Let AI Agents Translate Networks, Not Reason About Them

Typographical Network Modeling with Automated and Verifiable Axiomatization

Hongyu Hè
Princeton University

Maria Apostolaki
Princeton University

Abstract

A formal model enables verifying reachability, localizing an outage, or anticipating the blast radius of a change. Yet, virtually no production network has one, since writing a model by hand demands rare expertise and is hard to keep current as the network changes frequently. At its core, network modeling is a typographical exercise: it translates network artifacts (*e.g.*, configurations, topology, and routing state) into rules in formal logic. Translation of this kind is what large language models (LLMs) nowadays do well. Unlike free-form AI reasoning, such translation can be formally verified.

Once modeling is no longer the bottleneck, trusting AI to reason over large, complex networks no longer makes sense. Our position therefore cuts against the prevailing race to put autonomous AI agents in charge end-to-end. We instead confine AI to translation and rely on a solver for reliable long-horizon reasoning, building a reusable formal model of general network behavior that can then be specialized to specific tasks, *e.g.*, root-cause analysis (RCA). We build TypoNet that constructs and validates a symbolic model of an emulated production-scale WAN from the network's own artifacts. Our preliminary evaluation shows TypoNet helps in two ways. On its own, TypoNet answers operational questions (*e.g.*, reachability verification and change-impact analysis) faster, more cheaply, and more reliably than an LLM. As a tool for an AI agent, TypoNet boosts fault localization at lower cost. The result makes the case for AI that builds verifiable network models and relies on a solver for reliable long-horizon reasoning.

1 Introduction

Running a large network is an exercise in answering questions about how it behaves. An operator must know whether a service is still reachable, which links a customer's traffic crosses, what a planned configuration change will break, and, when an outage strikes, which device caused it. Answering them precisely requires a formal model of the network, a machine-checkable description of how its configurations, topology, and routing state combine to forward packets [4, 13, 26, 27]. Given such a formal model, a solver can verify reachability, localize a fault, or predict the blast radius of a change; without one, operators fall back on ad hoc scripts and tribal knowledge. Yet for almost every production network no such model exists, because building one by hand requires rare expertise in both formal methods and networking, demands prohibitive manual efforts, and is hard to keep current as the network changes frequently [8, 30].

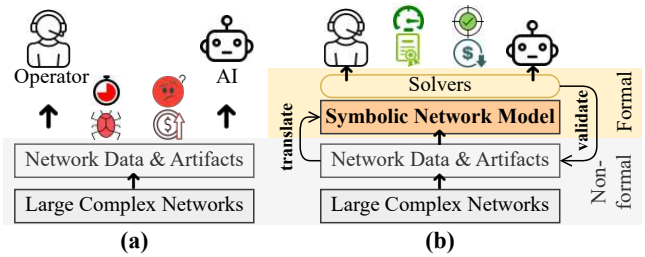

Figure 1: How large networks are operated today, and what reliable operation actually needs. (a) Status quo. Working directly from raw artifacts is slow and error-prone for operators and treacherous for AI agents: the data volume overflows the context window, cascading hallucinations derail reasoning, and reprocessing the same data each time is costly. (b) TypoNet. It *translates* the artifacts into rules in formal logic and *validates* each against the network's own evidence, yielding a symbolic model that operators and AI agents query by offloading formal reasoning to a solver.

The current trend in the community is to skip the formal model and put an AI agent in charge. The underlying hope is AI agents that watch the live network, reason about what is wrong, and act to repair it end to end [19, 38]. Large language models (LLMs) sit at the center of that ambition, and a fast-growing line of work asks them to configure, diagnose, and manage networks directly [21, 47, 50]. For critical network infrastructure, this approach is misplaced, because it trusts the LLM where it is weakest. LLMs hallucinate, and their errors compound over long chains of reasoning, so a single confident misstep derails an entire diagnosis [11, 12, 23, 52]. They also cannot hold a hyperscale network in view, since millions of devices and hundreds of regions do not fit in any context window. Accuracy degrades further as the relevant facts recede into a long prompt [31, 33]. Fault localization, the heart of troubleshooting, is exactly where cascading errors and missing context do the most damage. Fig. 1a depicts the regime we distrust, in which an AI agent works directly from the raw artifacts of a large network and its accuracy degrades as the network grows.

This paper makes a different bet: do not ask the LLM to reason about the network at all. Use it to build a formal model

that a solver can check, and let the solver do the complex, long-horizon reasoning. The LLM is confined to one job: translating the network's heterogeneous artifacts into logical rules. TypoNet then tests each rule against independent network evidence before trusting it. Two guarantees then stand apart. The solver is sound with respect to the symbolic model, so it correctly derives what the rules entail. When a rule is wrong, the refutation that exposes it names the specific axiom at fault, so an error localizes to a single rule the loop can then repair. The symbolic model's faithfulness to the real network is a separate matter, since it holds only as far as the evidence that tried and failed to refute it. Core reasoning moves from the LLM to the solver, and the symbolic model becomes the subject under scrutiny. *The LLM translates; the solver decides.*

The bet is principled, because a network is a manmade, typographical artifact. Its configurations, topology records, the logical rules describing its behavior, and the program a solver runs are all strings. Building a formal model is therefore a translation among those symbol systems. *A translation can be checked in a way that open-ended AI reasoning cannot.* LLMs are the strongest translators between symbol systems we have [16, 41]. The one risky step, writing the formal model, thus becomes one we can validate.

