Just-in-Time Logic Enforcement

A new paradigm of combining statistical and symbolic reasoning for network management

Hongyu Hè
Princeton University

Maria Apostolaki
Princeton University

ABSTRACT

Managing computer networks is increasingly complex. While machine learning offers scalability and can learn from unlabeled data, it remains error-prone, even making mistakes obvious to humans. This challenge is frustrating in networking, which is governed by clear logic rules. Prior attempts to combine ML with logic fall short: they handle only simple rules, demand retraining or heavy engineering per task, and still either compromise fidelity or offer no rule compliance guarantees.

In this paper, we argue that the most effective way to combine ML with logic is to enforce logic during inference (or Just-In-Time) rather than during training or post-generation. This approach decouples logic extraction and enforcement from ML design and training, allowing more expressive rules with little engineering overhead and achieving a better trade-off between fidelity and compliance guarantees.

To demonstrate the potential of Just-In-Time Logic Enforcement, we design LeJIT, a proof-of-concept framework that transforms the same GPT-2 model into either a synthetic data generator or a telemetry imputer by applying different sets of logic rules at inference time. At its core, LeJIT interleaves an SMT solver into the LLM's inference process, guiding generation step-by-step to enforce domain-specific rules. By performing on par with task-specific systems, LeJIT offers early evidence that logic-driven inference may be the key to harnessing ML in networking—and opens the door to imagining a foundation model for networking that we can shape through logic rules, rather than costly retraining or task-specific architectures.

1 INTRODUCTION

Machine learning (ML), and large language models (LLMs) in particular, hold promise for simplifying network management from router configuration [31] and intent translation [50], to incident response [18] and scheduling [20, 52], thanks to their ability to learn from unlabeled data of various modalities such as traffic traces [7], time series [29], and even RFCs [40]. However, network operators remain skeptical, as these models often hallucinate [11, 21, 41, 61] or struggle to follow rules [3, 9, 23, 24], making mistakes that a human expert would never make. They are also highly data-hungry, and training or even fine-tuning them is resource-intensive. Initial hopes for a universal model that could handle diverse networking tasks without increasing training costs are fading, as we increasingly see specialized, task-specific solutions [22, 27, 42, 59].

Unlike fields such as biology, networking is entirely human-engineered, not a natural phenomenon: almost every bit transmitted reflects deliberate choices in protocol design, hardware, and software. Solely relying on ML to infer all this structure from data (and then criticizing it when it fails to replicate or predict behaviors well-understood by experts) is ill-conceived at best. As prior work has noted, a natural solution is to combine ML with logic [6, 15, 60], *i.e.,* explicit rules. Existing approaches typically follow one of two design paradigms. On one end, logic can be taught through training, *i.e.,* pre-inference. For example, Zoom2Net [16] augments the loss function with logic constraints inspired by physics-informed neural networks [8, 37]. While this approach is efficient at runtime, it requires re-training, white-box access to the model, limits rules to differentiable forms, and most importantly, does not guarantee rule compliance during inference. On the other end, logic can be enforced after inference. NetDiffusion [24] takes this route, applying rule-based corrections post-generation to fix invalid outputs. This method is model-agnostic and supports more expressive rules, but it may fail to find a valid correction [23], as often observed in NetDiffusion, which relies on a deterministic one-pass algorithm.

This paper argues that logic should guide ML *during inference*, rather than during training or as a post-processing step. Autoregressive models such as transformers and LLMs are particularly well-suited to this approach, as their sequential generation allows rules to be enforced step-by-step. Concretely, we suggest that rules guide the model's generation process, dynamically pruning its token options at each step to enforce rule compliance. This approach enables easy repurposing of models by modifying the rules 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, as we will show, enforcing rules during inference can be minimally invasive and preserve ML fidelity.

While conceptually straightforward, intercepting and guiding an LLM's generation process using logic is highly challenging. First, network rules are complex, as they involve arithmetic constraints, conditional logic, and relationships across multiple input variables, making them incompatible with token-based filtering, which is recently supported by some

LLMs to ensure syntactic compliance [2, 12, 13, 34, 38, 51]. To cope with this complexity and benefit from the rich domain knowledge of networking, we posit that a true constraint solver must natively join the LLM's inference process. Following this principle, we built LeJIT, a framework in which an SMT solver intersects the LLM's token-by-token inference to guide it towards rule-compliant generation. Before each token generation, the solver dynamically computes the set of valid next tokens based on the applicable logic rules, and the already generated tokens. Critically, the solver also looks ahead before computing the valid tokens to ensure that there is a path to a valid complete output, *i.e.,* token sequence. As a result, LeJIT is minimally invasive, gently nudging the LLM away from mistakes that lead to dead ends without overwriting decisions that would not have led to rule violations, thereby preserving the LLM's original (valid) decisions.

