ARTIFACT EVALUATED USENIX AVAILABLE
ARTIFACT EVALUATED USENIX FUNCTIONAL
ARTIFACT EVALUATED USENIX REPRODUCED

# Making Logic a First-Class Citizen in Generative ML for Networking

Hongyu Hè
*Princeton University*

Minhao Jin
*Princeton University*

Maria Apostolaki
*Princeton University*

## Abstract

Generative ML models are increasingly popular in networking for tasks such as telemetry imputation, prediction, and synthetic trace generation. Despite their capabilities, they suffer from two shortcomings: *(i)* their output is often visibly violating well-known networking rules, which undermines their trustworthiness; and *(ii)* they are difficult to control, frequently requiring retraining even for minor changes.

To address these limitations and unlock the benefits of generative models for networking, we propose a new paradigm for integrating explicit network knowledge, in the form of first-order logic rules, into ML models used for networking tasks. Rules capture well-known relationships among observed signals, *e.g.,* that increased latency precedes packet loss. While the idea is conceptually straightforward, its realization is challenging: networking knowledge is rarely formalized into rules, and naively injecting rules into ML models often hampers their effectiveness. This paper introduces NetNomos, a multi-stage framework that *(i)* learns rules directly from data (*e.g.,* measurements); *(ii)* filters them to select semantically meaningful ones; and *(iii)* enforces them through collaborative generation between an ML model and a Satisfiability Modulo Theories (SMT) solver.

We show that NetNomos learns diverse, meaningful rules from four real-world datasets and is 1.6–6.5× more scalable than DuoAI, a state-of-the-art (SOTA) rule-learning method. By enforcing these rules on a generic GPT-2 model, NetNomos achieves performance on par with or even surpassing specialized SOTA systems such as Zoom2Net and NetShare across three networking tasks: telemetry imputation, traffic forecasting, and synthetic data generation.

## 1 Introduction

Generative machine learning (ML) models are increasingly popular for networking tasks such as measurement imputation, prediction, and synthetic trace generation. Their appeal lies in their adaptability, their capacity to learn directly from abundant network data, and their efficiency during inference.

Despite their potential, generative models often fall short in networking applications. They lack the strict correctness guarantees essential for network operations and offer limited controllability, severely restricting their real-world utility. Consequently, they can make egregious errors that degrade downstream performance, erode user trust, and are notoriously difficult to fix. For example, even state-of-the-art (SOTA) synthetic generators might produce UDP packets with TCP flags, skewing network sketch evaluations, and frustrating operators. Worse, fixing these mistakes typically requires expensive fine-tuning or complete retraining, assuming a fix is even possible [29, 40, 69].

Infusing generative models with domain knowledge can steer them away from nonsensical outputs, guarantee compliance with established networking principles, and reduce training overhead by eliminating the need to learn basic rules purely from data. However, this infusion is exceptionally challenging. First, networking knowledge is rarely formalized in a machine-readable format. While written sources like RFCs could theoretically be converted into such a format [3], they typically only describe a single protocol in isolation, ignoring the complex cross-layer interactions of real-world networks. Second, enforcing rules within ML pipelines is non-trivial. Existing constraint methods are often either too rigid—stifling the model's predictive ability (a known issue in large language models [72, 89])—or too loose, failing to provide meaningful guarantees. For example, best-effort enforcement in systems like NetDiffusion [40] still permits frequent rule violations, as we demonstrate in experiments (§7).

This paper introduces NetNomos[1], a novel, modular pipeline for designing controllable network data generators that provide strict correctness guarantees. NetNomos bridges this gap by combining the predictive power of ML with the underlying formal rules governing network data.

**NetNomos's Knowledge Mining.** Since formalized networking knowledge is scarce, we extract rules directly from

[1] Nomos comes from a Greek word meaning rule or law, reflecting NetNomos 's goal of learning and enforcing the underlying network principles.

data (*e.g.,* packet headers, measurements), which naturally obey constraints dictated by protocols and deployment policies. Viewing data samples as feasible solutions to these hidden constraints, NETNOMOS automatically mines the underlying rules. To balance scalability and expressiveness, NETNOMOS defines a first-order logic (FOL) grammar over observable variables and reduces the rule-learning task to the minimum hitting set problem [31].

**NETNOMOS's Filtering.** A critical challenge is that rules learned purely from data can be coincidental and lack true semantic meaning. However, unlike opaque ML weights, logic rules are discrete and auditable. NETNOMOS uses LLMs—grounded in networking texts—to filter and select meaningful rules. By restricting the LLM to choose only from mathematically mined rules, we ensure that consistency guarantees are preserved.

**NETNOMOS's Knowledge Enforcement.** NETNOMOS introduces a new approach to generation by embedding a Satisfiability Modulo Theories (SMT) solver directly into the language model's token-by-token generation process. Unlike prior methods that bake rules into training or attempt post-hoc corrections, NETNOMOS provides strict guarantees with minimal invasion to the underlying model, preserving its fidelity. This inference-time enforcement also allows models to be easily repurposed for new tasks by simply swapping the rule set, eliminating the need for retraining.

We evaluate NETNOMOS's ability to scalably learn meaningful rules and improve data generation across diverse datasets. To support this, we built and open-sourced new benchmarks for networking rules [32] alongside an extensive suite of baselines from three domains. Our results show that traditional database and ML approaches (*e.g.,* FastDC [13,55], H-Mine [54], FlowChronicle [22]) lack the expressiveness needed to formalize network knowledge. Furthermore, NETNOMOS is 1.6× to 6.5× more scalable than DuoAI [82], a SOTA, equally expressive rule-learning approach from the distributed systems domain.

Finally, we demonstrate the full NETNOMOS pipeline using a lightweight language model (GPT-2) across three use cases: telemetry imputation, synthetic data generation, and prediction. NETNOMOS reliably produces compliant network data—unlike heavily tailored SOTA systems (Zoom2Net [27], NetDiffusion [40], NetShare [84])—while matching or exceeding their performance metrics. Crucially, enforcing rules exclusively at inference is sufficient to tailor a single, basic GPT-2 model to all three tasks. This highlights the power of decoupling network knowledge extraction from ML inference, rather than relying on an opaque, end-to-end black box.

## 2 Motivation

We first examine two networking use cases that demand a neurosymbolic approach. While generative models have unlocked new capabilities for these use cases, they still require explicit knowledge infusion to be truly reliable. We then outline the fundamental pitfalls of prior designs that fail to effectively bridge machine learning and symbolic reasoning, hence fail to realize a truly neurosymbolic approach.

### 2.1 Motivating Use Cases

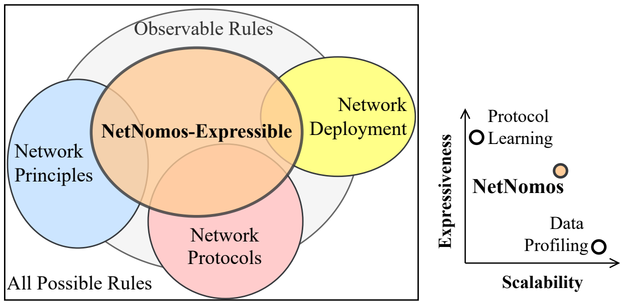
**Synthetic data generation.** Generative models can produce synthetic network traces (*e.g.,* packet headers and timestamps) by learning the latent structures and dependencies in real traffic [12, 40, 42, 46, 84]. These synthetic data generators (SynGens) allow third parties to optimize downstream applications without accessing sensitive raw data. However, most SynGens rely on end-to-end pipelines that force the model to learn all networking rules purely from data. This approach is data-hungry, computationally expensive, and offers no correctness guarantees. For example, SOTA systems like NetShare generate UDP packets with TCP flags or DNS packets with out-of-range IPs, severely undermining user trust. While recent systems like NetDiffusion [40] attempt post-generation corrections to preserve semantics, they still commit glaring errors. As detailed in §6, we found that NetDiffusion frequently breaks sequence number continuity and mishandles TCP handshakes despite its explicit post-hoc constraints.

**Telemetry imputation.** Recent efforts improve software-based network monitoring by recovering fine-grained time series from coarse-grained telemetry [27]. This is possible because concurrent network signals (*e.g.,* queue lengths, ECN marks, and retransmissions) act as correlated symptoms of underlying states like congestion. Zoom2Net [27] leverages generative models to learn these correlations and impute missing details. To improve trustworthiness, it incorporates domain knowledge via manually crafted symbolic rules. However, Zoom2Net's pipeline has two major drawbacks. First, because rules are embedded in the training loss function, the model requires complete retraining for even minor rule adjustments, such as changing the input collection granularity. Second, its reliance on a small, manually curated rule set leaves room for basic errors. For example, when trained on Meta's datacenter dataset to impute ingress bytes per millisecond, Zoom2Net occasionally generates ingress byte counts that are impossibly smaller than the retransmitted bytes in the same interval (§6).

### 2.2 Pitfalls in Generative ML for Networks

Existing approaches fail to integrate ML and formal reasoning for three key reasons:

**End-to-end reliance.** Driven by advancements like the attention mechanism [77], designers frequently use a single model to learn all correlations directly from data. While minimizing manual effort, this reliance makes systems *uncontrollable* and *brittle*. For instance, operators cannot instruct NetShare



(a) (b)

Figure 1: (a): NetNomos finds rules that connect observable variables; these can stem from network principles, protocols, deployment decisions, or their combination. (b): NetNomos strikes a delicate balance between expressiveness and scalability in the trade-off space for learning network rules.

to obey basic handshake semantics and must entirely retrain Zoom2Net simply to swap an input signal.

**Limited rule coverage.** Systems that do incorporate domain-specific rules typically include only a fraction of the necessary constraints. First, they rely on manually specifying rules, which will inevitably lead to poor coverage. Furthermore, many critical networking rules are either non-differentiable or too complex to encode in standard ML loss functions, strictly limiting systems like Zoom2Net to differentiable rules.

**Knowledge competing with ML.** Rules are frequently applied post-hoc to "correct" outputs, placing symbolic knowledge in direct competition with the ML model. This forces a binary choice between trusting the model or the rules. In NetDiffusion [40], for example, if post-processing cannot enforce a rule within a few iterations, the system defaults to the model's output, inevitably yielding data that violates fundamental networking principles.

## 3 Overview

Motivated by the potential of infusing domain knowledge into ML for networking (a neurosymbolic approach), and informed by a clearer understanding of the pain points that deter designers from effectively integrating symbolic and neural components, our vision is to systematize this integration. The goal of this paper is to lay the foundations of a framework that automates rule discovery and seamlessly integrates these rules with generative models. Such a framework will empower network operators to *(i)* leverage the rich domain knowledge associated with networks in their systems because codifying it will not be a labor-intensive task; *(ii)* benefit from generative models while maintaining their control (for example, they can always add a rule); and *(iii)* avoid choosing between the correctness guarantees offered by domain knowledge (rules) and the predictive capability offered by ML. Next, we highlight the key insights that allow NetNomos to realize this vision.

**NetNomos mines rules directly from network data.** We observe that the network itself can be seen as a data-generating process constrained by underlying principles and rules (e.g., capacity limits, routing behavior, protocols) where measurements, packet traces, and other monitoring vectors are feasible solutions of these constraints [27,40,41]. Hence, NetNomos reframes the problem of formalizing network knowledge into logic constraints from a manual, and error-prone task, to a constraint learning problem. At a high level, NetNomos analyzes network data (*e.g.,* measurements, packet headers) and infers rules (formulas over observable fields from data as variables). Observable variables (*e.g.,* measurements, packet headers) obey rules that stem from protocol definition (*e.g.,* TCP handshake), principles (*e.g.,* packet drops happen after queue build-up), and deployment decisions (*e.g.,* vantage point locations), but also from their combinations. These are the exact rules that NetNomos aims to find, as also illustrated in Fig. 1a. Observe that NetNomos cannot find all rules contained in RFCs or textbooks because they might connect variables that are not in the data, *i.e.,* are not measured. Conversely, many of the rules NetNomos can find and stream from interactions across components cannot be mined from text, which typically explains concepts in isolation. By design, NetNomos produces rules that are easier to use for guiding ML models. These models are already limited to observable variables during training and are often tailored to specific deployments.

**NetNomos reduces rule learning to the minimal hitting set problem.** Treating the network as a constrained generation process introduces a trade-off between expressiveness and scalability. Capturing complex relationships requires logical expressiveness, but greater expressiveness enlarges the search space and hurts scalability. For instance, while most network rules can be expressed in first-order logic (FOL), learning arbitrary FOL rules is undecidable [6].

To address this challenge, NetNomos defines a constraint language $\Gamma$ that restricts expressiveness to a guarded, finite-domain fragment of first-order logic (FOL). This fragment is expressive enough to capture useful rules, yet structured so that rule learning reduces to a minimal hitting set problem [65] (Fig. 2). Concretely, $\Gamma$ ranges over a finite set of dataset variables with finite domains (*e.g.,* fixed-bitwidth header fields, categorical fields, and bounded counters). Since all quantification in $\Gamma$ is over this finite universe, each $\Gamma$-constraint can be grounded exactly by enumeration, yielding a propositional formula whose literals correspond to ground predicate instances (including ground arithmetic comparisons over bounded domains). This grounding is information-preserving under the finite-domain semantics, so satisfaction of a $\Gamma$-formula on a record is unchanged.