We realize the idea in TypoNet, which constructs and validates a symbolic model of a network from its existing artifacts. The construction loop is adversarial, inspired by counterexample-guided inductive synthesis (CEGIS) [43]. An LLM-driven Constructor proposes logical rules, each a formal statement about one facet of how the network behaves. A Detractor refutes them with counterexamples drawn from the network's own evidence, *e.g.,* a reachability measurement that contradicts a proposed rule. The loop repeats until the symbolic model withstands attack and satisfies the network's established invariants, the properties a correct network must uphold in every valid state. Each rule is *axiomatized* bottom-up, built on simpler facts and states the loop has already validated. Behavior therefore grows from ground facts into the multi-hop relations operators ask about. We call such a validated set of rules a *theory*: a conjunction of rules, closed under entailment [7, 20]. A solver reasons over the theory to answer questions about the network's behavior. TypoNet builds the symbolic model *compositionally*, in layers: a reusable foundation theory of general network behavior, then vendor- and deployment-specific refinements, and finally per-task specializations. Root-cause analysis (RCA) is one such specialization: we teach the foundation theory how faults manifest by injecting them and modeling the resulting cause-symptom links.

We prototype TypoNet on an emulated WAN of tens of autonomous systems (ASes), scaled to approximate a production deployment's thousands of core devices. The symbolic model checks the invariants operators rely on, *e.g.,* all-pairs reachability and policy compliance. Used as a tool by an AI agent, the symbolic model localizes injected faults with a verified solver query at near-zero token cost [9, 25]. We measure the resulting gains in token cost and fault-localization accuracy against LLM-only baselines and SOTA agentic RCA solutions. The same symbolic model can also serve reachability verification, change-impact analysis, and configuration-drift detection, so it is built once and reused across tasks [8, 51]. AI agents keep a role as users of formal models, and the reasoning we have to trust runs on the solver. Fig. 1b shows the paradigm we advocate, in which operators and AI agents alike offload formal reasoning to a solver that runs over automatically constructed symbolic models.

We argue that automatically constructing a formal model turns network modeling from a bespoke, expert-driven craft into an automated and verifiable process. We give preliminary evidence on three fronts. The construction loop converges into a foundation theory that matches held-out network behavior and transfers almost unchanged to a WAN deployment twice as large. Once built, the theory answers operational questions such as blast radius in milliseconds and at no token cost. As a tool for RCA, it sharpens fault localization for every frontier AI agent we test, and lifts a small, inexpensive model to competitive accuracy at a fraction of a frontier agent's cost. We close with the open questions that decide how far the idea reaches, from keeping the symbolic model synchronized under constant network change to certifying the network models an AI creates.

2 Background and Motivation

Modern network operations already run on a written record of the network. Every large operator maintains a source-of-truth (SoT) database that stores, as typed and linked records, each device, interface, link, address block, and routing session, together with the roles and policy intent behind them. Meta's Robotron [44] and Google's MALT [40] are two examples, and comparable systems run at other hyperscalers [36]. Provisioning, monitoring, and configuration generation all read from that record, which makes it the closest thing a network has to a single authoritative description. Three trends now make that record the natural starting point for an automatically built formal model.

SoT databases capture structure, but leave semantics implicit. A SoT database records what the network is made of, but leaves how it behaves unstated. For instance, it states that two routers share a routing session and that an interface carries an access list. It does not explicitly say whether a destination is reachable, which path a flow takes, which flows a given link carries, or what breaks when a device fails. Properties of this kind are entailed by the records yet never written in a form a machine can compute over, *e.g.,* reachability, waypointing, impact cones, blast radius, and the causal

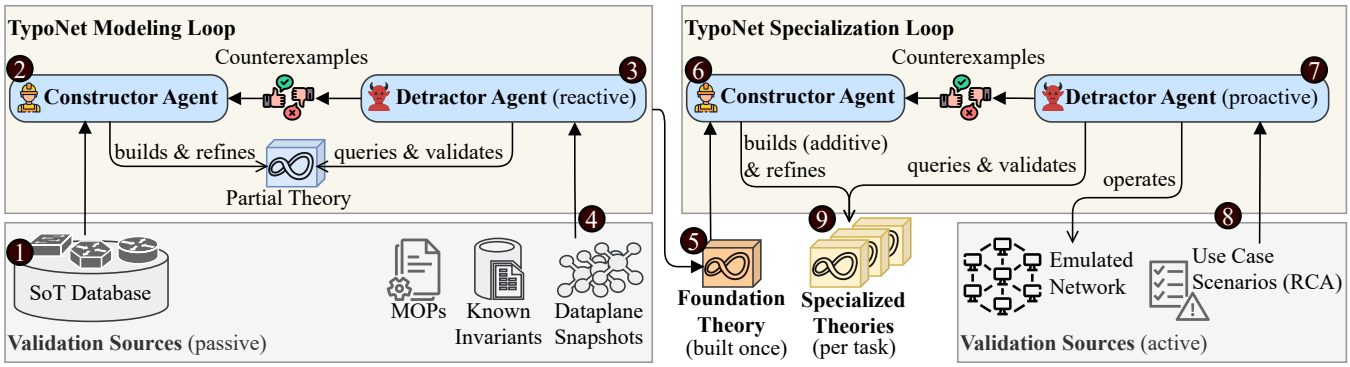

Figure 2: TypoNet's two agentic CEGIS loops turn a network's own records into a verifiable symbolic model. A *Modeling Loop* (left) builds a reusable Foundation Theory from passive evidence, and a *Specialization Loop* (right) adds thin per-task layers validated on an emulated network.

