36\Big) \tag{4.22}$$

  The score of 0.36 captures the compounded epistemic uncertainty: while the source tool is reliable, the network segment itself has never been empirically confirmed and may differ from the actual subnet configuration of the remote host. This contrast — 0.72 for directly cached ARP data versus 0.36 for the derived CIDR — demonstrates the discriminative power of the bipartite credibility model across triples extracted from the same input payload $\omega_{\texttt{PC1}}$.

4.3.3 Fusion Layer: reasoning

The core of the knowledge fusion phase relies on a finite set of domain-specific deductive rules, denoted as $\mathcal{R}$. These rules abstract the infrastructural logic required to resolve entity identities (e.g., merging nodes representing the same physical asset) and to discover implicit topological links across disjoint observation graphs.

Let $\Phi_{\Sigma,\mathcal{R}}$ be the deductive closure operator. Given an arbitrary RDF graph G, $\Phi_{\Sigma,\mathcal{R}}(G)$ yields the fully materialized graph containing all explicit assertions of G plus all implicit assertions logically entailed by the T-Box Σ and the rule set $\mathcal{R}$.

To maintain strict technology agnosticism, the framework formalizes the fusion process to accommodate the two primary execution paradigms employed by modern semantic engines:

Batch Processing and Incremental Processing.

Let the sequence of structurally valid observation graphs ingested during the operational window be $\mathbf{s} = \langle G_1, G_2, \ldots, G_m \rangle$.

1. Batch fusion (deferred materialization): In this paradigm, all observation subgraphs are first collected and unioned. The reasoning operator is then applied once over the aggregated dataset. The final fused graph G^*_{batch} is defined as:

$$G^*_{batch} = \Phi_{\Sigma,\mathcal{R}} \left(\bigcup_{i=1}^{m} G_i \right)$$

2. Incremental fusion (streaming materialization): In this paradigm, the reasoning operator is applied iteratively as each new observation graph arrives, fusing it with the pre-existing accumulated state. The state of the fused graph at step k, denoted as G^*_k, is defined recursively:

$$G^*_0 = \emptyset$$

$$G^*_k = \Phi_{\Sigma,\mathcal{R}}(G^*_{k-1} \cup G_k) \quad \text{for } 1 \leq k \leq m$$

The final graph under incremental fusion is defined as $G^*_{incremental} = G^*_m$.

Axiom of monotonic equivalence: By the property of monotonicity inherent in standard semantic reasoning regimes, the addition of new facts cannot invalidate previously inferred facts. Consequently, the deductive closure of the global union is mathematically equivalent to the recursive deductive closure of the sequential parts. Thus, the framework formally establishes that the resulting graph is strictly invariant to the underlying engine's computational strategy:

$$G^* = G^*_{batch} = G^*_{incremental}$$

The resulting graph G^* represents the completely materialized Knowledge Graph. Once the operator $\Phi_{\Sigma,\mathcal{R}}$ has fully resolved $\mathcal{R}$ over the inputs to produce G^*, the graph is passed to the semantic and structural validation engine to guarantee its global topological integrity.

4.3.3.1 Concrete instantiation: knowledge fusion

To programmatically bridge the architectural fragmentation of the infrastructure, the Fusion Layer executes sequential inference pipelines over the global union of ingested sub-

graphs. We formalize two interconnected domain rules and track their operational execution over the already seen client graph $G_{\text{neigh\_PC1}}$ (Section 4.3.2.1) and another graph $G_{\text{sh\_int\_R1}}$ collected from observation $O_{\texttt{sh\_int\_R1}} = (id_{\texttt{sh\_int\_R1}}, G_{\texttt{sh\_int\_R1}})$ produced from the processing of the command `show interfaces` on the Cisco gateway router (Figure 4.1). First, we introduce the structural typing rule, denoted as R_{inferL2}, which acts as a class promotion mechanism. It dictates that any untyped individual within the graph that possesses a hardware MAC address attribute is logically inferred to be a link-layer network interface. Formally, $\forall x$ and $\forall d \in D_{mac} \subseteq \texttt{xsd:string}$:

$$p_{\text{mac}}(x, d) \implies C_{\text{iface}}(x) \tag{4.23}$$

Second, we define the identity resolution rule, denoted as R_{sameIP}. This rule evaluates cross-layer topology assertions to collapse redundant nodes representing identical physical assets. It specifies that if two distinct individuals, both recognized as functional network interfaces, share the same logical IP address, those interfaces and their corresponding address nodes are semantically equivalent. Formally, $\forall x, y \in C_{\text{iface}}$, $\forall a_1, a_2 \in C_{\text{addr}}$, and $\forall d \in D_{ip} \subseteq \texttt{xsd:string}$:

$$\begin{aligned}\Big(p_{\text{netprop}}(x, a_1) \land p_{\text{addr}}(a_1, d) \land p_{\text{netprop}}(y, a_2) \land p_{\text{addr}}(a_2, d) \land x \neq y\Big) \\ \implies \Big(\texttt{owl:sameAs}(x, y) \land \texttt{owl:sameAs}(a_1, a_2)\Big)\end{aligned} \tag{4.24}$$

To analyze the state of the knowledge base prior to fusion, consider the relevant assertion fragment within the subjective graph $G_{\texttt{neigh\_PC1}}$-previously extracted from the `ip neigh` observation $O_{\texttt{neigh\_PC1}}$ (Section 4.3.2.1) generated by the client workstation:

$$\begin{aligned}&(\texttt{:NeighIface\_02},\ \texttt{:hasMAC},\ \texttt{"02:df:7f:e2:2d:b0"\^\^xsd:string}) \\ &(\texttt{:NeighIface\_02},\ \texttt{:hasNetworkProperty},\ \texttt{:Addr\_10.46.124.1}) \\ &(\texttt{:Addr\_10.46.124.1},\ \texttt{rdf:type},\ \texttt{:NetworkAddress}) \\ &(\texttt{:Addr\_10.46.124.1},\ \texttt{:hasAddress},\ \texttt{"10.46.124.1"\^\^xsd:string})\end{aligned} \tag{4.25}$$

Note that within $G_{\texttt{neigh\_PC1}}$, the individual `:NeighIface_02` lacks an explicit `rdf:type` declaration. Concurrently, the execution of the `show interfaces` configuration query on

the gateway router yields the authoritative graph segment $G_{\texttt{sh\_int\_R1}}$ - extracted from the corresponding observation $O_{\texttt{sh\_int\_R1}}$, which models its native configuration space:

$$\begin{aligned}
&(\texttt{:Iface_R1_FE1},\ \texttt{:hasMAC},\ \texttt{"08:c1:cb:c4:52:52"\^\^xsd:string}) \qquad (4.26)\\
&(\texttt{:Iface_R1_FE1},\ \texttt{:hasNetworkProperty},\ \texttt{:Addr_R1_FE1})\\
&(\texttt{:Addr_R1_FE1},\ \texttt{rdf:type},\ \texttt{:NetworkAddress})\\
&(\texttt{:Addr_R1_FE1},\ \texttt{:hasAddress},\ \texttt{"10.46.124.1"\^\^xsd:string})
\end{aligned}$$

When the global union $G = G_{\texttt{neigh\_PC1}} \cup G_{\texttt{sh\_int\_R1}}$ is submitted to the reasoner engine, the deductive closure operator $\Phi_{\Sigma,\mathcal{R}}$ processes the rule dependencies through two sequential execution phases:

- **Phase 1: Structural class inference**
  The engine scans the dataset and detects that the literal property of `:NeighIface_02` satisfies the premise of rule R_{inferL2} (4.23), mapping variable $x \mapsto$ `:NeighIface_02` and $d \mapsto$ `"02:df:7f:e2:2d:b0"`. The rule fires immediately, materializing the missing taxonomic type assertion directly into the working graph space:

  $$\text{Implicit Statement:}\quad (\texttt{:NeighIface\_02},\ \texttt{rdf:type},\ \texttt{:NetworkInterfaceL2})$$

  Similarly, the reasoning engine evaluates the authoritative graph segment and identifies that `:Iface_R1_FE1` also possesses a literal MAC address property (4.26). Applying the identical premise, the engine maps the variables $x \mapsto$ `:Iface_R1_FE1` and $d \mapsto$ `"08:c1:cb:c4:52:52"`. The rule triggers concurrently, materializing the explicit taxonomic type for the router's interface directly into the assertion space:

  $$\text{Implicit Statement:}\quad (\texttt{:Iface\_R1\_FE1},\ \texttt{rdf:type},\ \texttt{:NetworkInterfaceL2})$$

- **Phase 2: Semantic identity matching**
  Following the materialization of Phase 1, the individual `:NeighIface_02` is now recognized as a member of C_{iface}, satisfying the preconditions of the identity resolution

rule R_{sameIP} (4.24). The rule engine successfully binds the structural pattern variables across the combined graphs:

$$\begin{aligned}\text{Premise Match:}\quad & x \mapsto \texttt{:NeighIface\_02}, \quad y \mapsto \texttt{:Iface\_R1\_FE1} \\ & a_1 \mapsto \texttt{:Addr\_10.46.124.1}, \quad a_2 \mapsto \texttt{:Addr\_R1\_FE1} \\ & d \mapsto \texttt{"10.46.124.1"\^\^xsd:string}\end{aligned}$$

Because the inequality constraint ($\texttt{:NeighIface\_02} \neq \texttt{:Iface\_R1\_FE1}$) evaluates to true , the engine evaluates the rule implication and materializes the equivalence corollaries into the final synchronized Knowledge Graph G^*:

$$\begin{aligned}\text{Implicit Statement:}\quad & (\texttt{:NeighIface\_02},\ \texttt{owl:sameAs},\ \texttt{:Iface\_R1\_FE1}) \\ & (\texttt{:Addr\_10.46.124.1},\ \texttt{owl:sameAs},\ \texttt{:Addr\_R1\_FE1})\end{aligned}$$

- **Phase 3: Epistemic conflict materialization**

  An attentive analysis of the newly synchronized graph G^* reveals a critical structural consequence of this identity resolution. Because `:NeighIface_02` and `:Iface_R1_FE1` are now semantically equivalent, the unified interface entity inherits all properties originally asserted by both disjoint subgraphs.

  Specifically, the merged individual now concurrently possesses two distinct MAC addresses, (4.25) and (4.26). However, because the Semantic Lifting operator embedded RDF* metadata into the assertions, these competing facts are structurally contextualized by their epistemic weight.

  From PC1 `ip neigh`:

  $$\Big(\big(\texttt{:NeighIface\_02},\ \texttt{:hasMAC},\ \texttt{"02:df:7f:e2:2d:b0"\^\^xsd:string}\big),\ p_{cred},\ 0.72\Big)$$

  From R1 `sh int`:

  $$\Big(\big(\texttt{:Iface\_R1\_FE1},\ \texttt{:hasMAC},\ \texttt{"08:c1:cb:c4:52:52"\^\^xsd:string}\big),\ p_{cred},\ 1.00\Big)$$

Through this recursive execution flow, the incomplete, subjective neighbor entity discovered by the client workstation is mathematically unified with the authoritative physical interface of the gateway router. While the resulting epistemic collision—one physical interface

possessing two distinct MAC addresses—is perfectly valid under the OWA governing the Fusion Layer, it introduces operational ambiguity into the topological model. However, the preservation of the differing credibility scores perfectly primes the graph for the subsequent Output Layer, where this exact redundancy will be deterministically resolved to distill a single, high-fidelity operational truth.

4.3.4 Output Layer: Knowledge Synthesizer

Following the completion of the reasoning phase, the fully materialized KG, G^*, holds a comprehensive, multi-perspective representation of the infrastructure. However, because the fusion layer aggregates heterogeneous observations—retaining all syntactically valid data under the OWA—the graph may contain redundant or epistemically conflicting assertions describing the same physical entity.
The primary objective of the Output Layer is to distill G^* into a deterministic, high-confidence operational model by executing a cascade of credibility-aware extraction queries.

Entity-centric graph traversal

The synthesis process distills the final operational model by executing targeted semantic queries over the validated KG, G^*.
Let $\mathbf{N}_{root} \subset G^*$ be a designated set of focal nodes representing the starting entities for the analysis.
Originating from this focal set $\mathbf{N}_{root}$, the system evaluates a recursive traversal query $\mathcal{Q}$. Let $E(x)$ define the extraction closure for a given target node $x \in \mathbf{N}_{root}$. The function $E(x)$ systematically navigates the outward edges of the graph, traversing established topological property paths to discover connected entities.
Consequently, $E(x)$ yields a specialized subgraph encompassing all structurally and semantically dependent information logically tethered to the node x (such as its addressing configurations, subnet bindings, and discovered neighbor adjacencies). This entity-centric traversal ensures that the synthesis naturally bounds itself to the relevant contextual perimeter, providing the precise scope required for the subsequent epistemic filtering phase.

Epistemic filtering via maximum credibility

Because the traversal closure $E(x)$ aggregates facts originating from varied diagnostic modalities, it may surface competing structural assertions for functional properties (i.e., properties where an entity should logically possess only one true value, such as a primary MAC address or an OS version). For example, an active remote scanner and a local configuration tool might report disparate OS fingerprints for the same resolved node, each carrying a different epistemic weight.

To resolve these observational discrepancies mathematically, the Knowledge Synthesizer evaluates the embedded RDF* credibility annotations to filter the extraction cascade.

Let $T(s, p) \subset G^*$ denote the set of all competing assertion triples sharing the same subject resource s and the functional predicate p:

$$T(s, p) = \{t_i = (s, p, o_i) \mid t_i \in G^*\} \tag{4.27}$$

The synthesizer defines an epistemic selection operator, denoted as σ_{max}. This operator inspects the reified metadata $\chi(t_i)$ embedded during the Semantic Lifting phase (Section 4.3.2.1) and uniquely isolates the triple possessing the absolute highest epistemic reliability. The deterministic, synthesized assertion $t^*_{s,p}$ is formally selected as:

$$t^*_{s,p} = \sigma_{max}(T(s, p)) = \arg \max_{t_i \in T(s,p)} \chi(t_i) \tag{4.28}$$

In the boundary case of an epistemic tie—where multiple distinct observations yield the exact same maximum credibility score—the synthesizer defaults to a configurable tie-breaking heuristic (e.g., temporal prioritization, selecting the assertion with the most recent execution timestamp τ).

Final model generation

The ultimate operational view, denoted as the refined output graph G_{out}, representing the infrastructure knowledge, is constructed by applying the topological query chain $\mathcal{Q}$ over the roots $\mathbf{N}_{root}$, strictly gated by the selection operator σ_{max} for all functional properties. This mechanism guarantees that G_{out} is not merely an amalgamation of raw telemetry, but a deterministic, semantically validated structure composed exclusively of the most

epistemically credible intelligence captured during the operational window.

4.3.4.1 Concrete instantiation: synthesis

To finalize the operational cycle of the methodology, we trace the extraction and filtering pipeline over the fully unified KG G^*, explicitly targeting the client workstation `PC1` introduced in the initial motivating example.

Phase 1: entity-centric traversal

The Knowledge Synthesizer initiates the synthesis by defining the focal root set as the target device: $\mathbf{N}_{root} = \{\texttt{:dev\_PC1}\}$.

The extraction closure $E(\texttt{:dev\_PC1})$ systematically expands the topological context through a hop-by-hop traversal across the architectural layers of the graph. Starting from the physical host entity, the query steps outward to discover its local Layer 2 interfaces (e.g., `:Iface_PC1_eth0`), ascends into the Layer 3 semantic space by resolving the bound logical addresses, and ultimately reaches the overarching `:GlobalNetworkL3` entities (e.g., `:Net_10.46.124.0_24`). These Layer 3 networks act as semantic hubs, conglomerating all logically connected nodes within the same broadcast domain.

Phase 2: epistemic filtering via σ_{max}

As $E(\texttt{:dev\_PC1})$ aggregates the structural facts surrounding the newly resolved gateway interface, it inevitably retrieves the competing assertions for its functional properties that were materialized during the fusion phase.

Let $s = \texttt{:Iface\_R1\_FE1}$ be the canonical subject representing the merged gateway interface, and let the functional property under evaluation be $p_{\text{mac}} = \texttt{:hasMAC}$. The traversal retrieves two distinct, structurally valid triples, populating the conflict set $T(s, p_{\text{mac}})$:

1. The subjective assertion t_1, originating from the remote ARP observation of `PC1` (`ip neigh`). As calculated in Section 4.3.2.1, this remote observation over the wire dynamically captured the MAC address `"02:df:7f:e2:2d:b0"` and carries an epistemic credibility of $\chi(t_1) = 0.72$ (4.25).

2. The authoritative assertion t_2, originating from the local configuration query (`show interfaces`) executed directly on the gateway `R1`. Because this represents a direct OS configuration state querying the hardware (*Local OS Config*), it captured the burned-in MAC address `"08:c1:cb:c4:52:52"` and its credibility is absolute: $\chi(t_2) = 1.00$ (4.26).

The Knowledge Synthesizer mathematically resolves this observational redundancy by applying the epistemic selection operator:

$$t^*_{s,p_{\text{mac}}} = \sigma_{max}(T(s, p_{\text{mac}})) = \arg \max_{t_i \in \{t_1, t_2\}} \{0.72, 1.00\} = t_2 \tag{4.29}$$

Consequently, while G^* safely preserved both perspectives to satisfy the OWA during reasoning, the filtered operational graph G_{out} strictly commits to the MAC address assertion t_2, anchored to the gateway's absolute epistemic certainty. The definitive structural truth is successfully isolated, effectively shedding the probabilistic uncertainty—and potential ARP spoofing artifacts—inherent in the client's localized network cache.

Chapter 5

Architecture Implementation

While the previous chapter defined the theoretical foundation of the framework, this chapter presents its practical implementation alongside the engineering work required to systematically test the framework.

5.1 Experimental Setup and Technological Stack

This section presents the specific technologies and software tools utilized to implement the proposed architecture.

Infrastructure emulation

To rigorously evaluate the framework's ability to resolve complex Layer 2 and Layer 3 topologies without the logistical overhead of a physical laboratory, the target network infrastructure was emulated using Graphical Network Simulator-3 (GNS3) [29]. GNS3 was explicitly selected for its hybrid virtualization architecture, which permits the seamless integration of distinct emulation paradigms within a single, unified broadcast domain. Specifically, lightweight network endpoints were instantiated using Docker [26].