The tight integration of the SMT solver brings LeJIT substantially more flexibility and power but comes with its own challenges, some addressed and many still ahead. One such hurdle is the mismatch in granularity: LLMs operate over vocabularies of tokens, while SMT solvers reason over higher-level variables such as network measurements and packet header fields. To bridge this gap, LeJIT constructs a character-level transition system on the fly to exert finer-grained control than the granularity of variable-level network rules.

Several challenges remain, especially related to improving solver speed, including designing near-lossless abstractions, identifying which rules are most helpful for a given task, and refining provided rules to avoid dead ends more effectively.

Even in its proof-of-concept form, LeJIT shows significant promise. It turns the same GPT-2 model into either a synthetic data generator or a telemetry imputer simply by applying different sets of logic rules at inference time. Notably, LeJIT-guided GPT-2 delivers performance on par with heavily engineered pipelines like Zoom2Net and NetShare—while producing outputs that are more accurate from a knowledge-consistency perspective. This result points to a compelling vision: instead of chasing ever-larger opaque models, we could build a single reusable, task-adaptable foundation model for networking and work on guiding it with logic (rather than raw GPU power).[1]

2 MOTIVATION

We begin with a motivating use case introduced in recent work [16] to illustrate the benefits of combining ML with logic in networking. We use this use case to highlight why existing approaches fall short of fully realizing that potential and validate our intuition in our preliminary results (§4).

[1]The authors have nothing against GPUs—just a shortage.

2.1 Why Networking Needs Logic

Example. A datacenter operator seeks to analyze fine-grained burst behavior [14], but only coarse-grained measurements (*e.g.,* ingress volume, ECN marked byte count) at 50 ms intervals are available. To recover missing millisecond-level ingress bytes $I_0..I_4$, the operator uses a telemetry imputer—an ML model trained to infer fine-grained signals from coarse-grained ones [16]. This task is feasible because many network metrics are correlated [14, 16].

As illustrated in Fig. 1a, the operator chooses to use a LLM, leveraging its recent advances. Given inputs such as $\texttt{TotalIngress}_T = 100$ and $\texttt{Congestion}_T = 8$ over a window $T = 5$, the LLM predicts $I_0..I_4 = [20,15,25,70,8]$. This output violates two key rules: $I_3 = 70$ exceeds the bandwidth limit (BW=60), and the total sum (138) exceeds $\texttt{TotalIngress}_T$.

As prior work has noted [16, 24], rather than faulting an ML model for violating known rules, a better solution is to explicitly encode those rules into the model's pipeline. In our example, these include: $\forall t < T : \ 0 \leq I_t \leq \text{BW}$ (R1); $\sum_{t=0}^{T-1} I_t = \texttt{TotalIngress}_T$ (R2); and $(\texttt{Congestion}_T > 0) \implies \max_{t=0}^{T-1}\{I_t\} \geq \frac{1}{2}\text{BW}$ (R3).

R1 ensures that the ingress volume (I_t) at any given time is non-negative and does not exceed the bandwidth (BW). R2 states that the sum of all I_t within a time window T must equal the total observed ingress $\texttt{TotalIngress}_T$. R3 specifies that if ECN-markings (`Congestion`) are detected during the window T, there must be a burst event where at least one I_t exceeds half the bandwidth [14].

2.2 Where Prior Methods Fall Short

Enforcing logical rules, such as R1–R3, on ML models in a way that leverages their complementary strengths without putting them at odds is challenging. To better understand this problem, we examine three fundamentally different strategies explored in prior networking and ML research: (1) correcting model outputs after inference, (2) teaching model rules at training time, and (3) constraining the model decoding process. We omit the discussion on prompt engineering for LLMs, which is inherently ad-hoc and provides no guarantees.

Enforcing rules post-inference. A natural way of integrating logic rules in any ML task is to allow the generative model to operate freely and then correct its output after generation to satisfy these rules. The correction can be done using a fast deterministic algorithm as in [24], an ILP as in [16] or a full constraint solver (*e.g.,* an SMT solver) as done in other domains [5, 10, 45, 49].