The resulting propositional clauses are combined into candidate constraints $\mathbf{C}$ as disjuncts. NetNomos mutates these candidates (*e.g.,* by adding or removing clauses) and retains

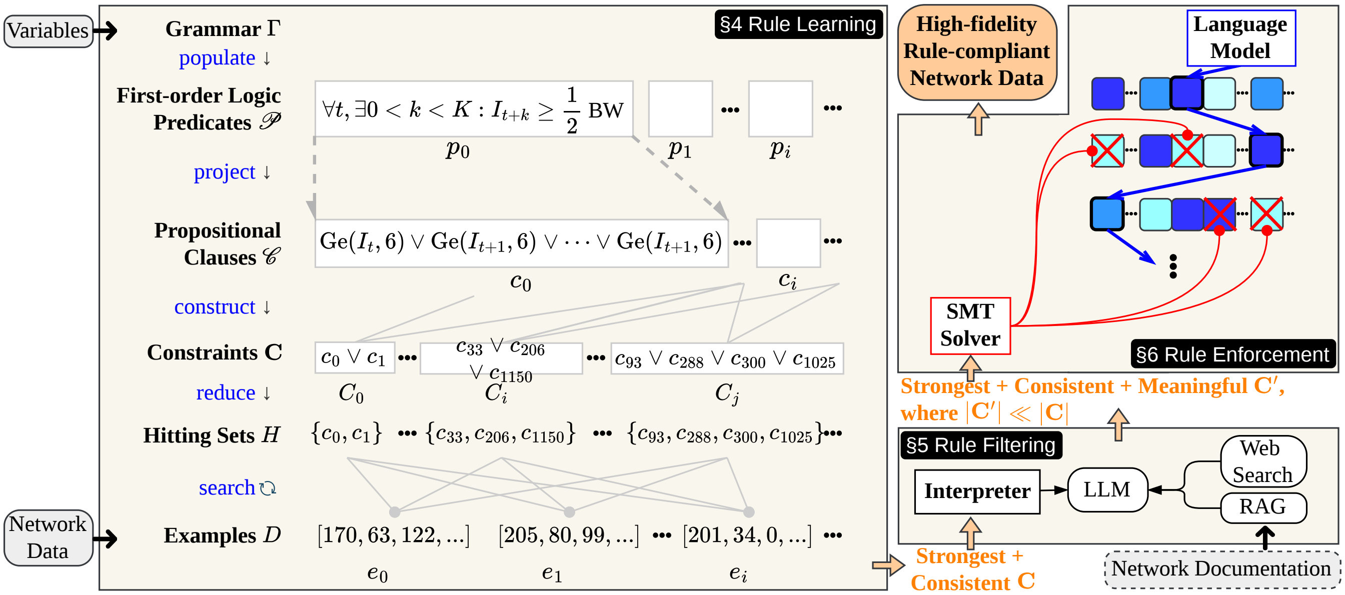

Variables
Grammar Γ
populate ↓
First-order Logic Predicates 𝒫
$\forall t, \exists 0 < k < K : I_{t+k} \ge \frac{1}{2}$ BW
$p_0$
$p_1$
$p_i$
project ↓
Propositional Clauses 𝒞
$\mathrm{Ge}(I_t, 6) \lor \mathrm{Ge}(I_{t+1}, 6) \lor \cdots \lor \mathrm{Ge}(I_{t+1}, 6)$
$c_0$
$c_i$
construct ↓
Constraints **C**
$c_0 \lor c_1$
$c_{33} \lor c_{206} \lor c_{1150}$
$c_{93} \lor c_{288} \lor c_{300} \lor c_{1025}$
$C_0$
$C_i$
$C_j$
reduce ↓
Hitting Sets $H$
$\{c_0, c_1\}$
$\{c_{33}, c_{206}, c_{1150}\}$
$\{c_{93}, c_{288}, c_{300}, c_{1025}\}$
search
Network Data
Examples $D$
[170, 63, 122, ...]
[205, 80, 99, ...]
[201, 34, 0, ...]
$e_0$
$e_1$
$e_i$
§4 Rule Learning
High-fidelity Rule-compliant Network Data
Language Model
SMT Solver
§6 Rule Enforcement
Strongest + Consistent + Meaningful **C**′, where |**C**′| ≪ |**C**|
§5 Rule Filtering
Interpreter
LLM
Web Search
RAG
Strongest + Consistent **C**
Network Documentation

Figure 2: NETNOMOS consists of three stages: **Rule Learning**, where NETNOMOS identifies the minimum set of constraints that are consistent with data (*i.e.,* are consistent and strongest) after reducing the problem into the minimum hitting set problem; **Rule Filtering**, where an LLM (or a human) filters out some of the learned rules as meaningless; and **Rule Enforcement**: where an SMT solver enforces rules during the token-by-token generation of a language model by invalidating the tokens that if selected by the LM would result in an inconsistent output (*e.g.,* a sequence of packets with header fields that violate protocol rules or an imputed fine-grained vector of measurements that defy network principles).

those consistent with the dataset while concise. We solve the search as a minimal hitting set problem (§4.5), followed by semantic filtering (§5).

**NETNOMOS interjects an SMT solver into the token-by-token generation of a language model (LM).** Instead of using knowledge for training or post-inference, NETNOMOS includes a true constraint solver that intersects the LM's token-by-token inference to guide it towards rule-compliant generation as shown in Fig. 2 top right. Tokens are illustrated as blue squares, with color intensity reflecting the probability assigned by the LM. Before each token is selected according to these probabilities by the model, the solver dynamically computes the set of valid next tokens based on the logic rules learned in earlier NETNOMOS steps and the sequence of tokens generated so far. This approach enables easy repurposing of LMs by modifying the rules that are applied during inference rather than retraining or fine-tuning. It also allows network operators to focus on defining useful rules without worrying whether they are differentiable, appear enough in training, or can be embedded in prompts. Finally, NETNOMOS enforces rules during inference in a minimally invasive way and preserves the statistical fidelity of ML-learned data distribution. The process of enforcing rules is explained in §6.

### 3.1 End-to-end View of NETNOMOS

To better understand NETNOMOS end-to-end, let's consider an operator seeking to recover fine-grained (1ms) granularity ingress byte counts of server ports using coarse-grained (50ms) timeseries of congested, retransmitted, dropped, ingress, and egress packet counts. This is the imputation use case described in §2.1. Instead of using Zoom2Net [27], which requires the operator to manually write rules, the operator uses the first stage of NETNOMOS with two inputs: Meta's dataset [26] and the set of variables among the observable ones (those in the dataset) that they believe are correlated. NETNOMOS first stage computes a set of non-redundant (later defined as strongest) constraints that are consistent with that dataset $D$. Let us assume that NETNOMOS first stage finds the following rules:

$$\forall t, 0 \le k < K:\ 0 \le I_{t+k} \le \mathrm{BW}, \tag{R0}$$

$$\forall t : \texttt{Ingress}_t^K = \sum_{k=1}^{K-1} I_{t+k}, \tag{R1}$$

$$\forall t : \texttt{Congestion}_t^K > 0 \implies \exists 0 \le k < K : I_{t+k} \ge \frac{1}{2}\mathrm{BW}, \tag{R2}$$

$$\forall t : \texttt{Connections}_k^K > \mathrm{TH} \implies (\texttt{Congestion}_t^K > 0 \lor \texttt{InRxmit}_t^K > 0), \tag{R3}$$

$$\forall t : \texttt{Egress}_t^K > \texttt{InRxmit}_t^K + \texttt{OutRxmit}_t^K, \tag{R4}$$

where BW stands for link bandwidth and TH stands for connection count threshold for incast.

In the second stage, an LLM will filter out meaningless rules in our example, R4. The remaining constraints $\mathbf{C}'$ are passed to an SMT solver, which computes the set of invalid tokens before each new layer of token generation. At each step, the SMT solver incorporates the tokens already generated into the constraint problem at hand. The output of NetNomos is an input Ingress byte sequence that follows logic rules enforced by the SMT solver and statistical properties enforced by the LM. Visually, NetNomos aims to constrain the generation to a region that is subject to learned rules (*i.e.,* shaded region) in Fig. 5. Note that the ground truth is always within the shaded region, whereas Zoom2Net's output is not.

## 4 Extracting Knowledge from Network Data

### 4.1 Formulation of Constraint Modeling

We view the network as a data-generating process constrained by rules arising from three sources: network principles, protocols, and deployment specifics. Each rule is expressed in formal logic as a symbolic *constraint* $C$. The network data $D$ thus consists of examples that satisfy these constraints.[2] An example $e \in D$ depends on the data type: a flow record in NetFlow traces, a sequence of packet headers in PCAP traces, or a time series in performance measurements. $C$ is *consistent* if it is satisfied by all examples: $D \models C$.

Given a set $\mathbf{C}$ of consistent constraints, each constraint corresponds to a model set $\mathcal{M}(C) = \{e \in D : e \models C\}$. A constraint $C$ is stronger than $C'$ if $\mathcal{M}(C) \subset \mathcal{M}(C')$, and two constraints $C, C'$ are equivalent if $\mathcal{M}(C) = \mathcal{M}(C')$. A constraint is in its strongest form if there is no non-equivalent $C' \in \mathbf{C}$ such that $\mathcal{M}(C') \subset \mathcal{M}(C)$. Formally, $C$ is strongest iff $\forall C' \in \mathbf{C},\ C' \not\equiv C \implies \mathcal{M}(C) \subset \mathcal{M}(C')$.

A constraint $C$ is redundant if there exists some $C' \in \mathbf{C}$ with $C' \models C$, in which case $C$ adds no new information beyond what is already entailed by $C'$. Equivalently, redundancy arises whenever $\mathcal{M}(C) \supseteq \mathcal{M}(C')$. Therefore, a constraint is non-redundant precisely when it is maximally strong: it cannot be replaced by a strictly stronger consistent constraint while preserving equivalence.

Each $C$ captures relationships among fields in $D$, which we treat as *variables* $V$, such as, source/destination IPs and ports, frame/packet/segment sizes, or ECN marked bytes in collected measurements. Relationships among $V$ can be complex, as they arise from various sources shown in Fig. 1a. Capturing these intricate relationships in a concise and straightforward way requires $C$ to be expressed in first-order logic (FOL).

**Goal.** Given a network dataset $D$ and its variables of interest $V$, we aim to learn a *minimal constraint theory*, defined as $\mathrm{Th}(\mathbf{C}) := \bigwedge_{C \in \mathbf{C}'} C$, where $\mathbf{C}^{'} \subseteq \mathbf{C}$ is the subset of consistent and strongest constraints. This formulation ensures that $\mathrm{Th}(\mathbf{C})$ avoids thousands of weaker reformulations of the same constraint, which contribute no additional information.

| | | |
|---|---|---|
| $\langle\texttt{type}\rangle$ | $::=$ | $\tau \in \{\mathrm{TIME}, \mathrm{SIZE}, \mathrm{ID}, \mathrm{FLAG}, \mathrm{COUNT}\}$ |
| $\langle\texttt{variable}\rangle$ | $::=$ | $v^{\langle\texttt{type}\rangle} \in V$ |
| $\langle\texttt{index}\rangle$ | $::=$ | $0 \mid 1 \mid \ldots \mid K-1$ |
| $\langle\texttt{svar}\rangle$ | $::=$ | $\langle\texttt{variable}\rangle^{\{\langle\texttt{index}\rangle\}}$ |
| $\langle\texttt{context}\rangle$ | $::=$ | $\{\langle\texttt{svar}\rangle\ (,\ \langle\texttt{svar}\rangle)^*\}$ |
| $\langle\texttt{constant}\rangle$ | $::=$ | $\mathcal{D}(V)\ \cup\ \mathcal{B}(D)$ |
| $\langle\texttt{operator}\rangle$ | $::=$ | $+ \mid \times \mid < \mid > \mid \leq \mid \geq \mid = \mid \neq$ |
| $\langle\texttt{connective}\rangle$ | $::=$ | $\vee \mid \wedge \mid \implies$ |
| $\langle\texttt{aggregator}\rangle$ | $::=$ | $\max \mid \min \mid \sum \mid \mathrm{avg}$ |
| $\langle\texttt{avar}\rangle$ | $::=$ | $\langle\texttt{aggregator}\rangle(\langle\texttt{svar}\rangle^+)$ |
| $\langle\texttt{lhs}\rangle$ | $::=$ | $[\langle\texttt{constant}\rangle\langle\texttt{operator}\rangle]\langle\texttt{svar}\rangle^{\langle\texttt{type}\rangle}$ |
| $\langle\texttt{term}\rangle$ | $::=$ | $\langle\texttt{constant}\rangle$ |
| | $\mid$ | $[\langle\texttt{constant}\rangle\langle\texttt{operator}\rangle]\langle\texttt{svar}\rangle^{\langle\texttt{type}\rangle}$ |
| $\langle\texttt{rhs}\rangle$ | $::=$ | $\langle\texttt{term}\rangle \mid [\langle\texttt{constant}\rangle\langle\texttt{operator}\rangle]\langle\texttt{avar}\rangle$ |
| $\langle\texttt{predicate}\rangle$ | $::=$ | $(\ \langle\texttt{lhs}\rangle^\tau\ \langle\texttt{operator}\rangle\ \langle\texttt{rhs}\rangle^\tau\ )$ |
| $\langle\texttt{pred\_lit}\rangle$ | $::=$ | $[\neg]\langle\texttt{predicate}\rangle$ |
| $\langle\texttt{predicates}\rangle$ | $::=$ | $\langle\texttt{pred\_lit}\rangle\ (\ \langle\texttt{connective}\rangle\ \langle\texttt{pred\_lit}\rangle\ )^*$ |
| $\langle\texttt{qf}\rangle$ | $::=$ | $\langle\texttt{predicates}\rangle$ |
| $\langle\texttt{qblock}\rangle$ | $::=$ | $\varepsilon$ |
| | $\mid$ | $\forall\ \langle\texttt{context}\rangle \mid \forall\ \langle\texttt{context}\rangle\ \exists\ \langle\texttt{svar}\rangle$ |
| $\langle\texttt{constraint}\rangle$ | $::=$ | $\langle\texttt{qblock}\rangle\ :\ \langle\texttt{qf}\rangle$ |

Table 1: Grammar $\Gamma$ for the phrase structure of network rules with context window $K$, and user-defined aggregations (shaded). Well-formedness of $\Gamma$ requires that variables in $\langle\texttt{lhs}\rangle$ and $\langle\texttt{rhs}\rangle$ of a predicate share the same type $\tau$.

**Complexity.** Learning FOL formulas from $D$ is an *undecidable* problem, and therefore, one has to restrict FOL to a decidable fragment, *e.g.,* the Bernays-Schönfinkel class [64]. Learning a minimal constraint theory even within a decidable fragment is NP-complete [20, 36, 50]. Consequently, the effectiveness of a rule-leaning method comes down to the efficiency of its search algorithm operating in a huge combinatorial space.