Telemetry orchestration

To simulate the decentralized, inherently noisy collection of operational network intelligence, the framework utilizes Ansible [25] as its primary automation orchestrator. Operating as an agentless configuration management engine, Ansible systematically connects to

the emulated GNS3 nodes via SSH.

Triplestore and reasoner

At the core of the framework's logical architecture is Ontotext GraphDB [27], deployed as the centralized semantic database. GraphDB was strategically chosen because it natively fulfills the stringent theoretical requirements established during the formalization phase. First, it provides out-of-the-box support for the OWL2-RL (Section 2.3.2) reasoning profile, enabling the seamless integration of the custom domain-specific rule file ($\mathcal{R}$) required to execute the continuous, incremental entity resolution logic. Second, GraphDB fully supports the RDF* syntax, empowering the Epistemic Credibility Manager to append, store, and query mathematical confidence weights directly as reified metadata on structural edges. Finally, its robust REST API facilitates the strict database-level segregation dictated by the framework's architecture, effortlessly managing both the isolated staging repositories for raw observations and the final reconstructed KG. The rule-based inference logic, conceptually aligned with the SWRL paradigm discussed in the state of the art, is implemented via GraphDB's native Pie ruleset and SPARQL CONSTRUCT queries, which provide equivalent expressiveness with full native integration into the triplestore.

5.2 Validation Strategy and Experimental Objectives

Empirical verification of an infrastructure reconstruction framework presents a fundamental methodological challenge: to evaluate the accuracy of the synthesized infrastructure, the experimental environment must provide an absolute, error-free reference topology. In real-world corporate networks, obtaining a verified, complete, and static architectural map is practically unfeasible due to ongoing operational changes and the very visibility limitations this framework aims to solve. Furthermore, manually mapping a large-scale network onto the domain ontology to create an evaluation baseline is an intensely time-consuming, error-prone, and non-scalable process.

To overcome these limitations, this work introduces a fully automated end-to-end evaluation and deployment pipeline as illustrated in Figure 5.1.

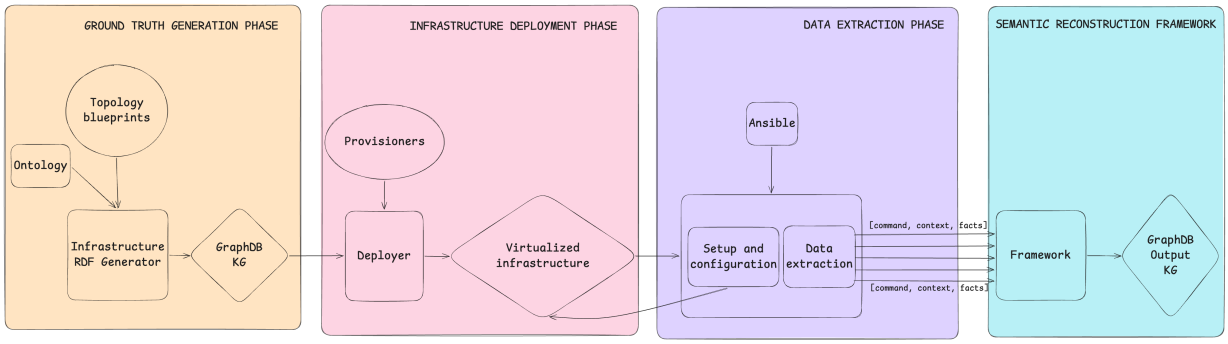

Figure 5.1: The complete automated evaluation pipeline architecture.

The lifecycle of the validation strategy is orchestrated across five distinct procedural phases:

- **Ground truth generation phase (Section 5.2.1):** The pipeline begins by parsing abstract topology blueprints to automatically synthesize a complex network infrastructure model. This model is immediately ingested into a primary semantic repository, establishing the absolute architectural benchmark against which the framework will be measured.

- **Infrastructure deployment phase (Section 5.2.2):** Rather than manually configuring virtual environments, a dedicated deployer engine queries the ground truth knowledge graph to extract structural parameters. It then programmatically instantiates the live virtualized infrastructure utilizing containerized nodes and isolated network domains.

- **Data extraction phase (Section 5.2.3):** Once the virtualized infrastructure is active, the Ansible automation orchestrator connects to the decentralized nodes. It applies specific routing policies, assigns dynamic interface states, and sets up real-world operational environments. Following configuration, the orchestrator executes a suite of native diagnostic commands across the live environment. This process extracts the raw, heterogeneous telemetry payloads capturing the local, vantage-point dependent views of the individual hosts.

- **Semantic Reconstruction Framework (Section 5.2.4):** The extracted payloads are injected into the Semantic Reconstruction Framework. The engine processes the

discovery data through the lifting, fusion, and reasoning layers, generating a finalized knowledge graph.

5.2.1 Ground truth generation phase

The ontology mapping problem

To evaluate the framework's capability to correctly reconstruct complex network architectures, a rigorous comparison against an absolute structural baseline is required.

Because the framework operates on a proprietary, specialized domain ontology optimized for cross-layer cybersecurity reasoning, there are no off-the-shelf automated discovery or inventory tools capable of natively exporting real-world corporate configurations directly into this specific T-Box target schema. Attempting to manually map a live production infrastructure—or even an extensive virtualized laboratory setup—into a large number of precise semantic assertions is completely unfeasible, highly error-prone, and inherently non-scalable. Without an existing translation layer or a pre-defined reference dataset, verifying the accuracy of the reasoning engine becomes impossible.

To break this circular dependency, the evaluation pipeline reverses the workflow: rather than attempting to semantically map an already existing network, the system programmatically synthesizes the network topology directly out of the ontology itself.

Procedural synthesis pipeline

The programmatic generation engine addresses this challenge through a multi-stage synthesis pipeline that combines structural variety with strict logical validity. The workflow transitions from abstract architectural design down to concrete RDF triple materialization across two primary components:

1. **Stochastic Topology Generation Engine:** This component functions as an autonomous generative model designed to synthesize arbitrary, highly dynamic network structures from scratch. The engine natively executes a *tree-based generation and model merging* approach. It algorithmically constructs independent tree-like hierarchical graphs, combines them to simulate interconnected network zones, and applies

randomized graph mutations—such as injecting unexpected cross-domain bridging links, varying peripheral subnet counts, or manipulating local connection densities. Concurrently, the engine probabilistically assigns granular device profiles to the newly spawned graph nodes, ensuring significant topological heterogeneity and structural entropy that closely mimic real-world corporate environments.

2. **Deterministic Attribute Allocation Engine:** Once the physical and logical skeleton of the graph is fixed by the stochastic engine, the structural layout is passed to a deterministic allocation compiler. This component steps through the synthesized graph to compute and distribute strict networking attributes. It maps out contiguous, non-overlapping IP address spaces, defines CIDR network boundaries, builds deterministic Layer 2 MAC address assignments, and calculates static routing tables for every device. This determinism guarantees that despite the random structural mutations, the underlying network remains mathematically valid, free of routing loops, and topologically consistent.

Knowledge graph serialization

The final phase of the generator compiles these in-memory structural blueprints and calculated attributes into a definitive, idealized reference KG. The engine serializes the complete A-Box dataset into a standardized RDF/Turtle (`.ttl`) format file.

This file represents the absolute architectural truth of the network model, completely unmarred by observation blind spots, missing telemetry, or vantage-point dependencies. The resulting Turtle file, representing the ground truth KG, is directly uploaded to a dedicated repository within the semantic database, namely `gt_repo`. This graph provides the baseline reference necessary to calculate the accuracy of the framework.

5.2.2 Infrastructure deployment phase

Once the definitive reference model is serialized and committed to the GraphDB, the pipeline transitions from abstract representation to live physical execution. This phase is governed by an automated deployer engine that operates under the paradigm of model-

driven Infrastructure as Code (IaC).
Instead of relying on static, hardcoded configuration files, the deployer engine executes parameterized SPARQL queries against the semantic database's A-Box to dynamically reconstruct the network's asset inventory, dependency trees, and interconnection matrices. The engine parses these semantic relations and compiles them into a structured, reproducible physical execution plan. This compilation layer maps every semantic individual and property onto specific virtualization resources, ensuring that the target lab environment is a direct, deterministic manifestation of the ontology.

Heterogeneous virtualization strategy
Building upon the experimental setup introduced in Section 5.1, the deployment framework orchestrates a strictly partitioned, heterogeneous virtualization architecture within the GNS3 emulation environment to achieve high fidelity while balancing computational overhead:

- **Containerized endpoint provisioning:** Every infrastructure entity classified as a peripheral or processing node rather than a core networking component—encompassing user clients, database instances, web hosts, and internal server architectures—is provisioned using lightweight Docker container environments. The deployer engine maps these semantic entities to dedicated container images, utilizing dynamic configuration templating to initialize execution privileges, user accounts, and local application states.

- **Network appliance emulation:** Conversely, standard container environments cannot accurately replicate complex link-layer control frames, proprietary encapsulation behaviors, or advanced routing mechanics. Therefore, core networking components—such as enterprise switches, edge routers, and boundary firewalls—are instantiated using full-stack virtualization. The engine provisions these nodes as dedicated virtual machines running authentic network operating systems, specifically utilizing Cisco virtual appliance images.

Parameter and topological fidelity

To ensure the validity of the downstream data extraction and semantic lifting phases, the live virtualized laboratory environment must perfectly replicate the operational metrics defined within the ground truth knowledge graph. The deployment engine systematically configures every active virtual interface to match its corresponding semantic literal value. This total fidelity covers:

- **Interface identifiers:** Native hardware naming conventions are preserved exactly as defined in the graph.
- **Link-Layer parameters:** Maximum Transmission Unit (MTU) sizes, interface descriptions, and hardware MAC addresses are explicitly assigned to the virtual interfaces.
- **Network-Layer properties:** Precise IPv4/IPv6 address allocations, subnet mask boundaries, default gateways, and static routing table matrices are directly injected into each node's running configuration.

By eliminating any structural divergence between the reference graph and the live virtual link constraints, the framework ensures that any subsequent telemetry represents an authentic observation of the ground truth.

5.2.3 Data extraction phase

Orchestration workflow and operational realism

Once the virtualized lab environment is fully provisioned and stable, the pipeline initializes the telemetry collection phase. This phase is entirely governed by the Ansible automation orchestrator, which executes declarative playbooks across the distributed topology. To prevent the evaluation from testing an idealized, sterile environment, the orchestrator first applies real-world operational constraints before executing any discovery tasks.

Using targeted configuration playbooks, the engine dynamically injects firewall policies and introduces deliberate routing anomalies, such as interface packet filters and intentionally erroneous ARP table entries. This ensures that the virtual infrastructure mimics

the structural entropy, conflicting facts, and restrictive visibility typical of real enterprise environments, establishing a rigorous testing ground for the resilience of the reasoning framework.

Active scanning and telemetry probing

Following the configuration and anomaly injection phase, the orchestrator triggers decentralized discovery playbooks across all live nodes simultaneously. Each node acts as an independent vantage point, executing a standard suite of native diagnostic commands and active network probes to capture the state of its local network domain.

To bridge the gap between raw textual terminal streams and the mathematical formalization defined in Section 4.3.1, the extraction scripts immediately ingest these standard outputs. For each execution instance, the orchestrator programmatically binds the raw structural evidence to its operational metadata, compiling the telemetry into a highly structured JSON format that explicitly materializes the formal input payload 3-tuple:

$$\omega = (\pi, k, \mathbf{f}) \tag{5.1}$$

Where the command π specifies the diagnostic utility executed, the context k encapsulates the executing host's FQDN, privilege level, and execution timestamp, and the fact sequence $\mathbf{f}$ contains the normalized, attribute-value JSON mappings of the command's standard output.

Virtualization artifact filtering and pre-processing

A critical challenge in executing an automated validation pipeline inside an emulated laboratory is the inadvertent generation of virtualization noise. Because the endpoints are instantiated using containerized platforms, the underlying container engines inherently inject non-production network infrastructure into the local routing tables and neighborhood caches—most notably the default Docker bridge gateways (e.g., the standard `172.17.0.1` management gateway), internal virtual ethernet pairs, and overlay orchestration control loops.

Since these management interfaces and routing paths are purely technical artifacts required to maintain the virtualization scaffold, they do not exist within the ground truth.

To eliminate this divergence, a dedicated Ansible tasks sanitize the raw telemetry data prior to writing the final file, utilizing deterministic regex filters and subnet mask exclusions to programmatically strip out all virtualization management gateways and internal docker bridge references. This proactive pre-processing guarantees that the final serialized JSON files exclusively contain authentic, production-equivalent network telemetry, making the fact sequence $\mathbf{f}$ perfectly aligned with the ground truth schema and ready for unbiased ingestion by the Semantic Lifting operator $\mathcal{M}$.

5.2.4 Semantic Reconstruction Framework

The Semantic Reconstruction Framework acts as the centralized computational core of the architecture. It ingests the pre-processed, decentralized telemetry payloads and executes the multi-layered synthesis pipeline formalized in Chapter 4. By sequentially applying Semantic Lifting, rule-based identity fusion, and deductive reasoning, the framework transforms fragmented observations into a fully materialized KG.

Upon completion of the deductive closure, the unified topology is serialized and committed to a dedicated semantic repository on GraphDB. This final interconnected graph serves as the definitive operational model required to mathematically measure the reconstruction accuracy against the ground truth baseline.

The comprehensive software architecture, modular design, and concrete engineering details of this system are thoroughly detailed in the following section (Section 5.4).

5.3 Ontological Design for Infrastructure Representation and Data Fusion

Table 5.1: T-Box specification schema.

Symbol	Ontology IRI	Domain	Inheritance / Type
CONCEPTS ($\mathcal{C}$)			
C_{asset}	`:SupportingAsset`	—	`rdfs:subClassOf :Thing`
C_{dev}	`:PhysicalDevice`	—	`rdfs:subClassOf :SupportingAsset`
C_{netdev}	`:NetworkDevice`	—	`rdfs:subClassOf :PhysicalDevice`
$C_{\text{appliance}}$	`:PhysicalAppliance`	—	`rdfs:subClassOf :PhysicalDevice`
C_{server}	`:PhysicalServer`	—	`rdfs:subClassOf :PhysicalDevice`
$C_{\text{workstation}}$	`:PhysicalWorkstation`	—	`rdfs:subClassOf :PhysicalDevice`
C_{iface}	`:NetworkInterface`	—	`rdfs:subClassOf :SupportingAsset`
C_{ifaceL2}	`:NetworkInterfaceL2`	—	`rdfs:subClassOf :NetworkInterface`
C_{ifaceL3}	`:NetworkInterfaceL3`	—	`rdfs:subClassOf :NetworkInterfaceL2`
$C_{\text{logifaceL2}}$	`:LogicalNetworkInterfaceL2`	—	`rdfs:subClassOf :NetworkInterface`
$C_{\text{logifaceL3}}$	`:LogicalNetworkInterfaceL3`	—	`rdfs:subClassOf :LogicalNetworkInterfaceL2`
C_{prop}	`:NetworkProperty`	—	`rdfs:subClassOf :SupportingAsset`
C_{addr}	`:NetworkAddress`	—	`rdfs:subClassOf :NetworkProperty`
C_{route}	`:NetworkRoute`	—	`rdfs:subClassOf :NetworkProperty`
C_{netL3}	`:NetworkL3`	—	`rdfs:subClassOf :SupportingAsset`
C_{globnet}	`:GlobalNetworkL3`	—	`rdfs:subClassOf :NetworkL3`
C_{locnet}	`:LocalNetworkL3`	—	`rdfs:subClassOf :NetworkL3`
OBJECT PROPERTIES ($\mathcal{P}_{\text{obj}}$)			
p_{hasL2NWI}	`:hasL2NWI`	C_{dev}	C_{iface}
p_{netprop}	`:hasNetworkProperty`	$C_{\text{ifaceL3}} \sqcup C_{\text{logifaceL3}}$	C_{prop}
p_{connects}	`:connectsTo`	C_{addr}	C_{netL3}
DATATYPE PROPERTIES ($\mathcal{P}_{\text{data}}$)			
p_{name}	`:hasName`	C_{asset}	`xsd:string`
p_{fqdn}	`:hasFQDN`	C_{dev}	`xsd:string`
p_{iface}	`:hasIface`	C_{iface}	`xsd:string`
p_{mac}	`:hasMAC`	C_{iface}	`xsd:string`
p_{mtu}	`:hasMTU`	C_{iface}	`xsd:int`
p_{bcast}	`:hasBroadcast`	$C_{\text{iface}} \sqcup C_{\text{addr}}$	`xsd:string`
p_{nettype}	`:hasNetworkType`	C_{netL3}	`xsd:string`
p_{cidr}	`:hasCIDR`	C_{netL3}	`xsd:string`
p_{proto}	`:hasProtocol`	C_{prop}	`xsd:string`
p_{addr}	`:hasAddress`	C_{addr}	`xsd:string`
p_{mask}	`:hasNetmask`	C_{addr}	`xsd:int`
p_{dst}	`:hasNetworkDst`	C_{route}	`xsd:string`
p_{gw}	`:hasGateway`	C_{route}	`xsd:string`
p_{metric}	`:hasMetric`	C_{route}	`xsd:string`
p_{cred}	`:hasCredibility`	—	`xsd:decimal`

The core philosophy of the Fusion Layer dictates that a domain ontology must function as an active, deductive execution engine rather than a static database schema. When reconstructing an infrastructure from decentralized, incomplete, and noisy telemetry streams, the ontology must provide the structural scaffolding necessary to handle data fragmentation, drive taxonomy promotions, and enforce strict real-world identity constraints.
The structural foundation of this framework is based on a proprietary enterprise ontology. This reference model was optimized and refactored leveraging advanced OWL 2 RL constructs, class inheritance hierarchies, and metadata annotation to directly support cross-layer identity resolution and automated data fusion.