We illustrate this post-inference in the lower part of Fig. 1a, where the LLM's invalid output is fed to an SMT solver (③) together with R1–R3. We use an SMT solver because it is the most general. The solver's job is to modify the LLM's output to make it compliant with all provided rules. Unless provided

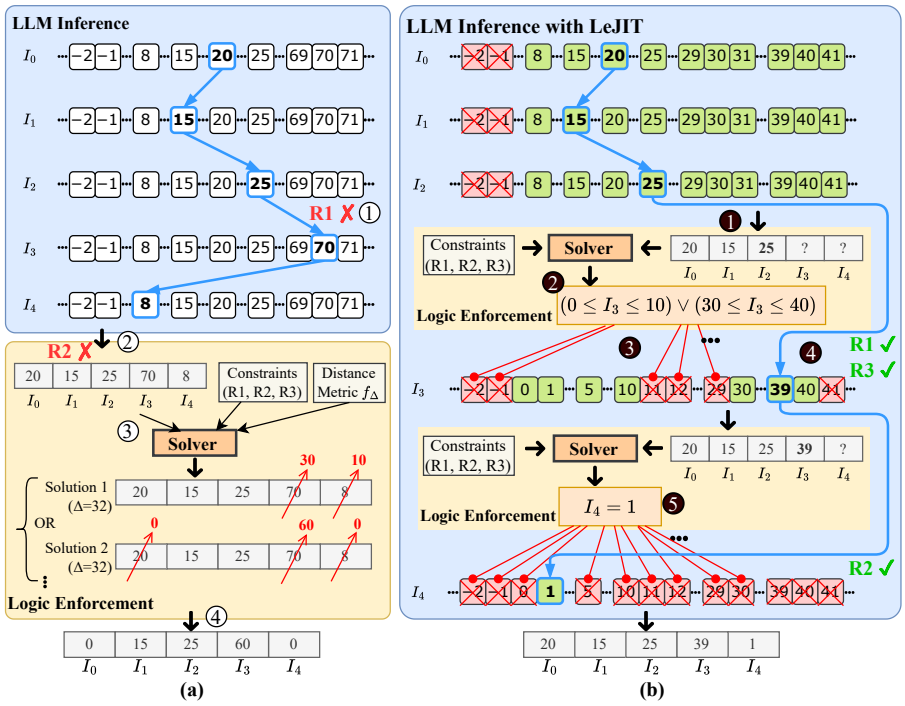

Figure 1: Example of using an LLM for network telemetry imputation under three rules R1-R3. (a) A pure LLM (blue frame) generates imputed samples $[I_0..I_4]$ that violate fundamental networking rules such as R1–R3. We can enforce logic post-inference (yellow frame), but not without hurting the statistical fidelity of the imputed sample. (b) Instead of enforcing rules post-inference, LeJIT invokes an SMT solver before every token generation to filter out tokens, that if selected by the LLM, will result in rule violations, effectively enforcing logic Just-In-Time.

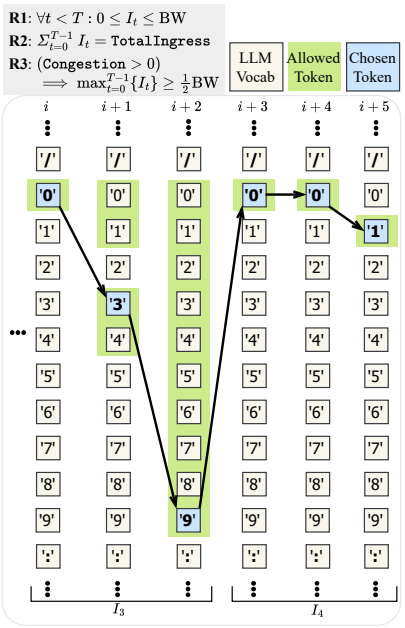

Figure 2: Character-level transition system constructed by LeJIT on the fly when imputing I_3 and I_4. LeJIT operates on tokens, while the SMT solver on variables. Carefully aligning them allows LeJIT to be minimally invasive.

with a specific optimization goal, the SMT solver would select an arbitrary solution among all compliant ones, not the most likely solution based on historical data. In other words, it will not leverage the LLM's learned distribution. One possible mitigation is to define a distance metric (f_Δ) and ask the solver to find a solution that satisfies the constraints while remaining as close as possible to the original output of the LLM. While this method is relatively straightforward in domains like vision, where simple metrics like L2 distance often suffice, various fields in networking are far more complex to compare. Semantic meaning does not align with numerical distance [9, 23, 25], making it challenging to define a meaningful metric for each field of interest.

