Invariant Discovery for Networked Systems

Hongyu Hè*
Princeton University

Alexander Krentsel*
UC Berkeley / Google

Sylvia Ratnasamy
UC Berkeley / Google

Maria Apostolaki
Princeton University

Abstract

Invariants, the relations expected to hold among measured signals of a network, underpin applications from verification to traffic generation, telemetry imputation, and input validation, yet writing them by hand demands rare expertise in both formal logic and networking. Automatic miners can help but fall short on two fronts: they still require the hardest input (the grammar of admissible invariants) and they learn only exact, "hard" rules, struggling with real-world approximation caused by inherent noise in data. LLMs are tools that can provide semantic reasoning over data, but are non-deterministic and opaque in their learning. Our key idea is to partition the invariant search problem into an AI-driven grammar "discovery" problem, followed by a statistics-driven "search" problem within the learned grammar. Taken together, this allows non-deterministic, hallucination-prone AI to help produce auditable invariants with formal guarantees. We design and implement such a system, Autogram, and evaluate it on both public and production telemetry data, recovering expert-derived invariants with high coverage and low false positives. We close with discussion on open problems on the path toward fully open-ended discovery.

1 Introduction

Network invariants, the relations that hold across a network's data, are the quiet foundation of many networked-system tasks: verification checks them [2, 3, 12, 33, 41] to find misconfigurations or faults, traffic generators follow them [6, 18, 21, 22] to ensure fidelity, telemetry imputation relies on them to fill gaps and avoid hallucinations [14, 15, 17], and data-pipeline validators use them to catch corrupt inputs before they spread [25, 26, 40]. Invariants capture expected truths about network data, such as "a node's total output equals the sum of its outgoing links," which makes otherwise opaque data machine-checkable; these checks have been applied to catch corrupted controller inputs behind many WAN outages [25], correct drifting sensors [38], or flag industrial process anomalies before equipment is damaged [11].

However, deriving these invariants for a target system remains incredibly difficult. It demands rare expertise in both formal logic and the target network, and is labor-intensive and error-prone. Even with expert knowledge, usable invariants must account for the noise inherent in measurements, which requires non-trivial manual data analysis [25]: real-world data is messy, reflecting collection error, systematic bias, and even implementation bugs, so intuition-derived invariants alone are not enough. In production, the invariants that matter are usually *soft*: they hold within a tolerance and on most observations, not exactly and always.

Automatic invariant mining methods promise to remove this burden by learning invariants directly from data [16, 18, 42, 43]; however, they fall short on two critical fronts. First, they still require a human expert to author the *grammar* of admissible invariants that constrains the search, which requires considerable expertise in formal methods and networking, and is tedious work. Second, even with an expert-curated grammar that can, in theory, express relations, existing invariant mining methods struggle to find them when masked by real-world data noise (as we discuss in §2).

Grammar writing is crucial to invariant discovery, yet surprisingly difficult, because the grammar dictates which relations are even expressible over a collection of observations. Consider an invariant I_1 that captures flow conservation at a node X: the total output equals the traffic on the links leaving it, *i.e.,* $\text{out}_X \approx \text{egress}(X \to Y) + \text{egress}(X \to Z)$. Merely to *express* this candidate, the grammar must encode (1) *types*, so a packet count is never summed with a byte count; (2) *locality*, grouping links into the family leaving X, which the raw counters do not encode; (3) an n-ary *sum* whose arity changes with the node's degree; and (4) *softness*, an approximate equality with a tolerance, since real counters never agree exactly. The raw measurements reveal none of these, so hand-authoring the grammar demands deep domain expertise and considerable effort.

The challenge of defining grammar has been that it is fundamentally a *semantic judgment*; that is, a good grammar follows from what the variables *mean*, and their meaning is carried in the knowledge the operator has. Thus in many domains [9, 18, 19], the hypothesis space of admissible invariants has always been drawn by a human who fixes the boundary in advance, after which an algorithm searches within it. We observe that LLMs can now provide the semantic judgment previously required of humans to aid in invariant discovery: LLMs have general knowledge of the

*Equal contribution.

world and reasoning capabilities, and unstructured metadata an operator already keeps (such as variable names) can expose system-specific knowledge. However, this raises an immediate objection: an LLM is non-deterministic and hallucinates, so how can invariants it helps produce ever be reproducible, auditable, or carry guarantees?