5.3.1 Structural inheritance

The optimized ontology establishes a strict taxonomy that separates infrastructure components across functional layers, rooted under the abstract concept of a `:SupportingAsset`. This layered model allows the inference engine to dynamically aggregate evidence collected from completely different network vantage points.
A primary example of this design is the structural inheritance chain implemented for network boundaries, modeled under the physical and logical `:NetworkInterface` branches. Focusing on the physical track, the hierarchy is structured as follows:

```
:SupportingAsset
    |__ :NetworkInterface
          |__ :NetworkInterfaceL2
                |__ :NetworkInterfaceL3
```

Rather than using subclassing merely to distribute static literal attributes, this hierarchical progression is engineered explicitly to drive semantic classification during data fusion. Within this schema, root configuration properties are bound directly to the base `:NetworkInterface` class, meaning they are shared equally by all descendants across both the physical and logical branches. The intentional separation into discrete L2 and L3 sub-classes serves a distinct structural purpose:

- `:NetworkInterface` acts as the generic taxonomic anchor, establishing the fundamental operational parameters common to any network boundary node.
- `:NetworkInterfaceL2` represents a base link-layer instantiation. An ingested interface individual remains categorized strictly at this level as long as the extracted telemetry only contains link-layer configurations, tracking the boundary asset within a pure Layer 2 scope.
- `:NetworkInterfaceL3` inherits all underlying link-layer attributes from father class but functions as a semantic extension point. It restricts the domain of the object property `:hasNetworkProperty`, designating this class to specifically represent a physical interface that has been enriched with Layer 3 parameters.

5.3.2 Physical-logical duality and real-world constraints

A common pitfall in network modeling is treating all interfaces and subnets identically. In a real-world enterprise infrastructure, technical constraints that are absolute for physical hardware are completely invalid for logical or virtualized components. For instance, while a physical network interface must possess a globally unique MAC address, a logical interface (such as a loopback interface, a tunnel endpoint, or a Docker virtual bridge) can reuse duplicate MACs or lack a hardware address entirely.
To map this behavior without introducing logical contradictions, the optimized ontology implements a strict structural duality between physical and logical entities under both the interface and network branches:

Table 5.2: Physical-logical duality separation within the ontology schema.

Functional Layer	Physical / Global Scope	Logical / Local Scope
Layer 2 Interfaces	`:NetworkInterfaceL2`	`:LogicalNetworkInterfaceL2`
Layer 3 Interfaces	`:NetworkInterfaceL3`	`:LogicalNetworkInterfaceL3`
Layer 3 Subnets	`:GlobalNetworkL3`	`:LocalNetworkL3`

This class separation enables the targeted application of OWL 2 uniqueness constraints (`owl:hasKey`). Uniqueness keys are highly destructive if applied too broadly; if a loopback

interface were evaluated under a global MAC constraint, the engine would incorrectly collapse entirely separate devices into a single node.
As defined in the optimized schema, identity keys are bound exclusively to physical and global classes, acting as the semantic anchors for data fusion:

```
:PhysicalDevice     owl:hasKey ( :hasFQDN ) .
:NetworkInterfaceL2 owl:hasKey ( :hasMAC ) .
:GlobalNetworkL3    owl:hasKey ( :hasCIDR ) .
```

Under this logic, if two independent logs instantiate a `:NetworkInterfaceL2` sharing the exact same `:hasMAC` string, the engine applies the key constraint to automatically infer an `owl:sameAs` relation, merging the nodes. Conversely, if two logs instantiate a `:LogicalNetworkInterfaceL2`, the key constraint is bypassed, safely preserving the independent identities of the virtual endpoints.

5.3.3 RDF* credibility metadata

Fusing heterogeneous network telemetry introduces significant noise due to transient scan artifacts, stale cache entries, or deliberate configuration anomalies. To mitigate this without relying on cumbersome standard RDF reification models, the optimized ontology leverages RDF* annotations to implement the mathematical credibility framework formalized in Section 4.3.2.1. By treating entire triples as subjects, the Semantic Layer utilizes the datatype property `:hasCredibility` to directly bind a computed contextual weight (between 0.0 and 1.0) to each serialized assertion.

5.3.4 Topological mapping of the running example

To synthesize the structural design detailed throughout this section, we project the theoretical T-Box constraints onto the concrete motivational scenario originally introduced in Section 4.1. Figure 5.2 illustrates the resulting A-Box graph when the localized network telemetry from that specific setup is materialized using the optimized ontology.

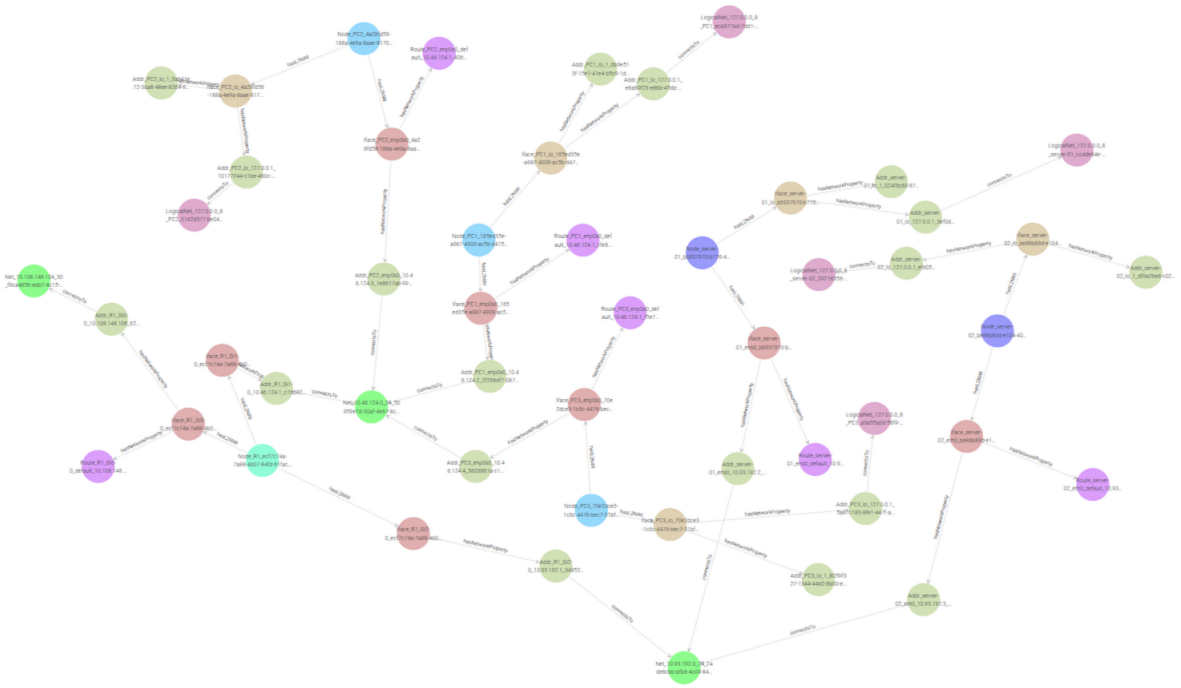

Figure 5.2: A-Box topological skeleton of the running example.

For visual clarity and to strictly emphasize the topological skeleton of the infrastructure, taxonomic bindings (such as `rdf:type` assertions) and literal datatype properties (e.g., `:hasMAC`, `:hasAddress`) have been intentionally omitted from this render.

5.4 Framework Implementation

This section transitions from the theoretical formalisms established in Chapter 4 and the ontological design detailed in Section 5.3 to their concrete software implementation. It details the end-to-end architecture of the Semantic Reconstruction Framework, demonstrating how decentralized observations are programmatically ingested, semantically lifted, and fused into a cohesive KG.

5.4.1 Input Layer

The foundational stage of the implementation involves standardizing the highly heterogeneous raw network telemetry into a structured, machine-readable format. As formalized in the mathematical model, the Input Layer must strictly organize the decentralized data by its observation vantage point to prepare isolated context blocks for semantic transformation.

To materialize the theoretical command set Π introduced in the methodology, the implementation defines the active operational toolset by partitioning it into two distinct execution subsets, based on the architectural role of the target nodes. Let Π_{router} be the subset of authoritative diagnostic commands executed strictly on routing backbone entities (Cisco IOS), defined as:

$$\Pi_{router} = \{\texttt{show interfaces}, \texttt{show arp}, \texttt{show ip route}\} \tag{5.2}$$

Conversely, let Π_{host} represent the subset of commands executed on standard Linux-based endpoints (e.g., leaf nodes and servers). This subset leverages native OS utilities alongside highly optimized, low-latency active scanning arguments to minimize execution overhead:

$$\Pi_{host} = \{\texttt{ip a}, \texttt{ip route}, \texttt{ip neigh}, \texttt{nmap}\} \tag{5.3}$$

The total global diagnostic toolset employed during the ingestion cycle is therefore instantiated as the union of these operational subsets: $\Pi = \Pi_{router} \cup \Pi_{host}$.

Practically, the ingestion process is managed through a strictly partitioned file system hierarchy. Telemetry extracted from the virtualized environment via these command subsets is segregated into isolated directories named after their source execution nodes (e.g., `/telemetry/PC1/`, `/telemetry/RouterA/`).

5.4.1.1 Example trace: input format

To contextualize this extraction process within the running example (Section 4.1), Listing 5.1 presents the concrete JSON payload generated by executing the `ip neigh` command on the `PC1` container. This artifact directly maps to the theoretical 3-tuple: the `command`

key represents π, the `context` object captures k, and the `facts` array represents $\mathbf{f}$, housing the three ARP entries corresponding to ϕ_1, ϕ_2, and ϕ_3 formalized in Section 4.3.1.1.

Listing 5.1: Structured JSON telemetry extracted from PC1 (command: `ip neigh`).


```
{
    "command": "ip neigh",
    "context": {
        "FQDN": "PC1",
        "os": "alpine",
        "role": "client",
        "timestamp": "2026-05-28T12:27:21Z",
        "user": "root"
    },
    "facts": [
        {
            "dev": "eth0",
            "dst": "10.46.124.3",
            "lladdr": "88:d3:d6:0b:69:5e",
            "state": [
                "REACHABLE"
            ]
        },
        {
            "dev": "eth0",
            "dst": "10.46.124.4",
            "lladdr": "64:46:ee:6c:ad:ae",
            "state": [
                "REACHABLE"
            ]
        },
        {
            "dev": "eth0",
            "dst": "10.46.124.1",
            "lladdr": "02:df:7f:e2:2d:b0",
            "state": [
                "PERMANENT"
            ]
        }
    ]
}
```


5.4.2 Semantic Layer

Modular parsing architecture

The programmatic implementation of the Semantic Lifting operator $\mathcal{M}$ relies on a modular architecture designed to systematically translate the standardized JSON payloads $\omega = (\pi, k, \mathbf{f})$ into formal semantic observations. To efficiently handle the heterogeneity of network diagnostic tools, the framework structures the ingestion logic using a dynamic registry pattern combined with an inheritance-based parser hierarchy.

A central dispatcher component evaluates the command string π and automatically in-

stantiates the specific child parser registered for that utility (e.g., dispatching an `ip neigh` payload to a dedicated neighbor cache parser). The generation workload is then structurally divided: a foundational base class provides the core routines for handling the context metadata k, while the specialized child classes inherit this foundation to implement the tool-specific fact extraction logic.

Observation generation and T-Box compliance

Once the correct child parser is instantiated by the dispatcher, the framework coordinates the programmatic construction of the observation graph. During this initialization, the dispatcher first mints a unique observation identifier to guarantee global uniqueness and strict data provenance. Rather than relying on abstract formulations, this identifier is dynamically generated at runtime via string interpolation, systematically concatenating the executing host's FQDN and the specific command string to the designated base namespace of the ontology (e.g., `http://example.com/EX/fqdn/command`).

Anchored by this namespace, the specific child parser proceeds to construct the localized graph payload. It first invokes the base class routine to process the execution context, deterministically instantiating the foundational device entity. Following this contextual grounding, the child parser executes its specific extraction logic, iterating over the raw JSON arrays to map telemetry details—such as MAC addresses and IP subnets—into graph nodes.

Crucially, the entire generation process strictly enforces T-Box compliance at the code level. By utilizing predefined namespace bindings and programmatic schemas, the framework ensures that every minted IRI is explicitly typed to a valid class from the optimized ontology. Similarly, edge creation routines strictly validate assignments against predefined object and datatype property domains, ensuring that the resulting payload is correct-by-construction before it is dispatched to GraphDB.

Credibility Manager

To bridge the formal epistemic model with the software architecture, the framework implements a dedicated module acting as the credibility manager within the data ingestion

pipeline. As the parser extracts structural relationships from the raw telemetry, it delegates the evaluation to the manager.

The Credibility Manager accesses a centralized configuration dictionary that maps the extracted command types and observation modalities to their respective static weights. Utilizing these parameters, it computes the numerical confidence score on the fly. Leveraging native RDF* programming interfaces, the manager dynamically constructs the nested graph statements. It programmatically encapsulates the base topological triple and injects the computed score via the `:hasCredibility` data property, ensuring that the semantic payload dispatched to the GraphDB repository is completely annotated and ready for the downstream filtering queries.

Observation serialization and staging isolation

To streamline the data ingestion process and prepare for deferred reasoning, the framework aggregates the individual observation graphs generated by the child parsers into a unified batch payload. Once the parsing layer completes the generation and credibility annotation of all observation tuples, the framework serializes the accumulated dataset and pushes it to the semantic database via the GraphDB REST API in a single, comprehensive transaction. Crucially, to preserve the integrity of the evaluation pipeline prior to the formal reasoning phase, this aggregated raw telemetry is committed to a completely isolated staging repository, namely `raw_repo`. This strict database-level segregation guarantees that the noisy, un-reconciled inputs do not prematurely contaminate the immutable ground truth reference or the globally resolved output KG.

5.4.2.1 Example trace: semantic observation generation

To contextualize the Semantic Lifting operator ($\mathcal{M}$) within the running example, we examine the transformation of the JSON payload introduced in Listing 5.1.

When the dispatcher routes this payload to the `ip neigh` parser, the framework generates the unique observation identifier from the execution context. Following the IRI generation rule, the resulting identifier is instantiated as `http://example.org/EX/PC1/ip_neigh`.

The parser then iterates over the three ARP facts $\langle \phi_1, \phi_2, \phi_3 \rangle$ to populate the A-Box graph

$G_{\texttt{neigh\_PC1}}$. Listing 5.2 presents the resulting RDF/Turtle serialization, showing how each ARP entry is translated into a typed `:NetworkInterfaceL2` individual carrying its MAC address, linked via `:hasNetworkProperty` to a `:NetworkAddress` individual, which is in turn connected to the inferred `:GlobalNetworkL3` subnet node — mirroring the formal pattern instantiated in Section 4.3.2.1.

Concurrently, the Credibility Manager intercepts the generated triples and embeds the credibility scores via RDF* nested annotations, as shown in Listing 5.3. The three credibility tiers derived in the Section 4.3.2.2 are directly observable in the output: local interface assertions receive 0.90, remotely cached ARP data receives 0.72, and the algebraically inferred CIDR block receives 0.36. Finally, Figure 5.3 provides the GraphDB visual representation of this subgraph, confirming the correct-by-construction topological alignment prior to the Fusion Layer.

Listing 5.2: Observation of PC1 generated from the `ip neigh` command.

```
@prefix : <http://example.org/EX/> .

:NeighIface_02_df_7f_e2_2d_b0_from_PC1 :hasMAC "02:df:7f:e2:2d:b0" ;
    :hasNetworkProperty :Addr_10.46.124.1 .

:NeighIface_64_46_ee_6c_ad_ae_from_PC1 :hasMAC "64:46:ee:6c:ad:ae" ;
    :hasNetworkProperty :Addr_10.46.124.4 .

:NeighIface_88_d3_d6_0b_69_5e_from_PC1 :hasMAC "88:d3:d6:0b:69:5e" ;
    :hasNetworkProperty :Addr_10.46.124.3 .

:Node_PC1 a :PhysicalWorkstation ;
    :hasFQDN "PC1" ;
    :hasL2NWI :Iface_PC1_eth0 .

:Addr_10.46.124.1 :connectsTo :Net_10.46.124.0_24 ;
    :hasAddress "10.46.124.1" .

:Addr_10.46.124.3 :connectsTo :Net_10.46.124.0_24 ;
    :hasAddress "10.46.124.3" .

:Addr_10.46.124.4 :connectsTo :Net_10.46.124.0_24 ;
    :hasAddress "10.46.124.4" .

:Addr_Inferred_PC1_eth0_10.46.124.0_24 :connectsTo :Net_10.46.124.0_24 .

:Iface_PC1_eth0 a :NetworkInterfaceL2 ;
    :hasIface "eth0" ;
    :hasNetworkProperty :Addr_Inferred_PC1_eth0_10.46.124.0_24 .

:Net_10.46.124.0_24 a :GlobalNetworkL3 ;
    :hasCIDR "10.46.124.0/24" .
```