links between a fault and its symptoms. Blast radius is a representative example: before every deployment operators must estimate the reach of a change. A single edit to one core device can cascade many hops away, and a wrong estimate risks an outage [14, 15, 37]. Predicting that reach reliably needs long-horizon reasoning operators can trust, which is where a sound formal model, and not a manual or LLM guess, earns its place. The barrier is semantic: a SoT stores ground facts but lacks the rules, entailment, and recursion that behavior requires. *The behavior is latent in the records but never expressed in a computable form.* Surfacing it as a checkable formal model is the gap TypoNet bridges.

Formal methods stall on the hand-built model. Two decades of research can verify reachability, synthesize configurations, and localize faults once a formal model exists [2–5, 13]. The hard part is obtaining that model. Existing tools assume a hand-crafted translation from raw artifacts into formal semantics, which demands scarce dual expertise and decays as the network changes [8]. Hand-crafting also cannot keep up with scale. A single hyperscale network spans tens of millions of devices and hundreds of datacenters [18, 24, 32], and its records change continuously as configurations, failures, and repairs land. A hand-built model is therefore stale the moment it is finished, so operators report that verification tooling sees little real-world use [30]. Only an automated method can build a formal model at that scale and keep it synchronized [14, 28, 29, 48].

LLMs have become good enough at translation. The missing capability was a way to produce the formal model automatically, cheaply, and without having to trust its author. LLMs now convert among configurations, intents, and formal artifacts with enough fidelity to be useful. Because a translation can be checked and corrected, their mistakes need not be believed [16, 38, 41]. Mature solvers stand ready to consume millions of variables and reason over them soundly [6, 7, 10]. For the first time, the translator and the reasoner co-exist.

TypoNet differs from prior tools in where the semantics come from. Two other lines of work also turn a network into something a machine can reason about, and TypoNet parts from both in the origin of its semantics. Configuration-parsing verifiers such as Batfish derive behavior by parsing vendor configurations through hand-written parsers that encode each vendor's semantics [4, 13]. Those vendor parsers carry the whole translation, stay hand-coded, and are never checked against the network's own evidence. An error in a parser therefore becomes a silent error in every answer. Tool-using AI agents go the other way and let the LLM reason over the live network directly, calling tools to fetch state [19, 38]. The semantics then live inside the LLM's reasoning, no step is validated, and the least reliable component is trusted the most. TypoNet instead keeps the semantics in an explicit symbolic model, synthesizes it automatically, and validates every rule against independent network evidence before a solver reasons over it.

3 Automated & Verifiable Network Modeling

TypoNet constructs a formal model of a network automatically, and it makes the symbolic model trustworthy even though an LLM writes it. Fig. 2 shows how the two agentic loops reconcile these goals end to end, and five principles, stated below, make the design work.

Build the symbolic model from the network's own records. A production network already documents itself in a SoT database (§2), so the raw material for a symbolic model exists before TypoNet runs. TypoNet treats each such record as a ground fact ❶ and asks the LLM to do one thing only: compose those facts into the rules that explain how the network behaves ❷. Bottom-up *axiomatization* gives the symbolic model three layers: ground facts at the bottom ❶, rules that derive operational states from them, and rules that combine those states into multi-hop relations. Because each rule is defined

only over facts and states already validated beneath it, definitions never turn circular, and TypoNet validates the symbolic model one layer at a time. Grounding the symbolic model in existing records removes the manual labor, since no expert transcribes the network and the LLM only translates.

Refute a rule before trusting it. A rule is only as trustworthy as our ability to attack it. TypoNet pairs the LLM Constructor with an adversarial Detractor (❸) that refutes each proposed rule against what the network actually does. A rule the evidence refutes is discarded however plausible it reads, so the network's own evidence is the validation oracle that keeps the symbolic model tied to the real network. Every refutation returns a counterexample the Constructor must repair (§1), and the loop stops once a full validation pass raises no new counterexample. The Detractor draws on two kinds of evidence: *passive* sources (❹) already recorded about the network, and *active* sources (❽) it produces on demand from an emulated network. TypoNet's trust base is therefore small and explicit: the network's own evidence, an emulated network, and the solver. The LLM's output is never part of it, since a rule it proposes counts for nothing until the evidence corroborates it.

Ground the foundation in operators' invariants and procedures. The reactive Detractor builds the Foundation Theory from passive sources alone. Reproducing every recorded case is not enough, since a symbolic model can match them all and still break on the one that matters. TypoNet therefore also grades the symbolic model against the network's known invariants (§1), the laws every deployed network upholds. Such invariants include all-pairs reachability, loop freedom, and header-precise access-control compliance [24, 34]. A second source is the Method of Procedures (MOPs) operators run for planned changes, whose steps are virtually all validated in production [32, 35, 42]. Operators author the invariants and MOPs independently of the symbolic model, so satisfying them is external evidence that the symbolic model reflects the real network.