Teaching the model to follow rules. One way to encourage constraint satisfaction (R1–R3) is to embed rules into the training process, typically by adding them to the loss function as regularization terms [3, 16, 26, 39, 54]. The model is penalized for violations during training, with the hope that it generalizes rule compliance at inference time.

However, this approach has major drawbacks. It offers no guarantee of constraint satisfaction during inference, and applies only to differentiable rules or their approximations—problematic in networking, where most rules are non-differentiable. For instance, constraints R1–R3 require approximations such as sigmoid functions or fuzzy logic [58] to be included in the loss. Moreover, scaling to many constraints is difficult: each rule must be manually encoded and weighted, which complicates optimization [3, 26]. This is especially limiting in domains like networking, where describing a single protocol may involve hundreds of rules [19, 23, 28, 55]. Lastly, this strategy lacks flexibility—any update to the rule set requires retraining or fine-tuning, making it ill-suited for dynamic environments.

Enforcing rules during decoding. The inability of ML and LLMs in particular to follow explicit rules has prompted the ML research community to develop specialized techniques to help them adhere to standardized output formats such as

JSON or knowledge triplets [2, 12, 13, 34, 38, 51]. Formally, constraint decoding cannot be used to enforce networking rules because there is no theoretical foundation for converting them into compatible forms, such as individual automata or their unions. In other words, constrained decoding typically filters tokens based on immediate validity (like matching a grammar), but it cannot perform arithmetic calculations or ensure that a future token can satisfy the constraint model. Encoding a constraint with $\sum$ as rule R2 into a decoding process would mean tracking the running total and pruning any continuation that makes the final sum impossible—essentially doing search or backtracking. Even keeping track of a single such rule is far beyond the capability of standard token-by-token decoding.

3 JUST-IN-TIME LOGIC ENFORCEMENT

Having shown that enforcing logic post-inference or during training compromises either correctness (compliance with rules) or fidelity (learned distributions), and constraint encoding is inadequate for network constraints, this section presents LeJIT: a framework for Enforcing Logic Just-In-Time. It intersects the LLM's token-by-token inference to guide it towards rule-compliant generation as shown in Fig. 1b. While incorporating the SMT solver introduces some inference delay, it provides a valuable balance between neural and symbolic reasoning. It maximizes the contribution of symbolic reasoning by enabling the enforcement of arbitrary constraints, including arithmetic, non-differentiable, and global ones, without placing additional burden on the operator. Same as using an LLM, despite its training cost, LeJIT maximizes the contribution of statistical learning.

To better understand how LeJIT works, let us revisit the example of imputing $[I_0, ..., I_4]$ but now as generated with LeJIT's guidance. After generating a complete value (*e.g.,* I_2 at ❶), LeJIT invokes the solver with the provided constraints, *instantiated using the values generated so far.* This dynamic partial instantiation is crucial for determining which constraints are relevant and what conditions must be met to ensure valid output going forward. For example, suppose the LLM had already produced values satisfying $\exists t < 3 : I_t \geq 30$; in that case, R3 would already be met and thus deactivated when determining the feasible region for I_3. If no such value has been generated—as is the case in our example—the solver considers all three rules when computing the valid range for I_3 (❷). Then, LeJIT invalidates all candidate values of I_3 that fall outside this feasible region (❸), effectively guiding the model toward valid generation paths. As a result, the resulting model output $I_3 = 39$ is always guaranteed to satisfy all constraints (❹). Moreover, in the presence of global aggregation constraints such as R2, this guided inference process often concludes with only a single valid value remaining for the LLM to emit (❺).

LeJIT provides a little guidance, but it goes a long way. Over-constraining the LLM, for example, through partial completions or rigid templates, disrupts its natural reasoning path and undermines its generative strength. Still, even a well-trained model is highly likely to produce invalid outputs, since a single incorrect token can render the entire sequence invalid. As illustrated in Fig. 1a, the sequence becomes invalid as early as the generation of I_3. LeJIT strikes a balance by filtering out rule-violating tokens at each generation step, intervening only when the model is about to make a critical mistake. This approach preserves the LLM's natural behavior while enforcing compliance with constraints.

LeJIT offers LLM-native generation with character-level control. A key challenge in guiding an LLM with the solver is the mismatch in granularity between the model's generation process and the solver's reasoning. LLMs produce output token by token, and these tokens, defined by the tokenizer, are often opaque and lack semantic clarity. In contrast, SMT solvers operate over well-defined, interpretable variables (such as ingress bytes or ECN markings) expressed through explicit logical constraints. This discrepancy makes it difficult to enforce constraints without interfering with the LLM's native decoding behavior.