Our key idea is to incorporate LLMs with a strict division of labor that allows the *grammar* to be initialized and iteratively refined by the LLM, while maintaining invariant identification exclusively via statistical methods. In our design, the LLM proposes only the grammar, never an invariant, and everything downstream is deterministic and provable: the pipeline is *exhaustive*, enumerating the bounded hypothesis space identically on every run; *sound*, reporting only invariants that a solver judges non-trivial and that clear a statistical confidence bar on the data; and *monotone complete* (§3), guaranteed to find any unknown invariant that holds at least as often, on at least as much data, as one the operator already trusts. The shift is not that a machine does the searching, which it always did, but that (1) the hypothesis space itself is now proposed by an AI, and (2) the space can grow and change at runtime, while the data, not the LLM, decides which candidates actually hold.

We instantiate this design in AUTOGRAM, whose LLM agent induces a bounded, typed grammar from variable names and the richer metadata operators already keep, such as topology, role, and vendor annotations. A deterministic engine then enumerates that grammar, a solver screens each candidate for logical triviality, contradiction, and redundancy, and a statistical test decides soft acceptance. None of these steps is straightforward: AUTOGRAM must control false discoveries with no ground truth to check against. It reads each invariant's tolerance from that invariant's own residuals, and calibrates only the generic acceptance bar on synthetic proxies, small datasets we generate with known planted relations and injected noise, never on the invariants it hopes to recover, so discovery stays unbiased by prior knowledge.

Running AUTOGRAM on public network telemetry and a large production WAN, we recover *every* invariant the operator already reported and uncover useful relations no one had recorded, as validated by the network operators (§4). AUTOGRAM is a promising step toward open-ended discovery, growing its own hypothesis space where prior approaches stay confined to a fixed, hand-drawn one. Two boundaries remain and frame our agenda (§5): AUTOGRAM cannot yet compose discovered invariants into relational structures none of them expresses in isolation, and its expressiveness is capped by the *decidability* of existing solvers, leaving the rich space of non-linear structures largely unexplored.

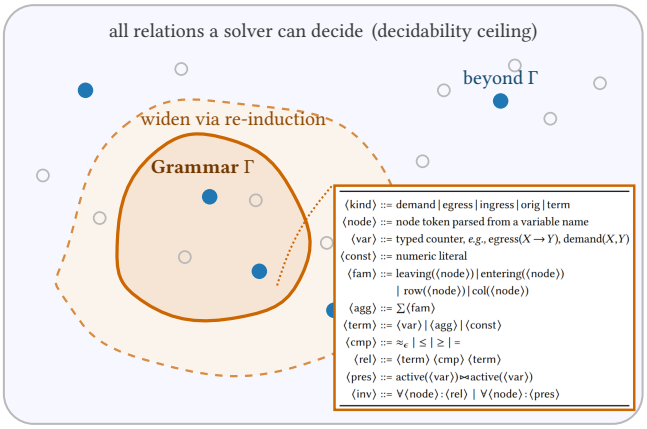

Figure 1: The grammar defines a *space*. An LLM proposes a bounded region Γ (a grammar; the inset shows a concrete one) inside the universe of all relations a solver can decide, and AUTOGRAM searches only *within* Γ, so a grammar *confines* the hypothesis space rather than enlarging it. The data decides which points hold (• invariants, ∘ candidates), and when known invariants fall outside Γ, re-induction widens it (dashed) but never past the decidability ceiling.

2 Background and Motivation

What a soft invariant is. A *soft network invariant* is a typed relation over named variables that holds on most snapshots of a network's data within a tolerance, so it carries a hold-rate between zero and one. The relations range from numeric conservation identities, such as a node's output equaling the sum of its egress links, through magnitude bounds, such as a queue length never exceeding its buffer, to logical presence pairings, such as a route advertised only when its next-hop is up. A soft invariant is therefore a *runtime check*, executable and falsifiable on any observation and returning a verdict, a residual, and a calibrated hold-rate, exactly what a monitor, validator, or anomaly detector consumes. Hard invariants are only the special case with a hold-rate of one at zero tolerance, so soft invariants extend the classical notion. Each still reduces to a symbolic relation an operator can read and a machine can evaluate on live data.

Why soft invariants are worth discovering. Soft invariants are the checkable contracts that keep large production systems honest, and the most valuable ones are those nobody has written down. In a wide-area backbone, an SDN controller engineers traffic from an aggregated demand matrix, and checking that input against the routers' local counters catches the incorrect inputs that are a leading cause of major outages [25]. In a multi-tenant telemetry pipeline, the conservation relation input = output + backlog + loss separates real, persistent byte loss from the transient dips that bursty

machine-learning workloads produce, which a fixed threshold cannot. In IoT sensing such as data-driven agriculture, invariants relating correlated measurements validate and correct a drifting sensor before it misleads a downstream model [38], and in an industrial control system the physical invariants of normal operation flag an anomaly or attack before it causes physical damage [11]. Every one of these relations is *soft* for two distinct reasons: zero-mean measurement noise scatters residuals symmetrically about zero (aleatoric uncertainty [8, 20, 23]), while systematic bias, such as interface counters that fold in header bytes the compared volumes omit, offsets a conservation identity by a few percent at every node. The two are useful in different ways, since a near-exact invariant is a tight correctness check while a systematic offset is a diagnostic that localizes an accounting or platform quirk, so a miner that accepts only exact rules misses exactly the invariants that matter.