Compose the symbolic model from a reusable foundation and thin refinement layers. A network model should cover many vendors, deployment conditions, and downstream tasks, more than any single hand-built artifact can carry. TypoNet therefore takes a *compositional* approach. It builds a Foundation Theory (❺) of general network behavior once, then *composes* thin refinement layers onto it (❻) for a given deployment's policy and a given vendor's quirks. The symbolic model is thus an assembly of independently built and independently validated theories (❾). The composition works because of how the rules are written: each rule quantifies over facts and names no specific devices. The Foundation Theory therefore states behavior every IP network shares, *e.g.,* how forwarding follows the installed routes, without reference to any one topology. Specializing to a concrete network is then mostly a matter of supplying that network's facts, since TypoNet loads the new SoT under the unchanged rules. Only a genuinely new behavior, such as a vendor feature the foundation does not cover, needs a fresh layer. The same reusable core then serves many downstream applications, *e.g.,* reachability verification, change-impact analysis, and root-cause analysis.

Confine specialization to an emulated network. The proactive Detractor (❼) builds the specialization layers, and here it does more than react. It actively drives an emulated network through use-case scenarios (❽) and reads the resulting states, because the knowledge it needs does not exist until a scenario is run. Learning how a network behaves under a fault means causing that fault. TypoNet therefore confines every perturbing operation to an *emulated network* and lets production contribute read-only artifacts alone. For RCA, TypoNet injects a fault, observes which invariants break, and reverts it. It records the resulting fault-to-symptom relationship as a new specialization layer, which lets the symbolic model reason backward from an incident's symptoms to the smallest set of faults that explain them. The emulated network need not be a full copy of production: a high-fidelity digital twin or a smaller network that exercises the same behaviors suffices. Such emulation at this scale is common practice in production [17, 22, 30, 39, 48].

An emulated network alone does not replace the symbolic model: it runs only forward, so it shows what a fault does but cannot invert an observed symptom into its cause the way RCA requires [6, 14, 32]. The emulated network is thus the offline apparatus TypoNet learns from. The symbolic model, on the other hand, is the lightweight, synchronized artifact operators query online.

Operators query the finished symbolic model by entailment. A query q holds under *cautious* entailment, that is, when q is true in every stable model of the theory Th. Concretely, $\mathsf{Th} \models q$, which TypoNet decides on the theory's stratified fragment by refutation, checking that $\mathsf{Th} \wedge \neg q$ has no stable model. Each answer is a sound chain of rule applications: it starts from ground facts, derives operational states, and composes them into the multi-hop relations operators ask about. One query can therefore traverse many devices and hops no operator could follow by hand.

4 Preliminary Results

Implementation. We implement TypoNet in Answer Set Programming (ASP), executed by a proprietary backend in the spirit of Network-Optimized Datalog [34] for performance. ASP's support for default negation and abductive inference suits not only property checking but also the backward, cause-seeking reasoning RCA needs [1, 25, 46].

Theory construction measurement	Value
Foundation rules (LLM-authored / human-seeded)	128 (121 / 7)
Counterexamples raised by the Detractor	214
Refutation rounds per rule (avg.)	2.8
One-time construction cost	3 h, $18.6
Held-out agreement with ground truth	97.4%
without the Detractor (ablation)	41%
Planted-wrong rules caught (negative control)	48 / 50
Rules reused unchanged, 30-AS → 70-AS	118 / 128 (92%)

Table 1: Adversarial refutation is what makes the automatically built Foundation Theory trustworthy. Disabling the Detractor collapses held-out agreement 58%; once validated, 92% of the rules transfer unchanged to a 2× larger deployment.

Setup. We deliberately build the foundation theory and the specialized theory on different networks, so the experiment tests both the feasibility and the transferability of TypoNet's compositional modeling. We build the foundation theory on a 30-AS emulated WAN of 515 devices. We then transfer that theory to a more than 2× larger 70-AS deployment of 1,191 devices and specialize it there, a scale on par with the core of a production WAN [14, 28, 29, 48].

4.1 Constructing Valid Foundation Theory

The construction loop converges to a Foundation Theory at a one-time, offline cost, and it agrees with held-out behavior (Table 1). The Detractor is what ties the theory to evidence: disable it, and the Constructor still produces plausible-looking rules, but the theory admits vacuous rules that fire on no real state. As a negative control, we plant deliberately wrong rules by mutating a correct rule's predicate or direction, and the Detractor catches nearly all of them.