LeJIT addresses this issue by offering fine-grained, character-level guidance, even when constraints are specified at the level of semantic variables. To achieve this level of control, LeJIT treats numeric values as plain text [36] and uses a character-level tokenization scheme [44], generating each number digit by digit. As shown in Fig. 2, LeJIT constructs a character-level transition system[4, 46, 48] on the fly during inference. Specifically, given a feasible range for a target variable as determined by the solver, LeJIT builds an unlabeled transition system where the current state reflects the last token selected by the LLM, and the set of next states includes all tokens that would maintain the value within the valid region.

A single LLM to "rule" them all? A key side benefit of applying rules at inference time is that modifying the rules enables repurposing an existing LLM—originally trained for one task—for a different task, without retraining or fine-tuning. For example, an LLM trained to impute fine-grained ingress volumes can be readily adapted to generate synthetic coarse-grained signals by simply changing the constraints: instead of enforcing rules on fine-grained ingress values I_t that rely on access to coarse-grained signals, we can substitute rules that capture relationships among the coarse-grained signals themselves. In our preliminary evaluation (§4.2), we demonstrate that a generic LLM trained for telemetry imputation can, under the guidance of LeJIT, achieve competitive performance with SOTA specialized data generators.

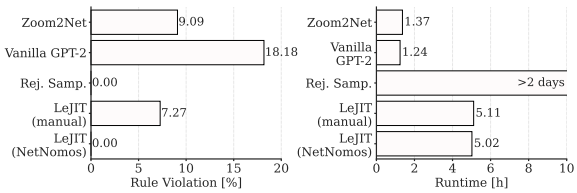

Figure 3: Rule violations in imputed time series (left) and runtime for 30K samples (right). LeJIT ensures 100% rule compliance with a moderate runtime overhead.

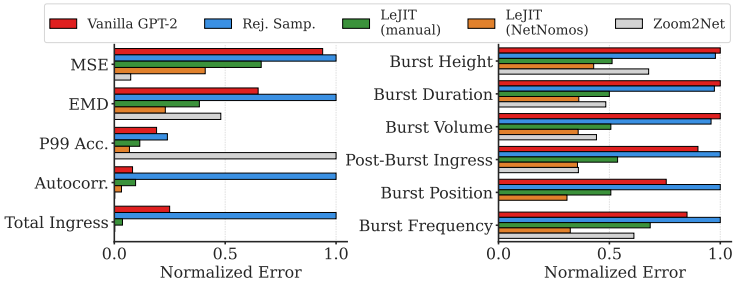

Figure 4: LeJIT improves both imputation accuracy (left) and downstream task performance (right) of the generic GPT-2 via logic enforcement, achieving on-par results with Zoom2Net [16].

4 EARLY RESULTS

As a proof of concept, we prototype and empirically evaluate the effectiveness of LeJIT in experiments.

Dataset. We conduct all experiments using the data center data released by Meta [14], following the same evaluation setup as that of Zoom2Net [16].The test set contains over 30,000 data points.

Network rules. For the network telemetry imputation task (§4.1), we use 55 rules which describe relationships between coarse-grained signals (*e.g.,* retransmissions) and fine-grained ingress measurements I_t. For the synthetic network data generation task (§4.2), we use 171 rules that capture relationships among the coarse-grained signals themselves. Both sets of rules were provided by the authors of NetNomos [23].

LeJIT implementation. We evaluate LeJIT on a less powerful LLM, namely, GPT-2 [35].We train GPT-2 from scratch on the aforementioned datacenter dataset [14] and adopt character-level tokenization [44] to support fine-grained control. Importantly, we repurpose the same trained model for two distinct tasks by applying task-specific rule sets through LeJIT, without any retraining or fine-tuning.

Baselines. We use the following baseline for both use cases: *(i)* Vanilla GPT-2: The original GPT-2 model without LeJIT; *(ii)* Rejection Sampling: A naive approach that discards all outputs violating network rules and repeatedly samples from GPT-2 until a valid output is produced;*(iii)* "`manual`" rules: Instead of using the automatically discovered rules from NetNomos [23], this baseline enforces the four manually specified rules (C4–C7) used by Zoom2Net [16]. For each use case, we compare against SOTA task-specific frameworks. For network measurement imputation (§4.1), we evaluate against Zoom2Net [16].For synthetic network data generation (§4.2), we compare against a diverse set of SOTA data generators: NetShare [56], E-WGAN-GP [17], CTGAN [53], TVAE [53], and the GPT-2-based REaLTabFormer [43].