Listing 5.3: Observation credibility of PC1 generated from the `ip neigh` command.

```
@prefix : http://example.org/EX/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<< :Node_PC1 rdf:type :PhysicalWorkstation >> :hasCredibility 0.9 .
<< :Node_PC1 :hasFQDN "PC1" >> :hasCredibility 0.9 .
<< :Node_PC1 :hasL2NWI :Iface_PC1_eth0 >> :hasCredibility 0.9 .

<< :Iface_PC1_eth0 rdf:type :NetworkInterfaceL2 >> :hasCredibility 0.9 .
<< :Iface_PC1_eth0 :hasIface "eth0" >> :hasCredibility 0.9 .
<< :Iface_PC1_eth0 :hasNetworkProperty :Addr_Inferred_PC1_eth0_10.46.124.0_24 >> :
    hasCredibility 0.9 .

<< :Addr_Inferred_PC1_eth0_10.46.124.0_24 :connectsTo :Net_10.46.124.0_24 >> :
    hasCredibility 0.9 .

<< :Net_10.46.124.0_24 rdf:type :GlobalNetworkL3 >> :hasCredibility 0.36 .
<< :Net_10.46.124.0_24 :hasCIDR "10.46.124.0/24" >> :hasCredibility 0.36 .

<< :Addr_10.46.124.1 :hasAddress "10.46.124.1" >> :hasCredibility 0.72 .
<< :Addr_10.46.124.1 :connectsTo :Net_10.46.124.0_24 >> :hasCredibility 0.36 .

<< :Addr_10.46.124.3 :hasAddress "10.46.124.3" >> :hasCredibility 0.72 .
<< :Addr_10.46.124.3 :connectsTo :Net_10.46.124.0_24 >> :hasCredibility 0.36 .

<< :Addr_10.46.124.4 :hasAddress "10.46.124.4" >> :hasCredibility 0.72 .
<< :Addr_10.46.124.4 :connectsTo :Net_10.46.124.0_24 >> :hasCredibility 0.36 .

<< :NeighIface_02_df_7f_e2_2d_b0_from_PC1 :hasMAC "02:df:7f:e2:2d:b0" >> :hasCredibility
    0.72 .
<< :NeighIface_02_df_7f_e2_2d_b0_from_PC1 :hasNetworkProperty :Addr_10.46.124.1 >> :
    hasCredibility 0.36 .

<< :NeighIface_88_d3_d6_0b_69_5e_from_PC1 :hasMAC "88:d3:d6:0b:69:5e" >> :hasCredibility
    0.72 .
<< :NeighIface_88_d3_d6_0b_69_5e_from_PC1 :hasNetworkProperty :Addr_10.46.124.3 >> :
    hasCredibility 0.36 .

<< :NeighIface_64_46_ee_6c_ad_ae_from_PC1 :hasMAC "64:46:ee:6c:ad:ae" >> :hasCredibility
    0.72 .
<< :NeighIface_64_46_ee_6c_ad_ae_from_PC1 :hasNetworkProperty :Addr_10.46.124.4 >> :
    hasCredibility 0.36 .
```

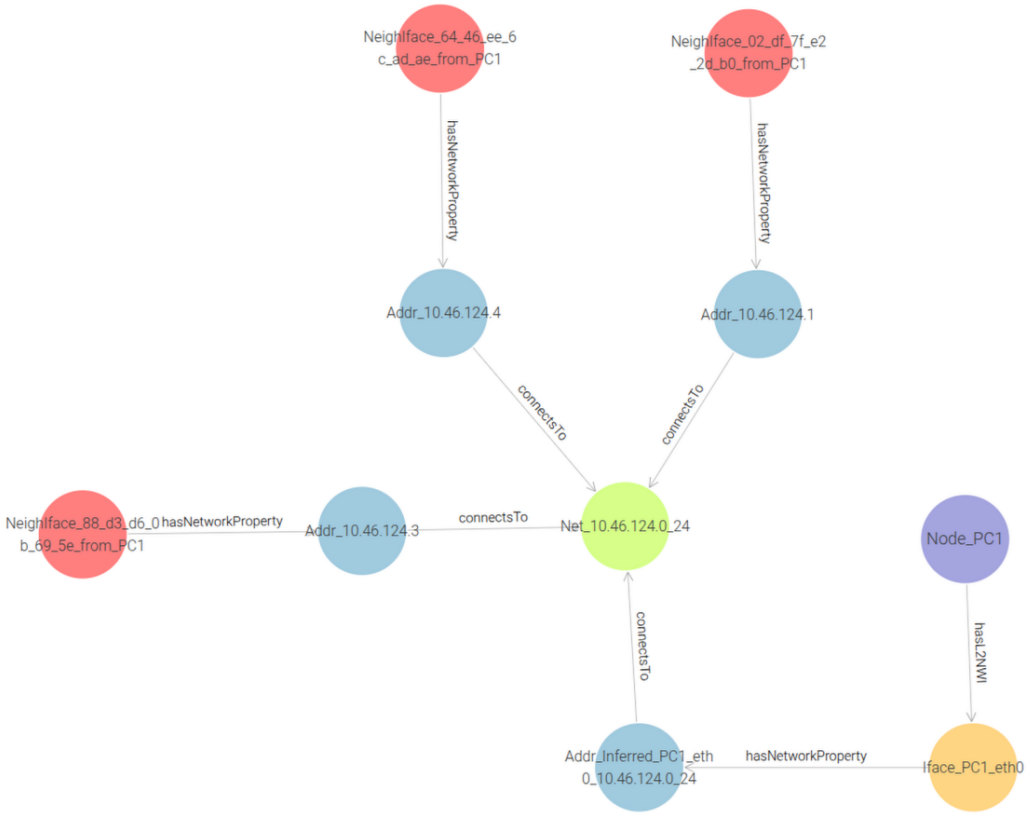

Figure 5.3: GraphDB visualization of the generated observation graph for `ip neigh`.

5.4.3 Fusion Layer

The programmatic realization of the deductive closure operator $\Phi_{\Sigma,\mathcal{R}}$ is achieved by configuring the underlying `raw_repo` with a customized inference engine. This phase is responsible for autonomously transforming the isolated, fragmented observations into the fully materialized and consolidated KG, G^*.

Inference engine configuration and custom ruleset ($\mathcal{R}$)

To implement the infrastructural logic required to resolve network identities and discover implicit topological links, the framework's reasoner is driven by a specifically engineered rule file. This file acts as the concrete implementation of the deductive rule set $\mathcal{R}$.

The semantic engine is bootstrapped using the pre-compiled OWL2-RL profile as its foun-

dation. This standard ruleset is highly optimized for scalable data fusion and autonomously handles universal ontological operations, such as enforcing domain/range constraints, structurally promoting individuals across class hierarchies, and managing basic `owl:sameAs` equivalence chains.

However, standard OWL2-RL is inherently infrastructure-agnostic. To enforce real-world networking behavior, we extended this foundation by injecting a custom set of domain-specific deductive rules. These custom rules act as a strictly functional complement to the structural constraints already embedded within the optimized T-Box Σ (such as the `owl:hasKey` constraint enforcing MAC address uniqueness). While the T-Box dictates valid data shapes and triggers structural identity locks, the custom rules explicitly execute the cross-layer merging logic, such as recursively merging separate `:PhysicalDevice` nodes when they share the exact same physical `:NetworkInterfaceL2` entity.

Execution paradigm: batch materialization

Operationally, the framework is architected to execute the **Batch Fusion** paradigm, strictly adhering to the mathematical formulation established in Section 4.3.3. Rather than applying deductive closure continuously over an incoming telemetry stream, the system defers reasoning until the complete data collection cycle concludes.

Once all localized observation subgraphs are collected and logically unioned within the staging repository, the GraphDB reasoning engine is invoked a single time over the aggregated dataset. Driven by the customized ontological rule file, the engine resolves cross-layer identities and infers implicit topological links to produce the final materialized state G^*_{batch}. By centralizing the deductive workload into a single computational pass, this execution strategy minimizes transactional overhead and drastically optimizes the database's throughput when resolving extensive enterprise topologies.

5.4.3.1 Example trace: fusion engine

Building on the deductive pipeline formalized in Section 4.3.3.1, this section traces how the running scenario is concretely executed against the `raw_repo`. For typographical clarity, lengthy MAC-based IRIs are abbreviated (e.g., `:NeighIface_02...PC1`).

Following the batch materialization paradigm, the two initially isolated observation graphs, $G_{\texttt{neigh\_PC1}}$ (Figure 5.4) and $G_{\texttt{sh\_int\_R1}}$ (Figure 5.5), are loaded into the triplestore's staging area. The framework then submits their global union $G = G_{\texttt{neigh\_PC1}} \cup G_{\texttt{sh\_int\_R1}}$ to GraphDB's forward-chaining OWL 2 RL reasoner, which materializes the inferred statements directly into the default inference graph.

- **Phase 1 (type promotion):** The engine fires rule R_{inferL2} (4.23). The node `:NeighIface_02...PC1`, which initially lacked an explicit `rdf:type` in the subjective ARP observation, is formally promoted to `:NetworkInterfaceL2`. This satisfies the strict domain preconditions required for the identity resolution step.
- **Phase 2 (identity resolution):** The engine fires rule R_{sameIP} (4.24). The reasoner detects that both the ARP-derived interface `:NeighIface_02...PC1` and the router's physical interface `:Iface_R1_FastEthernet1_0` are bound to the exact same IP address (`10.46.124.1`). Consequently, it materializes the semantic equivalence links:

  (`:NeighIface_02...PC1`, `owl:sameAs`, `:Iface_R1_FastEthernet1_0`)

  (`:Addr_10.46.124.1`, `owl:sameAs`, `:Addr_R1_FastEthernet1_0_10.46.124.1`)

The resulting epistemic conflict—the merged interface entity inheriting two competing MAC addresses with distinct credibility scores (0.72 and 1.00)—is preserved seamlessly within G^* under the OWA, keeping its RDF* annotations intact. Figure 5.6 illustrates the final synchronized graph rendered by GraphDB, where the `owl:sameAs` bridge merging the two previously disjoint subgraphs is clearly observable. While the credibility metadata is visually abstracted in the figure for structural clarity, its definitive resolution is deferred to the Output Layer (Section 5.4.4).

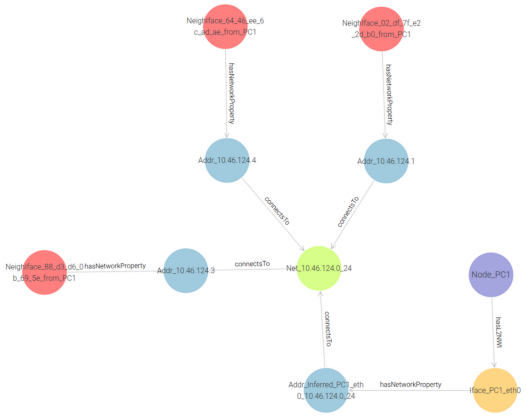

Figure 5.4: Observation graph $G_{\texttt{neigh\_PC1}}$ from PC1's `ip neigh` payload.

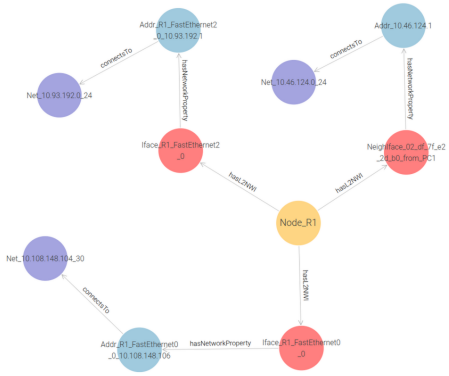

Figure 5.5: Observation graph $G_{\texttt{sh\_int\_R1}}$ from R1's `show interfaces` payload.

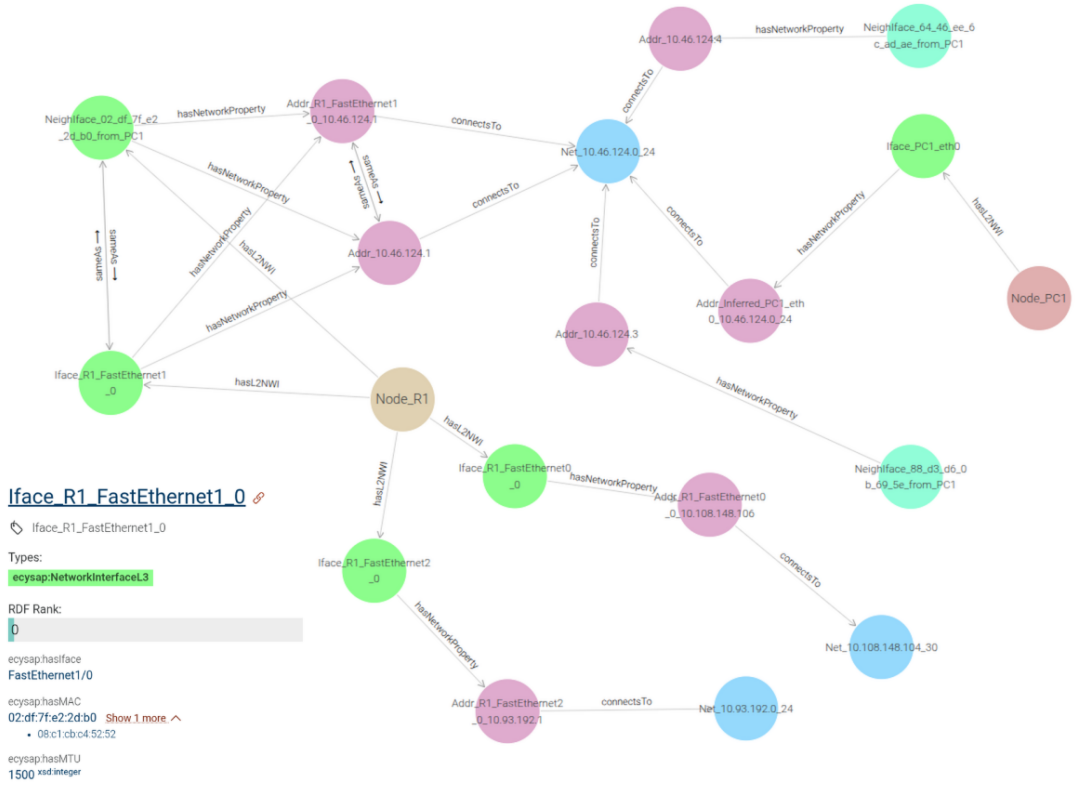

Figure 5.6: The inferred knowledge graph G^*.

5.4.4 Output Layer

The final phase of the framework implementation operationalizes the extraction of the definitive KG. Acting as the ultimate programmatic filter, the Output Layer queries the fully materialized staging graph (G^*) to shed redundant telemetry and export the final operational model (G_{out}).

Query management and execution architecture

To ensure modularity and maintainability, the programmatic synthesis is driven by a decoupled software architecture consisting of a Query Factory and a Synthesis Executor. A dedicated factory class encapsulates the complex SPARQL query chains $\mathcal{Q}$, acting as a centralized dictionary of traversal logic. Decoupled from the definitions, the Synthesis Executor acts as the operational engine: it retrieves these pre-defined queries and systematically orchestrates their execution against the GraphDB REST API.

Entity-centric traversal and epistemic filtering

The execution cascade directly maps to the formal entity-centric traversal methodology. The Synthesis Executor first evaluates the database to establish the focal root set $\mathbf{N}_{root}$. Programmatically, this initializes by targeting all resolved `:PhysicalDevice` entities alongside their core `:NetworkInterface` nodes.

Crucially, at every step of this extraction process, the SPARQL queries are engineered to act as the concrete implementation of the epistemic selection operator σ_{max}. Leveraging GraphDB's native support for querying RDF* nested triples, the queries actively inspect the `:hasCredibility` annotations appended to each structural link. When the traversal navigates functional properties—such as resolving the authoritative MAC address, mapping the correct IP configuration, or discovering neighbor adjacencies—specific SPARQL clauses dynamically evaluate the epistemic weights. This enforces a strict maximum-credibility filter, ensuring that competing assertions are resolved mathematically on the fly. From the initial root devices, the executor recursively traverses the graph outward, mapping the entire interconnected topology using strictly the most authoritative facts.

Serialization and final export

Once the query chain has successfully navigated and filtered the entire logical perimeter, the resulting deterministic view forms the refined operational model G_{out}. The Synthesis Executor serializes this distilled topology into a pristine RDF file format. Finally, to strictly separate the validated synthesis from the raw telemetry, the framework automatically exports and commits this serialized model into a final, dedicated repository within GraphDB, namely `out_repo`.

5.4.4.1 Example trace: SPARQL extraction

Building on the conflict state preserved in G^* (Section 5.4.3.1), the Output Layer executes the query cascade $\mathcal{Q}$ formalized in Section 4.3.4.1 against `raw_repo`.

Starting from the device roots `:Node_R1` and `:Node_PC1`, the Knowledge Synthesizer traverses the outward topological edges via the SPARQL extraction closure. When the traversal reaches the consolidated gateway interface, it issues a credibility-aware property resolution query: for each functional property p in conflict, the query retrieves all competing $(t_i, \chi(t_i))$ pairs from the RDF* annotation layer and applies σ_{max} inline.

For the `:hasMAC` conflict on the merged gateway interface, the query engine evaluates the two assertions materialized in the previous phase:

$$\chi(t_1) = 0.72 \quad (\texttt{"02:df:7f:e2:2d:b0"}, \text{PC1 } \texttt{ip neigh})$$
$$\chi(t_2) = 1.00 \quad (\texttt{"08:c1:cb:c4:52:52"}, \text{R1 } \texttt{show interfaces})$$

The operator deterministically selects t_2, suppressing the ARP-cache artifact from the output projection. The `owl:sameAs` bridge, no longer needed for property resolution, is collapsed in the output view: the merged entity surfaces as a single, unambiguous node with one MAC address and one IP address.

Figure 5.7 shows the resulting graph G_{out} as rendered by GraphDB: the `sameAs` edges are absent, each interface carries exactly one MAC address, and the epistemically inferior assertion has been silently discarded.