4.2 Answering Operational Questions

Once validated, the foundation theory answers operational questions with no LLM in the loop. It chains many facts across devices and hops, where a manual check or a free-form LLM loses track of a cascading consequence. A single query decides all-pairs reachability, for instance, and pinpoints the pair and hop at which it breaks. Blast radius is a representative question, because one change at a single device propagates through a long chain of consequences. As a real example, an operator plans to take a core interface out of service, modeled as $\neg \mathit{IsUp}(\texttt{spineB},\texttt{p17})$, where switch `swX` prefers the uplink `eth2` toward it and holds `eth1` as a backup for the video class, while the control class has none. TypoNet applies the change and unrolls the cascade one validated axiom at a time:

$$\neg \mathit{IsUp}(\texttt{spineB},\texttt{p17}) \wedge \mathit{ConnectsTo}(\texttt{swX},\texttt{eth2},\texttt{spineB},\texttt{p17}) \models \neg \mathit{IsUp}(\texttt{swX},\texttt{eth2}) \models \neg \mathit{CanForward}(\texttt{swX},\texttt{eth2},\texttt{c\_vid}),$$

$$\mathit{Preferred}(\texttt{swX},\texttt{c\_vid},\texttt{eth2}) \wedge \mathit{Backup}(\texttt{swX},\texttt{c\_vid},\texttt{eth1}) \wedge \mathit{CanForward}(\texttt{swX},\texttt{eth1},\texttt{c\_vid}) \models \mathit{IsRedirectedTo}(\texttt{swX},\texttt{c\_vid},\texttt{eth1}),$$

$$\mathit{IsRedirectedTo}(\texttt{swX},\texttt{c\_vid},\texttt{eth1}) \wedge \mathit{DropRate}(\texttt{swX},\texttt{eth1},\texttt{high}) \models \mathit{IsDegraded}(\texttt{p\_vid}),$$

$$\neg\exists i : \mathit{CanForward}(\texttt{swX},i,\texttt{c\_ctrl}) \models \neg \mathit{IsReachable}(\texttt{swX},\texttt{p\_ctrl}) \models \neg \mathit{IsReachable}(\texttt{torA},\texttt{p\_ctrl}).$$

The single change fans out into a *full impact cone*: the video prefix `p_vid` survives but degrades on the overloaded backup, while the control prefix `p_ctrl`, which had no backup, goes dark and drags an upstream top-of-rack down with it. Estimating that reach by hand or by an LLM's guess is the error-prone, long-horizon step that motivates TypoNet [14]. In practice, the same chained reasoning also drives alerting, where TypoNet elevates a log only when it entails a degraded prefix or lost reachability.

Per-query cost. Each query runs on the solver alone with no LLM in the loop, so its cost is a few milliseconds and no tokens at all. TypoNet answers four representative queries over the running example (healthy and post-change reachability, blast radius, and the video class's waypoint), each correct and within 5–34 ms. The blast-radius query alone chains 23 rules across 11 hops, whereas a single-shot LLM given the same ground facts clears only the one-hop reachability case.

4.3 Agentic RCA with a Specialized Theory

We specialize the foundation theory into a fault-to-symptom layer and give it to an AI agent as a tool for root-cause analysis. To evaluate it, we port the NIKA open benchmark [49] into our emulated WAN and adopt its scoring exactly as the SADE agent does [45]. Each AI agent inspects a live incident and names the faulty devices and the fault type.

Method. We split incidents into train and test by fault type and topology region, so no fault family or region seen during specialization reappears at test time. The test set holds 96 held-out incidents and 40 healthy controls that a trustworthy AI agent should leave alone. Every configuration runs under one shared prompt and tool schema, repeated 5 times, and we report the average values.

Configurations. We run three frontier LLMs, GPT-5.6 Sol, Opus 4.8, and the smaller Sonnet 4.6, each unaided (baseline) and each equipped with the TypoNet tool. We add the SADE skill library on the two frontier LLMs and the SADE agent itself, for nine configurations in all. Every configuration faces the same held-out incidents and the same healthy controls,

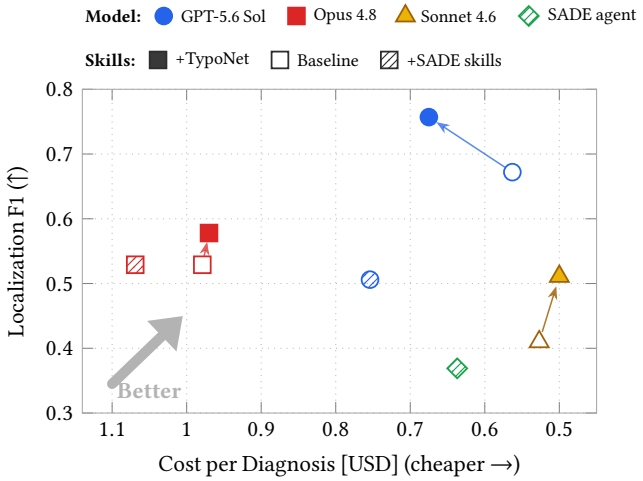

Figure 3: TypoNet's specialized theory improves fault localization for every LLM. GPT-5.6 Sol with TypoNet is the most accurate at cost below any Opus 4.8 configuration, and Sonnet 4.6 with TypoNet is the cheapest point above 0.5 localization F1 on the accuracy/cost frontier.

and Fig. 3 plots each one's localization F1 against its cost per run.

Insight 1: Even frontier AI agents localize faults more accurately and raise fewer false alarms with TypoNet's formal reasoning.

