No Hyperscaler? No Problem. Your Students also Break (and Fix) Networks

Constantine Doumanidis
Princeton University

Hongyu Hè
Princeton University

Maria Apostolaki
Princeton University

ABSTRACT

Network operators diagnose faults by issuing commands, forming hypotheses, and iterating to a fix. Yet, no dataset captures this diagnostic process. Production traces are locked down, and synthetic benchmarks reduce debugging to single injected faults on homogeneous topologies. We propose mining students' actions during the completion of networking course assignments as a source of realistic, labeled, and shareable debugging traces. We test this idea in the mini-Internet, an open-source platform in which students configure their part of a miniature of the Internet, naturally generating cascading, multi-protocol failures and the trajectories that fix them. Realizing this opportunity requires confronting two challenges: what to log and at what granularity, and how to extract clean diagnostic trajectories from noisy, concurrent student sessions. We describe our initial design and the open research problems ahead. Our preliminary results show that non-trivial faults such as misconfigurations and diagnostic patterns can be mined from student logs automatically, and that the resulting guidance improves the root cause analysis accuracy of LLM agents.

CCS CONCEPTS

• **Networks** → **Network manageability**; • **Information systems** → **Data mining**; • **Computing methodologies** → **Artificial intelligence**.

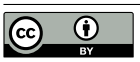
KEYWORDS

trace mining, debugging traces, network troubleshooting, network misconfiguration, root-cause analysis, network management education, agentic network operations

1 INTRODUCTION

Managing networks is hard. Operators face a wide range of tasks, from routine work like adding a new server and steering traffic across inter-domain paths, to diagnosing routing and configuration mistakes, to untangling cascading protocol failures whose root cause spans multiple systems and teams. The rise of LLMs has renewed interest in automating these tasks, yet even on purpose-built troubleshooting benchmarks, today's agents still struggle to localize faults and pinpoint root causes [36]. A key reason is data scarcity. Production environments generate rich operational traces, but those traces stay locked behind confidentiality walls because even synthesized network traces can leak sensitive source-level topology and traffic correlations [19]. What is studied in research takes one of two forms: single injected faults on simulated homogeneous topologies, or sanitized industry snapshots scrubbed of most operational context. The messy, cascading, multi-party failures, the routine configuration steps an operator takes to configure a new router, and the sequences of diagnostic actions that define real network management are simply absent.

We argue that the networking research community is sitting on an overlooked gold mine: academic networking courses. Consider a course built around the mini-Internet [17], where each student group operates its own Autonomous System, initializes all routers' configurations, establishes BGP sessions with neighboring ASes, and keeps their slice of the Internet alive all under the pressure of a semester deadline. Critically, these ASes are interconnected and share one server: a mistake in one AS becomes an outage in another, and a crashed container becomes someone else's unexplained link failure. Thus, failures emerge organically, not according to a predetermined script, from the interaction of dozens of concurrent operators (other groups) making independent decisions: Group A's route policy clashes with Group B's, while Group C's link goes down because their containers ran out of memory. Additionally, students perform fault diagnosis and mitigation, which often resembles genuine diagnostic reasoning similar to what we would expect from experienced operators. Students are, of course, not seasoned engineers, and we do not claim their faults or their reasoning match a production network run by one. What a course provides is a scarce ingredient: dozens of independent people with real stakes making concurrent decisions on a shared substrate, recorded. This cannot be emulated with, say, graduate students or engineers solving open-source challenges or injected faults. Even when student faults are simpler than production incidents, their traces are still valuable; for instance an autonomous Root Cause Analysis (RCA) agent that fails on them cannot be trusted in production. Critically, course-generated traces



This work is licensed under a Creative Commons Attribution 4.0 International License.

are also shareable: per-group topologies are identical, so no group is identifiable; IPs are abstracted; and any fault can be replayed on a fresh instance of the open-source platform. By contrast, even if provided, incident details from proprietary industry settings rarely transfer directly to another organization's topology and tooling.

Turning this observation into a research asset is not straightforward and requires solving two families of problems that mirror open challenges in production monitoring. The first is instrumentation: deciding what to log, at what granularity, and how to do so without disrupting the students or the platform itself. The second is mining: extracting clean, meaningful patterns of faults or of diagnostic trajectories from sessions that are noisy, temporally ambiguous, and interleaved across concurrent operators. Two key insights drive our approach. To minimize load, we need to monitor what is strictly needed to reproduce students' reasoning; hence, we only rarely run expensive global experiments such as PingMesh-like measurements but record each keystroke of students' command lines. Recording every command allows us to capture not just the final network state, but the full operational process, including the sequence of commands, hypotheses, and configuration changes that led there. To make sense of recorded sessions, we transform them through abstraction steps into a symbolic command sequence and use structured templates to identify recurring patterns in the debugging processes found across student groups [14, 16].