### 4.2 Expressive Grammar for Network Data

**Rich expressiveness.** Within the decidable fragment of FOL, we define the grammar $\Gamma$ such that it is sufficiently expressive to capture most network rules observable from data.

Table 1 summarizes the phrase structures of $\Gamma$. It supports arbitrary combinations of variables and constants as terms, together with a variety of arithmetic operations. While being flexible, $\Gamma$ tags every $v \in V$ with one of five types and enforces type checking on predicates. This typing avoids meaningless constraint candidates, for example, `FlowBytes` $>$ `Duration`, where `FlowBytes` is of type COUNT while `Duration` is of type TIME. Constants in $\Gamma$ come from two sources: (1) known variable domains $\mathcal{D}(V)$ derived from existing documentation/specifications, and (2) background knowledge $\mathcal{B}_D$

[2] A constraint $C$ is a purely syntactic object. We use the term *rule* $R$ to refer to its semantics (either meaningful or meaningless). A meaningful rule must be consistent with the data, but not every consistent rule is meaningful.

obtained via profiling. By default, $\Gamma$ includes a set of well-known constants (*e.g.,* ports, protocols, valid combinations of TCP flag bits). For variables without predefined domains, NETNOMOS extracts constants directly from $D$: it selects the top-10 most frequent values for discrete $V$, and for continuous $V$, the five quartiles plus the 90th percentile (p90). Finally, $\Gamma$ supports user-defined functions such as numerical aggregations. During evaluation on a concrete record (or context window), we treat these functions as uninterpreted operators and compute their outputs directly, without applying symbolic rewriting or additional constraints.

**Restrictions.** Without restrictions, a constraint grammar leads to an unbounded search space of candidate formulas [15]. Even with bounded arity $a$ (number of variables), the predicate space $\mathscr{P}$ grows as $|\mathscr{P}| = O(r|V|^2)$, where $r$ is the number of operators. The corresponding formula space $\mathscr{F}$ then explodes doubly exponentially, $|\mathscr{F}| = 2^{2^{|\mathscr{P}|}}$. An overly general grammar design leads to such intractable complexity. Therefore, it is imperative to specialize $\Gamma$ by enforcing restrictions on its phrase structure, so that the search space becomes tractable while still expressive enough to capture most observable network rules.

$\Gamma$ imposes two main restrictions on constraint candidates. First, predicates may only contain terms with a single variable on one side of the operator and a single or aggregated variable on the other side. We observe that such this form of predicates is sufficient in capturing most relationships we can observe from four diverse network datasets (Table 2, 5–7). This restriction limits the size of the predicate space ($|\mathscr{P}|$), which in turn bounds the combinatorial search space of all possible constraints, *i.e.,* $O(2^{|\mathscr{P}|})$. Moreover, it confines the computation domain to Linear Rational Arithmetic (LRA), which keeps $\Gamma$ decidable [9]. Second, the quantifier $\exists$ may only appear after $\forall$ within the context size $K$. In other words, $\Gamma$ disallows global $\exists$ quantification; instead, $\exists$ is only effective within $K$ consecutive observations. We impose this restriction for two reasons: (1) evaluating $\exists$ is prohibitively expensive, even within a limited time window (*e.g.,* a protocol execution [30, 82, 83]); and (2) network data $D$ typically contains millions of records and is *unbounded in time* (*i.e.,* does not have the notion of a single run/execution), making global existential learning impractical.

We argue that the two restrictions do not fundamentally undermine the expressiveness of NETNOMOS. In practice, the wide variety of predicates ($|\mathscr{P}|$) supported by $\Gamma$ still reaches the thousands even with restrictions, whereas prior work [13, 43, 47, 55, 83] supports fewer than 100 predicates. Moreover, while global existential properties are valuable, network operators rarely analyze them in practice since doing so requires substantial compute and/or memory [24, 26, 34, 45, 85, 87].

## 4.3 Limitations of Existing Work

Existing methods for learning logical rules fall into two categories: data profiling and protocol invariant learning. The former lacks expressiveness, while the latter fails to scale to the complexity of network data.

Decades of work in data profiling (*e.g.,* association rule learning [54,79,86], functional dependency discovery [52,53]) has produced methods with limited formal grammars. These methods cannot capture the diversity of network rules. As shown in Fig. 6a, their expressiveness bounds the rules they can learn, preventing full coverage of network behaviors.

Protocol learning methods (*e.g.,* I4 [47], FOL-IC3 [43], DistAI [83], SWISS [30], DuoAI [82], Basilisk [88]) are expressive but unsuitable for network data. They face three fundamental limitations. First, they assume protocol models are *known* and use these as ground truth with verifiers like IVy [51]. NETNOMOS, in contrast, has no model of the entire network. Second, they rely on *direct* input-output traces from simulation. NETNOMOS operates in a passive setting, learning only from collected traces. Without interactive feedback, learning an equivalent automaton is provably infeasible [2]. Third, they are designed for learning one protocol at a time. This assumption restricts their predicate space $\mathscr{P}$ to variables of a single entity, keeping $|\mathscr{P}|$ small. Network data $D$ is heterogeneous, containing many entities and their interactions. As a result, NETNOMOS 's predicate space is 2–15× larger, and since the search space is exponential in $|\mathscr{P}|$, scalability becomes the key barrier. As shown in Fig. 6b, a SOTA protocol learning method learns only a fraction of benchmark rules, while NETNOMOS learns nearly all within the same time budget.

## 4.4 From Constraints to Hitting Sets

Prior work enumerates the formula space $\mathscr{F}$. This approach does not scale for two fundamental reasons. *First*, evaluating the strength of each candidate constraint requires scanning *all* examples. The cost is $O(|D| \cdot 2^{|\mathscr{P}|})$. In networking, $|D|$ is often in the millions. *Second*, instantiating all constraints at the same strength causes *exponential level-wise growth*. Weakening produces more candidates at the next level than the current level provides. For example, if $P_1 \vee P_2$ is too strong (where $P_i$ are predicates), weakening by adding $P_3$ yields $|\mathscr{P}|$ new candidates to evaluate.

NETNOMOS takes an indirect route, reducing the search for strongest constraints to a Minimal Hitting Set problem [13,65]. We design $\Gamma$ so that all variables range over finite domains (*e.g.,* bounded counters, finite-width header fields, and a finite context window $K$). Under these finite-domain semantics, any $\Gamma$-constraint can be grounded exactly by enumerating quantified variables, yielding a finite Boolean formula whose literals are ground predicate instances (including ground arithmetic comparisons). This grounding is information-preserving for

Γ and enables solving the search via hitting set. Thus every constraint $C$ can be represented as a set of *clauses* in propositional logic, *e.g.,* $C = (c_0 \vee c_1 \vee c_2) \equiv \{c_0, c_1, c_2\}$. A *clause* is composed of one or more propositions, *e.g.,* protocol is TCP: $\mathrm{Eq}(\texttt{Proto}, TCP)$. Fewer clauses mean a stronger constraint: $C' = \{c_0, c_2\}$ is stronger than $C$, *i.e.,* $C' \vdash C$. Each clause $c$ satisfies a subset of examples. Define its evidence set as $E_c := \bigcup_e^D e \models c$. All clauses in a consistent $C$ together must satisfy $D$. Equivalently, $C \models D \iff \left(\bigcup_c^C E_c = D\right)$.

Intuitively, learning a consistent constraint means finding clauses whose evidence sets include $D$. This process reduces to the Hitting Set problem: given a set of clauses, find a subset $H$ of size at most $s$ whose evidence sets together "hit" all examples in $D$. The bound $s$ limits the number of clauses a constraint may contain.

> **Theorem 1.** *Learning a consistent constraint $C$ on examples $D$ is equivalent to finding a hitting set $H$ of clauses whose evidence sets hit $D$.*[3]

A hitting set $H$ is *minimal* if no proper subset of $H$ is also a hitting set. In that case, $C := \bigwedge_c^H c$ uses the fewest possible clauses and is therefore the strongest:

> **Lemma 2.** *A minimal hitting set corresponds to a constraint that is the strongest.*

This reduction gives NetNomos two scaling advantages. First, it avoids exhaustive per-candidate evaluation on $D$. As explained in §4.5, NetNomos scans $D$ once to construct evidence sets, yielding complexity $O(|D| \cdot |\mathscr{P}|)$ instead of $O(|D| \cdot 2^{|\mathscr{P}|})$. Second, it traverses the search space without level-by-level instantiation by strength, thereby avoiding exponential search space explosion. Next, we explain the learning process step by step.

## 4.5 Rule Learning Method of NetNomos

**Search space projection.** The first step of the learning process is constructing the search space. NetNomos starts by populating the space $\mathscr{P}$ of FOL predicates that are allowed by Γ: $\mathscr{P} := \{\texttt{Ingress}_t^K = \sum_{k=0}^{K-1} I_{t+k},\ \texttt{Ingress}_t^K > \sum_{k=0}^{K-1} I_{t+k},\ \texttt{Congestion}_t^K = 0,\ \texttt{Congestion}_t^K > 0,\ \exists 0 \leq k < K-1 : I_{t+k} \geq \frac{1}{2}\mathrm{BW}, \ldots\}$.

Then, NetNomos propositionalizes $\mathscr{P}$ by projecting all predicates therein to propositional clauses ($\mathscr{C}$). Specifically, this process unrolls quantifier ∃ in FOL and translates aggregation functions to basic propositions:

$$\begin{aligned}
\mathscr{C} := \{&\mathrm{Eq}(\texttt{Ingress}_t^K, I_t + I_{t+1} + \cdots + I_{t+K-1}), && (c_0)\\
&\mathrm{Gt}(\texttt{Ingress}_t^K, I_t + I_{t+1} + \cdots + I_{t+K-1}),\ \ldots && \\
&\mathrm{Eq}(\texttt{Congestion}_t^K, 0), && (c_2)\\
&\mathrm{Gt}(\texttt{Congestion}_t^K, 0),\ \ldots, && \\
&(\mathrm{Ge}(I_t, 6) \vee \mathrm{Ge}(I_{t+1}, 6) \vee \ldots \vee \mathrm{Ge}(I_{t+K-1}, 6)), && (c_5)\\
\ldots\}, &&&
\end{aligned}$$

where constants are materialized in Gbps for simplicity.

This process produces a clause space $\mathscr{C}$, from which constraint formulas are constructed.

**Building evidence sets.** For each $c \in \mathscr{C}$, NetNomos builds an evidence set $E_c$ containing indices of all examples satisfying $c$: $\forall i \in E_c : D_i \models c$. In Fig. 3, six clauses $c_0$–$c_5$ and five examples (0–4) yield six sets $E_0$–$E_5$. For example, all examples satisfy $c_0$, while only $D_3$ satisfies $c_1$. This step runs in $O(|D| \cdot |\mathscr{P}|)$.

**Finding minimal hitting sets.** NetNomos then solves the minimal hitting set problem using branch and bound (Fig. 3). Clauses are ranked by *cover*, the number of uncovered examples they hit: $\mathrm{Cover}(c) = \mid E_c \setminus \bigcup_{c'}^H E_{c'} \mid$. The highest-cover clause becomes the pivot. For example, $c_0$ is chosen first, producing $H = c_0$ which hits all examples and forms a valid hitting set. Unhit examples are tracked as $M = D \setminus \bigcup_{c \in H} E_c$. When $M = \emptyset$, $H$ is valid; otherwise, NetNomos branches on new pivots. For instance, branching on $c_5$ yields $M = \{1, 2\}$, and further branching on $c_2$ produces another hitting set $\{c_5, c_2\}$. Non-minimal sets, such as $\{c_5, c_4, c_2\}$, are discarded.

**Bounding search.** To prune, NetNomos computes the maximum possible cover $\psi = \max_c \mathrm{Cover}(c)$ and stops search when $s - |H| < |M| / \psi$, where $s$ is the maximum formula size. This optimistic bound prevents exploring subtrees that cannot yield valid sets.

**Minimal constraint theory & proof system.** The process yields minimal hitting sets such as $\{c_0\}$ and $\{c_5, c_2\}$, which correspond to constraints R1 and R2:

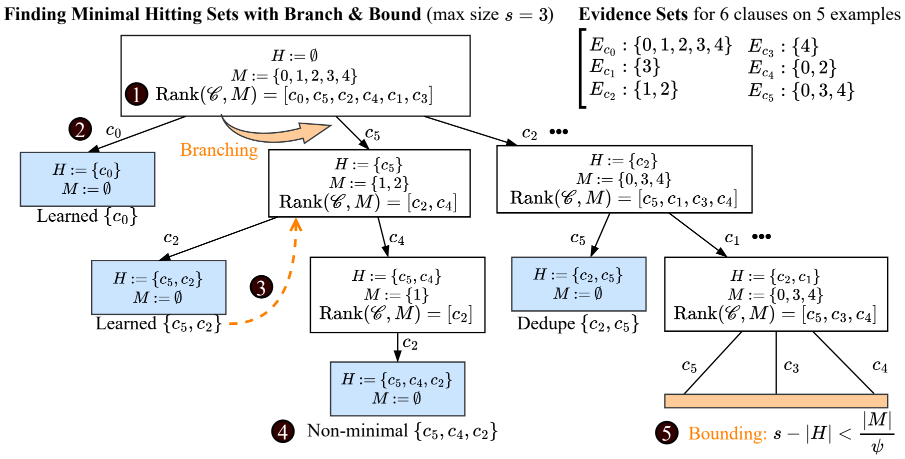
$$\begin{aligned}
\{c_0\} &\mapsto c_0 \equiv C1 \equiv \mathrm{R1}, \text{ and}\\
\{c_5, c_2\} &\mapsto (c_5 \vee c_2) \equiv (\neg c_5 \implies c_2) \equiv C2 \equiv \mathrm{R2}.
\end{aligned}$$

NetNomos then constructs a constraint theory $\mathrm{Th}(\mathbf{C}) = \bigwedge_{C \in \mathbf{C}} C$ and supports reasoning with the Fitch proof system [23], which is sound and complete for propositional clauses. Given a query $q$,

$$f_{\mathrm{Fitch}} : q \mapsto \{\top, \bot, ?\} = \mathrm{Th}(\mathbf{C}) \vdash q, \qquad (1)$$

where $f_{\mathrm{Fitch}}$ returns ⊤ if $q$ is derivable from $\mathrm{Th}(\mathbf{C})$, and ⊥ if it creates a contradiction, and ? if $q$ is contingent. Thus, all derived constraints are correct, and all entailed constraints are derivable.