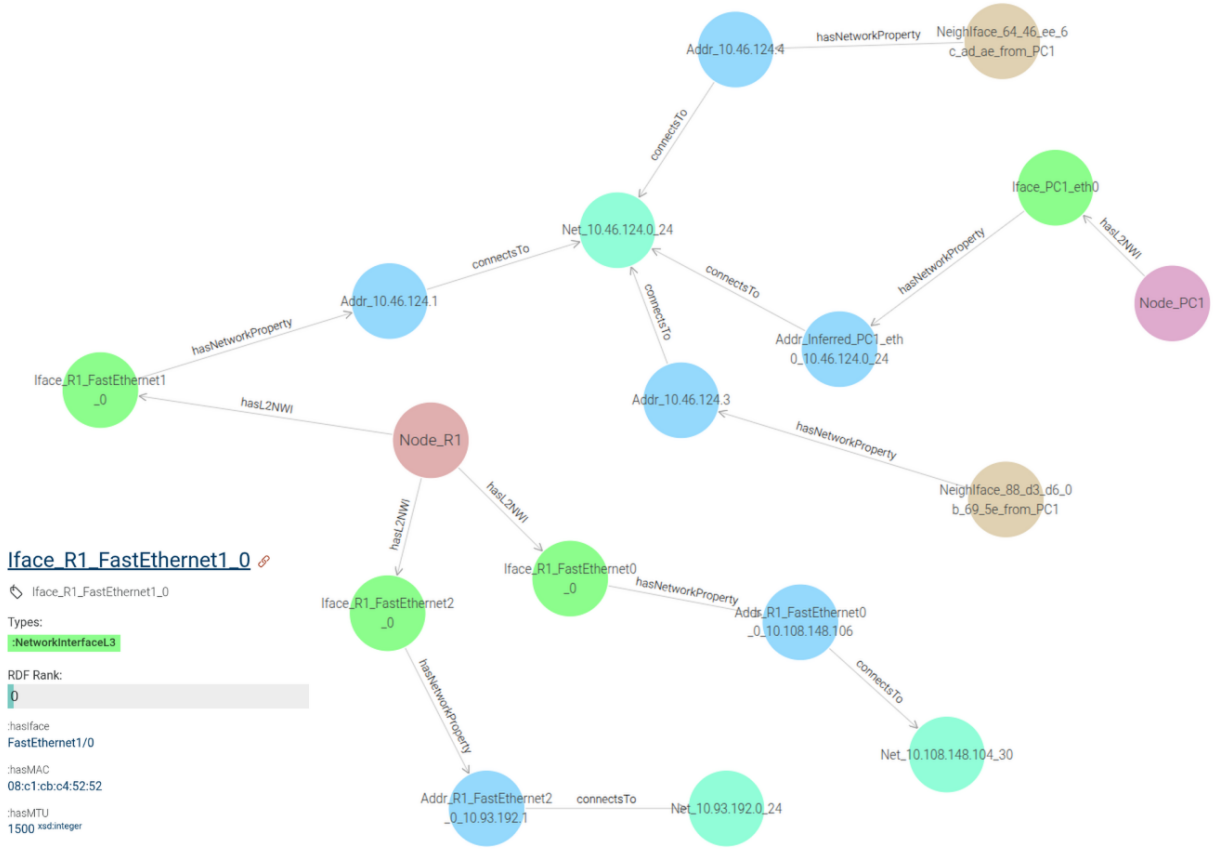

Figure 5.7: The final knowledge graph G_{out}.

Chapter 6

Experiments and evaluation

6.1 Empirical Scope and Emulation Environment

To bridge the theoretical methodology with the experimental validation, it is necessary to establish the exact empirical boundaries and technological constraints of the emulation environment. While the mathematical model formalized in Chapter 4 remains strictly hardware-agnostic, the reproducible evaluation of the framework relies on a designated set of standardized virtualization technologies and operational assumptions.

Infrastructure emulation and topology scaling

All target infrastructures evaluated in this chapter were programmatically synthesized using the stochastic mechanism detailed in Section 5.2.1 under the **Stochastic Topology Generation Engine**. The architectural components were instantiated using two distinct virtualization paradigms:

- **Routing devices:** All core and edge Layer 3 nodes were emulated using Cisco 7200 router images. This guarantees that the authoritative diagnostic commands generate authentic, vendor-specific telemetry.
- **Peripheral hosts:** All network leaf nodes were instantiated as lightweight Alpine Linux Docker containers, providing a standardized footprint for executing the generic OS utilities and active scanning tools.

Crucially, for all evaluated infrastructure configurations, the exponential scaling dimension N explicitly and exclusively defines the cardinality of the peripheral container nodes (the hosts). The core routing devices required to logically interconnect these subnets are dynamically generated by the stochastic engine and do not count toward the baseline dimension N.

Hardware allocation

The experimental simulations were orchestrated within a high-performance GNS3 virtual machine deployment. The underlying hardware was provisioned with 64 symmetric CPU cores and 128 GB of RAM. To ensure unconstrained deductive materialization, 32 GB of RAM were exclusively allocated to the GraphDB semantic reasoning engine. The remaining compute cores and memory pools were dynamically shared between the GNS3 network hypervisor, the Docker daemon, and the graph traversal query engine.

Contextual completeness hypothesis

Finally, all evaluation scenarios were conducted under the hypothesis of perfect spatial metadata extraction. Specifically, the operational context 5-tuple $k = (h, r, o, u, \tau)$, as strictly formalized in the methodology, is assumed to be fully and accurately populated for every telemetry payload injected into the Semantic Layer.

6.2 Evaluation Metrics and Validation Methodology

To rigorously assess the fidelity and resilience of the Semantic Reconstruction Framework, we establish a formal evaluation methodology that compares the synthesized operational graph G_{out} (Section 4.3.4) against a ground-truth graph G_{gt}, representing the absolute topological state of the target infrastructure. The evaluation is structured across five complementary dimensions: **informational metrics** (Section 6.2.3) assess the correctness of extracted values for aligned entity pairs; **population metrics** (Section 6.2.2) compare instance and assertion counts class by class; **topological metrics** (Section 6.2.4) evaluate global graph structure via connectivity; **semantic metrics** (Section 6.2.5) detect onto-

logical misclassifications via relational Weisfeiler-Lehman (WL) embeddings; and **computational scalability** (Section 6.2.6) measures the execution throughput and reasoning overhead across environments of exponentially increasing magnitude.

6.2.1 Experimental setup and observation strategies

To validate framework scalability, we leverage a set of target infrastructures defined by their host node count: $\mathcal{E} = \{E_4, E_8, E_{16}, E_{32}, E_{64}, E_{128}, E_{256}, E_{512}\}$. While the full set $\mathcal{E}$ assesses computational throughput, the evaluation of the framework's deductive resilience under constrained visibility is strictly isolated to the E_{64} topology. This environment provides the necessary structural complexity to test the reasoning engine.
For the E_{64} target, we simulate realistic operational deployments by applying four progressively constrained observation strategies:

- **S1 - Ideal Visibility:** The complete diagnostic toolset Π (as defined in Section 5.4.1) is executed locally on every device in the network (both routers and host nodes). This assumes absolute administrative access and establishes the structural ground truth.

- **S2 - Hybrid Reconnaissance:** The complete toolset Π is executed exclusively on the authoritative routers. Conversely, telemetry from the host nodes is heavily restricted, executing only the active network scanning component (i.e., `nmap`) via a single candidate device per subnet targeting its local segment.

- **S3 - Router-Only Visibility:** The complete toolset Π is executed exclusively on the authoritative routers. Telemetry collection from all peripheral host nodes is completely revoked. This scenario evaluates the reasoner's capacity to deduce edge connectivity and host presence relying strictly on the centralized routing and ARP tables of the core infrastructure.

- **S4 - Peripheral Scanning:** Data collection from routers is completely revoked. Telemetry is derived exclusively from the active `nmap` scans executed by the single

candidate device in each subnet, rigorously testing the reasoner's ability to reconstruct the core topology using purely external, peripheral data.

This tiered observation matrix is designed to evaluate the framework's deductive capabilities across varying gradients of infrastructural knowledge. By methodically isolating data sources, these scenarios allow us to explicitly quantify the structural impact of centralized routing intelligence versus localized peripheral telemetry on the accuracy of the final reconstructed knowledge graph.

6.2.2 Population metrics

While the informational metrics evaluate the factual accuracy of the extracted data values via IRI-agnostic multiset matching, population metrics address a complementary, macroscopic question: does G_{out} populate the ontological classes and their associated relationships to the same extent as G_{gt}?

Class population

For each ontological class $C \in \mathcal{C}$ we compare the instance counts:

$$n_C^{gt} = |\{x \in G_{gt} : \texttt{type}(x) = C\}|, \qquad n_C^{out} = |\{x \in G_{out} : \texttt{type}(x) = C\}|$$

From these two counts we derive two complementary indicators.

- The *ratio*
$$\rho_C = \frac{n_C^{out}}{n_C^{gt}}$$
measures relative volume, with $\rho_C = 1$ indicating exact population parity, $\rho_C < 1$ indicating under-population (missed instances), and $\rho_C > 1$ indicating over-population (spurious instances).

- The *coverage* κ_C is a symmetric score bounded in $[0, 1]$:
$$\kappa_C = \frac{\min(n_C^{gt},\, n_C^{out})}{\max(n_C^{gt},\, n_C^{out})}$$
which equals 1 only when $n_C^{gt} = n_C^{out}$ and degrades symmetrically regardless of the direction of the discrepancy.

Property population

An analogous analysis is applied independently to data properties and object properties. For each property $p \in \mathcal{P}_{data} \cup \mathcal{P}_{obj}$, we count the total number of assertions (i.e., triples of the form $\langle s, p, o \rangle$) present in each graph:

$$n_p^{gt} = |\{\langle s, p, o \rangle \in G_{gt}\}|, \qquad n_p^{out} = |\{\langle s, p, o \rangle \in G_{out}\}|$$

The same ratio and coverage defined for classes are computed for each property. This allows the evaluation to detect not only missing or spurious nodes, but also cases where nodes are present but systematically under-annotated (e.g., `PhysicalDevice` instances that lack `hasMAC` assertions) or over-annotated (e.g., spurious `connectsTo` edges that do not exist in the ground truth).

6.2.3 Informational metrics

The informational metrics dimension evaluates the accuracy and completeness of the explicit structural attributes assigned to the infrastructural entities. Specifically, it assesses the accuracy and the syntactic fidelity of the literal values mapped through datatype properties ($\mathcal{P}_{data}$) within the synthesized operational graph G_{out}, compared against the absolute truth defined in G_{gt}.

A fundamental methodological challenge in evaluating the synthesized KG lies in the inherent disparity of IRIs. In a conventional graph comparison scenario—where nodes are assumed to possess deterministic, globally synchronized identifiers—evaluation is a trivial exercise of 1:1 IRI matching to verify the associated properties. However, this assumption fundamentally contradicts the paradigm of decentralized infrastructure discovery. Because the Semantic Lifting operator dynamically constructs entity IRIs directly from their localized observational attributes, and since these subjective attributes may be incomplete or diverge from the absolute ground truth, achieving a direct entity-to-entity alignment between G_{out} and G_{gt} via strict IRI equivalence is analytically intractable. Rather than attempting to reconcile mismatched IRIs, the evaluation extracts and evaluates the global multiset of data property values natively associated with each ontological class.

Formalization of multiset value extraction

For any given taxonomic class $C \in \mathcal{C}$ (e.g., `:PhysicalDevice`, `:NetworkInterface`), let $\mathcal{P}_C \subset \mathcal{P}_{data}$ be the subset of datatype properties logically applicable to C.
We define a projection operator that queries a given graph G and compiles all structurally valid literal values for a specific property $p \in \mathcal{P}_C$ into a multiset, allowing for the natural retention of duplicate values.
The extracted value multisets for the ground-truth graph and the synthesized output graph are formalized as:

$$\mathcal{B}_{p_C}^{gt} = \{v \mid \exists s \in \mathcal{U}_{IRI} : (s, \texttt{rdf:type}, C) \wedge (s, p, v) \in G_{gt}\} \tag{6.1}$$

$$\mathcal{B}_{p_C}^{out} = \{v \mid \exists s \in \mathcal{U}_{IRI} : (s, \texttt{rdf:type}, C) \wedge (s, p, v) \in G_{out}\} \tag{6.2}$$

The consume-and-verify matching algorithm

Once the foundational property multisets are established, the framework executes a consume-and-verify matching algorithm. This IRI-agnostic approach iteratively evaluates the overlap between the literal pools to derive the standard classification metrics: True Positives (TP), False Positives (FP), and False Negatives (FN).
Because the pools are multisets, a correct match requires not only equivalence in value but also equivalence in multiplicity (i.e., a value appearing twice in the ground truth must be matched by two identical occurrences in the output graph). The metrics for a specific property p are computed via multiset arithmetic:

- **True Positives (TP_p):** The cardinality of the multiset intersection, representing correctly inferred property values.

$$TP_p = |\mathcal{B}_p^{gt} \cap \mathcal{B}_p^{out}|$$

- **False Negatives (FN_p):** The cardinality of the multiset difference, representing ground-truth values that the framework failed to discover or synthesize.

$$FN_p = |\mathcal{B}_p^{gt} \setminus \mathcal{B}_p^{out}|$$

- **False Positives (FP_p):** The cardinality of the inverse multiset difference, representing extraneous or hallucinated property values generated by the framework that do not exist in the ground truth.

$$FP_p = |\mathcal{B}_p^{out} \setminus \mathcal{B}_p^{gt}|$$

From these foundational counts, the system computes the **Precision (P_p)**, **Recall (R_p)**, and **F1-Score ($F1_p$)** for every datatype property.

6.2.4 Topological and structural metrics

Beyond node-level content, the KG encodes global network properties that require structural evaluation. To assess this, both G_{gt} and G_{out} are projected into directed, IRI-agnostic mathematical graphs: each RDF resource becomes an anonymous node, each object-property triple $\langle s, p, o \rangle$ becomes a directed edge labeled p, and all literal values are discarded. This strictly structural approach neutralizes the IRI mismatch problem, making all subsequent metrics completely invariant to the independent naming conventions of the two graphs.
The evaluation computes six families of topological metrics:

Structural counts

The fundamental structural indicators are the node count $|V|$ and edge count $|E|$ of each graph. Specifically, $|V|$ counts the distinct IRI-identified individuals (A-Box instances) present in the graph, while $|E|$ counts the distinct object-property assertions linking them. Their respective ratios:

$$\rho_V = \frac{|V_{out}|}{|V_{gt}|}, \qquad \rho_E = \frac{|E_{out}|}{|E_{gt}|} \tag{6.3}$$

quantify the degree of structural completeness of the reconstructed topology relative to the ground truth, where $\rho_V = \rho_E = 1$ indicates a perfect match in size.

Degree distribution similarity

Structural fidelity at the distributional level is evaluated by comparing the out-degree

sequences of the two graphs. Let $\mathbf{d}_{gt}$ and $\mathbf{d}_{out}$ denote the empirical out-degree distributions of G_{gt} and G_{out} respectively. The similarity is computed as:

$$\delta_{deg} = 1 - \frac{W_1(\mathbf{d}_{gt},\, \mathbf{d}_{out})}{\max(\mathbf{d}_{gt} \cup \mathbf{d}_{out})} \tag{6.4}$$

where W_1 denotes the first Wasserstein distance[1]. This metric quantifies the minimal mathematical "work" required to transform the probability distribution of the ground truth degrees into the reconstructed one. By normalizing it against the maximum observed degree, δ_{deg} provides a robust similarity score ($\delta_{deg} = 1$ indicating identical distributions) that is highly sensitive to missing hubs (e.g., core routers) without requiring direct node alignment.

Path length and diameter

Average Path Length (APL) and graph diameter are computed on the largest Weakly Connected Component (WCC) of the undirected projection. A WCC is formally defined as a maximal subgraph where any two vertices are connected to each other by some path, strictly disregarding the direction of the edges. APL measures the mean shortest-path distance between all reachable node pairs. To maintain directional consistency with the other similarity metrics, the APL comparison is expressed as:

$$\delta_{\text{APL}} = 1 - \Delta_{\text{APL,norm}}, \quad \Delta_{\text{APL,norm}} = \frac{|\text{APL}_{out} - \text{APL}_{gt}|}{\max(\text{APL}_{out},\, \text{APL}_{gt})} \tag{6.5}$$

where $\delta_{\text{APL}} = 1$ indicates perfect path-length fidelity. The analogous quantity δ_{diam} captures similarity in peripheral eccentricity.

Approximate Graph Edit Distance

An IRI-agnostic, upper-bound estimate of the Graph Edit Distance (GED) is computed strictly on size metrics:

$$\widehat{\text{GED}}(G_{gt}, G_{out}) = \big||V_{gt}| - |V_{out}|\big| + \big||E_{gt}| - |E_{out}|\big| \tag{6.6}$$

[1]Wasserstein distance is defined as a metric that evaluates the distance between a reference distribution Q and an arbitrary distribution L based on the p-norm, representing the minimal cost of transforming one distribution into the other.

This formulation counts the minimum absolute number of node and edge insertions or deletions required to match the graph sizes, providing a conservative but universally valid upper bound on the true GED. Normalized and converted to a similarity score consistent with the other metrics:

$$\delta_{\text{GED}} = 1 - \widehat{\text{GED}}_{\text{norm}}, \quad \widehat{\text{GED}}_{\text{norm}} = \frac{\widehat{\text{GED}}}{|V_{gt}| + |V_{out}| + |E_{gt}| + |E_{out}|} \tag{6.7}$$

where $\delta_{\text{GED}} = 1$ indicates structurally identical graphs and $\delta_{\text{GED}} = 0$ indicates maximal divergence.

6.2.5 Semantic metrics

The metrics introduced in the previous sections do not capture a subtler class of reconstruction errors: the misuse of the ontological type hierarchy. The ontology defines a rich inheritance structure—`PhysicalDevice` specialises into `PhysicalServer`, `NetworkDevice`, and so on—and a reconstruction that instantiates a node under the correct parent class but the wrong leaf class goes undetected by all previous metrics, yet encodes a false semantic assertion that may propagate incorrect inferences through OWL reasoning. Semantic metrics address this gap by evaluating whether entities are typed at the correct level of specificity and whether their relational context is consistent with the semantic commitments of G_{gt}.

Feature extraction via relational-WL embedding

Simple random walks often fail to capture the branching complexity of network infrastructure. Instead, we map the localized subgraph of each node into a continuous latent space using a relational WL adaptation tailored to RDF knowledge graphs.

The embedding function maps each node v to a fixed-size real-valued vector:

$$f_h : V \to \mathbb{R}^{|\mathcal{C}|+|\mathcal{P}_{obj}|} \tag{6.8}$$

where $\mathcal{C}$ is the set of all ontological classes defined in the T-Box (Section 5.3), $\mathcal{P}_{obj}$ is the set of all object properties, and h is the WL iteration depth. The vector has therefore two halves: the first $|\mathcal{C}|$ dimensions encode *what kind of node* v is, the second $|\mathcal{P}_{obj}|$ dimensions encode *how it is relationally connected* to its neighborhood.