4.1 LeJIT for Network Telemetry Imputation

Finding 1: Unlike task-specific models, which (at best) comply with a few hand-picked rules, LeJIT comply with all 55 rules, while achieving on-par performance in imputation accuracy and downstream tasks.

We apply LeJIT on the task of network telemetry imputation and evaluate its effectiveness in enforcing network rules, overhead and accuracy.

Rule violation. Fig. 3 (left) reports rule violation rates. Vanilla GPT-2, lacking any constraints, shows the highest violation rate at 18%. Zoom2Net, despite using a constraint enforcement module (CEM), relies on limited and soft `manual` rules, resulting in over 7% violations—similar to LeJIT when only `manual` rules are used. With the full set of NetNomos rules, LeJIT reduces violations to 0%.

Runtime overhead. As shown in Fig. 3 (right), rejection sampling achieves perfect compliance but takes over two days, as it repeatedly discards invalid outputs without guiding the model. In contrast, LeJIT completes over 30K imputations in 5 hours by guiding inference directly. Zoom2net's runtime performance is not directly comparable because it enforces a fraction of the rules. While LeJIT incurs significant overhead compared to unguided GPT-2, our prototype remains unoptimized and offers opportunities for future speedups (§5).

Imputation accuracy. LeJIT with `manual` rules substantially improves GPT-2's accuracy (Fig. 4, left), though it still trails Zoom2Net due to limited domain coverage. Rejection sampling hurts accuracy by distorting the LLM's distribution, suppressing near-correct outputs and forcing sampling from unrelated regions. With full NetNomos rules, LeJIT matches and even surpasses Zoom2Net on EMD and p99 accuracy, while also improving burst analysis metrics across the board. When guided by LeJIT, GPT-2 outperforms Zoom2Net on all metrics except Burst Position. These results show that LeJIT enforces rules effectively at inference time, with performance improving as rule quality increases. The remaining gap on

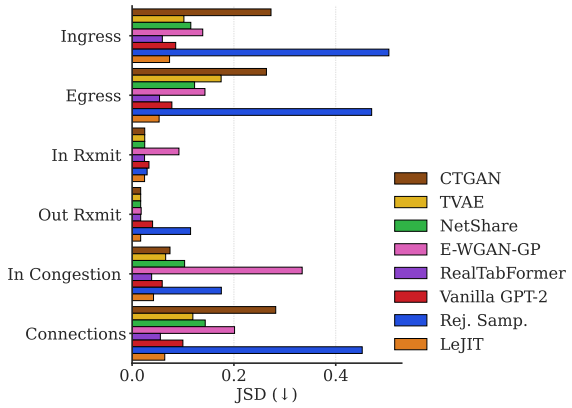

Figure 5: LeJIT generates samples of high fidelity (on-par with SOTA) while complying with all 171 rules (unlike SOTA).

time-sensitive metrics (*e.g.,* autocorrelation, Burst Position) likely stems from GPT-2's general-purpose architecture and the limited temporal expressiveness of the extracted rules by NetNomos [23]. Advancing methods for learning richer temporal constraints remains a key direction for future work and will unlock more benefits for LeJIT.

4.2 LeJIT for Network Data Synthesis

Finding 2: LeJIT preserves, and in some cases improves, the statistical fidelity of synthetic time series generated by tailored generators, while ensuring the time series follow hundreds of rules (unlike tailored data generators). Importantly, LeJIT's underlying model is not task-specific.

We now apply LeJIT to the task of synthetic data generation and evaluate its effectiveness in enforcing network rules. Unlike the imputation, this generation task is unconditional: the models are not provided with any input signals (*e.g.,* prompts no longer fed into GPT-2), and the data they generate depends solely on the learned input distributions.

As shown in Fig. 5, we compare various GPT-2 variants (vanilla, with rejection sampling, and with LeJIT's guidance) against five aforementioned SOTA data generators. From each model, we draw 30K samples and compute the Jensen–Shannon divergence (JSD) with respect to the original data distribution. The results demonstrate that LeJIT preserves the generative behavior of the base LLM while enforcing all 171 network rules. Rejection sampling significantly distorts the learned distribution, while the other data generators not only violate a large number of network rules [23], but also fail to offer clear advantages in approximating the target distribution. In contrast, LeJIT enables the base LLM to outperform its vanilla counterpart in most cases. This result suggests that embedding domain knowledge through inference can improve the quality of generated data distributions.