Adding the TypoNet tool to the same LLM helps both frontier AI agents: GPT-5.6 Sol's localization F1 climbs from 0.672 to 0.757, while Opus 4.8 gains most in naming. The edge widens on the hardest incidents, where the symptom surfaces far from its cause: there TypoNet lifts localization F1 from 0.59 to 0.82 and roughly doubles Opus 4.8's. A matched ablation on GPT-5.6 Sol isolates the source of the gain, holding the prompt, tool schema, and call budget fixed. The theory rendered as plain text reaches F1 0.710, a randomly shuffled theory falls to 0.55, and the validated theory behind a solver reaches 0.757. The gain therefore comes from the validated formal layer and the solver that uses it. The same formal grounding also curbs false alarms: the baseline frontier AI agents flag a fault on almost *every* healthy network scenario. Using TypoNet as a tool cuts that false-alarm rate by 2–3×. This observation serves as a preliminary warning for anyone using an AI agent for monitoring.

Insight 2: With TypoNet, a small, less powerful LLM reaches a competitive accuracy/cost point.

On its own Sonnet 4.6 localizes poorly (F1 0.41), but adding TypoNet raises it to 0.511 at just $0.50 per run. It undercuts the GPT-5.6 Sol baseline and roughly halves the per-run cost of the Opus 4.8 baseline at comparable localization. The uplift is largest on the hardest incidents, where Sonnet 4.6 with TypoNet reaches F1 0.83 and clears both *unaided* frontier baselines. Naming stays the small LLM's weak point.

5 Research Agenda

Keep the symbolic model in step with a network that changes frequently. A production WAN is reconfigured continuously, so a symbolic model correct when built drifts within hours. A stale symbolic model is worse than none, since it can certify a property that the live network no longer satisfies. Synchronization is therefore a first-class requirement [14, 44]. Since most changes touch a small part of the network, the open question is recompiling only the affected layers soundly and fast enough to track the network in near real time.

Turn vendor-specific behavior into theory automatically. Vendor-specific features cause over 30% of production failures [14, 48], yet a general Foundation Theory leaves them out. The proactive Detractor can learn them by driving an emulated network through scenarios that exercise a feature. Each scenario surfaces gaps between the symbolic model's prediction and the emulator's behavior, which the Detractor turns into new rules. Two questions stay open: which scenarios expose a vendor's key behaviors, and how a learned layer transfers across deployments on the same platform.

Confront the incompleteness of established knowledge. The established knowledge our Foundation Theory draws on is abundant but underspecified, written in prose that is often silent on corner cases. The theory is therefore inevitably incomplete, and specialization narrows the gap without ever closing it. A deeper obstacle is that observation does not imply causation: a mined fault-to-symptom relationship is only a correlation between an injected condition and its symptom. Isolating the true cause needs controlled experiments that vary one factor at a time, itself hard at network scale. Until we bound what the theory does not know, a verified answer holds only relative to the current theory.

Reach up the stack and fold in telemetry. TypoNet today mostly models Layer 2 and Layer 3 forwarding, leaving higher layers outside the theory. Stateful middleboxes such as NAT, firewalls, and load balancers are the first stress test, since a stateless forwarding theory cannot express their connection state. Datalog-based verification already reaches host-level reachability [34], so the same logical style can climb the stack. A second frontier is numeric telemetry: congestion and SLA violation would extend a representation like MALT [40] with numeric facts and rules. Both raise fidelity while enlarging the solver's search, so which properties earn their cost remains an open question.

References

[1] Atocha Aliseda. 2006. *Abductive reasoning*. Vol. 330. Springer.

[2] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. 2014. NetKAT: Semantic Foundations for Networks. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '14)*. ACM, 113–126. doi:10.1145/2535838.2535862

[3] Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang Harry Liu, Jitu Padhye, Boon Thau Loo, and Geoff Outhred. 2018. 007: Democratically finding the cause of packet drops. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 419–435.

[4] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2017. A General Approach to Network Configuration Verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. ACM, 155–168. doi:10.1145/3098822.3098834

[5] Ryan Beckett, Ratul Mahajan, Todd D. Millstein, Jitendra Padhye, and David Walker. 2016. Don't Mind the Gap: Bridging Network-wide Objectives and Device-level Configurations. In *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 328–341. doi:10.1145/2934872.2934909

[6] Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. 2024. CaDiCaL 2.0. In *Computer Aided Verification – 36th International Conference (CAV 2024), Part I*. Springer, 133–152. doi:10.1007/978-3-031-65627-9_7

[7] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh (Eds.). 2021. *Handbook of Satisfiability* (second ed.). Frontiers in Artificial Intelligence and Applications, Vol. 336. IOS Press. doi:10.3233/FAIA336

[8] Rüdiger Birkner, Dana Drachsler-Cohen, Laurent Vanbever, and Martin Vechev. 2020. Config2Spec: Mining Network Specifications from Network Configurations. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 969–984. https://www.usenix.org/conference/nsdi20/presentation/birkner

[9] Cristiano Calcagno, Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. 2009. Compositional Shape Analysis by means of Bi-Abduction. In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2009)*. ACM, 289–300. doi:10.1145/1480881.1480917

[10] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings 14*. Springer, 337–340.

[11] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang (Lorraine) Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. 2023. Faith and Fate: Limits of Transformers on Compositionality. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, Vol. 36. Curran Associates, Inc., 70293–70332.

[12] Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. 2024. Detecting hallucinations in large language models using semantic entropy. *Nature* 630, 8017 (2024), 625–630.