Hop 0 — semantic class initialization

At iteration $h = 0$, each node is initialized with a vector whose first half encodes the node's ontological class and its ancestors, and whose second half is set to zero (relational context enters only during aggregation):

$$\mathbf{v}^{(0)} = \underbrace{\Big[\, \mathbf{c}(v) \,\Big\|\, \mathbf{0}_{|\mathcal{P}_{obj}|} \,\Big]}_{\text{size } |\mathcal{C}|+|\mathcal{P}_{obj}|} \in \mathbb{R}^{|\mathcal{C}|+|\mathcal{P}_{obj}|} \tag{6.9}$$

Where $\mathbf{c}(v)$ denotes the taxonomic signature vector of the node, capturing its explicit and inferred ontological hierarchy. To formalize its construction, let $\mathcal{C} = \{c^{(1)}, c^{(2)}, \ldots, c^{(|\mathcal{C}|)}\}$ be the fixed-order vocabulary of ontological classes defined in the T-Box (e.g., `:PhysicalDevice`, `:NetworkInterfaceL3`, `:NetworkAddress`, ...), and let $c_v \in \mathcal{C}$ denote the most specific declared class of node v (e.g., C_{netdev} for a router). The vector $\mathbf{c}(v) \in \mathbb{R}^{|\mathcal{C}|}$ has one dimension per class in the vocabulary: its k-th element, corresponding to class $c^{(k)}$, is defined as:

$$\mathbf{c}(v)_k = \begin{cases} 1 & \text{if } c^{(k)} = c_v \\ \dfrac{1}{2^{d(c_v,\, c^{(k)})}} & \text{if } c^{(k)} \text{ is an ancestor of } c_v \text{ at depth } d(c_v, c^{(k)}) \\ 0 & \text{otherwise} \end{cases} \tag{6.10}$$

where $d(c_v, c^{(k)})$ is the number of `rdfs:subClassOf` hops separating c_v from its ancestor $c^{(k)}$. The vector $\mathbf{c}(v)$ is then ℓ_2-normalized so that all node vectors lie on the unit hypersphere, making cosine similarity well-defined regardless of hierarchy depth.

This soft hierarchical encoding ensures that semantically proximate classes produce similar but distinguishable representations: for instance, C_{netdev} (`:NetworkDevice`) and C_{server} (`:PhysicalServer`) both share the ancestor C_{dev} (`:PhysicalDevice`) and will therefore produce vectors with partial overlap, while structurally unrelated classes such as C_{addr} (`:NetworkAddress`) and C_{dev} produce near-orthogonal representations.

Hops $1 \ldots h$ — relational aggregation

At each WL iteration $i \in \{1, \ldots, h\}$, the embedding of node v is updated by aggregating

contributions from both its outgoing and incoming edges. For each neighboring node $k \in \mathcal{N}(v)$ connected via object property r_{vk}, we form a *relational contribution vector*:

$$\mathbf{e}_{v,k}^{(i)} = \underbrace{\mathbf{v}_k^{(i-1)}\big|_{\mathcal{C}}}_{\text{class part of neighbor } k} \Big\| \underbrace{\mathbf{p}(r_{vk})}_{\text{one-hot of property } r_{vk}} \in \mathbb{R}^{|\mathcal{C}|+|\mathcal{P}_{obj}|} \tag{6.11}$$

where $\mathbf{v}_k^{(i-1)}\big|_{\mathcal{C}}$ denotes the first $|\mathcal{C}|$ dimensions of neighbor k's embedding at the previous iteration (i.e., *who the neighbor is*), and $\mathbf{p}(r_{vk})$ is the one-hot encoding of the connecting property over the vocabulary $\mathcal{P}_{obj}$ (i.e., *how it is connected*). The operator $\|$ denotes vector concatenation.

The neighborhood $\mathcal{N}(v)$ is bidirectional: it includes both the nodes that v points to (outgoing edges) and the nodes that point to v (incoming edges), so that every node correctly incorporates context from both directions of the directed graph.

The new embedding is then computed as a residual-weighted mean of all contributions, followed by ℓ_2-normalization:

$$\mathbf{v}^{(i)} = \frac{\alpha \cdot \mathbf{v}^{(i-1)} + (1-\alpha) \cdot \dfrac{1}{|\mathcal{N}(v)|} \sum_{k \in \mathcal{N}(v)} \mathbf{e}_{v,k}^{(i)}}{\left\| \alpha \cdot \mathbf{v}^{(i-1)} + (1-\alpha) \cdot \dfrac{1}{|\mathcal{N}(v)|} \sum_{k \in \mathcal{N}(v)} \mathbf{e}_{v,k}^{(i)} \right\|_2} \tag{6.12}$$

where $\alpha = 0.4$ is a residual weight that balances the node's own identity against the aggregated neighborhood context. After h iterations, $\mathbf{v}^{(h)}$ encodes both the semantic type of v and the relational structure of its h-hop neighborhood.

WL depth calibration

The WL depth h is calibrated per ontological target class to reflect the relational traversal depth mandated by the T-Box:

- **Physical Devices ($C_{\mathbf{dev}}$):** $h = 2$. A two-hop traversal is required to fully encapsulate a device's operational state: the first hop follows p_{hasL2NWI} to reach C_{iface}, while the second follows p_{netprop} to retrieve C_{addr} and C_{route}.
- **Network Classes ($C_{\mathbf{netL3}}$):** $h = 2$. To capture the full semantic scope of a network segment, a two-hop traversal is necessary. The first hop retrieves the C_{addr} instances

bound to the network, and the second hop proceeds to the associated C_{iface} nodes linked to each address. This traversal depth ensures that the embedding effectively captures the binding between logical IP addresses and their underlying interface-level properties within the subnet.

Cross-graph alignment and GSS score

Since IRIs carry no correspondence across graphs, alignment operates entirely in the embedding space. For a target class C, let V_C^{gt} and V_C^{out} be the node populations of class C (or any subclass) in each graph. The pairwise cosine similarity between any two nodes is:

$$S(v_i, u_j) = \frac{\mathbf{v}_i^{(h)} \cdot \mathbf{u}_j^{(h)}}{\|\mathbf{v}_i^{(h)}\| \, \|\mathbf{u}_j^{(h)}\|} \in [0, 1]$$

Let $\mathbf{S}$ be the matrix of pairwise similarities such that $\mathbf{S}_{ij} = S(v_i, u_j)$. A strict 1-to-1 assignment is enforced by solving the optimal bipartite matching problem on the cost matrix $\mathbf{W} = \mathbf{1} - \mathbf{S}$ via the Kuhn-Munkres (Hungarian) algorithm:

$$\sigma^* = \underset{\sigma \text{ injective}}{\operatorname{argmin}} \sum_{i=1}^{|M|} \big(1 - S(v_i, u_{\sigma(i)})\big)$$

where $|M| = \min(|V_C^{out}|, |V_C^{gt}|)$. The Global Structural Similarity (GSS) score is then:

$$\text{GSS}(C, h) = \frac{1}{|M|} \sum_{(i, \sigma^*(i)) \in M} S\big(v_i, u_{\sigma^*(i)}\big)$$

GSS measures the *quality* of extracted nodes independently of how many were extracted; the complementary coverage metric $\text{Recall}_C = |V_C^{out}|/|V_C^{gt}|$ quantifies completeness. For graphs where $|V_C^{out}|$ exceeds a tractability threshold, the exact Hungarian algorithm is replaced by an Approximate Nearest-Neighbour index with greedy deduplication, reducing complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(n \log n \cdot d)$.

6.2.6 Computational scalability

To ensure the framework remains operationally viable in enterprise environments, its computational footprint is systematically profiled against the exponentially scaled target infrastructures $E_N \in \mathcal{E}$ introduced in Section 6.2.1.

To precisely isolate performance bottlenecks, the execution pipeline for each target environment is partitioned into two discrete operational phases:

1. **Phase 1 - reasoning:** Encompasses the ingestion of the isolated observation subgraphs into the inference engine and the execution of the deductive closure operator $\Phi_{\Sigma,\mathcal{R}}$ to resolve topological identities and fully materialize the implicit knowledge.

2. **Phase 2 - extraction:** Encompasses the execution of the entity-centric traversal queries $\mathcal{Q}$ over the materialized graph, applying the epistemic selection operator σ_{max} to resolve observational conflicts and synthesize the final operational graph G_{out}.

For each defined infrastructure E_N, the evaluation continuously monitors the resource utilization across both individual phases and the global pipeline execution. The following hardware-agnostic metrics are recorded:

- **Execution time (T):** The chronological duration required to complete the logical operations, measured independently for the reasoning phase (T_{reason}), the extraction phase ($T_{extract}$), and as an absolute total duration ($T_{total} = T_{reason} + T_{extract}$).

- **CPU utilization (L):** The processor load exclusively driven by the semantic database engine during the materialization of inferences and the execution of traversal queries. It is recorded as both the continuous average utilization (L_{avg}) and the absolute peak stress threshold (L_{peak}), isolating the engine's footprint from the underlying host operating system.

- **Memory consumption (M):** The RAM footprint directly allocated by the semantic database engine to maintain the active graph state in memory and execute deductive logic. It is tracked as both the average memory allocation (M_{avg}) and the peak memory spike (M_{peak}) to evaluate the true spatial complexity and operational ceiling of the reasoning framework.

6.3 Results Analysis

This section presents a comprehensive empirical evaluation of the proposed Semantic Reconstruction Framework. Each subsection systematically analyses the results obtained under the four observation strategies {**S1**, **S2**, **S3**, **S4**} introduced in Section 6.2.1, progressively characterizing the framework's behavior from macroscopic population fidelity down to semantic correctness and computational viability.

6.3.1 Population metrics results

As formalized in Section 6.2.2, the population metrics evaluate the quantitative coverage of the Semantic Reconstruction Framework, assessing its ability to instantiate ontological classes and properties across the four observation strategies. To visually synthesize these results, Figure 6.1 and Figure 6.2 present the coverage scores κ for the extracted classes and properties, respectively, compared against the ground truth. Furthermore, to overcome the symmetric nature of κ, the visualizations integrate the *ratio* metric via directional arrows ($\uparrow$ for over-generation, $\downarrow$ for under-generation), providing immediate insight into the specific direction of the discrepancies.

Under the **S1** scenario, Figure 6.1 demonstrates that the framework achieves perfect coverage ($\kappa_C = 1.00$) across the populated classes. This confirms that when the complete diagnostic toolset Π is executed locally on every node, the framework successfully discovers and exactly instantiates every ontological class with the correct number of elements. However, as revealed by Figure 6.2, this structural fidelity is not absolute across all relational dimensions. While the vast majority of object and data properties exhibit complete saturation under **S1**, the property heatmap exposes a precise limitation in the current telemetry implementation. Datatype properties pertaining strictly to underlying hardware specifications, namely `hasCPUCores` and `hasRAMTotal`, consistently record a coverage of 0.00 across all observation strategies, including the otherwise exhaustive S1 scenario. This indicates that the currently selected diagnostic command pool Π lacks the explicit operating system utilities required to probe deep physical hardware resources.

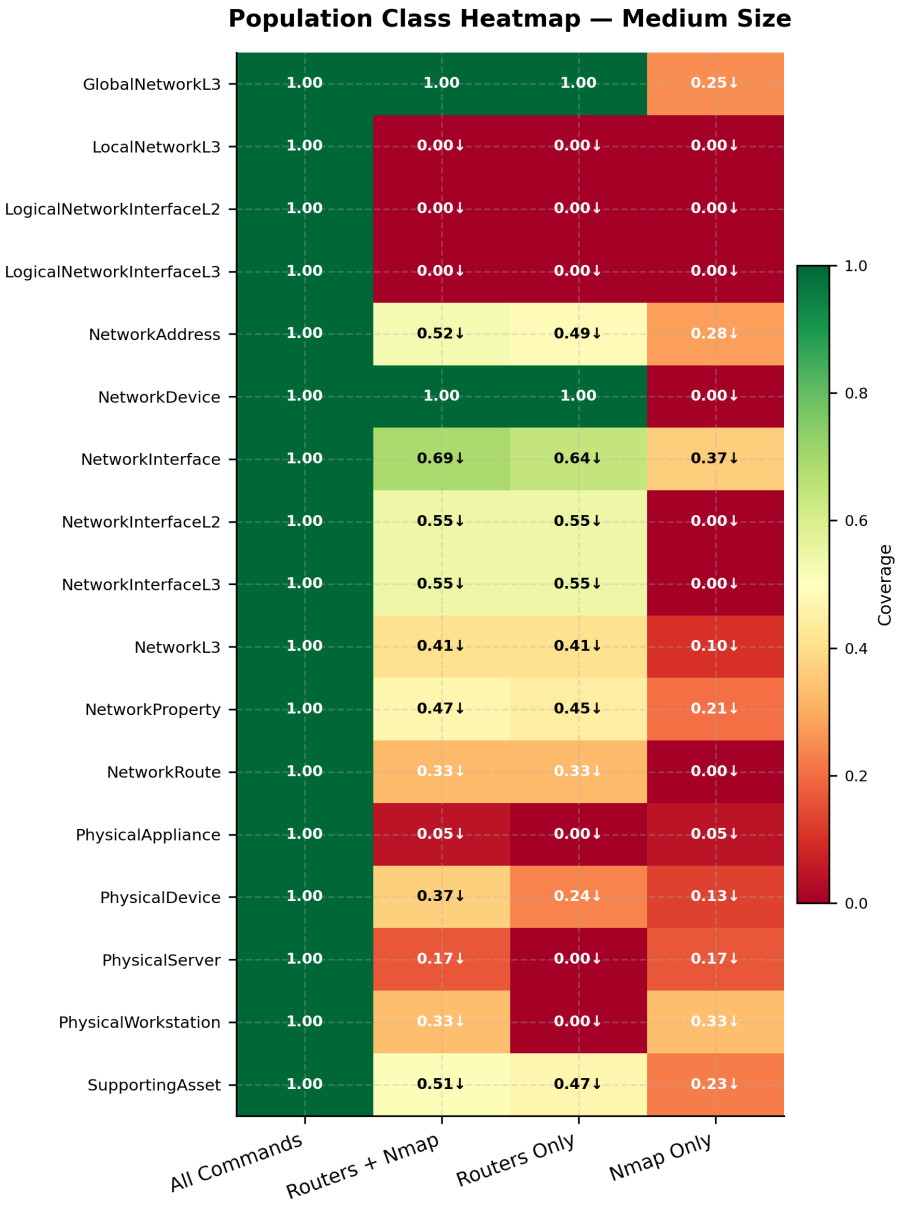

Figure 6.1: Population class coverage heatmap.

The **S2** and **S3** strategy scenarios reveal the significant deductive power derived from centralized infrastructure configurations. As observed in the middle columns of both heatmaps, authoritative routing knowledge alone (S3) is almost entirely sufficient to reconstruct the core complexity of the Layer 3 topology. Specifically, as illustrated in Figure 6.1, the back-

bone routing entities successfully discover all global network segments. However, because the specific local networks within our evaluated infrastructure reside exclusively on the peripheral host nodes, the routers lack direct visibility into these isolated edge segments. Consequently, as evidenced in Figure 6.2, the `hasCIDR` property fails to achieve absolute coverage under the S3 observation strategy. The introduction of active peripheral scanning via `nmap` in **S2**—restricted to a single candidate device per subnet—yields only a marginal quantitative improvement over **S3**.

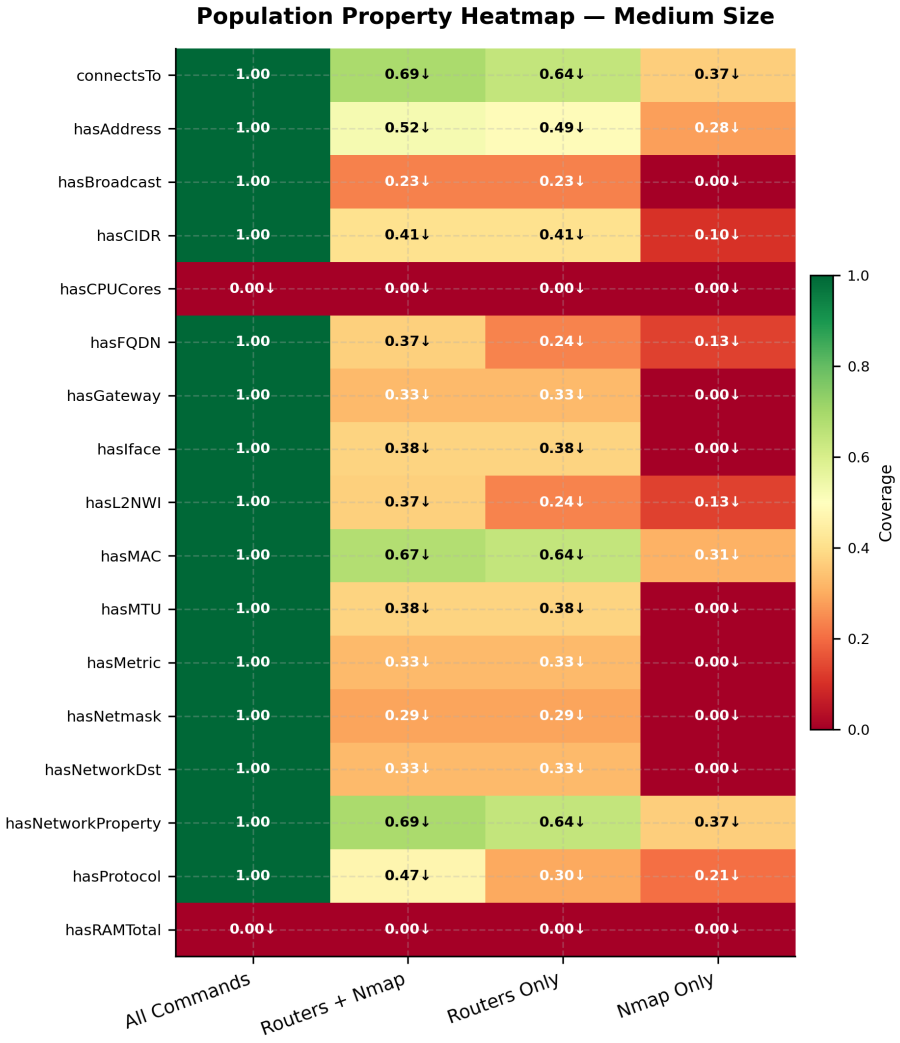

Figure 6.2: Population property coverage heatmap.