We instrumented one semester of our course: 78 ASes, 26 of them run by student groups, 26 devices each. Three templates alone mine 10090 occurrences (§4). The mined faults are not toy benchmarks: given full documentation and shell access to every device, a GPT-5.4-mini agent localizes our L2 subnetting fault in 3 of 5 runs, never localizes our BGP traffic engineering fault, and fails to even detect the latter in 4 of 5 runs. Runbooks generated automatically from the mined patterns lift root cause accuracy to 100% on both, with 32% and 38% fewer input tokens. Upgrading to a stronger model, GPT-5.5, yields a similar picture: runbooks raise RCA success rate from just 20% to 100% in the L2 subnetting fault, while reducing token usage by 85% in the BGP traffic engineering fault. Neither of our two problems is specific to the mini-Internet: what to log and how to mine it are the questions production monitoring has not answered either.

2 MOTIVATION

Next, we outline three concrete applications enabled by the data we collect from monitoring students as they work on the mini-Internet. While already useful in this setting, these applications reach their full potential only when the same methodology is applied to other populations (trained engineers, researchers) and other platforms, from educational testbeds to production networks.

Realistic, labeled, and shareable fault benchmark. The monitoring system produces a fault corpus that can serve as a reliable benchmark to develop, evaluate, and compare RCA agents. Unlike recent troubleshooting benchmarks built from curated or injected incidents [36], this corpus has properties no existing dataset offers. First, it contains real cascading failures caused by different individuals' actions across multiple routing protocols and hardware faults. For instance, Group A's BGP route policy interacts with Group B's OSPF configuration, while Group C's link fails because its containers crashed. Second, the dataset is reproducible and shareable: because we log the full configuration state and topology, any fault can be replayed on a fresh instance of the open-source mini-Internet platform. We ensure that no student or group is identifiable through their logs because internal AS topologies are identical and IP addresses are abstracted away, avoiding the source-level privacy leakage that affects shared or synthesized network traces [19]. Irrelevant commands or accidental inputs are removed, and students were informed of our recording, which also reduced cheating attempts. Third, the dataset is naturally labeled because students eventually solve each problem, and the configuration fix diff reveals the root cause without manual annotation.

Diagnosis Trajectories as demonstration data for network agents. Diagnosis trajectories, which capture sequences of diagnostic commands, configuration changes, and their effects on connectivity, are useful for training network copilots or serving as a realistic human baseline for evaluating RCA systems. Recent work reinforces their value: agents that explicitly encode a step-by-step diagnostic methodology or ground their decisions in formal models and runtime safety checks substantially outperform free-form reasoning at root-cause localization and control [32], showing that the diagnostic process itself carries useful signal beyond the eventual fix. Yet such trajectories are not available today beyond locked-down production environments that cannot be shared. By recording students' terminal sessions alongside configuration snapshots and ping-mesh measurements, we obtain both successful trajectories that serve as demonstrations from symptom to fix and failed trajectories that capture plausible hypotheses tried and ruled out as negative examples. Together, these trajectories provide the reward signal needed for imitation learning or RLHF-style fine-tuning, along with a built-in human baseline for evaluation. Although students may make more exploratory mistakes than veteran engineers, both populations must query the same routing and forwarding state to isolate a fault, and our abstraction and replay steps filter out dead-end commands before extracting demonstrations.

Evidence-based Monitoring Design. Network monitoring systems today are designed by intuition: operators add dashboards, metrics, and alerts until something feels sufficient, with no principled way to know whether they are collecting too much, too little, or the wrong things entirely. Even when telemetry imputers, anomaly detectors, and invariant miners extract structure from partial measurements [10, 16], operators still lack empirical evidence showing which signals actually guide human troubleshooting. A data-driven approach would allow operators to identify redundant sources that can be dropped and gaps where critical signals are missing. By aggregating which telemetry commands human operators reach for across hundreds of debugging episodes, and which commands actually help them, we obtain that signal directly: if nobody ever checks a particular metric, it is likely dead weight; if two commands always appear together, one may be redundant; if a fault type consistently requires information that no standard command surfaces, that gap points to missing monitoring support. Of course, student decisions and preferences might not reflect production monitoring value perfectly, yet they also reveal gaps in student understanding. More importantly, we believe this analysis represents a novel application that a logging system like the one we propose can enable.

Designing better network courses. In short: We can mine common error patterns that imply certain concepts need better instruction / reinforcement by instructors. We can also study how and when students employ the use of agents in their workflows, and understand what worked and what did not.

3 INITIAL DESIGN

Our goal is to extract user action trajectories, which we later also evaluate as RCA benchmarks and guides for RCA agents (§4). This section describes the challenges we encountered and the insights that influenced the design of our data collection and interpretation methods. We stress that these hold beyond the mini-Internet and even the classroom setting.

3.1 Instrumentation Challenges & Insights

Scale logging by avoiding redundancy: The full state of our mini-Internet network can be captured by recording extensive network data such as link statistics, ARP tables, BGP databases, and switch configuration files. However, such comprehensive data collection would be unscalable because it would compete with students for system resources. We observe that there is substantial redundancy in raw network telemetry because many unrecorded signals can be reconstructed from sparse measurements and configuration state or derived by re-running the mini-Internet. For instance, we frequently log configuration files of all routers, but we only infrequently collect global data-plane measurements such as pings and traceroutes.