[3] Proof sketch: Appendix 1.


Finding Minimal Hitting Sets with Branch & Bound (max size $s = 3$)
Evidence Sets for 6 clauses on 5 examples
$E_{c_0} : \{0,1,2,3,4\}$ $E_{c_3} : \{4\}$
$E_{c_1} : \{3\}$ $E_{c_4} : \{0,2\}$
$E_{c_2} : \{1,2\}$ $E_{c_5} : \{0,3,4\}$
$H := \emptyset$
$M := \{0,1,2,3,4\}$
1 Rank($\mathscr{C}, M$) = $[c_0, c_5, c_2, c_4, c_1, c_3]$
2 $c_0$
Branching
$c_5$
$c_2$ •••
$H := \{c_0\}$
$M := \emptyset$
Learned $\{c_0\}$
$H := \{c_5\}$
$M := \{1,2\}$
Rank($\mathscr{C}, M$) = $[c_2, c_4]$
$H := \{c_2\}$
$M := \{0,3,4\}$
Rank($\mathscr{C}, M$) = $[c_5, c_1, c_3, c_4]$
$c_2$
$c_4$
$c_5$
$c_1$ •••
$H := \{c_5, c_2\}$
$M := \emptyset$
Learned $\{c_5, c_2\}$
3
$H := \{c_5, c_4\}$
$M := \{1\}$
Rank($\mathscr{C}, M$) = $[c_2]$
$H := \{c_2, c_5\}$
$M := \emptyset$
Dedupe $\{c_2, c_5\}$
$H := \{c_2, c_1\}$
$M := \{0,3,4\}$
Rank($\mathscr{C}, M$) = $[c_5, c_3, c_4]$
$c_2$
$c_5$ $c_3$ $c_4$
$H := \{c_5, c_4, c_2\}$
$M := \emptyset$
4 Non-minimal $\{c_5, c_4, c_2\}$
5 Bounding: $s - |H| < \frac{|M|}{\psi}$

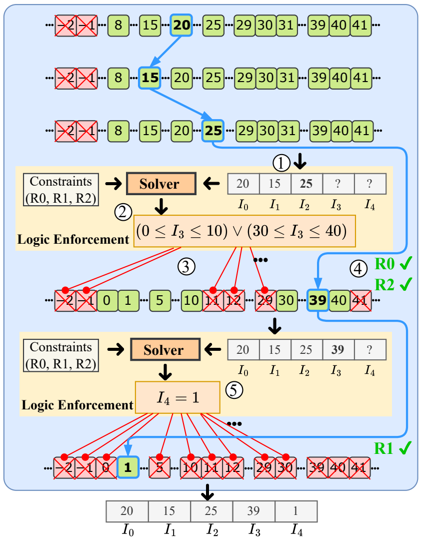


Figure 3: NetNomos learns complex FOL constraints by systematically finding minimal hitting sets.

Figure 4: NetNomos invokes a solver during inference to filter out invalid tokens that will cause rule violations.

## 5 Semantic Filtering

A learned constraint can be syntactically valid but semantically meaningless (for example, R4). This occurs because NetNomos, like all rule-learning methods, reasons about syntax but not meaning. The problem is severe: thousands or even millions of valid formulas may align with the data [4, 30, 47, 55, 82, 83], and the issue grows with grammar expressiveness.

The consequences are twofold: (1) reduced interpretability of the learned rules, and (2) heavy overhead when applying or enforcing them. Although experts can often spot meaningless rules, manually filtering such large sets is infeasible.

NetNomos addresses this challenge using large language models (LLMs).[4] While LLMs can hallucinate, the risk here is bounded: (1) they only filter rules already valid with respect to the data, and (2) they are tasked with semantic reasoning, not generation, which plays to their strength.

The semantic filter (Fig. 2) takes as input the constraints learned by NetNomos. It first interprets raw SMT-LIB formulas into logical expressions with semantic values. For example, `(assert (forall ((e Flow)) (=> (not (= (Proto e) 6)) (= (Flags e) 0))))` becomes "$\forall e : e.$`Proto`$\neq$`TCP`$\Rightarrow e.$`Flags`$=$`0`" (Rule #6, Table 6). This can also be translated into plain English, though experiments (Fig. 7) show no added benefit for filtering. Next, NetNomos queries an external LLM with the interpreted rules and an instruction to identify semantically meaningful ones.[5] The LLM can also be augmented with lightweight tools such as web search or retrieval-augmented generation (RAG) over RFCs, specifications, and related papers. Rules marked as meaningful are kept; the rest are discarded.

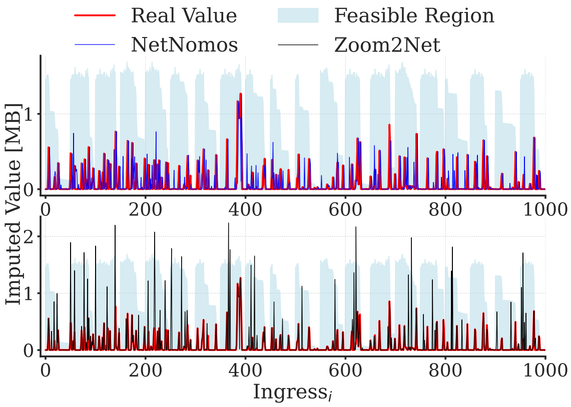


Figure 5: (Upper) NetNomos enforces learned rules at inference time, guaranteeing that model outputs remain within the feasible region defined by constraints. (Lower) In contrast, outputs of Zoom2Net [27] frequently exceed the boundaries.

As shown in Fig. 7, this approach is highly effective. Leveraging the interpretability of NetNomos 's rules, the semantic filter removes meaningless rules with over 98% precision and under 0.73% false positive rate.

## 6 Rule Enforcement

We present NetNomos 's inference-time rule enforcement. The method interleaves with the LM's token-by-token de-

[4] We use LLM to refer to models with billions of parameters trained on massive corpora, while LM denotes any model over discrete vocabulary.

[5] This step involves prompt design, which we do not optimize.

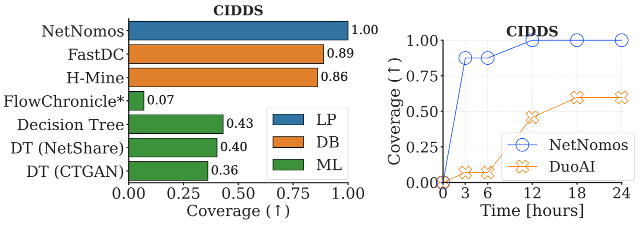
coding and steers generation toward rule-compliant outputs (Fig. 4). An SMT solver adds some latency, yet it lets NetNomos combine neural and symbolic reasoning. Symbolic reasoning enforces diverse constraints, including arithmetic and non-differentiable ones, without manual rule checks by operators. At the same time, NetNomos preserves the benefit of statistical learning by intervening only when needed.

**Example.** Consider imputing $[I_0, \ldots, I_4]$ under constraints R0–R2. After the LM emits a complete value (*e.g.,* $I_2$ at ①), NetNomos calls the solver with all three constraints, instantiated with the values generated so far. This partial instantiation identifies which constraints are active and what they imply for the next symbol. If a rule such as R2 is already satisfied by earlier values, NetNomos deactivates it when computing the feasible region for $I_3$. If not, the solver uses all relevant rules to derive the valid range for $I_3$ (②). NetNomos then prunes any candidate value of $I_3$ that lies outside this region (③). The chosen value (*e.g.,* $I_3 = 39$) is therefore guaranteed to satisfy all constraints (④). With aggregation rules such as R1, the feasible region may collapse to a single value, which the LM then emits (⑤).

**Fine-grained, minimally invasive control.** A key challenge is granularity mismatch: The LM generates opaque tokens from a tokenizer, while the solver reasons over interpretable variables such as ingress bytes or ECN markings. For example, the solver may require $I_4 = 6$ to satisfy R2, while the LM's vocabulary may not contain a standalone token "6". Instead, the digit may appear only inside subword tokens such as "062" or "␣6". This mismatch can force invasive control that harms generation quality [5, 25, 57].

NetNomos *resolves this mismatch with per-field vocabularies and character-level control.* For each variable in the dataset, NetNomos instantiates a field-specific vocabulary. For numerical fields (*e.g.,* ingress volume, packet count), NetNomos treats numbers as plain text [63] and uses character-level tokenization [73], generating digits one by one. For categorical fields (*e.g.,* protocols, ports), NetNomos tokenizes entire values as single units. For example, TCP is never split into T and CP. This design gives NetNomos control that is at least as fine as the solver's variable granularity.

During inference, NetNomos builds token transition systems on the fly. Given solver-derived feasible ranges, it constructs a labeled transition system [14, 75, 76] across variables and an unlabeled one within the digits of a numerical variable. The current state corresponds to the last emitted token, and the next states include all tokens that keep the partial value within the feasible region. Fig. 5 shows this process in action. NetNomos keeps imputed values within the valid (shaded) region at every step, while outputs of Zoom2Net [27] frequently exceed the boundaries.



(a) (b)

Figure 6: (a): NetNomos is much more expressive in capturing network rules than existing SOTA methods from other domains. Theoretically, Γ can express all benchmark rules (§7.1), and its coverage result is bounded by scalability. (b): NetNomos outscales DuoAI, achieving >75% rule coverage in half the time and up to 6.5× better scalability compared to DuoAI. NetNomos enables practical learning of complex real-world network rules from data within a reasonable time budget. (Full results: Fig. 11, 12. Appendix F, G).

## 7 Evaluation

Our evaluation answers the following questions:

E1.1 Can NetNomos learn complex network rules that require sufficient grammatical expressiveness?

E1.2 Is NetNomos more scalable than equally expressive SOTA rule-learning methods?

E2. Can NetNomos reliably filter out syntactically valid but semantically meaningless rules?

E3. Can the same GPT-2 model, without retraining or fine-tuning, achieve on-par performance for distinct tasks (imputation, generation, prediction) with SOTA systems tailored to each task, only by enforcing rules at inference?

Testbed setup for the experiments is reported in Appendix N.

### 7.1 Benchmark Rulesets

**Rule learning rulesets.** To address E1.1 and E1.2, we curate a comprehensive benchmark ruleset for each dataset to compare NetNomos against baseline rule-learning methods. We leverage the benchmark rules from prior papers on the corresponding datasets, as well as from RFCs and IANA standards. As shown in Table 2 in Appendix B, these rulesets together cover all three types of network rules (Fig. 1a). We manually translate the rules into formal logic; example rules and their semantics are presented in Tables 5–7 in Appendix C. For all datasets, we set the formula size limit to 12 for NetNomos.
**Rule filtering rulesets.** To answer E2., we build rulesets that contain both meaningful and meaningless rules for evaluating the semantic rule filter of NetNomos. Meaningful rules are consistent by definition, while not all consistent rules are meaningful (§4.1). We generate meaningless rules from the benchmark rulesets (Table 2). The benchmark rules

are both consistent and meaningful. To create meaningless ones, we apply Semantic Fusion [80]. Specifically, we mix the premises and conclusions of logical implications. For example, given two benchmark rules $X \implies Y$ and $Z \implies Q$, we construct $X \implies Q$ and $Z \implies Y$. We then evaluate these fused rules on the corresponding datasets. The ones that remain consistent with the network data are treated as meaningless rules. This process produces cases such as $\texttt{SrcPt} = 80 \implies \texttt{SrcPt} = 433$, which is consistent but meaningless. Table 3 in Appendix D summarizes the resulting rulesets used for evaluating the semantic filter.

## 7.2 Rule Learning

**Baselines.** We compare NetNomos with representative rule-learning methods from three domains: logic programming (LP), databases (DB), and machine learning (ML). Specifically,, we evaluate LP-based DuoAI [82]; DB-based FastDC [13, 55] and H-Mine [54]; ML-based FlowChronicle [18] and Decision tree (DT). Details are provided in Appendix E.

**Metric.** We evaluate the coverage of all rule-learning methods on the benchmark rulesets (§7.1). We run NetNomos, as well as baselines, for 24 hours on each dataset. We then collect their learned rules **C** and feed them into NetNomos's proof system (Eqn. 1) for evaluation. We then query the proof system with each benchmark rule. A rule is counted as *learned* if the system returns $\top$ (*i.e.,* the benchmark rule is derivable from $\text{Th}(\mathbf{C})$). Otherwise, we consider that rule as not learned, *i.e.,* the result is $?$ or $\perp$. Rule coverage is then computed as: $\text{Coverage} = \frac{\#\text{ of learned rules}}{\text{Total }\#\text{ of rules in benchmark}}$.

**Expressiveness results.** We first evaluate the expressiveness of NetNomos. All baselines, except for DuoAI, are limited in expressiveness; that is, their underlying rule-learning methods cannot capture all types of network rules in the benchmarks. Fig. 6a compares NetNomos against these expressiveness-bounded methods (full results shown in Fig. 11, Appendix F). NetNomos is able to capture nearly all benchmark rules, missing only 2 rules in the Netflix benchmark and 6 rules in MAWI. The grammar $\Gamma$ of NetNomos can, in principle, express all benchmark rules. The few rules not learned are long and complex protocol rules that exceed the current scalability of NetNomos. Improving scalability to cover such rules is an avenue for future work.

**Scalability results.** Fig. 6b depicts the rule coverage and running time for NetNomos and DuoAI (full results in Fig. 12, Appendix G). Both methods are bound by scalability rather than expressiveness. Within the first 12 hours, NetNomos achieves over 75% coverage on all four benchmarks, while DuoAI reaches at most 60% coverage after 24 hours. The advantage of NetNomos is most pronounced on the Netflix and MAWI benchmarks, where constraint sizes are mostly larger than 8: here, NetNomos scales 5.7–6.5× better than DuoAI. In contrast, the CIDDS and MetaDC benchmarks consist mostly of smaller constraints of size less than 5, where the gap is relatively narrower.