[13] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. 2015. A General Approach to Network Configuration Analysis. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 469–483. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/fogel

[14] Zhaoyu Gao, Anubhavnidhi Abhashkumar, Zhen Sun, Weirong Jiang, and Yi Wang. 2024. Crescent: Emulating Heterogeneous Production Network at Scale. In *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI '24)*. USENIX Association, 1045–1062. https://www.usenix.org/conference/nsdi24/presentation/gao-zhaoyu

[15] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. 2011. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. In *Proceedings of the ACM SIGCOMM 2011 Conference*. ACM, 350–361. doi:10.1145/2018436.2018477

[16] Fengchen Gong, Divya Raghunathan, Aarti Gupta, and Maria Apostolaki. 2023. Towards Integrating Formal Methods into ML-Based Systems for Networking. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*. 48–55.

[17] Paulo Gouveia, João Neves, Carlos Segarra, Luca Liechti, Shady Issa, Valerio Schiavoni, and Miguel Matos. 2020. Kollaps: Decentralized and Dynamic Topology Emulation. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20)*. ACM, 23:1–23:16. doi:10.1145/3342195.3387540

[18] Ramesh Govindan, Ina Minei, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. 2016. Evolve or Die: High-Availability Design Principles Drawn from Google's Network Infrastructure. In *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 58–72. doi:10.1145/2934872.2934891

[19] Pouya Hamadanian, Behnaz Arzani, Sadjad Fouladi, Siva Kesava Reddy Kakarla, Rodrigo Fonseca, Denizcan Billor, Ahmad Cheema, Edet Nkposong, and Ranveer Chandra. 2023. A Holistic View of AI-driven Network Incident Management. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*. 180–188.

[20] Hongyu Hè, Minhao Jin, and Maria Apostolaki. 2026. Making Logic a First-Class Citizen in Generative ML for Networking. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*.

[21] Zhiyuan He, Aashish Gottipati, Lili Qiu, Xufang Luo, Kenuo Xu, Yuqing Yang, and Francis Y Yan. 2024. Designing Network Algorithms via Large Language Models. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks*. 205–212.

[22] Thomas Holterbach, Tobias Bühler, Tino Rellstab, and Laurent Vanbever. 2020. An Open Platform to Teach How the Internet Practically Works. *ACM SIGCOMM Computer Communication Review* 50, 2 (2020), 45–52. doi:10.1145/3402413.3402420

[23] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems* 43, 2 (2025), 1–55.

[24] Karthick Jayaraman, Nikolaj Bjørner, Jitu Padhye, Amar Agrawal, Ashish Bhargava, Paul-Andre C. Bissonnette, Shane Foster, Andrew Helwer, Mark Kasten, Ivan Lee, Anup Namdhari, Haseeb Niaz, Aniruddha Parkhi, Hanukumar Pinnamraju, Adrian Power, Neha Milind Raje, and Parag Sharma. 2019. Validating Datacenters at Scale. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*. ACM, 200–213. doi:10.1145/3341302.3342094

[25] John R Josephson and Susan G Josephson. 1996. *Abductive inference: Computation, philosophy, technology*. Cambridge University Press.

[26] Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Header Space Analysis: Static Checking for Networks. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. USENIX Association, San Jose, CA, 113–126. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/kazemian

[27] Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. 2012. VeriFlow: Verifying Network-Wide Invariants in Real Time. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks (HotSDN '12)*. ACM, 49–54. doi:10.1145/2342441.2342452

[28] Alexander Krentsel, Rishabh Iyer, Isaac Keslassy, Bharath Modhipalli, Sylvia Ratnasamy, Anees Shaikh, and Rob Shakir. 2026. CrossCheck: Input Validation for WAN Control Systems. In *Proceedings of the 23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI '26)*. USENIX Association, 647–667. https://www.usenix.org/conference/nsdi26/presentation/krentsel

[29] Alexander Krentsel, Rishabh Iyer, Isaac Keslassy, Sylvia Ratnasamy, Anees Shaikh, and Rob Shakir. 2024. The Case for Validating Inputs in Software-Defined WANs. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks*. 246–254.

[30] Alexander Krentsel, Oliver Ye, Anthony Tafoya, Xuqian Ma, Sylvia Ratnasamy, and Anees Shaikh. 2025. Towards Accessible Model-Free Verification. In *Proceedings of the 24th ACM Workshop on Hot Topics in Networks (HotNets '25)*. ACM. doi:10.1145/3772356.3772380

[31] Mosh Levy, Alon Jacoby, and Yoav Goldberg. 2024. Same Task, More Tokens: the Impact of Input Length on the Reasoning Performance of Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Bangkok, Thailand, 15339–15353. doi:10.18653/v1/2024.acl-long.818

[32] Hongqiang Harry Liu, Yibo Zhu, Jitu Padhye, Jiaxin Cao, Sri Tallapragada, Nuno P. Lopes, Andrey Rybalchenko, Guohan Lu, and Lihua Yuan. 2017. CrystalNet: Faithfully Emulating Large Production Networks. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. ACM, 599–613. doi:10.1145/3132747.3132759

[33] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173. doi:10.1162/tacl_a_00638

[34] Nuno P. Lopes, Nikolaj Bjørner, Patrice Godefroid, Karthick Jayaraman, and George Varghese. 2015. Checking Beliefs in Dynamic Networks. In *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15)*. USENIX Association, 499–512. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/lopes