The grammar is the real bottleneck. The scarce input is not the data but the *grammar*, the set of production rules over typed variables that fixes which relations can even be written down (Fig. 1). A usable grammar must identify what each variable measures, define the *locality families* that group related variables such as the links leaving a node, permit n-ary aggregation over such a family, and specify which typed terms may be compared under which operator and tolerance. Each is a semantic decision about what the variable names *mean*, and it must be remade for every new set of counters and every new network, which is exactly the expert labor that does not scale. The barrier is semantic: the raw numbers barely say that two egress counters form the family "links leaving X" whose sum should be compared to X's output.

Where prior methods fall short. No existing line discovers our target: a typed, soft invariant induced without a hand-authored grammar and certified despite a non-deterministic LLM in the loop. Network verification only decides whether a *given* property holds [3, 12, 24, 41, 45], so it presupposes the very artifact we produce. The methods that do discover relations all fix the hypothesis space by hand: functional-dependency and constraint miners search a predetermined template [4, 10, 19, 32], likely-invariant detectors such as Daikon test a fixed pattern library [9], and the closest network miner hand-authors a grammar per counter set and accepts only exact rules, unable to express a locality-grouped n-ary sum or a soft hold-rate [18]. Symbolic regression fits a single-target function, the wrong shape for an implicit identity over a whole family [5]; association-rule mining has soft acceptance but quantifies over discrete items, not typed numeric aggregation [1, 37]; and Markov logic learns rule weights but over predicate types rather than variable-name semantics, yielding no per-candidate tolerance [35]. Protocol-invariant inference is automated but targets boolean predicates over discrete state checked against a crisp oracle [42, 43]. Closest in spirit, LLM-driven evolutionary program search (FunSearch, AlphaEvolve, OpenEvolve) has an LLM edit candidate *programs* that an evolutionary loop scores against a hand-written objective [27, 29, 36], but its artifact is opaque code whose properties are in general undecidable, it presupposes the trusted objective we lack, and it calls the model once per candidate, whereas Autogram returns a readable relation a solver can decide and lays out the whole candidate space in a handful of calls. The common thread is that the hypothesis space is fixed by hand, and any guarantee evaporates the moment an LLM emits the invariant itself.

Why not just learn a model. If invariants are soft and their grammar is hard to find, why not train a model to predict the data directly? Because the two yield fundamentally different objects. An invariant paired with a hold-rate is an *auditable* contract an operator can read and any downstream task can reuse, whereas a black-box model's output is neither. One invariant serves verification, validation, and generation alike, while a model must be trained for each. Softness only sharpens the contrast: a black box *absorbs* measurement noise and hides the structure beneath, whereas a discovered invariant *surfaces* it and turns a systematic offset into a diagnostic rather than an error to smooth over. Discovery is hard with no ground truth to score against and spurious relations easy to accept, so Autogram must control false discovery while isolating the LLM's non-determinism and keeping its role minimal, so that the guarantees survive (§3).

3 Network Invariant Discovery

Autogram turns our key idea introduced in §1 into a working system by cutting the problem along a single seam: the one judgment that needs world knowledge, versus everything that is deterministic computation. Fig. 2 shows the design end to end, and numbered markers below point to its stages.

Problem formulation. The input is a dataset of network counters and measurements, treated as named *typed variables* observed over many snapshots, optionally paired with a set of known invariants the operator (the team that runs the network) already trusts. The output is a set of soft invariants over those variables, each augmented with a hold-rate and a tolerance. Invariant discovery after the grammar is fixed is more than a prior mining method behind an LLM. Specifically, soft acceptance with a per-candidate tolerance, calibration on synthetic proxies, and grammar re-induction together change how the logical formulas are learned, unlike hard-invariant miners that keep only rules holding exactly on every snapshot [16, 18, 42, 43].