5 RESEARCH AGENDA

Logic-Guided Foundation Models for Networking. We envision a future where one LLM can power a broad range of networking tasks (*e.g.,* configuration generation, security policy synthesis) simply by swapping in task-specific logic rules. Such a foundational, logic-guided model for networking would unify currently siloed ML efforts and vastly reduce engineering overhead [52]. Key questions include: (1) how to symbolically handle non-numeric or structured outputs (*e.g.,* tables, topology graphs) in a language-based model, (2) how to tokenize heterogeneous networking knowledge in a way that does not create misalignment between the model output and symbolic rules, and (3) how to efficiently switch or compose rule sets for different tasks on the fly. Success in this direction would be transformative—instead of maintaining bespoke ML solutions for every networking problem, operators could rely on a single powerful model that is made context-specific and trustworthy via JIT logic "plug-ins."

Constraint Learning and Solver Co-Design To improve JIT logic enforcement, two key directions are (1) improving LLM–solver integration and (2) improving the rule sets themselves. Current implementations rely on general-purpose SMT solvers external to the LLM, introducing significant inference delays [57]. Future work should enable tighter coupling through token-level solvers, solver-aware decoding paths, or hybrid neural-symbolic architectures, making JIT enforcement feasible for latency-sensitive applications. In parallel, network rules—which are currently static and manually defined—must become more expressive (*e.g.,* better support for temporal logic), data-driven, and adaptable. Systems could learn constraints from logs, refine them over time, or co-train them with model outputs.

Generalizing LeJIT beyond LLMs. While LeJIT currently targets autoregressive language models, many core networking tasks (*e.g.,* traffic forecasting, anomaly detection, routing, and protocol simulation) rely on non-language models like time-series regressors, GNNs, and diffusion models. However, this generalization is non-trivial. Unlike token-based LLMs, these models often produce continuous, high-dimensional outputs without an inherent notion of "next-step," making it unclear how to insert constraint checks or prune invalid predictions. One promising direction is to rethink the inference process itself in networking as a constrained optimization problem: for instance, projecting a model's unconstrained output onto the nearest point in the rule-compliant space via differentiable solvers [1, 32, 47] or gradient-based corrections. Similarly, generative models could be trained to emit semantic concepts [30, 33] that are easier to steer via symbolic logic, then decoded in a constraint-aware manner.

REFERENCES

[1] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. 2019. Differentiable convex optimization layers. *Advances in neural information processing systems* 32 (2019).

[2] Luca Beurer-Kellner, Marc Fischer, and Martin T. Vechev. 2024. Guiding LLMs The Right Way: Fast, Non-Invasive Constrained Generation. *International Conference on Machine Learning (ICML)* abs/2403.06988 (2024).

[3] Elliot Chane-Sane, Pierre-Alexandre Leziart, Thomas Flayols, Olivier Stasse, Philippe Souères, and Nicolas Mansard. 2024. Cat: Constraints as terminations for legged locomotion reinforcement learning. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 13303–13310.

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

[5] Andrea Coletta, Sriram Gopalakrishnan, Daniel Borrajo, and Svitlana Vyetrenko. 2023. On the constrained time-series generation problem. *Advances in Neural Information Processing Systems* 36 (2023), 61048–61059.

[6] Davide Corsi, Enrico Marchesini, and Alessandro Farinelli. 2021. Formal verification of neural networks for safety-critical tasks in deep reinforcement learning. In *Uncertainty in Artificial Intelligence*. PMLR, 333–343.

[7] Tianyu Cui, Xinjie Lin, Sijia Li, Miao Chen, Qilei Yin, Qi Li, and Ke Xu. 2025. TrafficLLM: Enhancing LLMs for Network Traffic Analysis. *arXiv preprint arXiv:2504.04222* (2025). Version 2025-04-05.

[8] Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. 2022. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing* 92, 3 (2022), 88.

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

[10] Priya L Donti, David Rolnick, and J Zico Kolter. 2021. DC3: A learning method for optimization with hard constraints. *arXiv preprint arXiv:2104.12225* (2021).

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

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

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

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

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

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

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

[18] Pouya Hamadanian, Behnaz Arzani, Sadjad Fouladi, Siva Kesava Reddy Kakarla, Rodrigo Fonseca, Denizcan Billor, Ahmad Cheema, Edet Nkposong, and Ranveer Chandra. 2023. A holistic view of ai-driven network incident management. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*. 180–188.

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

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

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

[22] Vojtěch Hudeček and Ondřej Dušek. 2023. Are LLMs all you need for task-oriented dialogue? *arXiv preprint arXiv:2304.06556* (2023).