Capture Students' Reasoning by recording terminals: Our goal is to study and understand the thought processes of students as they manage their networks. One way to do this is by asking students to keep detailed logs of their actions (config files, debug efforts, etc.). However, students might omit their failed attempts, or unnecessary actions, or might simply forget to do so. Our insight is that students' thought processes manifest in their terminal sessions, which can be logged via commodity terminal recorders, like asciinema [25]. We augment asciinema for our use by making it tag terminal lines with precise timestamps. Since students can have concurrent active terminal sessions, timestamping allows us to merge their actions into a single timeline. To ensure the completeness of our terminal logs, we modify asciinema to write out the full primary virtual screen buffer rather than just what is displayed on the screen. We also later reduce noise in our analysis, by discarding sessions that were either uneventful (*e.g.,* login-logout) or failed to properly start (*e.g.,* due to exhausted system resources). At the end, our pipeline yields a sequence of user terminal commands and their corresponding outputs.

Exploit (not avoid) Failures: Despite our best efforts to maintain high availability for mini-Internet containers running on our server, we occasionally observed that a small number of them crashed unexpectedly and needed to be restored. Rather than upgrading the server to eliminate these failures, we treated them as an opportunity to introduce random faults like those that commonly occur in real-world environments. Students were aware that such crashes could occur and were instructed on how to diagnose them and have their affected containers restored.

Action	Description	Example Commands
`create`	Initialize resources	`ip addr add ...`, `ovs-vsctl add-port...`
`read`	Inspect resources	`show ip bgp`, `do show running-config`
`update`	Modify resources	`ovs-vsctl set ...`, `restore_configs.sh`
`delete`	Remove resources	`ip addr del ...`, `no neighbor ...`
`probe`	Probe network	`ping 1.1.1.1`, `traceroute 1.1.1.1`
`navigate`	Shell context changes	`ssh root@ATLrouter`, `config terminal`
`system`	Misc. commands	`ls -l`, `cd`, `cat`, `chmod`, `save_configs.sh`

Table 1: Actions Taxonomy: abstracting commands by the effect they have, facilitates mining patterns.

Resource	Description	Example Commands
`interface`	Links, switch ports & tunnels	`ip tunnel add...`, `ifconfig eth0`,
`bridge`	OVS Bridges	`ovs-vsctl set-fail-mode br0 secure`
`address`	IPs, MAC addrs.	`ip addr add...`, `ip neighbor show`
`protocol`	Routing config. & static routes	`neighbor...`, `router bgp 100`
`policy`	Filter rules, ACLs & route maps	`route-map MAP permit 10`
`state`	Status checks, diagnostic probes	`ping 1.1.1.1`, `show ip bgp summary`
`config`	Static config files	`restore_configs.sh`
`system`	Misc system operations.	`ls -l`, `uname`

Table 2: Resource Type Taxonomy: this abstraction enables mining patterns by relating commands through the network resources they access or modify.

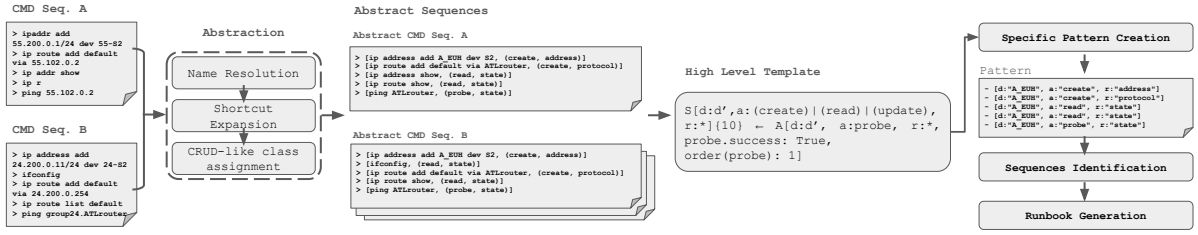


Figure 1: Student command sequences are transformed through a series of abstraction steps to abstract sequences that expose common patterns among individual command sequences. The corpus of abstract sequences is then matched against a set of high-level templates (*e.g.,* write after write with some commands in between). Each template can expose multiple individual patterns by constraining some of their parameters. Mined patterns can point to common faults such as configurationisms and/or guide LLMs in RCA, or routine tasks.

3.2 Interpretation Challenges & Insights:

Expose patterns by abstracting command sequences: Mining for patterns on raw commands is challenging since it is unlikely that any non-trivial sequence of commands appears verbatim more than a handful of times. Command strings can contain environment-specific identifiers (*e.g.,* IP addresses, interface numbers), reflect specific user preferences (*e.g.,* output formatting flags), and can have multiple other syntactically different yet functionally equivalent alternatives from which users can select. Consider a simple host configuration pattern that we would like to spot: a student adds an IP address to an interface, they then set a default gateway, then proceed to check if both of these settings were applied to the system, and finally check connectivity to the upstream router. Student A may choose to configure their `55_A_EUH` host using `ipaddr add 55.200.0.1/24 dev 55-S2`, followed by `ip route add default via 55.102.0.2`, then `ipaddr show` and `ip r`, and finally `ping 55.102.0.2`. Student B takes a different approach on their `24_A_EUH` host: `ip address add 24.200.0.11/24 dev 24-S2`, `ifconfig`, then `ip route add default via 24.200.0.254`, `ip route list default`, and finally `ping group24.ATLrouter`. Both sequences are underpinned by the same basic host configuration and verification workflow, yet their command sequences appear very different in their current form.