Only one step needs world knowledge. The single judgment that needs world knowledge is which relations are

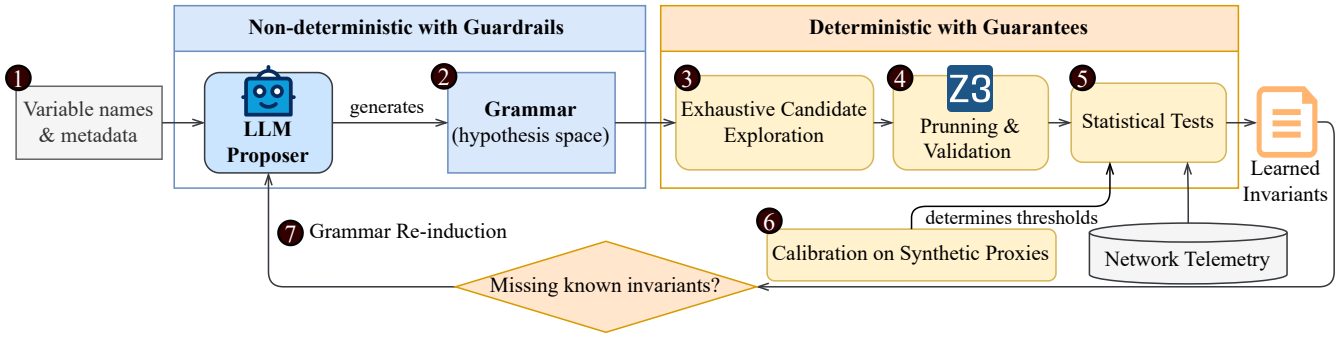

Figure 2: AUTOGRAM splits automatic invariant learning into a non-deterministic component, an LLM that proposes a *grammar* (a bounded space of candidate invariants) from variable names and metadata alone (❶, ❷), and a deterministic, provable component driven by formal derivation and statistics; the LLM never asserts that any relation holds. Downstream, AUTOGRAM enumerates the grammar (❸), screens candidates with a solver (❹), and accepts soft invariants by a data-calibrated hold-rate (❺), with generic thresholds set on synthetic proxies (❻), never on the invariants AUTOGRAM aims to recover. If coverage on a held-out split of the known invariants stalls, AUTOGRAM re-induces a more expressive grammar (❼); the data reaches only the evaluator, never the LLM.

worth considering, and that judgment is exactly the grammar: which quantities are comparable, how they group by locality, and which aggregations are allowed (❷). The LLM reads the variable names ❶ and returns a bounded, typed grammar but never *asserts* that any relation holds, so a hallucination can only mis-shape the space of candidates, never inject an accepted invariant that the deterministic search then decides.

Metadata is a rich, shareable signal. A grammar describes what the variables *are*, so the metadata an operator already keeps, names together with topology, role, and vendor annotations, is a rich and readily available signal for proposing it ❶. The same structure is latent in the raw measurements too, only harder to uncover (§5), so metadata is the cheaper starting point rather than the only source. Depending on metadata brings two practical benefits. Because the LLM sees only metadata and never the measurements, sensitive telemetry never leaves for the model, sidestepping the privacy concerns of feeding raw data to an LLM. And because a grammar is a property of names rather than values, it is portable across deployments with the same schema and is re-scored offline under many tuning settings without another LLM call ❷, keeping the untrusted, expensive component away from the large input.

How the grammar is induced, and re-induced. AUTOGRAM invokes the LLM as a bounded call for grammar induction (not an agentic loop). The prompt carries the variable names and the metadata an operator already keeps, such as topology, role, and vendor, and asks for a typed grammar in a fixed schema ❷. AUTOGRAM parses and type-checks the response before any measurement is touched, so a malformed or ill-typed proposal is rejected rather than run. When calibration recall stalls, AUTOGRAM re-invokes the LLM to re-induce a *strictly* more expressive grammar ❼ with more aggregations or higher-degree terms, and each re-induction is again only a grammar proposal, not a claim that a relation holds.

An invariant's tolerance is written in its residuals. Real invariants are soft, each in its own way (§2), so no single hand-picked tolerance serves them all; AUTOGRAM reads it from the data instead. A true invariant leaves residuals that cluster near zero, and the edge of that cluster fixes a per-candidate tolerance, a *capped conformal band* over the residuals [28, 30, 39] with coverage honest on a held-out split ❺; it fixes how soft a genuine invariant is, not whether the relation is genuine. A spurious relation's diffuse residuals would need a wide band, but the band can only tighten below a global ceiling, never widen past it. The ceiling is a generic default whose exact value does not matter, so a spurious relation cannot widen its tolerance to pass and must clear the same acceptance bar as everything else. Softness becomes something AUTOGRAM *observes*, not something the operator must specify in advance.