[23] Hongyu Hè, Minhao Jin, and Maria Apostolaki. 2025. Making Logic a First-Class Citizen in Network Data Generation with ML. arXiv:2506.23964 [cs.NI] https://arxiv.org/abs/2506.23964

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

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

[26] Yunho Kim, Hyunsik Oh, Jeonghyun Lee, Jinhyeok Choi, Gwanghyeon Ji, Moonkyu Jung, Donghoon Youm, and Jemin Hwangbo. 2024. Not only rewards but also constraints: Applications on legged robot locomotion. *IEEE Transactions on Robotics* (2024).

[27] Haitao Li, Qingyao Ai, Jia Chen, Qian Dong, Zhijing Wu, and Yiqun Liu. 2025. Blade: Enhancing black-box large language models with small domain-specific models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 24422–24430.

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

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

[30] Jiayuan Mao, Joshua B Tenenbaum, and Jiajun Wu. 2025. Neuro-Symbolic Concepts. *arXiv preprint arXiv:2505.06191* (2025).

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

[32] Geoffrey Négiar, Michael W Mahoney, and Aditi S Krishnapriyan. 2022. Learning differentiable solvers for systems with hard constraints. *arXiv preprint arXiv:2207.08675* (2022).

[33] Sagar Patel, Dongsu Han, Nina Narodytska, and Sangeetha Abdu Jyothi. 2024. Toward Trustworthy Learning-Enabled Systems with Concept-Based Explanations. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks*. 60–67.

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

[35] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. 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.

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

[37] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics* 378 (2019), 686–707.

[38] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. *arXiv preprint arXiv:2109.05093* (2021).

[39] Sungyong Seo, Sercan O. Arik, Jinsung Yoon, Xiang Zhang, Kihyuk Sohn, and Tomas Pfister. 2021. Controlling Neural Networks with Rule Representations. arXiv:2106.07804 [cs.LG]

[40] Prakhar Sharma and Vinod Yegneswaran. 2023. Prosper: Extracting protocol specifications using large language models. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*. 41–47.

[41] Rahul Anand Sharma, Ishan Sabane, Maria Apostolaki, Anthony Rowe, and Vyas Sekar. 2022. Lumen: a framework for developing and evaluating ML-based IoT network anomaly detection. In *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies*. 59–71.

[42] Junhong Shen, Neil Tenenholtz, James Brian Hall, David Alvarez-Melis, and Nicolo Fusi. 2024. Tag-LLM: Repurposing general-purpose LLMs for specialized domains. *arXiv preprint arXiv:2402.05140* (2024).

[43] Aivin V. Solatorio and Olivier Dupriez. 2023. REaLTabFormer: Generating Realistic Relational and Tabular Data using Transformers. *arXiv preprint arXiv:2302.02041* (2023).

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

[45] Paul Temple, José A Galindo, Mathieu Acher, and Jean-Marc Jézéquel. 2016. Using machine learning to infer constraints for product lines. In *Proceedings of the 20th International Systems and Software Product Line Conference*. 209–218.

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

[47] Kiwon Um, Robert Brand, Yun Raymond Fei, Philipp Holl, and Nils Thuerey. 2020. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. *Advances in neural information processing systems* 33 (2020), 6111–6122.

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

[49] David Wan, Chris Kedzie, Faisal Ladhak, Marine Carpuat, and Kathleen McKeown. 2020. Incorporating Terminology Constraints in Automatic Post-Editing. In *Conference on Machine Translation*.

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

[51] Brandon T Willard and Rémi Louf. 2023. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702* (2023).

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

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

[54] Chenxi Yang and Swarat Chaudhuri. 2022. Safe neurosymbolic learning with differentiable symbolic execution. *arXiv preprint arXiv:2203.07671* (2022).

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

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

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

[58] Lotfi Asker Zadeh. 1988. Fuzzy logic. *Computer* 21, 4 (1988), 83–93.

[59] Jiali Zeng, Fandong Meng, Yongjing Yin, and Jie Zhou. 2024. Teaching large language models to translate with comparison. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 19488–19496.

[60] Yedi Zhang, Yufan Cai, Xinyue Zuo, Xiaokun Luan, Kailong Wang, Zhe Hou, Yifan Zhang, Zhiyuan Wei, Meng Sun, Jun Sun, et al. 2024. The Fusion of Large Language Models and Formal Methods for Trustworthy AI Agents: A Roadmap. *arXiv preprint arXiv:2412.06512* (2024).

[61] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. 2023. Siren's song in the AI ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219* (2023).