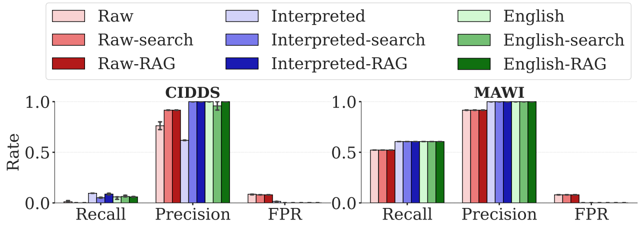


Figure 7: NetNomos's filtering stage can effectively avoid carrying meaningless (albeit consistent) rules to the next stage, achieving a 9.2% FPR. NetNomos also integrates simple interpretation (`Interpreted*`) and lightweight tools with the LLM such as web search or RAG, which leads to >98.1% precision and <0.73% FPR. (Full results: Fig. 13, Appendix I)

## 7.3 Rule Filtering

We use the GPT-4.1 API from OpenAI as the LLM component of NetNomos's semantic filter. The API is invoked with a fixed prompt (shown in Appendix M). Each query appends the rules in one of three forms: `Raw` formulas in SMT-LIB format, `Interpreted` formula expressions, or plain `English` translations.

**Metrics.** We collect the indices of rules that the LLM marks as meaningful. From these results, we compute recall (TP / (TP + FN)), precision (TP / (TP + FP)), and false positive rate (FP / (FP + TN)) with the following definitions:

(TP) True positives: correctly identified meaningful rules.
(FP) False positives: meaningless rules wrongly marked.
(TN) True negatives: correctly identified meaningless rules.

**Results.** Fig. 7 shows the performance of NetNomos's semantic filtering (full results in Fig. 13, Appendix I). Even raw logic formulas (`Raw`) achieve 82.5% precision with only 9.2% FPR. Interpreting the raw SMT-LIB format as logical formulas with semantic values further improves performance: precision rises by 5.9% (from 86.4% to 91.5%), and FPR drops by 633.6% (from 6.9% to 0.82%). We then improve filtering by enabling web search and giving the LLM access to a vector database that indexes relevant documents such as protocol RFCs [7, 8, 16, 17, 19, 33, 58–60, 74, 78] and dataset-related papers (*e.g.,* [11, 39, 66–68]). With this lightweight tool use, the semantic filter reaches at least 98.1% precision and at most 0.73% FPR. Translating formulas into plain English provides no additional gain, which indicates that NetNomos 's interpretable rules are already sufficient for reliable selection. The filtering performance on MetaDC is slightly lower since there is little standard documentation on network principles compared to the abundant protocol specifications.

The main drawback of this approach is recall: even with

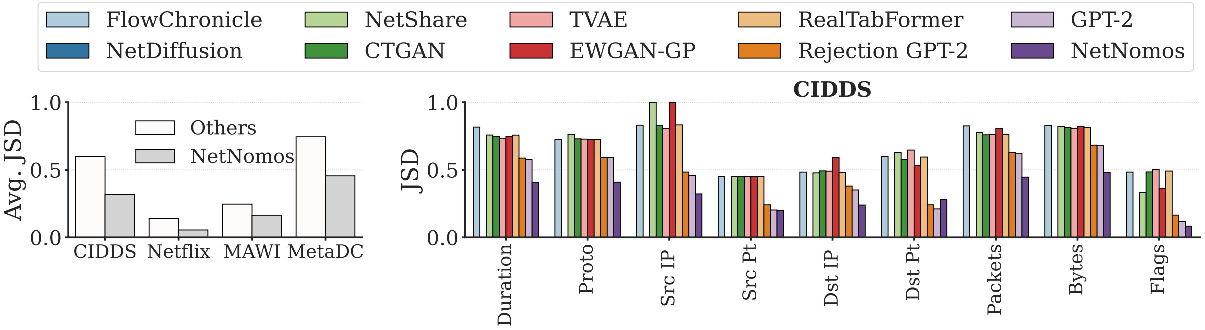


Figure 8: NetNomos generates synthetic data of higher fidelity, achieving both lower JSD and EMD across four datasets compared to NetShare [84], NetDiffusion [40], FlowChronicle [18], and other generative ML models. For the Netflix and MAWI datasets, the EMD values are truncated to improve the visibility of the lower range. Average (`Avg.`) JSD and EMD values are computed *without considering the maximum*. (Full results are reported in Fig. 14, Appendix J.)

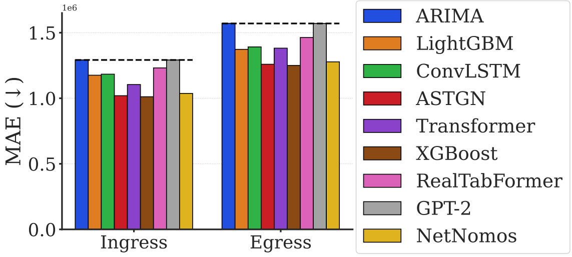


Figure 9: By enforcing learned rules, NetNomos is able to improve the generic GPT-2 (marked by dashed line) by 15.04%–42.67% across various metrics and achieve competitive performance against regression and the specialized model, ASTGN [56]. (All six evaluation metrics are reported in Fig. 15, Appendix K).

interpretation and tool use, recall across all four datasets remains below 76.1%. In practice, the LLM is more likely to reject a rule as meaningless, even when given supporting resources. We argue that this conservative behavior is beneficial. NetNomos often learns thousands of syntactically valid rules (Table 4 in Appendix H). Enforcing all of them during model inference would put excessive strain on the solver and increase latency. High precision in filtering therefore reduces overhead and ensures practical deployment. We acknowledge the recall limitation but improvements in LLM performance are likely to strengthen filtering in the future.

## 8 End-to-end NetNomos Case Studies

For E3., we evaluate NetNomos end-to-end on *(i)* data synthesis (§8.1); *(ii)* traffic forecasting (§8.2); and *(iii)* telemetry imputation (§8.3).

**NetNomos uses GPT-2.** NetNomos is LM-agnostic and can naturally benefit from a more powerful model. Still, to ensure that improvements come from rule learning and enforcement rather than the inherent strength of the LM, we deliberately choose a generic, weaker LM: GPT-2 [62].

**NetNomos tailors the GPT-2 per task by inference rules.** We use the *same* GPT-2 model, trained once on the network data synthesis task, for all three use cases without retraining or fine-tuning. What varies across tasks is the set of rules enforced by NetNomos to contain those that involve task-related variables. For data synthesis, all rules are applied because every field is generated. We acknowledge an approximately 5× inference-time efficiency cost from enforcing NetNomos rules [35], due to the overhead of validating the LM's decisions with an SMT solver. We leave performance optimization to future work.

### 8.1 Network Data Synthesis

**Baselines.** We evaluate NetNomos against three network-specific generators: NetShare [84], NetDiffusion [40], and FlowChronicle [18], as well as four general-purpose tabular data generators: E-WGAN-GP [28], CTGAN [81], TVAE [81], and REaLTabFormer [70].[6] This selection covers a broad range of generative models, including GANs, Diffusion models, Variational Autoencoders, and Transformers. For comparison, we also include vanilla GPT-2 [62] and GPT-2 with rejection sampling. In the latter, the model resamples whenever an inconsistent output is detected, repeating until enough consistent samples are produced.

**Metrics.** For each dataset, we generate 10K samples (flow records for CIDDS, packets for Netflix and MAWI, and measurements for MetaDC). We compare the Jensen–Shannon Divergence (JSD) and Wasserstein/Earth Mover's Distance (EMD) of the synthetic data generated from different frameworks against the distributions of the original data. For both metrics, the lower, the better.

**Fidelity results.** As shown in Fig. 8 (and Fig. 14 in Appendix J), NetNomos achieves on average 1.94× lower JSD

[6] FlowChronicle only supports CIDDS, and NetDiffusion only PCAP.

and 27.9× lower normalized EMD across the four datasets (excluding the maximum in each case). Unlike other frameworks such as NetDiffusion, which perform well on one metric (JSD) but poorly on another (EMD), NetNomos delivers consistently strong performance on both.

**Rule-compliance results.** Fig. 16 in Appendix L shows the evaluation of data generators in terms of violations against our benchmark rules. By enforcing rules during inference, NetNomos guarantees compliance and achieves zero violations on all datasets except MAWI. In contrast, other frameworks exhibit high violation rates, exposing serious breaks in network semantics within their generated data. Critically, network-specific frameworks do not outperform general-purpose models in rule compliance, suggesting that domain knowledge is not well integrated into their designs. Moreover, as shown in Fig. 17 in Appendix L, the CDF of rule violations reveals that more than 50% of the rules are violated by multiple generated samples.

## 8.2 Network Traffic Forecasting

**Baselines.** For traffic forecasting, we compare NetNomos against four traditional regression methods: XGBoost, LightGBM, ARIMA, and ConvLSTM, as well as vanilla Transformer, RealTabFormer [70], vanilla GPT-2 [62], and ASTGN (Attention-based Spatial-Temporal Graph Network) [56]. Note that ASTGN is specifically designed for time-series network traffic forecasting.

**Metrics.** We provide all models with network signals (ECN markings, connection count, and ingress/egress retransmissions) of the past five 50ms-intervals and asking them to predict ingress/egress traffic volume of the next 50ms interval. We evaluate forecasting performance using six metrics: mean absolute error (MAE), root mean squared error (RMSE), normalized scale-free RMSE (NRMSE), symmetric mean absolute percentage error (sMAPE), explained variance (R2), and linear correlation (PearsonR).

**Results.** Fig. 9 shows the performance of nine forecasting methods ranked from best to worst. Without enforcing rules, vanilla GPT-2 ranks near the bottom as generic LMs are not inherently good at handling numbers [1, 49, 61]. (Full results are reported in Fig. 15, Appendix K.) By enforcing learned network rules, NetNomos boosts the vanilla GPT-2 model by 15.04%–42.67% across different metrics, demonstrating knowledge enforcement that can reduce the need for expensive retraining for every task. While NetNomos is not the top performer, it achieves accuracy comparable to the specialized traffic predictor ASTGN [56].

## 8.3 Network Telemetry Imputation

**Methodology.** The goal of this use case is to impute 1ms ingress values given aggregated network signals sampled at 50ms intervals. We compare NetNomos against three baselines: the SOTA telemetry imputer Zoom2Net [27], vanilla GPT-2, and GPT-2 with rejection sampling. For NetNomos, we evaluate two rule sets: `manual` and `learned`. The `manual` rules correspond to the four constraints manually identified by the authors of Zoom2Net (C4–C7 in [27]), while the `learned` rules are automatically learned by NetNomos.

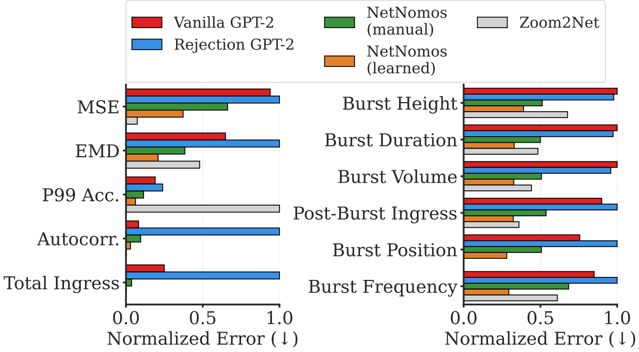


Figure 10: NetNomos improves both imputation accuracy (left) and downstream task performance (right) of the generic GPT-2 via logic enforcement, achieving on-par (and often better) results compared to SOTA Zoom2Net [27].

**Results.** Fig. 10 reports performance both on imputation accuracy and on a downstream task, burst analysis [26]. Enforcing `manual` rules improves GPT-2's accuracy (Fig. 10, left), but still lags behind Zoom2Net. Rejection sampling, by contrast, reduces accuracy: it distorts the LM's distribution by suppressing near-correct outputs and forcing sampling from unrelated regions. With its `learned` rules, NetNomos matches or surpasses Zoom2Net on EMD and p99 accuracy.When guided by NetNomos, GPT-2 outperforms Zoom2Net on all downstream tasks except Burst Position for which Zoom2net uses a specially crafted rule. This demonstrates that automatically learned rules improve the performance of the downstream tasks (not just direct output), even compared to hand-picked rules. The remaining shortfall on time-sensitive metrics (*e.g.,* autocorrelation, Burst Position) likely arises from two factors: GPT-2's general-purpose architecture and the limited temporal expressiveness of learned rules, constrained by the context window of Γ. Developing methods to learn richer temporal constraints is an important direction for future work and could unlock even greater benefits for NetNomos.

# 9 Conclusion

We present NetNomos, the first framework that automatically learns rules from large volumes of network data and enforces them during language model generation to produce compliant outputs. This work takes an important step toward building practical neurosymbolic systems for networking.

# Acknowledgment

We thank the anonymous reviewers for their helpful feedback. We also thank Aarti Gupta for her insightful suggestions and guidance during the revision of the manuscript. This work was supported by the National Science Foundation (NSF) through Grants CNS-2442625, and CNS-2319442.

# References

[1] Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. Large language models for mathematical reasoning: Progresses and challenges. *arXiv preprint arXiv:2402.00157*, 2024.

[2] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[3] David Basin, Nate Foster, Kenneth L. McMillan, Kedar S. Namjoshi, Cristina Nita-Rotaru, Jonathan M. Smith, Pamela Zave, and Lenore D. Zuck. It takes a village: Bridging the gaps between current and formal specifications for protocols. *Commun. ACM*, 68(8):50–61, July 2025.

[4] Matan Ben-Tov, Daniel Deutch, Nave Frost, and Mahmood Sharif. Cafa: Cost-aware, feasible attacks with database constraints against neural tabular classifiers. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 1345–1364. IEEE, 2024.