Logic and statistics do what only each can. Discovering soft invariants needs two kinds of reasoning that neither a classical miner nor a statistical test supplies alone. *Exact logic* discards candidates that are trivially true, self-contradictory, or logically entailed by a single kept invariant, and AUTOGRAM gets it from a solver [7] that decides validity over the enumerated candidates ❹; set-level redundancy, entailment by several invariants together, is a separate empirical matter the solver does not decide. *Calibrated statistics* decide when a surviving candidate holds *often enough* to be real,

which AUTOGRAM gets from a hold-rate with a conservative confidence interval ❺. The solver keeps the output free of logical junk while the confidence bar guards against accepting noise as structure, the false-discovery risk that no ground truth would otherwise leave unchecked; separately, neither suffices.

Calibrate the generic knobs without peeking at the answers. The per-candidate tolerance is read from data, but generic knobs such as the confidence level still have to be set, and tuning them against the invariants we hope to recover would be cheating. AUTOGRAM instead calibrates the knobs on synthetic proxies, small generated datasets with planted relations and injected noise ❻. The proxies are deliberately generic: each plants only a random subset of relation families, and the suite includes null datasets with no planted relation. The knobs therefore control false discovery in general and never fit the operator's specific invariants. The known invariants enter only in two disciplined ways, steering the search toward shapes worth proposing and measuring recovery on a *held-out* split the thresholds never saw.

AI-driven discovery with three guarantees. Since nondeterminism is confined to grammar proposal and the generic knobs are calibrated on proxies, three properties hold for whatever grammar is in force; we sketch their intuition and defer proofs to future work. Determinism and exhaustiveness: AUTOGRAM enumerates the entire bounded grammar identically on every run ❸ and keeps a representative of every accepted candidate up to logical equivalence, so it neither misses an expressible invariant nor silently drops an accepted one, and any result regenerates exactly for audit. Soundness: every returned invariant is non-trivial by the solver, supported by data, and clears the conservative confidence bar, so nothing is accepted on a point estimate alone. Monotone completeness: any unknown invariant that holds at least as often, on at least as much data, as a known one is provably accepted, since the confidence bound is monotone in both. As a result, a recovered known invariant *vouches* for every candidate that dominates it.

4 Preliminary Results

An SDN controller can only reject a malformed input if it knows what a well-formed input is, yet the relations that define validity are exactly the invariants no operator has written down. We use AUTOGRAM to discover those relations from a WAN's telemetry, and read the results below not as a scorecard but as probes, each testing whether the reframing survives real, noisy telemetry and pointing to a question a broader agenda must answer.

Setup. We treat every counter and measurement as a typed variable and ask each system to discover the relations they satisfy. We evaluate on two public WAN traffic-matrix datasets, Abilene and GEANT, packaged as CrossCheck-style snapshots [25], and on a production backbone we call "WAN prod." Each snapshot pairs an aggregated *demand matrix*, the controller's traffic-engineering input, with per-router *interface counters* measured locally in the dataplane, and every dataset carries the noise and systematic bias of real telemetry. Our ground truth is an expert-identified catalog of invariant families. Only two of them, two-end agreement and flow conservation, are *known* invariants that CrossCheck already reports; AUTOGRAM discovers the rest on its own, from non-negativity and zero self-demand to topology presence and demand conservation. Our baseline is NetNomos [18], a state-of-the-art miner we hand an expert-authored grammar that took over an hour of expert effort to write. AUTOGRAM removes that effort entirely: it receives only the counter *names* and induces the grammar once per dataset.

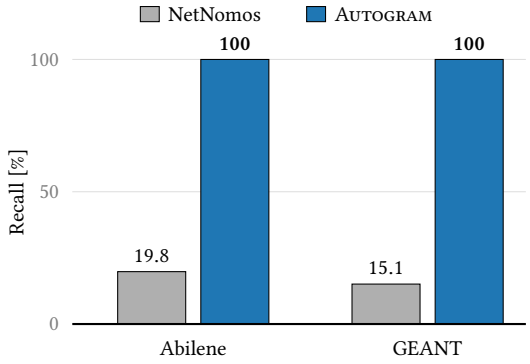

Figure 3: AUTOGRAM recovers every invariant on both networks, whereas NetNomos recovers barely a fifth because it keeps only rules that hold exactly.

Insight 1: The grammar is already latent in the variable names. Reading the counter names alone, AUTOGRAM recovers the full invariant catalog on both public networks, while NetNomos recovers under a fifth (Fig. 3), only the non-negativity and zero-self-demand families CrossCheck never even states. It misses every canonical CrossCheck invariant, since two-end agreement holds exactly on too few snapshots and flow conservation it cannot express at all. The hard part was never searching for invariants but writing the language that defines them, which an LLM can now do. If names alone go this far, handing the model the richer metadata operators already keep, such as topology and vendor, is a natural next step.