To extract our desired pattern, we must first lift this sequence to a higher level of abstraction. Concretely, we identify three necessary transformations: First, we map domain-specific identifiers, such as IP addresses and AS-specific hostnames, to symbolic counterparts that indicate their positions in the topology. We are able to do this because each student AS in the mini-Internet has the exact same internal structure. For instance, in the above example we map both `55.102.0.2` and `group24.ATLrouter` to `ATLrouter`, `55-S2` and `24-S2` become `S2`, and so on. We next transform all commands that have multiple (short-cut type) alternative forms and notations to a standard form; in our example, we overwrite `ipaddr add` with `ip address add`, and `ip r` with `ip route`. Finally, we map commands to equivalence classes that we define using (action, resource) tuples. Our actions taxonomy, shown in Table 1, is inspired by CRUD *i.e.,* the Create, Read, Update, and Delete actions from database systems [26]. In our context, these actions can be used to initialize, view, modify, and remove network-related configurations. We augment our actions to include probe, which represents network measurements through reachability tests and traces. To capture students moving between different shell environments (*e.g.,* SSH, privileged configuration modes), we defined the navigate action. Finally, to categorize commands unrelated to network management, we created a system action. Similar in spirit, our resource taxonomy in Table 2 represents the entities that students interact with at a high level in their assignment, including interfaces, addresses, protocols, policies, static configurations, etc. In our running example, `ipaddr show` and `ifconfig` both map to (read, address).

Mine patterns through carefully crafted templates: Identifying command patterns that are common is challenging because the potential candidates are too many to enumerate. Observe that all sequences of any size, composed of any type of command, and even their permutations, should be checked to determine if they are common. Straightforward splitting the command sequences in time to reduce the search space *e.g.,* by session does not work: students often use multiple sessions concurrently, or might not conclude their actions within one session.

Instead, we propose mining for patterns in two steps, each partitioning command sequences along different axes, similar to how domain-specific grammars and templates constrain invariant and rule discovery in network telemetry [14, 16]. We start by extracting sequences that match high-level *templates* which aim to frame a semantically meaningful job that a student completed, such as fixing a misconfiguration, effectively reducing its size. Hence, templates are composed

of interleaved sequences of *anchor* points and *unordered sets* of commands. Anchors aim to identify a change in state (*e.g.,* through a write–config change) or a trigger to action (*e.g.,* through a read–failed ping). Once sequences are partitioned and grouped under a given template, we proceed to the second step, where we identify concrete patterns by specifying more variables of a given template. For example, we can pin down device, action, and resource type. Through this process, template #2 shown in Table 3 eventually leads to mining the patterns shown in Table 4. To illustrate the usefulness of templates, recall in our running example that Student A ran their IP and gateway verification commands after their configurations, whereas Student B interleaved theirs. Because template #1 in Table 3 starts with an unordered set, it captures the configuration pattern of both students for this host.

Clean-up mined patterns via selective re-runs: Mined patterns identify commands that frequently occur together across our recordings. However, not all commands within a pattern contribute to the observed outcome. Furthermore, because of challenges related to recording and processing student terminal sessions, our command records may be incomplete. Without additional reasoning, it is hard to determine statically whether a mined pattern is *sufficient* to produce the observed outcome, and which of its commands are *necessary* for doing so. We overcome this by leveraging the reproducibility of the mini-Internet. For instance, in our running example we wish to determine which of the write commands in the pattern we mined are *sufficient and necessary* for A_EUH to ping host A_CHA which is connected to it via the same switch and has an IP address in the same /24 prefix. We restore the network state to the point in time before the student configured A_EUH, and evaluate every combination of the two `create` commands, followed by a ping from A_EUH to A_CHA and an environment reset. After these experiments, we can conclude that configuring only A_EUH's interface with an IP address was sufficient for it to successfully ping A_CHA, and that the default gateway configuration was not necessary for this objective.

4 INITIAL PROMISE

We demonstrate the feasibility of our method and showcase the potential of the patterns we mined from our class' mini-Internet environment. Our setup contains 78 virtual ASes, with 26 containing identical 26-device (*e.g.,* router, switches, hosts) topologies, and managed by student groups of 2 or 3.

Claim #1: Patterns exist and can be mined. Applying our pattern mining method to traces collected from the mini-Internet yields hundreds of recurring patterns, demonstrating that student behavior contains mineable structure. We illustrate this using the three example templates shown in Table 3. Template #1 targets *device configuration* patterns such as that of the example in Section 3. It is comprised of an unordered set of up to 10 commands that read or modify network state on a given device, followed by the first successful probe command executed on the same device. This template occurs 2087 times in our dataset. Template #2 mines *misconfiguration diagnosis* patterns: an anchor that modifies a specific resource on a given device is followed by commands that probe or read the network state, and are finally followed by an anchor command that overwrites the configuration of the same resource on the same device. This template manifests 3678 times in total in our data. Template #3 targets patterns that fix *connectivity issues*: a failed probe anchor is immediately followed by commands that modify network state. This template occurs 4325 times in the dataset. Across these three templates alone, our method discovers more than 1,500 unique patterns, indicating that student terminal sessions exhibit structure that can be mined automatically.