[5] Luca Beurer-Kellner, Marc Fischer, and Martin T. Vechev. Guiding llms the right way: Fast, non-invasive constrained generation. *International Conference on Machine Learning (ICML)*, abs/2403.06988, 2024.

[6] George S. Boolos, John P. Burgess, and Richard C. Jeffrey. *Computability and Logic*. Cambridge University Press, Cambridge, UK, 5 edition, 2007.

[7] E. Borman, B. Braden, V. Jacobson, and R. Scheffenegger. Tcp extensions for high performance. RFC 7323, September 2014.

[8] R. Braden. Requirements for internet hosts – communication layers. RFC 1122, October 1989.

[9] Aaron R Bradley and Zohar Manna. *The calculus of computation: decision procedures with applications to verification*. Springer, 2007.

[10] Francesco Bronzino, Paul Schmitt, Sara Ayoubi, Guilherme Martins, Renata Teixeira, and Nick Feamster. Inferring streaming video quality from encrypted traffic: Practical models and deployment experience. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(3):1–25, 2019.

[11] Kenji Cho, Koushirou Mitsuya, and Akira Kato. The mawi working group traffic archive. In *Proceedings of the 2000 USENIX Annual Technical Conference (FREENIX Track)*, pages 263–270, 2000.

[12] Andrew Chu, Xi Jiang, Shinan Liu, Arjun Bhagoji, Francesco Bronzino, Paul Schmitt, and Nick Feamster. Netssm: Multi-flow and state-aware network trace generation using state-space models. *arXiv preprint arXiv:2503.22663*, 2025.

[13] Xu Chu, Ihab F Ilyas, and Paolo Papotti. Discovering denial constraints. *Proceedings of the VLDB Endowment*, 6(13):1498–1509, 2013.

[14] Andreas Classen, Maxime Cordy, Pierre-Yves Schobbens, Patrick Heymans, Axel Legay, and Jean-François Raskin. Featured transition systems: Foundations for verifying variability-intensive systems and their application to ltl model checking. *IEEE Transactions on Software Engineering*, 39(8):1069–1089, 2012.

[15] David Cohen and Peter Jeavons. The complexity of constraint languages. In *Foundations of Artificial Intelligence*, volume 2, pages 245–280. Elsevier, 2006.

[16] A. Conta, S. Deering, and M. Gupta. Internet control message protocol (icmpv6) for the internet protocol version 6 (ipv6) specification. RFC 4443, March 2006.

[17] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire. Internet assigned numbers authority (iana) procedures for the management of the service name and transport protocol port number registry. RFC 6335, August 2011.

[18] Joscha Cüppers, Adrien Schoen, Gregory Blanc, and Pierre-Francois Gimenez. Flowchronicle: Synthetic network flow generation through pattern set mining. *Proceedings of the ACM on Networking*, 2(CoNEXT4):1–20, 2024.

[19] S. Deering and R. Hinden. Internet protocol, version 6 (ipv6) specification. RFC 8200, July 2017.

[20] Nachum Dershowitz, Ziyad Hanna, and Alexander Nadel. A scalable algorithm for minimal unsatisfiable core extraction. In *Theory and Applications of Satisfiability Testing-SAT 2006: 9th International Conference, Seattle, WA, USA, August 12-15, 2006. Proceedings 9*, pages 36–41. Springer, 2006.

[21] Pedro Domingos. *The master algorithm: How the quest for the ultimate learning machine will remake our world*. Basic Books, 2015.

[22] Andrew Ferraiuolo, Mark Zhao, Andrew C Myers, and G Edward Suh. Hyperflow: A processor architecture for nonmalleable, timing-safe information flow security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1583–1600, 2018.

[23] Frederic B Fitch. A logical analysis of some value concepts1. *The journal of symbolic logic*, 28(2):135–142, 1963.

[24] Aaron Gember-Jacobson, Wenfei Wu, Xiujun Li, Aditya Akella, and Ratul Mahajan. Management plane analytics. In *Proceedings of the 2015 Internet Measurement Conference*, pages 395–408, 2015.

[25] Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured nlp tasks without finetuning. *arXiv preprint arXiv:2305.13971*, 2023.

[26] Ehab Ghabashneh, Yimeng Zhao, Cristian Lumezanu, Neil Spring, Srikanth Sundaresan, and Sanjay Rao. A microscopic view of bursts, buffer contention, and loss in data centers. In *Proceedings of the 22nd ACM Internet Measurement Conference*, pages 567–580, 2022.

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

[28] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

[29] Satyandra Guthula, Roman Beltiukov, Navya Battula, Wenbo Guo, and Arpit Gupta. netfound: Foundation model for network security. *arXiv preprint arXiv:2310.17025*, 2023.

[30] Travis Hance, Marijn Heule, Ruben Martins, and Bryan Parno. 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)*, pages 115–131, 2021.

[31] Juris Hartmanis. Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson). *Siam Review*, 24(1):90, 1982.

[32] Hongyu Hè. Netnomos code repository. https://github.com/HongyuHe/NetNomos.

[33] R. Hinden and G. Fairhurst. Ipv6 hop-by-hop options processing procedures. RFC 9673, October 2024.

[34] Kevin Hsieh, Mike Wong, Santiago Segarra, Sathiya Kumaran Mani, Trevor Eberl, Anatoliy Panasyuk, Ravi Netravali, Ranveer Chandra, and Srikanth Kandula. {NetVigil}: Robust and {Low-Cost} anomaly detection for {East-West} data center security. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1771–1789, 2024.

[35] Hongyu Hè and Maria Apostolaki. 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)*, 2025.

[36] Alexey Ignatiev, Alessandro Previti, Mark Liffiton, and Joao Marques-Silva. Smallest mus extraction with minimal hitting set dualization. In *International Conference on Principles and Practice of Constraint Programming*, pages 173–182. Springer, 2015.

[37] Internet Assigned Numbers Authority. Assigned internet protocol numbers. https://www.iana.org/assignments/protocol-numbers/, 2025. Last Updated: 2025-07-11.

[38] Internet Assigned Numbers Authority. Service name and transport protocol port number registry. https://www.iana.org/assignments/service-names-port-numbers/, 2025. Last Updated: 2025-08-26.

[39] Arthur S. Jacobs, Roman Beltiukov, Walter Willinger, Ronaldo A. Ferreira, Arpit Gupta, and Lisandro Z. Granville. AI/ML for Network Security: The Emperor has no Clothes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22)*, Los Angeles, CA, USA, 2022. ACM.

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

[41] Minhao Jin and Maria Apostolaki. Robustifying ml-powered network classifiers with pants. In *34th USENIX Security Symposium (USENIX Security 25)*, 2025.

[42] Minhao Jin, Hongyu He, and Maria Apostolaki. Assessing user privacy leakage in synthetic packet traces: An attack-grounded approach. *arXiv preprint arXiv:2508.11742*, 2025.

[43] Jason R Koenig, Oded Padon, Neil Immerman, and Alex Aiken. First-order quantified separators. In *Proceedings*

*of the 41st ACM SIGPLAN conference on programming language design and implementation*, pages 703–717, 2020.

[44] Roger J Lewis et al. An introduction to classification and regression tree (cart) analysis. In *Annual meeting of the society for academic emergency medicine in San Francisco, California*, volume 14, page 106. Department of Emergency Medicine Harbor-UCLA Medical Center Torrance San . . . , 2000.

[45] Ruihan Li, Fangdan Ye, Yifei Yuan, Ruizhen Yang, Bingchuan Tian, Tianchen Guo, Hao Wu, Xiaobo Zhu, Zhongyu Guan, Qing Ma, et al. Reasoning about network traffic load property at production scale. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1063–1082, 2024.

[46] Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. Generating high-fidelity, synthetic time series datasets with doppelganger. *arXiv preprint arXiv:1909.13403*, 2019.

[47] Haojun Ma, Aman Goel, Jean-Baptiste Jeannin, Manos Kapritsos, Baris Kasikci, and Karem A Sakallah. I4: incremental inference of inductive invariants for verification of distributed protocols. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 370–384, 2019.

[48] Qian Ma, Ashwin Bharambe, Mor Harchol-Balter, and Alan Scheller-Wolf. Neural networks for modeling netflix's dynamic pricing system. In *Proceedings of the 2017 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 1–13, 2017.

[49] Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024.

[50] Alexander Nadel. Boosting minimal unsatisfiable core extraction. In *Formal methods in computer aided design*, pages 221–229. IEEE, 2010.

[51] Oded Padon, Kenneth L McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. Ivy: safety verification by interactive generalization. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 614–630, 2016.

[52] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proceedings of the VLDB Endowment*, 8(10):1082–1093, 2015.

[53] Thorsten Papenbrock and Felix Naumann. A hybrid approach to functional dependency discovery. In *proceedings of the 2016 International Conference on Management of Data*, pages 821–833, 2016.

[54] Jian Pei, Jiawei Han, Hongjun Lu, Shojiro Nishio, Shiwei Tang, and Dongqing Yang. H-mine: Fast and space-preserving frequent pattern mining in large databases. *IIE transactions*, 39(6):593–605, 2007.

[55] Eduardo HM Pena, Fabio Porto, and Felix Naumann. Fast algorithms for denial constraint discovery. *Proceedings of the VLDB Endowment*, 16(4):684–696, 2022.

[56] Yufei Peng, Yingya Guo, Run Hao, and Chengzhe Xu. Network traffic prediction with attention-based spatial–temporal graph network. *Computer Networks*, 243:110296, 2024.

[57] Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. Synchromesh: Reliable code generation from pre-trained language models. *arXiv preprint arXiv:2201.11227*, 2022.

[58] Jon Postel. User datagram protocol. RFC 768, August 1980.

[59] Jon Postel. Internet protocol. RFC 791, September 1981.

[60] Jon Postel. Transmission control protocol. RFC 793, September 1981.

[61] Jing Qian, Hong Wang, Zekun Li, Shiyang Li, and Xifeng Yan. Limitations of language models in arithmetic and symbolic induction. *arXiv preprint arXiv:2208.05051*, 2022.

[62] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[63] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[64] Frank P Ramsey. On a problem of formal logic. In *Classic Papers in Combinatorics*, pages 1–24. Springer, 1987.

[65] Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987.

[66] Markus Ring, Daniel Schlör, Dieter Landes, and Andreas Hotho. Flow-based network traffic generation using generative adversarial networks. *Computers & Security*, 82:156–172, 2019.

[67] Markus Ring, Sarah Wunderlich, Dominik Grüdl, Dieter Landes, and Andreas Hotho. Creation of flow-based data sets for intrusion detection. *Journal of Information Warfare*, 16:40–53, 2017.

[68] Adrien Schoen, Gregory Blanc, Pierre-François Gimenez, Yufei Han, Frédéric Majorczyk, and Ludovic Me. A tale of two methods: Unveiling the limitations of gan and the rise of bayesian networks for synthetic network traffic generation. In *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 273–286. IEEE, 2024.

[69] Nirhoshan Sivaroopan, Kaushitha Silva, Chamara Madarasingha, Thilini Dahanayaka, Guillaume Jourjon, Anura Jayasumana, and Kanchana Thilakarathna. A comprehensive survey on network traffic synthesis: From statistical models to deep learning. *arXiv preprint arXiv:2507.01976*, 2025.

[70] Aivin V. Solatorio and Olivier Dupriez. Realtabformer: Generating realistic relational and tabular data using transformers. *arXiv preprint arXiv:2302.02041*, 2023.

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

[72] Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung-yi Lee, and Yun-Nung Chen. Let me speak freely? a study on the impact of format restrictions on performance of large language models. *arXiv preprint arXiv:2408.02442*, 2024.

[73] Yi Tay, Vinh Q Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. Charformer: Fast character transformers via gradient-based subword tokenization. *arXiv preprint arXiv:2106.12672*, 2021.

[74] J. Touch. Recommendations on using assigned transport port numbers. RFC 7605, August 2015.

[75] Jan Tretmans. Model based testing with labelled transition systems. In *Formal Methods and Testing: An Outcome of the FORTEST Network, Revised Selected Papers*, pages 1–38. Springer, 2008.

[76] Johan Van Benthem and Jan Bergstra. Logic of transition systems. *Journal of Logic, Language and Information*, 3:247–283, 1994.

[77] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[78] Ed. W. Eddy. Transmission control protocol (tcp). RFC 9293, August 2022.

[79] Ke Wang, Liu Tang, Jiawei Han, and Junqiang Liu. Top down fp-growth for association rule mining. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 334–340. Springer, 2002.

[80] Dominik Winterer, Chengyu Zhang, and Zhendong Su. Validating smt solvers via semantic fusion. In *Proceedings of the 41st ACM SIGPLAN Conference on programming language design and implementation*, pages 718–730, 2020.

[81] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional gan. *Advances in neural information processing systems*, 32, 2019.

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

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

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

[85] Minlan Yu, Lavanya Jose, and Rui Miao. Software {Defined}{Traffic} measurement with {OpenSketch}. In *10th USENIX symposium on networked systems design and implementation (NSDI 13)*, pages 29–42, 2013.

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

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

[88] Tony Nuda Zhang, Keshav Singh, Tej Chajed, Manos Kapritsos, and Bryan Parno. Basilisk: Using provenance invariants to automate proofs of undecidable protocols. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, pages 1–17, 2025.

[89] Ruiwen Zhou, Wenyue Hua, Liangming Pan, Sitao Cheng, Xiaobao Wu, En Yu, and William Yang Wang. Rulearena: A benchmark for rule-guided reasoning with llms in real-world scenarios. *arXiv preprint arXiv:2412.08972*, 2024.

## A Proof for Theorem 1

We construct a proof using a modified formulation of the reduction from clause learning to finding minimal set covers [13]. Let $\mathcal{C}$ denote the set of propositional clauses derived from the grammar $\Gamma$. For each clause $c \in \mathcal{C}$, define its *evidence set*

$$E_c := \{e \in D \mid e \models c\}.$$

A constraint of disjuncts is a finite set of clauses $C \subseteq \mathcal{C}$ interpreted as $\bigvee_{c \in C} c$. By construction,

$$C \text{ is consistent with } D \iff \bigcup_{c \in C} E_c = D.$$

$\Longrightarrow$ Suppose $C \subseteq \mathcal{C}$ is consistent. Then by the equivalence above, $\bigcup_{c \in C} E_c = D$. Hence $C$ itself is a hitting set of clauses that covers $D$.