Insight 2: Even on clean data, prior approaches cannot express these invariants. Two obstacles hold a strictly-exact miner back. First, *noise*: NetNomos keeps only rules holding on 100% of snapshots, but two-end agreement holds exactly on barely a sixth once collection noise enters, and the conservation laws carry a systematic offset at every node, so

Invariant	Hold-rate	Origin
Two-end counter agreement	96.5%	CrossCheck
Router & network flow conservation	92.3–97.5%	CrossCheck
Non-negativity, zero self-demand	100%	Autogram
Topology & demand presence symmetry	92.4–97.3%	Autogram
Demand conservation (termination)	**73.2%**	**Autogram**
Demand conservation (origination)	**68.6%**	**Autogram**

Table 1: On a production network, Autogram recovers CrossCheck's invariants and discovers deeper ones it never reported (bold).

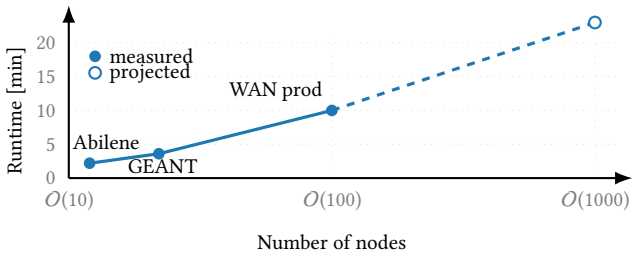

Figure 4: Autogram discovers a network's invariants in minutes, and the cost grows only gently as networks get larger.

both are discarded. Second, *expressivity*: even on noiseless data, its human-authored grammar sums through a fixed binary +, whereas the conservation invariants require aggregation over an entire family, of arity 11 on Abilene and 21 on GEANT. Autogram accepts an invariant that holds often enough within a data-driven tolerance, and its induced grammar supplies the n-ary aggregation and presence relations these families need, so lifting the expressivity cap is the first item on our agenda (§5).

Insight 3: A soft invariant is also a diagnostic. The traffic a router reports originating equals the row-sum of the demand matrix the controller consumes, and terminating traffic equals the column-sum, a routing-free boundary form Autogram finds directly from the counter names, without the topology or forwarding state that CrossCheck's path invariant requires. These laws run a uniform ∼2% short at every node, and Autogram reports the gap rather than discarding it: the interface counters are read locally while the demand matrix is an end-to-end aggregate, so the two disagree by a structural offset rather than noise (in production the counters even fold in header bytes the demand omits). An exact miner cannot represent such a law and a black-box model would bury it, whereas Autogram surfaces it as an auditable relation, so using discovered invariants to diagnose faults is a direction worth pursuing on its own.

Insight 4: With the right representation, completeness and readability stop competing. Each soft invariant Autogram reports is quantified over a locality family, so a single rule stands in for the many single-instance facts an exact miner must list. On GEANT, NetNomos emits 1,645 rules, of which 946 are vacuous disjunctions and only 27 match a known invariant, whereas Autogram reaches full recall with 101 rules, without trading recall for precision.

Insight 5: A production run shows both the promise and the limit. To test external validity, we run Autogram on a live operator WAN with $O(100)$ routers and $O(1000)$ links. From counter names alone, Autogram discovers 41 invariants end to end in ten minutes, spanning the same families as the benchmark catalog (Table 1) and again surfacing the demand-conservation laws with their ∼2% deficit, so the approach generalizes past the public benchmarks. Its runtime grows gently with network size (Fig. 4). Autogram also accepts a few relations that are not really invariants, and controlling those, with no ground truth to check against, is the open problem that stands between bounded and open-ended discovery.

5 Research Agenda

Discover more by annotating more. The semantic signal Autogram reads is metadata, so enriching it should enrich what Autogram discovers. Annotating each variable with side information the operator already tracks, such as router vendor, and letting the grammar range over it would surface relations no name-only view can express, for instance a counting imbalance that appears only across Arista-to-Juniper links.

When the annotations are wrong. Autogram reads metadata as ground truth, yet names and annotations drift, disagree across teams, and are sometimes simply wrong, and mislabeled telemetry is not a corner case because the annotations that carry the signal are hand-constructed and rarely audited [13]. Faulty semantics could mislead the grammar, but the same failure is diagnostic: a relation that holds everywhere except a few variables is strong evidence those variables are misannotated, so Autogram could use discovered invariants to flag or repair suspect labels [34]. The open question is how much annotation error the method absorbs before the signal degrades faster than the invariants can correct it.