Claim #2: Mined patterns are non-trivial RCA benchmarks. We evaluate two of our mined misconfiguration patterns as RCA benchmarks using GPT-5.4 mini and GPT-5.5-based RCA agents. Despite being provided with detailed network documentation and full access to the topology, the agents often struggle to identify the root causes of the faults. Our L2 Subnetting misconfiguration captures an error where students configure two router subinterfaces with overlapping IP prefixes, causing the traffic to one of the downstream VLANs to be dropped. Our BGP traffic engineering (BGP TE) misconfiguration requires configuring AS_PATH prepending to correctly steer inbound traffic. As shown in Table 5, the GPT-5.4 mini agent successfully localized the L2 Subnetting misconfigurations in only 60% of runs, and failed to do so for the BGP TE misconfiguration. The RCA success rates were likewise 60% and 0% respectively. These failures occurred despite substantial efforts: across 5 diagnosis runs for each misconfiguration, the agent took 10 steps, made more than 15 tool invocations, and consumed over 69,000 and 91,000 input tokens for the L2 subnetting and BGP TE misconfigurations respectively. Using a more capable frontier model, GPT-5.5, does not eliminate this difficulty. GPT-5.5 localizes the L2 subnetting misconfiguration in only 20% of runs despite using an average of 31.8 tool calls and near 110,000 input tokens per run. On the BGP TE misconfiguration, it only reaches 100% accuracy by brute-forcing the diagnosis using 17.6 steps and 48.6 tool calls while consuming over 221,000 input tokens and 18,000 output tokens.

Claim #3: Mined runbooks can guide RCA agents. We augment our RCA agents with access to runbooks that we automatically generated from the patterns corresponding to the mined misconfigurations and re-evaluate its performance. Each runbook is mined from a single pattern occurrence, and does not depend on the specifics of that occurrence (e.g.,

#	Template	# Occurrences
1	**S[**d:d', a:"(create)|(read)|(update)", r:*]**{10}** ← **A[**d:d', a:"probe", r:*, probe.success: True, order(probe): 1]	2087
2	**A[**d:d', a:"(create)|(update)", r:r'] - **S[**d:d', a:"(read)|(probe)", r:*]**{10}** ← **A[**d:d', a:"(create)|(update)", r:r']	3678
3	**A[**d:*, a:"probe", r:*, probe.success: False] → **S[**d:*, a:"(create)|(update)|(delete)", *, *]**{10}**	4325

Table 3: Example templates and number of occurrences. Notation: A - anchor command, S - unordered commands set. ← /→ - search order for S commands w.r.t the attached A. A, and S are parameterized by [d(evice), a(ction), r(esource), *other args.*]. S are also parameterized by their maximum size noted in {}.

Misconfiguration	Pattern	# Occurrences
L2 Subnetting	[d:"ATLrouter", a:"create", r:"address"] - [[d:*, a:"probe", r:"state"], [d:*, a:"read", r:"state"]] - [d:"ATLrouter", a:"delete", r:"address"]	41
BGP TE	[d:"MSProuter", a:"update", r:"protocol"] - [[d:*, a:"probe", r:"state"], [d:*, a:"read", r:"state"]] - [d:"MSProuter", a:"update", r:"protocol"]	89

Table 4: Patterns used to mine the L2 subnetting and BGP traffic engineering misconfigurations.

Model	Misconfiguration	Runbooks	Steps	# Tools	# In Tokens	# Out Tokens	DR(%)	LR(%)	RCA(%)
GPT-5.4 mini	L2 Subnetting	False	10.0	15.4	69302.8	890.4	100.0%	60.0%	60.0%
	L2 Subnetting	True	5.8	11.6	47096.8	664.2	100.0%	100.0%	100.0%
	BGP TE	False	10.6	15.6	91566.2	775.6	20.0%	0.0%	0.0%
	BGP TE	True	9.2	11.2	56882.2	611.4	100.0%	80.0%	100.0%
GPT-5.5	L2 Subnetting	False	13.6	31.8	109778.6	9186.8	60.0%	20.0%	20.0%
	L2 Subnetting	True	15.4	40.6	134159.4	8139.0	100.0%	100.0%	100.0%
	BGP TE	False	17.6	48.6	221144.0	18274.0	100.0%	100.0%	100.0%
	BGP TE	True	5.8	7.6	32444.8	1914.2	100.0%	100.0%	100.0%

Table 5: Mined misconfigurations are challenging benchmarks for RCA agents. Providing the agent with access to runbooks automatically generated from mined patterns leads to improved performance and/or decreased token usage. Notation: DR - detection success rate, LR - localization success rate, RCA - root cause analysis success rate.