As shown in Figure 6.2, peripheral `nmap` scans in **S2** marginally improve the discovery of IP (`hasAddress`, 0.49 → 0.52) and MAC bindings (`hasMAC`, 0.64 → 0.67) compared to **S3**. Consequently, Figure 6.1 reflects a proportional coverage increase in the corresponding `NetworkAddress` and `NetworkInterface` classes.

Notably, this interface discovery is confined strictly to the generic `NetworkInterface` superclass. Lacking the deep telemetry required to reliably distinguish physical from virtual interfaces via external `nmap` scans, the reasoner conservatively halts classification at the highest verifiable ontological level to prevent semantic hallucinations.

Finally, **S4** isolates the core deductive contribution of these localized scans. Relying exclusively on `nmap` without authoritative routing data yields heavily degraded coverage across both metrics. This confirms that peripheral scanning provides only fragmented knowledge, reinforcing the framework's foundational reliance on core routing intelligence for comprehensive topological reconstruction. Notably, all non-perfect scores across both heatmaps are exclusively accompanied by downward arrows (↓), indicating that every discrepancy stems strictly from under-coverage rather than spurious over-generation.

6.3.2 Informational metrics results

While the population metrics verified the macroscopic coverage of the infrastructural entities, the informational metrics shift the focus to the microscopic fidelity of the synthesized data. As formalized via the consume-and-verify multiset matching algorithm in Section 6.2.3, this evaluation assesses whether the specific literal values contained within the discovered nodes accurately reflect the absolute ground truth.

To visualize this validation, Figure 6.3 presents the F1-Scores for the extracted datatype properties across the four observation strategies, while Figure 6.4 provides a global aggregation of Precision, Recall, and F1-Score to illustrate the overarching behavior of the reasoning engine under constrained visibility.

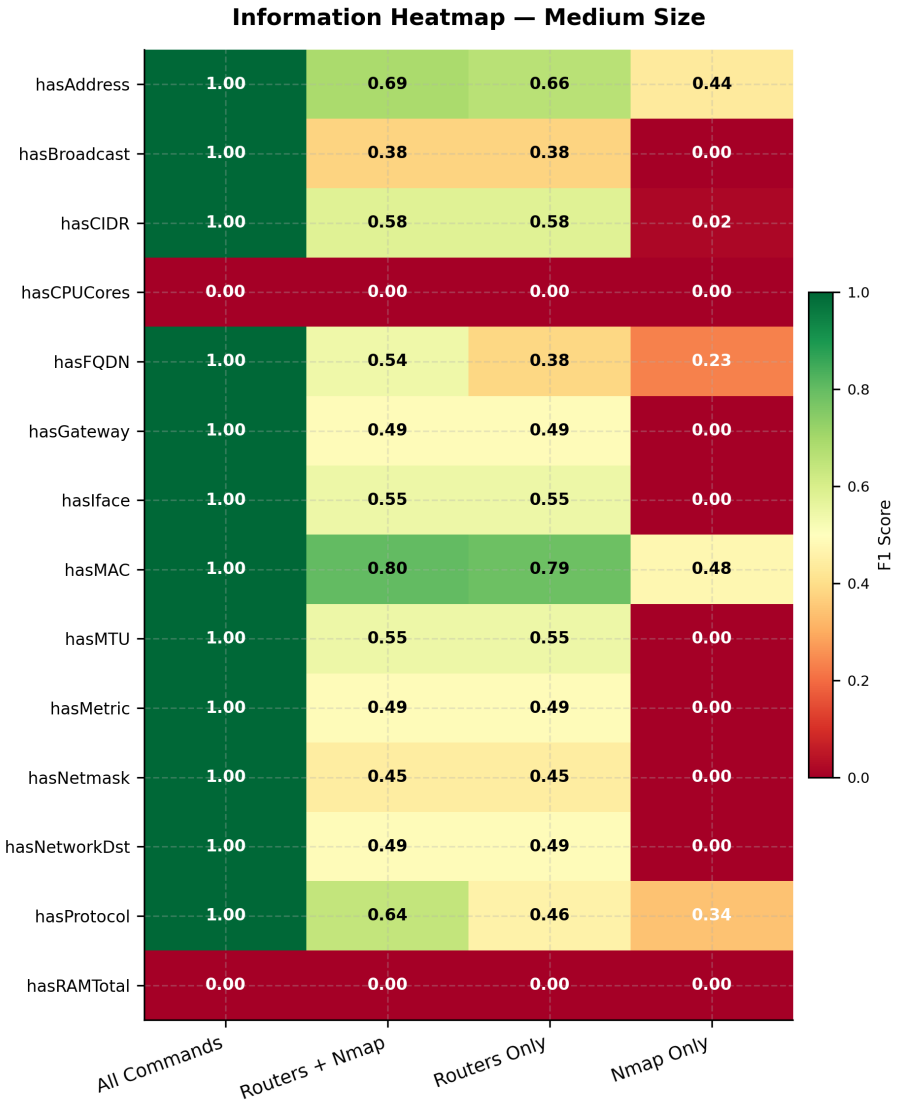

Figure 6.3: Informational F1-Score heatmap.

The informational heatmap Figure 6.3 completes the previous analysis, showing how the quantity of covered nodes also preserves the correct information. To provide a comprehensive understanding of these informational dynamics, Figure 6.4 plots the global multi-class metrics. The radar chart visually encapsulates a fundamental characteristic of the Semantic Reconstruction Framework: its epistemic conservatism.

The outer boundary perfectly traces the **S1** scenario, representing the theoretical optimum. For the partial visibility scenarios (**S2** and **S3**), the framework exhibits an asymmetric deformation: while Recall drops significantly (reflecting the inevitable loss of unobserved nodes and their internal values), Precision remains exceptionally high, pulling the **S2** and **S3** triangles tightly toward the top apex. This demonstrates that the epistemic selection operator actively prevents the reasoner from hallucinating data. Even when operating with severely limited topological knowledge, the literal values the framework deduce and assert are overwhelmingly correct, prioritizing structural truth over speculative completeness.

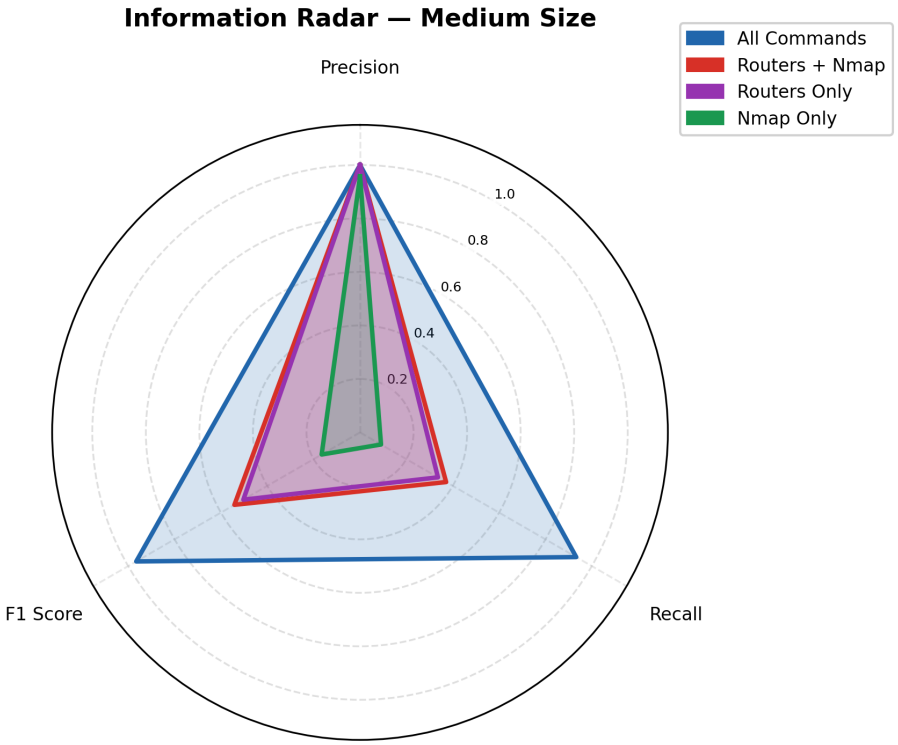

Figure 6.4: Information Radar Chart.

6.3.3 Topological metrics results

While the preceding evaluations focused on node-level factual accuracy and volumetric coverage, topological metrics assess the global structural fidelity of G_{out} against G_{gt}, independently of entity identifiers. Figure 6.5 visualizes performance across the five structural similarity scores defined in Section 6.2.4, where all axes are normalized in $[0, 1]$ with 1.0 indicating perfect alignment.

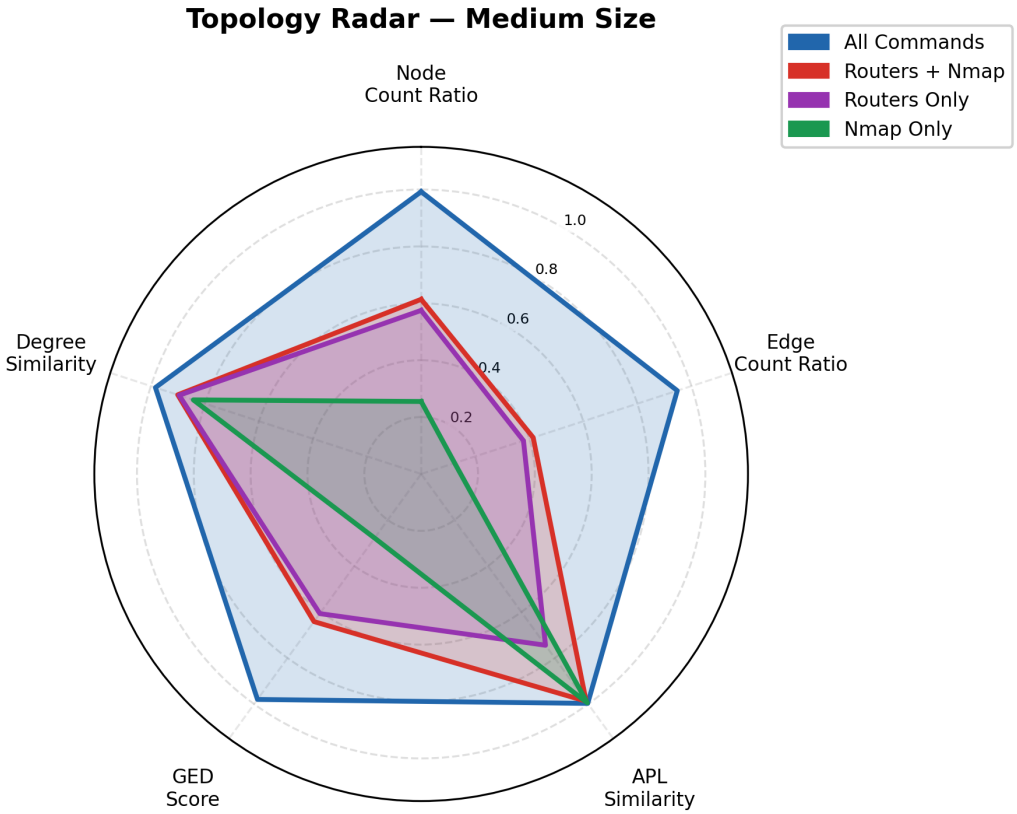

Figure 6.5: Topological Radar Chart.

The radar chart immediately confirms the deductive baseline established in the previous sections: under the **S1**, the framework achieves a high score across all five axes. This demonstrates that complete telemetry collection allows the Semantic Reconstruction Framework to perfectly capture not only the entities themselves but also the exact mathematical topology and connectivity patterns of the underlying infrastructure.

To systematically dissect the framework's behavior under constrained visibility, the structural metrics can be analyzed across three distinct analytical dimensions: **volumetric ratios**, **local relational coherence**, and **global navigational geometry**.

Volumetric ratios

The node count ratio and edge count ratio exhibit a proportional degradation as visibility becomes increasingly restricted. This uniform contraction across **S2**, **S3**, and **S4** perfectly mirrors the quantitative decline observed in the population metrics. Specifically, the decline in the node count directly corresponds to the drop in class population metrics, just as the reduced edge count reflects the decrease in property population metrics. Consequently, the approximate GED score—which is mathematically calculated from these absolute size differences—naturally degrades in parallel with these volumetric constraints.

Local relational coherence

A critical insight emerges from the degree similarity metric, which remains remarkably high and stable across all four scenarios—even in **S4**, where volumetric metrics collapse. In the semantic model of the reconstructed network, significant fluctuations in a node's degree are predominantly driven by an excess or deficit of connected `NetworkAddress` or `NetworkInterface` properties. The constancy of this metric proves that while the framework may fail to discover entire subgraphs (resulting in missing nodes and edges), the entities it successfully extract maintain a relational density highly coherent with the ground truth. There is no disproportionate hallucination of object properties; the localized topological footprint of discovered nodes remains structurally sound.

Global navigational geometry

The comparative analysis of **S2** and **S3** reveals a strong overall structural similarity, reaffirming that centralized routing intelligence dictates the core topology. However, a distinct divergence manifests in the APL similarity. In scenario **S3**, the synthesized graph is completely stripped of its peripheral host nodes. Structurally, hosts form dense, star-like clusters around their local subnets (where a single network node bridges multiple host IPs

alongside the gateway IP), which mathematically lowers the global average traversal distance. Their absence in **S3** leaves a sparser, linear routing backbone that artificially inflates the APL, causing a visible drop in similarity against G_{gt}. Conversely, in **S2**, the localized `nmap` telemetry discovers a portion of these leaf nodes, repopulating the dense geometric periphery and noticeably restoring the APL Similarity score toward the **S1** baseline.

6.3.4 Semantic metrics results

To conclude the structural evaluation, the semantic metrics dimension assesses whether the reconstructed graph G_{out} respects the formal semantic constraints and hierarchical type definitions mandated by the ontology, independently of volumetric completeness. As formalized in Section 6.2.5, this is quantified using GSS score, which leverages relational WL)embeddings to capture both the explicit ontological class of a node and the semantic signature of its relational neighborhood.

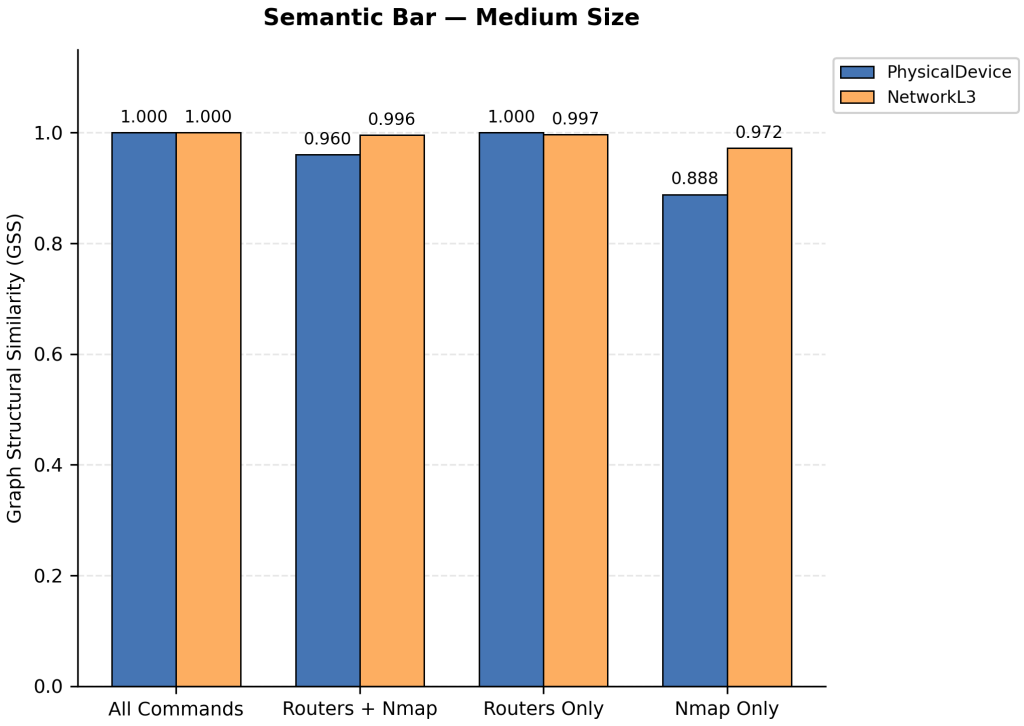

Figure 6.6: Semantic Bar Chart.

Figure 6.6 presents the GSS scores for the `PhysicalDevice` and `NetworkL3` target classes. The most prominent insight is the exceptionally high baseline across all observation strategies. This demonstrates a fundamental resilience of the framework: regardless of the visibility constraints or the volume of missing data, the nodes that are successfully discovered and instantiated are typed and relationally linked with strict semantic correctness. The reasoning engine guarantees that objects are mapped to valid ontological classes and interconnected via the correct object properties, completely preserving the logical coherence established by the T-Box.

A closer examination of the slight variations—specifically the minor degradation in the `PhysicalDevice` GSS for scenarios **S2** (0.960) and **S4** (0.888) compared to **S1** and **S3**—reveals the framework's designed response to observational ambiguity. As previously established in the population analysis, localized peripheral scanning via `nmap` often lacks the deep system-level telemetry necessary to decisively specialize a network interface into its exact leaf class (e.g., `NetworkInterfaceL2` or `NetworkInterfaceL3`).

Because the relational WL embedding aggregates both a node's explicit class and the classes of its neighbors, a `PhysicalDevice` in G_{out} connected to a generic `NetworkInterface` will exhibit a slight cosine distance from its G_{gt} counterpart, which is bound to a fully specialized leaf class. However, this mathematical variation strictly represents a graceful degradation in taxonomic specificity rather than a categorical semantic error. The framework's epistemic logic successfully avoids semantic hallucinations: while an instance might be modeled at a higher, more general level of the ontological hierarchy due to missing evidence, it is never assigned a fundamentally incorrect or incompatible semantic type.