[35] Nuno P. Lopes and Andrey Rybalchenko. 2019. Fast BGP Simulation of Large Datacenters. In *Verification, Model Checking, and Abstract Interpretation (VMCAI 2019)*. Springer, 386–408. doi:10.1007/978-3-030-11245-5_18

[36] Biao Lyu, Enge Song, Tian Pan, Jianyuan Lu, Shize Zhang, Xiaoqing Sun, Lei Gao, Chenxiao Wang, Han Xiao, Yong Pan, et al. 2024. POSEIDON: A Consolidated Virtual Network Controller that Manages Millions of Tenants via Config Tree. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association. https://www.usenix.org/conference/nsdi24/presentation/lyu

[37] Ratul Mahajan, David Wetherall, and Tom Anderson. 2002. Understanding BGP Misconfiguration. In *Proceedings of the 2002 ACM SIGCOMM Conference*. ACM, 3–16. doi:10.1145/633025.633027

[38] Sathiya Kumaran Mani, Yajie Zhou, Kevin Hsieh, Santiago Segarra, Ranveer Chandra, Srikanth Kandula, Trevor Eberl, Eliran Azulai, and Ido Frizler. 2023. Enhancing Network Management Using Code Generated by Large Language Models. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks (HotNets '23)*. Cambridge, MA, USA. doi:10.1145/3626111.3628183

[39] Congcong Miao, Yuejie Wang, Jianming Wang, Xuefeng Ji, Guozhi Shan, Sirui Li, Pan Fang, Yanke Zhang, Jialin Li, Xianneng Zou, and Guyue Liu. 2026. MirrorNet: High-fidelity and Scalable Network Emulation for Software-defined WAN. In *Proceedings of the 23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI '26)*. USENIX Association, 175–190. https://www.usenix.org/conference/nsdi26/presentation/miao

[40] Jeffrey C. Mogul, Drago Goricanec, Martin Pool, Anees Shaikh, Douglas Turk, Bikash Koley, and Xiaoxue Zhao. 2020. Experiences with Modeling Network Topologies at Multiple Levels of Abstraction. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 403–418. https://www.usenix.org/conference/nsdi20/presentation/mogul

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

[42] Gordon D. Plotkin, Nikolaj Bjørner, Nuno P. Lopes, Andrey Rybalchenko, and George Varghese. 2016. Scaling Network Verification using Symmetry and Surgery. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '16)*. ACM, 69–83. doi:10.1145/2837614.2837657

[43] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodík, Sanjit A. Seshia, and Vijay A. Saraswat. 2006. Combinatorial Sketching for Finite Programs. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XII)*. ACM, 404–415. doi:10.1145/1168857.1168907

[44] Yu-Wei Eric Sung, Xiaozheng Tie, Starsky H. Y. Wong, and Hongyi Zeng. 2016. Robotron: Top-down Network Management at Facebook Scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 426–439. doi:10.1145/2934872.2934874

[45] Kuan-Hao Tseng, Niruth Bogahawatta, Yasod Ginige, Kosta Dekic, Arunan Sivanathan, and Suranga Seneviratne. 2026. SADE: Symptom-Aware Diagnostic Escalation for LLM-Based Network Troubleshooting. arXiv:2605.04530 [cs.NI] https://arxiv.org/abs/2605.04530

[46] Douglas Walton. 2014. *Abductive reasoning*. University of Alabama Press.

[47] Changjie Wang, Mariano Scazzariello, Alireza Farshin, Simone Ferlin, Dejan Kostić, and Marco Chiesa. 2024. Netconfeval: Can llms facilitate network configuration? *Proceedings of the ACM on Networking* 2, CoNEXT2, 1–25.

[48] Dan Wang, Peng Zhang, Wenbing Sun, Wenkai Li, Xing Feng, Hao Li, Jiawei Chen, Weirong Jiang, and Yongping Tang. 2025. S2: A Distributed Configuration Verifier for Hyper-Scale Networks. In *Proceedings of the ACM SIGCOMM 2025 Conference (SIGCOMM '25)*. ACM, 796–808. doi:10.1145/3718958.3750516

[49] Zhihao Wang, Alessandro Cornacchia, Alessio Sacco, Franco Galante, Marco Canini, and Dingde Jiang. 2025. A Network Arena for Benchmarking AI Agents on Network Troubleshooting. arXiv:2512.16381 [cs.NI] https://arxiv.org/abs/2512.16381

[50] Duo Wu, Xianda Wang, Yaqi Qiao, Zhi Wang, Junchen Jiang, Shuguang Cui, and Fangxin Wang. 2024. NetLLM: Adapting Large Language Models for Networking. In *Proceedings of the ACM SIGCOMM 2024 Conference*. Sydney, NSW, Australia, 661–678. doi:10.1145/3651890.3672268

[51] Xieyang Xu, Yifei Yuan, Zachary Kincaid, Arvind Krishnamurthy, Ratul Mahajan, David Walker, and Ennan Zhai. 2024. Relational Network Verification. In *Proceedings of the ACM SIGCOMM 2024 Conference*. ACM, 213–227. doi:10.1145/3651890.3672238

[52] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. 2025. Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models. *Computational Linguistics* 51, 4 (2025), 1373–1418. doi:10.1162/coli.a.16