ASN, exact IP addresses). We keep runbook generation and evaluation data separate by mining the runbook from an occurrence in one student group's AS and evaluating it on the same misconfiguration in a different group's AS not used for runbook generation. In this setting, the runbooks substantially improve agent performance or decrease utilized tokens, or both (Table 5). The agents now achieve high RCA success rate on both L2 subnetting and BGP traffic engineering. The runbooks also reduce the effort required for diagnosis: the GPT-5.4 mini agent working on the L2 subnetting misconfiguration required approximately 6 steps and 12 tool calls, while consuming 32% fewer tokens. For the BGP traffic engineering misconfiguration, the agent required more than a step less, just over 11 tool calls and 38% fewer input tokens. The mined runbooks also strongly benefit GPT-5.5, lifting RCA success rate on the L2 subnetting misconfiguration from 20% to 100%. On the BGP traffic engineering misconfiguration, the runbook maintains GPT-5.5's 100% RCA accuracy and reduces diagnosis from 17.6 to 5.8 steps, and tool calls from 48.6 to 7.6, while reducing input tokens by 85%. These results suggest that mined patterns can be transformed into useful troubleshooting guidance for RCA agents.

5 RELATED WORK

Network troubleshooting and AI benchmarks: A long line of work automates fault diagnosis, anomaly detection, and network management. Prior systems localize faults from production telemetry such as endpoint TCP statistics [2], and end-host signals [1]. Recent systems drive operations with LLMs and learning-based controllers. These span failure localization in production networks [34], dialogue-based diagnosis [35], intent-driven management at hyperscale [37], management tasks code generation [23], incident management [12], and formal-methods guardrails that translate network artifacts into symbolic models or enforce runtime safety [10, 14, 15]. To measure progress, benchmarks such as NIKA replay curated troubleshooting incidents [36], and NetConfEval scores configuration synthesis from natural-language intent [33]. These systems learn from proprietary production incidents that are never released, and the benchmarks rely on author-curated injected faults. Neither pairs a fault with the commands, hypotheses, and dead ends of the human process, which is the signal we aim to capture.

Educational and emulation platforms: Reproducible environments for building and running networks are mature and broadly used in academia [17, 20, 22, 28, 30, 31]. Mininet prototypes single-machine SDNs [21], Emulab and GENI provision controlled experiments at scale [5, 38], and Kathará assembles container-based topologies [6]. The mini-Internet brings the same reproducibility to teaching, letting students jointly operate a miniature Internet [17]. We repurpose such a platform into an instrument that records and mines how

its operators reason. The platforms supply the substrate, but none captures the human debugging process.

Operational datasets and operator studies: Understanding real networks has relied on production data held by a single organization. Telemetry systems such as Pingmesh continuously measure large data centers [11], and measurement studies characterize why networks fail [9, 27]. Related work quantifies configuration and management complexity [4], analyzes the data plane implied by configurations [8], and mines network invariants to synthesize traces, and audit source-level privacy leakage [14, 16, 19]. Field studies instead observe how administrators work in practice [3]. Each source is locked to one provider, summarizes outcomes after the fact rather than the diagnostic trajectory, or reports qualitative findings that yield no labeled, minable traces.

Mining the human process: Our approach builds on educational data mining, where capturing the process rather than the final artifact has proven its worth. Repositories such as Blackbox record millions of novice programming sessions [7], compilation behavior has been mined across tens of thousands of users [18], and program trajectories drive automated feedback [29]. We carry the same principle into network operations. Sequential pattern mining offers a natural starting point for finding recurring structure [13, 24]. Standard methods assume a single ordered stream, however, and cannot cleanly separate the interleaved, concurrent sessions of our setting, motivating template-based mining (§3).

6 FUTURE AGENDA

What happens once the mini-Internet structure is gone? Our abstraction leans on two properties of the mini-Internet: *(i)* every group configures an AS with the same internal topology, so a name resolves to a position and a position to a role; *(ii)* every group completes the same assignment, giving our templates a frequency signal. Yet, a production network might not be symmetrical enough to resolve names to roles, and tasks in production are far more diverse. Finding abstractions is therefore an open question. Roles might come from text and metadata: textbooks, vendor documentation, and RFCs already describe networks the way we want to abstract them, naming roles rather than devices and decomposing tasks into canonical rules and invariants [16]. Significance through frequency might need to be replaced by significance through authority, where an observed sequence executed by an expert serves as a candidate pattern without requiring high repetition. Further focusing on patterns anchored in data-plane signals or observable forwarding effects offers a promising direction because those signals transfer across topologies: a newly failing probe or path degradation means the same thing regardless of which command configured it.

Networking courses as research infrastructure: Our work suggests that educational networking platforms can serve a second purpose beyond instruction: producing realistic, reproducible, and readily shareable operational datasets. This dual role raises a broader question for our community: how can we co-design networking courses to simultaneously optimize student learning and the generation of high-fidelity research artifacts? By introducing standardized topologies, reproducible assignments, and privacy-preserving instrumentation [14, 19], multiple institutions could collaboratively build corpora of configuration tasks, debugging trajectories, and failures that cannot be obtained from production networks. Such datasets would also directly benefit networking education by revealing misconceptions and instruction gaps.