Let the data shape the grammar. A grammar need not be invented from nothing when the data itself can suggest its shape. Autogram currently proposes relations from metadata before seeing a measurement, but a first statistical pass over the data, such as clustering variables that move together, could narrow the relations it plausibly supports and turn induction from free invention into grounded refinement [9, 46].

Compose invariants into new ones. Invariants compose: chaining or deriving from them can reach relations no single seed could express, turning bounded discovery into a foundation for open-ended discovery.

Proxies that do not presuppose the answer. Calibrating against recall of known invariants risks circularity: the proxy suite still targets a fixed catalog, so gains may reflect fit to it rather than a sharper instrument. A more defensible signal ties calibration to tasks whose objective never names an invariant, such as telemetry imputation, forecasting, or trace synthesis [15, 31, 40, 44], each rewarding a grammar that captures telemetry's joint structure, a signal defined over data alone and thus harder to overfit.

Acknowledgments

We thank Lily Tsai for her insightful feedback and comments, which meaningfully improved this paper.

References

[1] Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast Algorithms for Mining Association Rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*. 487–499.

[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] 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

[4] Nicolas Beldiceanu and Helmut Simonis. 2012. A Model Seeker: Extracting Global Constraint Models from Positive Examples. In *Principles and Practice of Constraint Programming*, Michela Milano (Ed.). Springer, Berlin, Heidelberg, 141–157. doi:10.1007/978-3-642-33558-7_13

[5] Miles Cranmer, Alvaro Sanchez Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. 2020. Discovering symbolic models from deep learning with inductive biases. *Advances in neural information processing systems* 33 (2020), 17429–17442.

[6] Joscha Cüppers, Adrien Schoen, Gregory Blanc, and Pierre-Francois Gimenez. 2024. FlowChronicle: Synthetic Network Flow Generation through Pattern Set Mining. *Proceedings of the ACM on Networking* 2, CoNEXT4 (2024), 1–20.

[7] 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.

[8] Armen Der Kiureghian and Ove Ditlevsen. 2009. Aleatory or epistemic? Does it matter? *Structural Safety* 31, 2 (2009), 105–112. doi:10.1016/j.strusafe.2008.06.020

[9] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao. 2007. The Daikon System for Dynamic Detection of Likely Invariants. *Science of Computer Programming* 69, 1-3 (2007), 35–45. doi:10.1016/j.scico.2007.01.015

[10] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. 2011. Discovering Conditional Functional Dependencies. *IEEE Transactions on Knowledge and Data Engineering* 23, 5 (May 2011), 683–698. doi:10.1109/TKDE.2010.154 Conference Name: IEEE Transactions on Knowledge and Data Engineering.

[11] Cheng Feng, Venkata Reddy Palleti, Aditya Mathur, and Deeph Chana. 2019. A Systematic Framework to Generate Invariants for Anomaly Detection in Industrial Control Systems. In *Network and Distributed System Security Symposium (NDSS)*.

[12] 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

[13] Benoît Frénay and Michel Verleysen. 2014. Classification in the Presence of Label Noise: A Survey. *IEEE Transactions on Neural Networks and Learning Systems* 25, 5 (2014), 845–869.

[14] 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 (HotNets '23)*. Association for Computing Machinery, New York, NY, USA, 48–55. doi:10.1145/3626111.3628188

[15] Fengchen Gong, Divya Raghunathan, Aarti Gupta, and Maria Apostolaki. 2024. Zoom2net: Constrained network telemetry imputation. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 764–777.

[16] Travis Hance, Marijn Heule, Ruben Martins, and Bryan Parno. 2021. Finding invariants of distributed systems: It's a small (enough) world after all. In *18th USENIX symposium on networked systems design and implementation (NSDI 21)*. 115–131.

[17] Hongyu Hè and Maria Apostolaki. 2025. Just-in-Time Logic Enforcement: A new paradigm of combining statistical and symbolic reasoning for network management. In *Proceedings of the 24th ACM Workshop on Hot Topics in Networks (HotNets 25)*. 184–192.

[18] 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)*.

[19] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. Tane: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. *Comput. J.* 42, 2 (Feb. 1999), 100–111. doi:10.1093/comjnl/42.2.100

[20] Eyke Hüllermeier and Willem Waegeman. 2021. Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods. *Machine Learning* 110, 3 (2021), 457–506. doi:10.1007/s10994-021-05946-3

[21] Xi Jiang, Shinan Liu, Aaron Gember-Jacobson, Arjun Nitin Bhagoji, Paul Schmitt, Francesco Bronzino, and Nick Feamster. 2024. Netdiffusion: Network data augmentation through protocol-constrained traffic generation. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 8, 1 (2024), 1–32.