$\Longleftarrow$ Conversely, suppose $H \subseteq \mathcal{C}$ is a hitting set: $\bigcup_{c \in H} E_c = D$. Consider the constraint $C_H = \bigvee_{c \in H} c$. For any $e \in D$, since $e \in \bigcup_{c \in H} E_c$, there exists $c \in H$ with $e \models c$. Thus $e \models C_H$, so $C_H$ is consistent with $D$.

Therefore, learning a consistent constraint is equivalent to finding a hitting set of clauses covering $D$. □

## B Network Dataset and Corresponding Benchmark Rulesets

Table 2 describes the datasets used for evaluation and the corresponding benchmark rulesets.

## C Samples of Benchmark Rules

Tables 5–7 provide samples of the network rules and their corresponding semantics.

## D Evaluation Rulesets for Evaluating Semantic Filtering

Table 3 summarizes the rulesets used for evaluating semantic filtering.

| Dataset | Format | Benchmark Ruleset | Description |
|---|---|---|---|
| CIDDS [66–68] | NetFlow | Total rules: 72<br>• Deployment: 30<br>• Protocol: 42<br>• Principle: 0 | Normal and attack traffic collected from an enterprise network. |
| Netflix [10] | PCAP | Total rules: 33<br>• Deployment: 0<br>• Protocol: 33<br>• Principle: 0 | Video streaming traffic with typical client-server interactions. |
| MAWI [11] | PCAP | Total rules: 32<br>• Deployment: 0<br>• Protocol: 32<br>• Principle: 0 | Backbone Internet traffic traces collected at a trans-Pacific link. |
| MetaDC [26] | Datacenter logs | Total rules: 10<br>• Deployment: 0<br>• Protocol: 0<br>• Principle: 10 | Millisecond-scale traffic traces from Meta's datacenters, capturing burstiness, buffer contention, and packet loss. |

Table 2: Datasets used together with statistics of their corresponding benchmark rulesets for evaluation. NETNOMOS generalizes across diverse real-world network datasets.

| Dataset | Meaningful Rules | Meaningless Rules |
|---|---|---|
| CIDDS | 72 | 176 |
| Netflix | 33 | 82 |
| MAWI | 35 | 24 |
| MetaDC | 10 | 18 |

Table 3: Rulesets used for evaluating semantic rule-filtering.

## E Baselines for Rule Learning

We compare NETNOMOS to five rule-learning methods from three domains, namely, logic programming (LP), database (DB), and machine learning (ML):

- (LP) DuoAI [82]: a SOTA method for learning invariants of distributed protocols. It constructs a minimal implication graph and enumerates the formula space, starting from the strongest and weakening iteratively. Because we do not assume the existence of formal models of the networks of the corresponding datasets, we omit DuoAI's post-learning refinement with IVy [51] and only compare against its method for learning candidate invariants.

- (DB) FastDC [13, 55]: a SOTA method for discovering denial constraints (DCs) in databases. DCs are the *most expressive* integrity constraints. We negate learned DCs to derive positive rules, since a DC specifies what must not happen.

- (DB) H-Mine [54]: a SOTA association rule mining method that learns "if-then" implication patterns between variables in large datasets. To apply it to network data, we encode records as boolean variable-value transactions. It does not support numerical variables.[7]

[7] Recent work on numerical association rule learning [71] did not yield

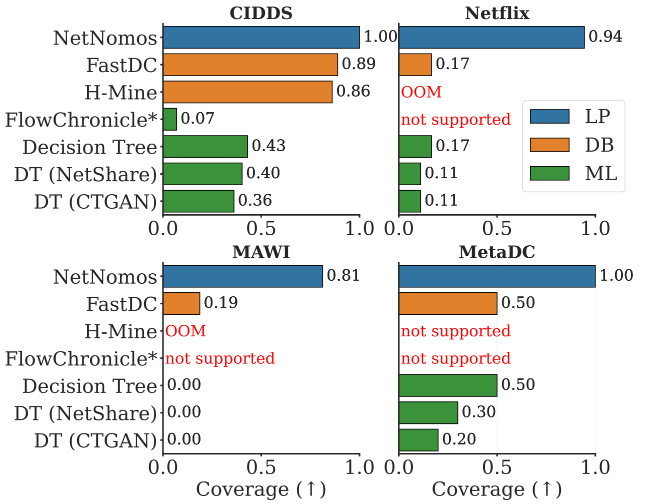


Figure 11: Expressiveness evaluated under four datasets. NetNomos is much more expressive than other existing works.

- (ML) FlowChronicle [18]: a SOTA network data generator that employs sequential pattern mining and defines a constraint language for network rules. Its implementation is tailored to the CIDDS dataset, and adapting it to other datasets would require substantial changes. Thus, we only evaluate it on CIDDS.
- (ML) Decision tree (DT): Decision paths in DTs correspond to linear combinations of propositions [21]. We train a CART-style DT [44] on each dataset and extract their decision paths as logical implications.
- (ML) DTs from GANs: We train DTs as student models on the decisions made by discriminators of NetShare [84] and CTGAN [81]. We then extract decision paths from these student models as learned rules.

## F Expressiveness of Rule Learning

Fig. 11 shows the full results of NetNomos's expressiveness evaluated on four network datasets.

## G Scalability of Rule Learning

Fig. 12 shows the full results of NetNomos's learning scalability evaluated on four network datasets.

## H Rule Filtering Rate

Table 4 describes the filtering rate when using a semantic filter against the learned rules associated with the data sets.

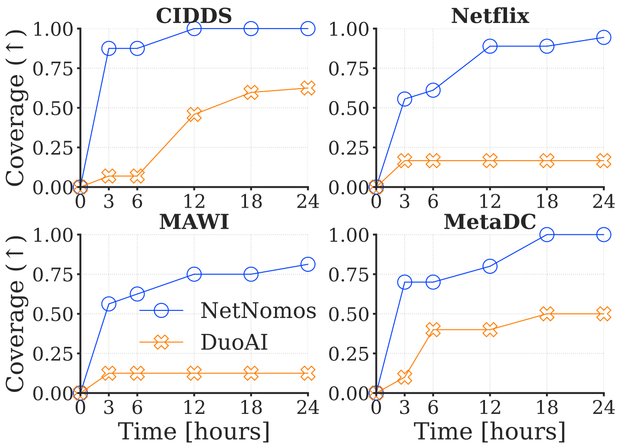


Figure 12: NetNomos outscales DuoAI under four evaluated datasets.

| Dataset | Learned Rules | Filtered Rules | Filtering Rate |
|---|---|---|---|
| CIDDS | 1,203 | 1,150 | 0.96 |
| Netflix | 11,479 | 10,098 | 0.88 |
| MAWI | 50,989 | 44,859 | 0.88 |
| MetaDC | 5,934 | 5,322 | 0.91 |

Table 4: Number of rules learned by NetNomos after 24h and number of rules filtered out by its semantic filter.

## I LLM Filtering Result

Fig. 13 describes the performance of the LLM filtering under four evaluated datasets.

## J Synthetic Data Fidelity

Fig. 14 shows the performance of various data generators across datasets.

## K Traffic Forecasting Performance

Fig. 15 shows traffic forecasting performance evaluated on the MetaDC dataset [26] with six metrics.

## L Rule Compliance of ML-Generated Data

Fig. 16 describes rule violation rates of various data generators. Fig. 17 is the CDF of the number of violating samples per rule.

## M Prompt Used for Semantic Filtering

We attach the prompt template used for semantic filtering below.

meaningful rules, so we omit its results.

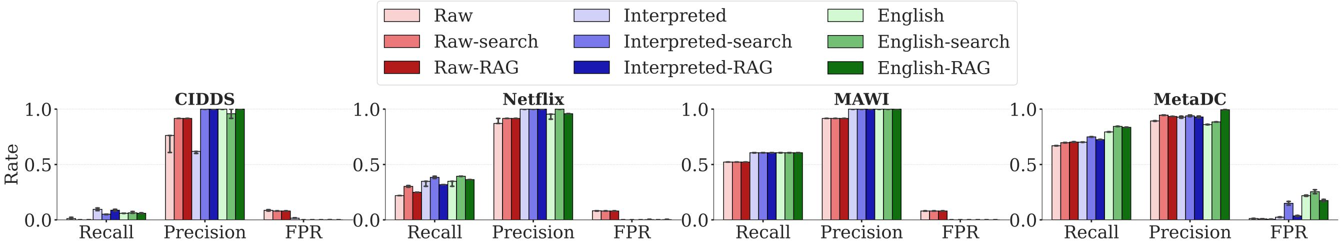


Figure 13: Similar to the Fig. 7, the use of LLM in NETNOMOS is able to filter out meaningless rules when testing under four different datasets.

You are given a list of logical rules extracted from network data.
These rules aim to describe relationships between fields in network data.
##Task
Identify which rules are semantically meaningful. A rule is meaningful only if it reflects real network behavior or expresses a sound integrity constraint.
##Sources of rules
1. Protocol specifications (e.g., RFCs) describing standard behaviors such as port assignments and TCP handshakes.
2. Deployment patterns specific to certain datasets such as no occurrence of public-to-public IP flows.
3. Principles such as queueing, congestion, bandwidth limits, etc.
##Output format
Return a Python-style list of the rule IDs that are consistent. Example: [0, 2, 5]
Do not return anything else and be critical.
##Rules: {rules}

## N Testbed

For fair evaluation, we conduct all the experiments on a two-socket server with 40 logical CPUs of Intel Xeon E5-2660v3 with a frequency of 2.60GHz and 256GiB of DRAM. Training and inference of ML models are all conducted on an A100 NVIDIA GPU.

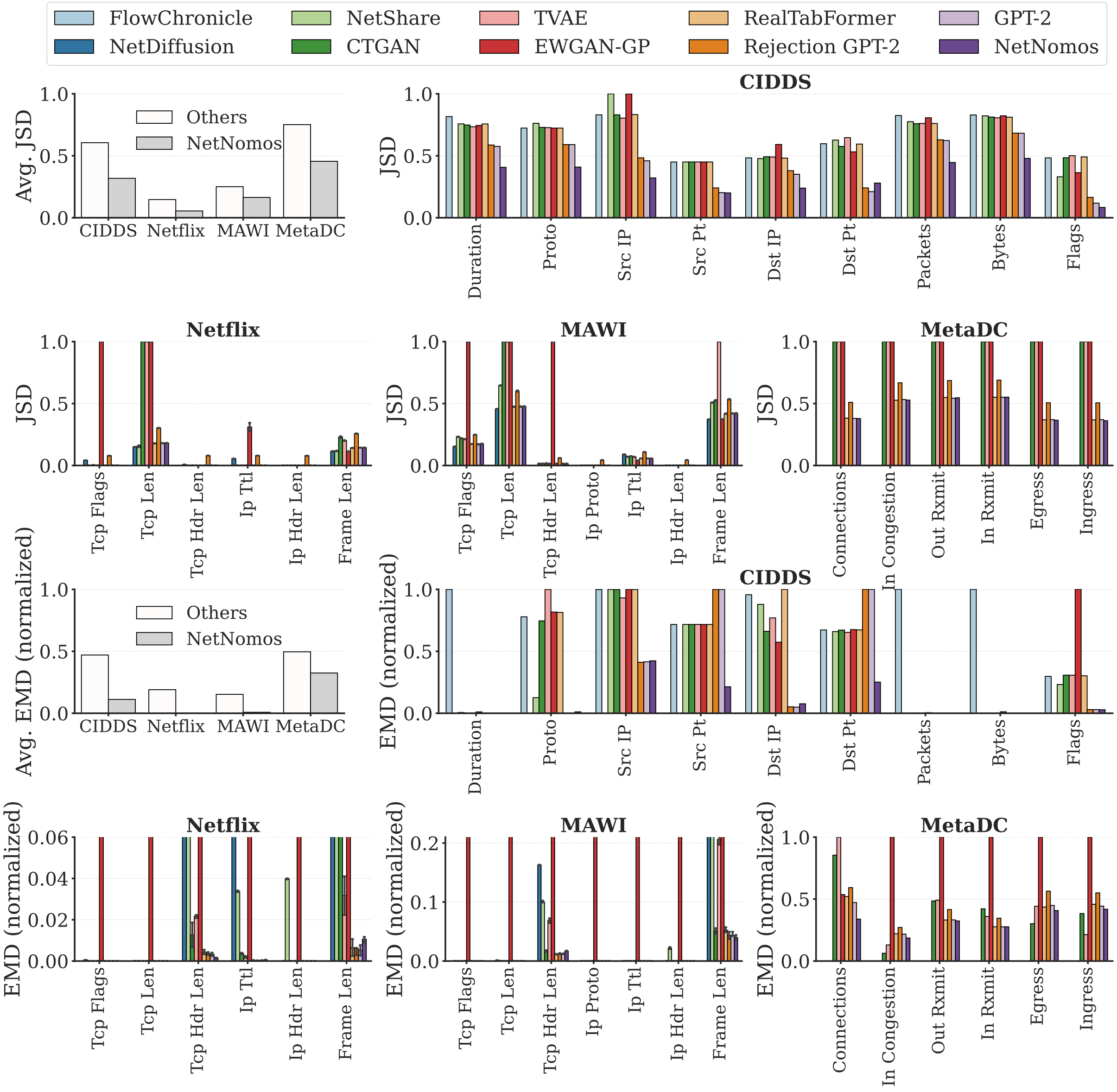


Figure 14: By imposing learned network rules, NetNomos generates high-fidelity synthetic data across diverse datasets compared to SOTA data generators.