Aligning academic and industry environments: Instruction environments serve students best, and yield more realistic and representative research artifacts when their design is informed by industry practice. Perfectly aligning the two is impossible: students are not network engineers, universities lack industry-scale resources, industry environment details are scarcely disclosed, and even within industry, artifacts from one setting may not generalize to another. Nevertheless, artifacts collected in academic environments remain valuable. A key research challenge is to identify which operational environment properties (e.g., scale, failure modes, objectives, troubleshooting constraints) matter for the studied behaviors, and how to incorporate them in educational platforms. This may involve scaling emulation platforms, extending their protocol and application support, and incorporating production-inspired constraints (e.g., SLAs, erroneous routing costs). Educational tasks could also be extended to encompass additional operations faced by network engineers (e.g., upgrading infrastructure, migrating live services). The impact of these design choices on mined datasets can be evaluated across variations of the same instruction environment and, where possible, using industry knowledge.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation (NSF) through Grants CNS-2442625 and CNS-2319442 and Office of Naval Research (ONR) through Grant N000142612552.

REFERENCES

[1] Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang Harry Liu, Jitu Padhye, Boon Thau Loo, and Geoff Outhred. 2018. 007: Democratically Finding the Cause of Packet Drops. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 419–435. https://www.usenix.org/conference/nsdi18/presentation/arzani

[2] Behnaz Arzani, Selim Ciraci, Boon Thau Loo, Assaf Schuster, and Geoff Outhred. 2016. Taking the Blame Game out of Data Centers Operations with NetPoirot. In *Proceedings of the 2016 ACM SIGCOMM Conference*. Association for Computing Machinery, 440–453. https:

//doi.org/10.1145/2934872.2934884

[3] Rob Barrett, Eser Kandogan, Paul P. Maglio, Eben M. Haber, Leila A. Takayama, and Madhu Prabaker. 2004. Field Studies of Computer System Administrators: Analysis of System Management Tools and Practices. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work (CSCW)*. Association for Computing Machinery, 388–395. https://doi.org/10.1145/1031607.1031672

[4] Theophilus Benson, Aditya Akella, and David A. Maltz. 2009. Unraveling the Complexity of Network Management. In *6th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 335–348. https://www.usenix.org/conference/nsdi-09/unraveling-complexity-network-management

[5] Mark Berman, Jeffrey S. Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. 2014. GENI: A Federated Testbed for Innovative Network Experiments. *Computer Networks* 61 (2014), 5–23. https://doi.org/10.1016/j.bjp.2013.12.037

[6] Gaetano Bonofiglio, Veronica Iovinella, Gabriele Lospoto, and Giuseppe Di Battista. 2018. Kathará: A container-based framework for implementing network function virtualization and software defined networks. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. 1–9. https://doi.org/10.1109/NOMS.2018.8406267

[7] Neil C. C. Brown, Michael Kölling, Davin McCall, and Ian Utting. 2014. Blackbox: A Large Scale Repository of Novice Programmers' Activity. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE)*. Association for Computing Machinery, 223–228. https://doi.org/10.1145/2538862.2538924

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

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

[10] Fengchen Gong, Divya Raghunathan, Aarti Gupta, and Maria Apostolaki. 2024. Zoom2Net: Constrained Network Telemetry Imputation. In *Proceedings of the ACM SIGCOMM 2024 Conference (SIGCOMM '24)*. Association for Computing Machinery.

[11] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, Zhi-Wei Lin, and Varugis Kurien. 2015. Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis. In *Proceedings of the 2015 ACM SIGCOMM Conference*. Association for Computing Machinery, 139–152. https://doi.org/10.1145/2785956.2787496

[12] 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 (HotNets)*. Association for Computing Machinery, 180–188. https://doi.org/10.1145/3626111.3628176

[13] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, Meichun Hsu, et al. 2001. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *proceedings of the 17th international conference on data engineering*. IEEE Piscataway, NJ, USA, 215–224.

[14] Hongyu Hè, Minhao Jin, and Maria Apostolaki. 2026. Making Logic a First-Class Citizen in Generative ML for Networking. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*. USENIX Association. https://www.usenix.org/conference/nsdi26/presentation/he

[15] Hongyu Hè, Minhao Jin, and Maria Apostolaki. 2027. Worst-Case Discovery and Runtime Protection for RL-Based Network Controllers. In *24th USENIX Symposium on Networked Systems Design and Implementation (NSDI 27)*. USENIX Association.

[16] Hongyu Hè, Alexander Krentsel, Sylvia Ratnasamy, and Maria Apostolaki. 2026. Invariant Discovery for Networked Systems. In *Proceedings of the 25th ACM Workshop on Hot Topics in Networks (HotNets '26)*. Association for Computing Machinery.