6.3.5 Computational scalability results

This section evaluates the operational performance and resource footprint of the framework across the scaled target infrastructures $\mathcal{E} = \{E_4, E_8, E_{16}, E_{32}, E_{64}, E_{128}, E_{256}, E_{512}\}$. For clarity and ease of visualization, these configurations are sequentially mapped to the following categorical labels: *nano*, *micro*, *small*, *compact*, *medium*, *standard*, *large*, and *xlarge*. Resource utilization is measured via a lightweight background monitor that continuously polls GraphDB's native REST endpoint `/rest/monitor/infrastructure` [28]

throughout the execution of each pipeline phase, extracting peak and average values. To ensure statistical robustness, all reported metrics represent the mean over 10 independent runs per configuration.

Execution time

Figure 6.7 presents the execution time profiles across the infrastructure scales, decomposed into T_{reason} (Phase 1) and $T_{extract}$ (Phase 2), alongside the aggregate T_{total}.

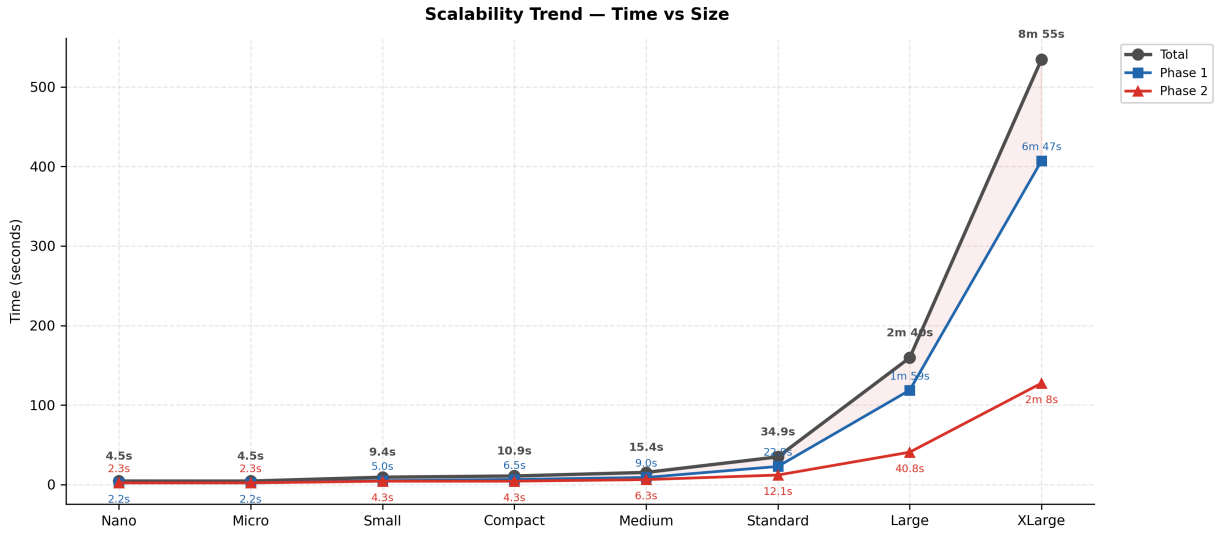

Figure 6.7: Scalability trend — execution time across infrastructure scales.

As the infrastructure scales from E_4 to E_{512}, T_{total} grows from 4.5 s to 8 m 55 s (535 s), with Phase 1 consistently representing the dominant cost: the deductive closure operator $\Phi_{\Sigma,R}$ alone accounts for 6 m 47 s (407 s) at the XLarge scale, reflecting the super-linear growth in inference materialization as the graph density increases. Phase 2 also scales upward but remains comparatively contained, reaching 2 m 8 s (128 s) at E_{512}, as the entity-centric traversal queries Q impose a more moderate overhead relative to the reasoning workload.

CPU Utilization

Figure 6.8 illustrates the peak and average CPU utilization recorded for both phases across all infrastructure scales.

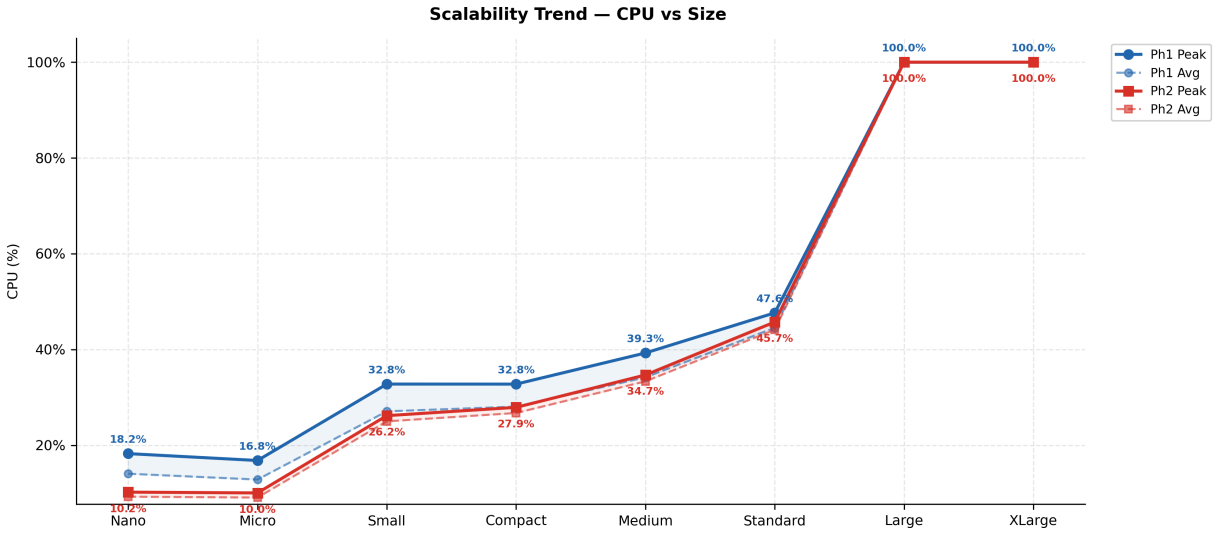

Figure 6.8: Scalability trend — CPU utilization across infrastructure scales.

CPU utilization grows steadily across both phases as the infrastructure scales, with Phase 1 consistently leading: its peak rises from 18.2% at E_4 to 47.6% at E_{64}, before saturating at 100% at the Large scale. Phase 2 follows a similar trajectory, starting from 10.2% and converging to the same ceiling at E_{256}. The concurrent saturation of both phases from the Large scale onward confirms that this environment represents the stress boundary of the current deployment configuration.

Memory Consumption

Figure 6.9 reports the peak and average RAM footprint allocated by the semantic database engine across both phases and all scales.

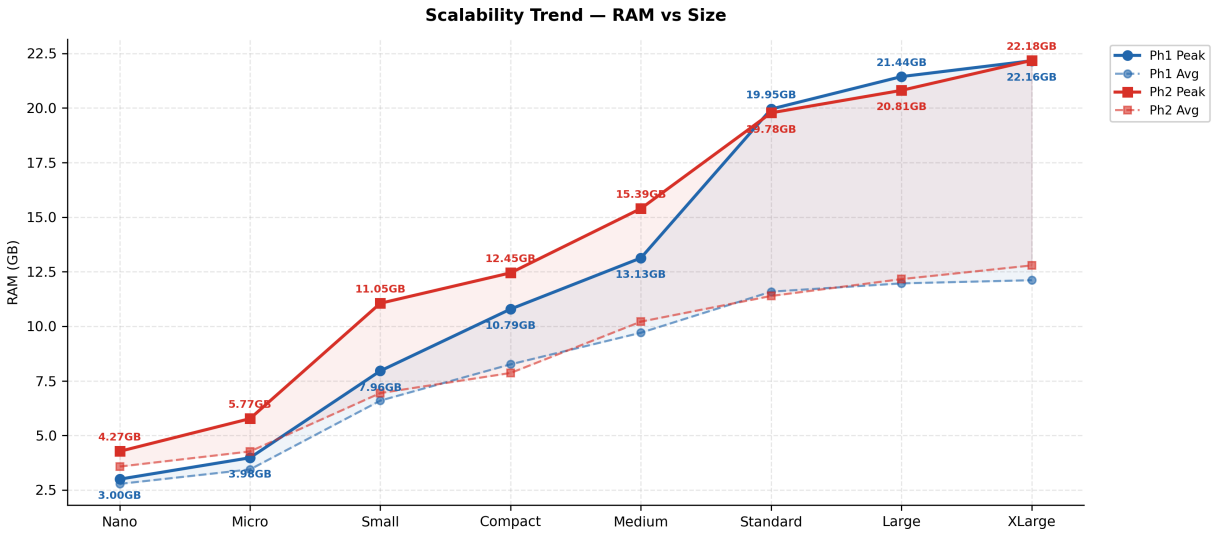

Figure 6.9: Scalability trend — RAM consumption across infrastructure scales.

Memory consumption increases with infrastructure size across both phases, reaching peak values of 22.16 GB and 22.18 GB for Phase 1 and Phase 2 respectively at E_{512}. Notably, Phase 2 exhibits a markedly higher peak footprint than Phase 1 at the smaller scales — reaching 11.05 GB versus 7.96 GB at the Small scale — as the extraction layer must simultaneously retain the fully materialized graph alongside the intermediate aggregation structures required by the epistemic selection operator σ_{max}. The two phases converge toward equivalent memory ceilings at larger scales, while the average utilization remains substantially lower than the peak across all scales, indicating that memory spikes are transient events bounded to specific pipeline stages rather than a sustained elevated footprint.

Chapter 7

Conclusions and future works

7.1 Conclusions

This thesis presented a formal semantic framework for the automated reconstruction of heterogeneous IT network infrastructures into unified KGs. Motivated by the growing inadequacy of fragmented, siloed discovery tools in the face of modern enterprise network complexity, the proposed system addresses a concrete and operationally critical gap: the absence of a principled, scalable methodology for synthesizing multi-source network observations into a coherent, auditable infrastructure model suitable for advanced security assessment.

The results of this thesis demonstrate that semantic data fusion, grounded in a formal ontology and driven by deductive reasoning, constitutes a viable and principled foundation for automated infrastructure modeling. By transforming fragmented, multi-perspective network observations into a unified, traceable, and logically consistent KG, the framework directly addresses the environmental visibility gap that underlies many of the most critical challenges in modern cybersecurity operations. As enterprise networks continue to grow in complexity and fluidity, the ability to maintain an accurate, machine-readable model of the operational environment will only increase in strategic importance — and the semantic approach developed in this thesis offers a rigorous starting point for that capability.

7.2 Future Works

The implemented framework provides a solid foundation for several future extensions aimed at increasing its expressive power and operational applicability. Naturally, integrating new tools or expanding to additional layers requires a corresponding extension of the underlying ontology to accurately capture the required level of detail.

OSI layer expansion

The current ontology is strictly scoped to OSI Layers 1–3. A natural extension is to expand the model upward to Layers 4–7 to represent active services, open ports, and application-layer protocols. This requires new ontological classes and OWL constraints to link these higher-level entities to their underlying network substrates.

Telemetry source expansion

To support the OSI layer expansion, the diagnostic command pool Π should be broadened to include tools for application fingerprinting, vulnerability scanning, and hardware profiling (e.g., `hasCPUCores`, `hasRAMTotal`). Each new tool will require dedicated semantic lifting specifications to map its raw output into the ontological observation layer.

LLM-assisted semantic lifting

Currently, parsing raw tool outputs relies on deterministic, hand-crafted extraction rules. Future work could integrate LLMs to autonomously interpret heterogeneous telemetry and generate ontological assertions. Using the formal ontology as a structured context would constrain the LLM's output space, mitigating the risk of hallucinations while drastically reducing the engineering effort needed to onboard new data sources.

Bibliography

[1] Martin J. O'Connor et al. "Supporting rule system interoperability on the semantic web with SWRL". In: *International Semantic Web Conference (ISWC 2005)*. Springer, 2005, pp. 974–986. DOI: 10.1007/11574620_69.

[2] Jens Bleiholder and Felix Naumann. "Data fusion". In: *ACM Computing Surveys (CSUR)* 41.1 (2009), pp. 1–41. DOI: 10.1145/1456650.1456651.

[3] Gordon "Fyodor" Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure.Com LLC, 2009. ISBN: 978-0-9799587-1-7.

[4] Pascal Hitzler et al. *OWL 2 Web Ontology Language Primer (Second Edition)*. W3C Recommendation. World Wide Web Consortium (W3C), 2012. URL: https://www.w3.org/TR/owl2-primer/.

[5] Boris Motik et al. *OWL 2 Web Ontology Language Profiles (Second Edition)*. W3C Recommendation. World Wide Web Consortium (W3C), 2012. URL: https://www.w3.org/TR/owl2-profiles/.

[6] Steve Harris and Andy Seaborne. *SPARQL 1.1 Query Language*. W3C Recommendation. World Wide Web Consortium (W3C), 2013. URL: https://www.w3.org/TR/sparql11-query/.

[7] Richard Cyganiak, David Wood, and Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation. World Wide Web Consortium (W3C), 2014. URL: https://www.w3.org/TR/rdf11-concepts/.

[8] Xin Luna Dong and Divesh Srivastava. "Big Data Integration". In: *Synthesis Lectures on Data Management* 6.3 (2015), pp. 1–198. DOI: 10.2200/S00578ED1V01Y201404DTM040.

[9] Olaf Hartig. "Foundations of RDF* and SPARQL*: An Alternative Approach to Statement-Level Metadata in RDF". In: *AMW 2017 – 11th Alberto Mendelzon International Workshop on Foundations of Data Management* (2017). URL: http://ceur-ws.org/Vol-1912/paper12.pdf.

[10] Holger Knublauch and Dimitris Kontokostas. *Shapes Constraint Language (SHACL)*. W3C Recommendation. World Wide Web Consortium (W3C), 2017. URL: https://www.w3.org/TR/shacl/.

[11] Ivan Brugere, Brian Gallagher, and Tanya Y. Berger-Wolf. "Network Structure Inference, a Survey: Motivations, Methods, and Applications". In: *ACM Computing Surveys* 51.2 (2018), pp. 1–39. DOI: 10.1145/3154524.

[12] E. Aghaei et al. "Network Topology Discovery: A Problem of Incomplete Data Improvement". In: *Proceedings of the IEEE Conference on Computer Communications Workshops*. 2019. DOI: 10.1109/INFCOMW.2019.8845166.

[13] Georgios Kavallieratos, Georgios Spathoulas, and Sokratis Katsikas. "Attack Path Analysis for Cyber Physical Systems". In: *Computer Security. ESORICS 2020 International Workshops*. Vol. 12513. LNCS. Springer, 2020, pp. 19–33. DOI: 10.1007/978-3-030-64330-0_2.

[14] Oscar Corcho et al. "A high-level ontology network for ICT infrastructures". In: *Proceedings of the 20th International Semantic Web Conference (ISWC 2021)*. Vol. 12922. LNCS. Springer, 2021, pp. 446–462. DOI: 10.1007/978-3-030-88361-4_26.

[15] Aidan Hogan et al. "Knowledge Graphs". In: *ACM Computing Surveys (CSUR)* 54.4 (2021), pp. 1–37. DOI: 10.1145/3447772.

[16] European Union. *Directive (EU) 2022/2555 of the European Parliament and of the Council of 14 December 2022 on measures for a high common level of cybersecurity across the Union (NIS2 Directive)*. Official Journal of the European Union, L 333/80. 2022. URL: https://eur-lex.europa.eu/eli/dir/2022/2555/oj.

[17] European Union. *Regulation (EU) 2022/2554 of the European Parliament and of the Council of 14 December 2022 on digital operational resilience for the financial sector (DORA)*. Official Journal of the European Union, L 333/1. 2022. URL: https://eur-lex.europa.eu/eli/reg/2022/2554/oj.

[18] William Stallings and Lawrie Brown. *Computer Security: Principles and Practice*. 4th ed. Pearson, 2022. ISBN: 978-0-13-751434-4.

[19] Tran Thi Huong Nguyen et al. "A Survey on Enterprise Network Security: Asset Behavioral Monitoring and Distributed Attack Detection". In: *IEEE Access* (2023). DOI: 10.1109/ACCESS.2023.3285727.

[20] Leslie F. Sikos. "Cybersecurity knowledge graphs". In: *Knowledge and Information Systems* 65 (2023), pp. 3511–3531. DOI: 10.1007/s10115-023-01860-3.

[21] Hao Wang et al. "Applications of large language models in networking". In: *arXiv preprint arXiv:2309.06342* (2023). URL: https://arxiv.org/abs/2309.06342.

[22] Zhenhao Ma et al. “LLMParser: An exploratory study on using large language models for log parsing”. In: *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE 2024)*. ACM, 2024. DOI: 10.1145/3597503.3639150.

[23] National Institute of Standards and Technology. *The NIST Cybersecurity Framework (CSF) 2.0*. NIST Cybersecurity White Paper (CSWP) 29. U.S. Department of Commerce, 2024. DOI: 10.6028/NIST.CSWP.29.

[24] Lionel Tailhardat, Raphaël Troncy, and Yoan Chabot. “NORIA-O: An ontology for anomaly detection and incident management in ICT systems”. In: *Proceedings of the 21st European Semantic Web Conference (ESWC 2024)*. Springer, 2024. DOI: 10.1007/978-3-031-60635-9_2.

[25] Ansible Community. *Ansible Community | Ansible documentation*. URL: https://docs.ansible.com/.

[26] Docker Inc. *Docker: Accelerated Container Application Development*. URL: https://www.docker.com/.

[27] Ontotext. *Ontotext GraphDB*. URL: https://www.ontotext.com/products/graphdb/.

[28] Ontotext. *System monitoring — GraphDB 11.3 Documentation*. URL: https://graphdb.ontotext.com/documentation/11.3/system-monitoring.html.

[29] SolarWinds Worldwide, LLC. *GNS3 | The software that empowers network professionals*. URL: https://www.gns3.com/.

[30] World Wide Web Consortium. *World Wide Web Consortium (W3C)*. W3C. URL: https://www.w3.org/.