[22] Minhao Jin and Maria Apostolaki. 2025. Robustifying ML-powered Network Classifiers with PANTS. In *34th USENIX Security Symposium (USENIX Security 25)*.

[23] Alex Kendall and Yarin Gal. 2017. What uncertainties do we need in Bayesian deep learning for computer vision?. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 30.

[24] 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

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

[26] Alexander Krentsel, Rishabh R. 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 (HotNets 24)*. ACM, 246–254. doi:10.1145/3696348.3696874

[27] Algorithmic SuperIntelligence Labs. [n. d.]. Open-source implementation of AlphaEvolve. https://github.com/algorithmicsuperintelligence/openevolve.

[28] Jing Lei, Max G'Sell, Alessandro Rinaldo, Ryan J. Tibshirani, and Larry Wasserman. 2018. Distribution-Free Predictive Inference for Regression. *J. Amer. Statist. Assoc.* 113, 523 (2018), 1094–1111.

[29] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. 2025. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131* (2025).

[30] Harris Papadopoulos, Kostas Proedrou, Volodya Vovk, and Alexander Gammerman. 2002. Inductive Confidence Machines for Regression. In *Machine Learning: ECML 2002*. Springer, 345–356.

[31] Konstantina Papagiannaki, Nina Taft, Zhi-Li Zhang, and Christophe Diot. 2005. Long-Term Forecasting of Internet Backbone Traffic. *IEEE Transactions on Neural Networks* 16, 5 (2005), 1110–1124.

[32] Thorsten Papenbrock and Felix Naumann. 2016. A Hybrid Approach to Functional Dependency Discovery. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 821–833. doi:10.1145/2882903.2915203

[33] Divya Raghunathan, Maria Apostolaki, and Aarti Gupta. 2025. A Layered Formal Methods Approach to Answering Queue-related Queries. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association, 1289–1304. https://www.usenix.org/conference/nsdi25/presentation/raghunathan

[34] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. Holistic Data Repairs with Probabilistic Inference. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1190–1201.

[35] Matthew Richardson and Pedro Domingos. 2006. Markov logic networks. *Machine Learning* 62, 1–2 (2006), 107–136.

[36] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. 2024. Mathematical Discoveries from Program Search with Large Language Models. *Nature* 625, 7995 (2024), 468–475.

[37] Žiga Stupan and Iztok Fister Jr. 2022. Niaarm: a minimalistic framework for numerical association rule mining. *Journal of Open Source Software* 7, 77 (2022), 4448.

[38] Deepak Vasisht, Zerina Kapetanovic, Jongho Won, Xinxin Jin, Ranveer Chandra, Sudipta Sinha, Ashish Kapoor, Madhusudhan Sudarshan, and Sean Stratman. 2017. FarmBeats: An IoT Platform for Data-Driven Agriculture. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 515–529.

[39] Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. 2005. *Algorithmic Learning in a Random World*. Springer.

[40] Xizheng Wang, Libin Liu, Li Chen, Dan Li, Yukai Miao, and Yu Bai. 2025. Resolving Packets from Counters: Enabling Multi-scale Network Traffic Super Resolution via Composable Large Traffic Model. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association, 1541–1561. https://www.usenix.org/conference/nsdi25/presentation/wang-xizheng-resolving

[41] 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

[42] Jianan Yao, Runzhou Tao, Ronghui Gu, and Jason Nieh. 2022. {DuoAI}: Fast, automated inference of inductive invariants for verifying distributed protocols. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 485–501.

[43] Jianan Yao, Runzhou Tao, Ronghui Gu, Jason Nieh, Suman Jana, and Gabriel Ryan. 2021. {DistAI}:{Data-Driven} automated invariant learning for distributed protocols. In *15th USENIX symposium on operating systems design and implementation (OSDI 21)*. 405–421.

[44] Yucheng Yin, Zinan Lin, Minhao Jin, Giulia Fanti, and Vyas Sekar. 2022. Practical gan-based synthetic ip header trace generation using netshare. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 458–472.

[45] Yifei Yuan, Soo-Jin Moon, Sahil Uppal, Limin Jia, and Vyas Sekar. 2020. {NetSMC}: A Custom Symbolic Model Checker for Stateful Network Verification. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 181–200.

[46] Chengqi Zhang and Shichao Zhang. 2002. *Association rule mining: models and algorithms*. Springer.