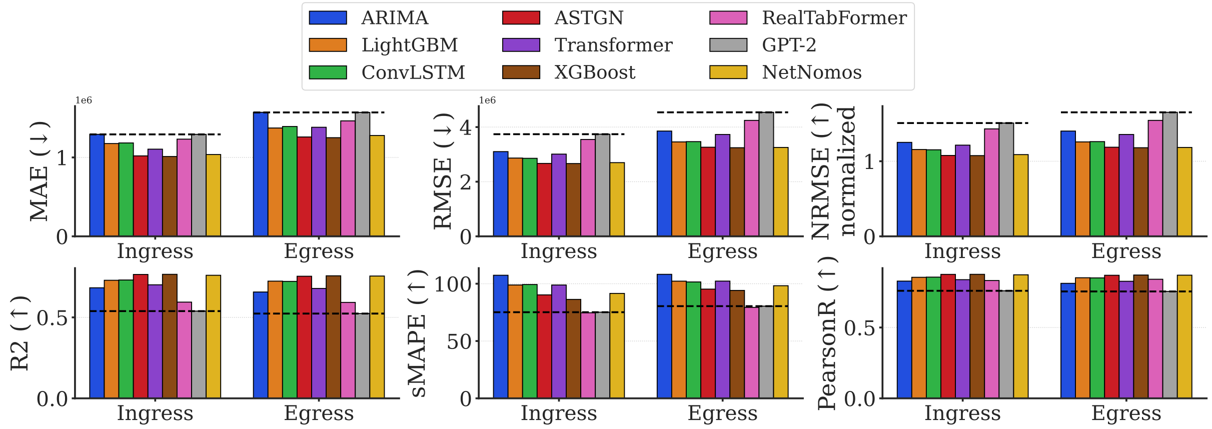


Figure 15: By enforcing learned rules, NETNOMOS improves generic GPT-2 (marked by dashed line) on traffic forecasting by 15.04%–42.67% across six metrics and achieve competitive performance against regression and specialized model, ASTGN [56].

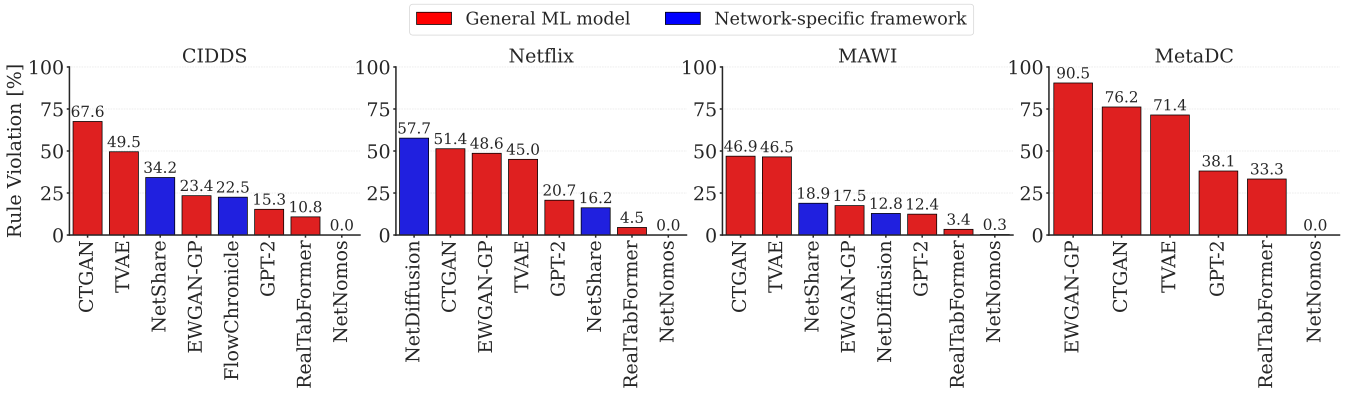


Figure 16: NETNOMOS ensures compliance with the learned rules by enforcing them during inference, achieving zero violations on all datasets except MAWI. In contrast, other frameworks show high violation rates, revealing severe infringements of network semantics in the generated data. Notably, *network-specific frameworks do not display a clear advantage over general models in preserving network semantics, suggesting insufficient integration of domain knowledge.*

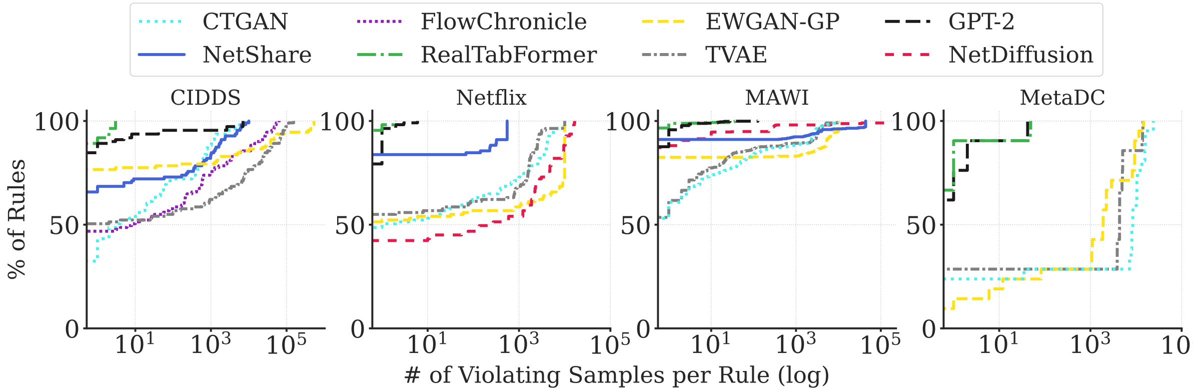


Figure 17: More than 50% of the rules are violated by multiple samples from the synthetic data generators. The presence of long tails further indicate limited diversity, as many samples tend to violate the same subset of rules. For instance, on the MetaDC dataset, GPT-2 and RealTabFormer generate considerably more diverse samples than the other models.

Table 5: Sample PCAP constraints from our Netflix and MAWI benchmark rulesets. SameBiFlow(·) and SameDir(·) are shorthand notations for flow matching.

| Network Constraint | Meaning | Expressible by | Reference |
|---|---|---|---|
| 1. $\forall t, p$ : $\text{SameBiFlow}(p_t, p_{t+1}, p_{t+2}) \wedge \mathit{SYN} \in p_t.\texttt{Flags} \wedge \mathit{SYN\text{-}ACK} \in p_{t+1}.\texttt{Flags} \wedge \mathit{ACK} \in p_{t+2}.\texttt{Flags} \implies p_{t+2}.\texttt{Seq} = p_{t+1}.\texttt{Ack} \wedge p_{t+1}.\texttt{Ack} = (p_t.\texttt{Seq} + 1) \wedge p_{t+2}.\texttt{Ack} = (p_{t+1}.\texttt{Seq} + 1)$ | **[Protocol]** TCP three-way handshake. | NetNomos | [40], [48], RFC 9293 [78], 1122 [8] |
| 2. $\forall t, p : p.\texttt{IpVersion} = 4 \vee p.\texttt{IpVersion} = 6$ | **[Protocol]** Correct IP version number. | NetNomos, H-Mine [54], FastDC [13, 55], Decision Trees | IANA [37], RFC 791 [59], 4443 [16], 8200 [19], 9293 [78] |
| 3. $\forall t, p$ : $\text{SameBiFlow}(p_t, p_{t+1}) \wedge \mathit{PSH\text{-}ACK} \in p_t.\texttt{Flags} \implies \mathit{ACK} \in p_{t+1}.\texttt{Flags} \vee \mathit{RST} \in p_{t+1}.\texttt{Flags}$ | **[Protocol]** PSH data packet followed by ACK. | NetNomos | RFC 1122 [8], 9293 [78] |
| 4. $\forall t, p$ : $\text{SameBiFlow}(p_t, p_{t+1}, p_{t+2}) \wedge \mathit{SYN} \in p_t.\texttt{Flags} \wedge \mathit{SYN\text{-}ACK} \implies p_t.\texttt{TcpWinSize} > 0 \wedge p_{t+2}.\texttt{TcpWinSize} > 0$ | **[Protocol]** TCP rwnd negotiation. | NetNomos | [48], RFC 1122 [8], 9293 [78], 7323 [7] |
| 5. $\forall t, p : p.\texttt{TcpUrgPointer} > 0 \iff \mathit{URG} \in p.\texttt{Flags} \vee \mathit{RST} \in p.\texttt{Flags}$ | **[Protocol]** Urgent pointer is set if only if URG flag bit is active. | NetNomos, FastDC [13, 55], Decision Trees | RFC 9293 [78] |
| 6. $\forall t, p$ : $p.\texttt{IpHdrLen}\%4 = 0 \wedge p.\texttt{TcpHdrLen}\%4 = 0$ | **[Protocol]** Header length alignment. | NetNomos | [11], RFC 791 [59], 9293 [78], 1122 [8], 8200 [19], 9673 [33] |
| 7. $\forall t, p$ : $\text{SameDir}(p_t, p_{t+1}) \wedge \mathit{SYN} \not\in \{p_t.\texttt{Flags} \cup p_{t+1}.\texttt{Flags}\} \wedge \mathit{FIN} \not\in \{p_t.\texttt{Flags} \cup p_{t+1}.\texttt{Flags}\} \implies p_{t+1}.\texttt{Seq} = p_t.\texttt{TcpLen} + p_t.\texttt{Seq}$ | **[Protocol]** TCP sequence number continuity. | NetNomos | [12], [40], RFC 1122 [8], 9293 [78] |
| 8. $\forall t, p : 0 \leq p.\texttt{IpTtl} \leq 255$ | **[Protocol]** Valid TTL number. | NetNomos | RFC 791 [59], 1122 [8], 8200 [19] |

Table 6: Sample NetFlow constraints from our CIDDS benchmark ruleset.

| **Network Constraint** | **Meaning** | **Expressible by** | **Reference** |
|---|---|---|---|
| 1. $\forall e : e.\texttt{SrcIp} \not\in \{\mathit{Multicast} \cup \mathit{Broadcast}\} \wedge e.\texttt{DstIp} \neq \texttt{0.0.0.0}$ | **[Protocol]** Multicast or broadcast IPs can only appear as destination IP addresses, and wildcard mask can only be used in source IP. | NetNomos | RFC 1122 [8], 791 [59], 4443 [16], 8200 [19] |
| 2. $\forall e : e.\texttt{DstPt} \in \{80, 443\} \implies \texttt{SrcIp} \in \mathit{Private}$ | **[Deployment]** Web traffic only originates from within internal network. | NetNomos,<br>H-Mine [54],<br>FastDC [13, 55],<br>Decision Trees | [39, 66, 68] |
| 3. $\forall e : e.\texttt{DstIp} \in \text{DNS} \implies e.\texttt{DstPt} = 53$ | **[Protocol]** DNS service employs port 53 by IANA convention. | NetNomos,<br>H-Mine [54],<br>FastDC [13, 55],<br>Decision Trees,<br>FlowChronicle [18] | RFC 768 [58], 7605 [74], IANA [38] |
| 4. $\forall e : e.\texttt{Bytes} \leq 65535 \times e.\texttt{Packets}$ | **[Protocol]** Total amount of transmitted data in a flow is bounded by the number of packets and MTU. | NetNomos | RFC 1122 [8], 791 [59], 9293 [78], 4443 [16] |
| 5. $\forall e : e.\texttt{DstPt} \in \{137, 138\} \implies e.\texttt{SrcIp} \in \mathit{Private} \wedge e.\texttt{DstIp} \in \mathit{Broadcast}$ | **[Deployment]** NetBIOS flows contain only broadcast packets and use ports 137/138. | NetNomos,<br>H-Mine [54],<br>FastDC [13, 55],<br>Decision Trees | [66, 68] |
| 6. $\forall e : e.\texttt{Proto} \neq \text{TCP} \implies e.\texttt{Flags} = \emptyset$ | **[Protocol]** Non-TCP flow records do not contain flag information. | NetNomos,<br>H-Mine [54],<br>FastDC [13, 55],<br>Decision Trees | RFC 9293 [78], 768 [58], 791 [59] |
| 7. $\forall e : e.\texttt{DstIp} \in \mathit{Public} \iff e.\texttt{SrcIp} \in \mathit{Private}$ | **[Deployment]** No public-to-public traffic, since the vantage point was located behind the gateway. | NetNomos,<br>H-Mine [54],<br>FastDC [13, 55],<br>Decision Trees | [39, 66, 68] |
| 8. $\forall e : e.\texttt{Proto} = \mathit{UDP} \implies e.\texttt{Bytes} \geq 8 \times e.\texttt{Packets}$ | **[Protocol]** A UDP flow entry contains at least one packet. | NetNomos | RFC 1122 [8] |

Table 7: Sample constraints from our MetaDC benchmark rulesets.

| Network Constraint | Meaning | Expressible by | Reference |
|---|---|---|---|
| 1. $\forall e : e.\texttt{Ingress} \geq e.\texttt{InRxmit} \wedge e.\texttt{Egress} \geq e.\texttt{OutRxmit}$ | **[Principle]** Retransmitted traffic is contained in the total traffic volume. | NETNOMOS,<br>FastDC [13, 55], | ——— |
| 2. $\forall t,e : e.\texttt{Congestion}_t > 0 \implies e.\texttt{Ingress}_t > 0$ | **[Principle]** Congestion markings indicate ingress traffic. | NETNOMOS,<br>FastDC [13, 55],<br>Decision Trees | ——— |
| 3. $\forall t,e : e.\texttt{Connections}_t > 26700 \implies (e.\texttt{InCongession}_t > 0) \vee (e.\texttt{InRxmit}_t > 0)$ | **[Principle]** Large number of active connections (p90) indicate congestion or retransmission (incast). | NETNOMOS,<br>Decision Trees | ——— |
| 4. $\forall t,e : e.\texttt{Connections}_t > 0 \implies (e.\texttt{Ingress}_t > 0) \vee (e.\texttt{Egress}_t > 0)$ | **[Principle]** Active connection leads to traffic. | NETNOMOS,<br>Decision Trees | ——— |
| 5. $\forall t,e : \sum_{k=1}^{K=50} e.\texttt{InCongestion}_{t+k} > 206305 \implies \exists i \in [1,5] : e.\texttt{IngressRate10ms}_{t+i} > 5357$ | **[Principle]** Heavy congestion implies micro-bursts. | NETNOMOS | ——— |