[17] Thomas Holterbach, Tobias Bühler, Tino Rellstab, and Laurent Vanbever. 2020. An Open Platform to Teach How the Internet Practically Works. *SIGCOMM Comput. Commun. Rev.* (2020). https://doi.org/10.1145/3402413.3402420

[18] Matthew C. Jadud and Brian Dorn. 2015. Aggregate Compilation Behavior: Findings and Implications from 27,698 Users. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research (ICER)*. Association for Computing Machinery, 131–139. https://doi.org/10.1145/2787622.2787718

[19] Minhao Jin, Hongyu Hè, and Maria Apostolaki. 2027. Cross-Flow Correlations Survive Synthesis: Measuring Source-Level Privacy Leakage in Synthetic Network Traces. In *Network and Distributed System Security Symposium (NDSS)*.

[20] Kathara. 2026. Success Stories. https://www.kathara.org/stories.html.

[21] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*. Association for Computing Machinery, Article 19, 6 pages. https://doi.org/10.1145/1868447.1868466

[22] Tie Ma, Long Luo, Hongfang Yu, Xi Chen, Jingzhao Xie, Chongxi Ma, Yunhan Xie, Gang Sun, Tianxi Wei, Li Chen, et al. 2024. Klonet: an {Easy-to-Use} and scalable platform for computer networks education. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 2025–2046.

[23] Sathiya Kumaran Mani, Yajie Zhou, Kevin Hsieh, Santiago Segarra, Trevor Eberl, Eliran Azulai, Ido Frizler, Ranveer Chandra, and Srikanth Kandula. 2023. Enhancing Network Management Using Code Generated by Large Language Models. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks (HotNets)*. Association for Computing Machinery, 196–204. https://doi.org/10.1145/3626111.3628183

[24] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. 1995. Discovering Frequent Episodes in Sequences. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD)*. AAAI Press, 210–215. https://aaai.org/papers/kdd95-024-discovering-frequent-episodes-in-sequences/

[25] Kulik Marcin. 2026. asciinema. https://github.com/asciinema/asciinema.

[26] J. Martin. 1983. *Managing the Data-base Environment*. Prentice-Hall. https://books.google.com/books?id=nMgmAAAAMAAJ

[27] Justin Meza, Tianyin Xu, Kaushik Veeraraghavan, and Onur Mutlu. 2018. A Large Scale Study of Data Center Network Reliability. In *Proceedings of the Internet Measurement Conference (IMC)*. Association for Computing Machinery, 393–407. https://doi.org/10.1145/3278532.3278566

[28] mininet. 2021. Teaching and Learning with Mininet. https://github.com/mininet/mininet/wiki/Teaching-and-Learning-with-Mininet.

[29] Chris Piech, Jonathan Huang, Andy Nguyen, Mike Phulsuksombati, Mehran Sahami, and Leonidas Guibas. 2015. Learning Program Embeddings to Propagate Feedback on Student Code. In *Proceedings of the 32nd International Conference on Machine Learning (ICML) (PMLR, Vol. 37)*. PMLR, 1093–1102. https://proceedings.mlr.press/v37/piech15.

html

[30] Maurizio Pizzonia and Massimo Rimondini. 2016. Netkit: network emulation for education. *Software: Practice and Experience* 46, 2 (2016), 133–165.

[31] SEED Labs. 2026. SEED Labs around the world. https://seedsecuritylabs.org/adoptions/.

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

[33] 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, Article 7 (2024), 25 pages. https://doi.org/10.1145/3656296

[34] Chenxu Wang, Xumiao Zhang, Runwei Lu, Xianshang Lin, Xuan Zeng, Xinlei Zhang, Zhe An, Gongwei Wu, Jiaqi Gao, Chen Tian, Guihai Chen, Guyue Liu, Yuhong Liao, Tao Lin, Dennis Cai, and Ennan Zhai. 2025. Towards LLM-Based Failure Localization in Production-Scale Networks. In *Proceedings of the ACM SIGCOMM 2025 Conference* (São Francisco Convent, Coimbra, Portugal) *(SIGCOMM '25)*. Association for Computing Machinery, New York, NY, USA, 496–511. https://doi.org/10.1145/3718958.3750505

[35] Haopei Wang, Anubhavnidhi Abhashkumar, Changyu Lin, Tianrong Zhang, Xiaoming Gu, Ning Ma, Chang Wu, Songlin Liu, Wei Zhou, Yongbin Dong, Weirong Jiang, and Yi Wang. 2024. NetAssistant: Dialogue Based Network Diagnosis in Data Center Networks. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 2011–2024. https://www.usenix.org/conference/nsdi24/presentation/wang-haopei

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

[37] Zhaodong Wang, Samuel Lin, Guanqing Yan, Soudeh Ghorbani, Minlan Yu, Jiawei Zhou, Nathan Hu, Lopa Baruah, Sam Peters, Srikanth Kamath, Jerry Yang, and Ying Zhang. 2025. Intent-Driven Network Management with Multi-Agent LLMs: The Confucius Framework. In *Proceedings of the ACM SIGCOMM 2025 Conference*. Association for Computing Machinery, 347–362. https://doi.org/10.1145/3718958.3750537

[38] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. 2002. An Integrated Experimental Environment for Distributed Systems and Networks. In *5th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 255–270. https://www.usenix.org/conference/osdi-02/integrated-experimental-environment-distributed-systems-and-networks