A Holistic View of AI-driven Network Incident Management

Pouya Hamadanian; Behnaz Arzani; Sadjad Fouladi; Siva Kesava Reddy Kakarla; Rodrigo Fonseca; Denizcan Billor; Ahmad Cheema; Edet Nkposong; Ranveer Chandra

Abstract — We discuss the potential improvement large language models (LLM) can provide in incident management and how they can overhaul how operators conduct incident management today. Despite promising results, initial work in this space only scratched the surface of what we can achieve with LLMs. We instead propose a holistic framework for building an AI helper for incident management and discuss the several avenues of future research needed to achieve it.

We support our design by thoroughly analyzing the fundamental requirements the community should think about when it designs such helpers. Our work is based on discussions with operators of a large public cloud provider and their prior experiences both in incident management and with attempts that have tried to improve the incident management experience through various forms of automation.

1 Introduction

"No design works unless it embodies ideas that are held common by the people for whom the object is intended."
—Adrian Forty

Frequent *incidents*—failures that compromise the reliability of essential services—pose a challenge for large-scale cloud operations [20, 24]. Prior research has focused on the Incident Management (IM) process which finds and mitigates these incidents quickly [1, 10, 13, 22, 26, 46]. This paper presents a network operations view of how Large Language Models (LLMs) can accelerate the IM process and reduce the impact of incidents on cloud tenants. We discuss the requirements that any LLM-based solution for IM must meet.

When automation or customers report an incident, the incident manager assigns it to an On-Call Engineer (OCE) who investigates the issue, mitigates it (*e.g.,* moves traffic away from the problematic area or takes a failed device offline), finds the root cause, and fixes it. The mitigation step is the most important: it stops the incident's impact on customers. Providers view Time to Mitigation (TTM) as the main indicator of efficiency and strive to keep it within 30 minutes (for more details on the IM process, see prior work [13, 22, 24]).

The IM process is time-consuming, exhausting, and stressful for the OCE because the set of possible root causes and monitoring data is large. OCEs also need to understand a diverse set of systems and their components and go through multiple hours of training to develop enough expertise to understand how different components interact.

A prevailing question is: should we utilize LLMs to improve the IM process [1, 13, 46], or in other words, create an LLM-powered OCE-helper? LLMs are well positioned to help address the challenges in this space. Works such as [8] show how LLMs can ingest large datasets, generate structured insights from unstructured text, automate workflows, and generate human-like written content: they introduce an unexplored design continuum in systems research. LLMs such as GPT-3.x [7] and PaLM [14] have already helped with tasks such as code generation [12], formal verification [21], database query generation [31, 48]. At the same time, LLMs struggle with long-term planning [8], and are prone to hallucination [8, 41].

Our answer to this question—whether one *should* use LLMs in this context—is nuanced: due to the unstructured format of IM, OCE-helpers are infeasible without LLMs, *but* cannot solve the end-to-end problem directly. We, as networking researchers with IM experience, investigated feasible OCE-helpers and the requirements for a successful design. This led us to a modular design where we incorporate LLMs to help in different stages of the IM workflow. Our design bears a similar insight to goal-driven LLM agents [37, 47, 55].

While the implications may not be as severe, IM process shares similarities to medical diagnosis. Operators (medical practitioners) want to mitigate incidents (treat symptoms), understand what caused the incident, and fix (cure) the problem. The space of possible hypotheses (underlying diseases) is large, and they cannot test each and every one. Instead, they pick a likely hypothesis, do targeted tests, and reassess their initial guess—doing so from incident summaries (symptoms) is non-trivial. They also have to be careful in how they apply mitigations (treatments) to avoid further harm to the system (patient)—the risks involved mean an OCE-helper cannot replace an operator (medical practitioner) outright. This is why we design the OCE-helpers as assistants instead of replacements for OCEs.

We interviewed multiple OCEs with 5+ years of experience and identified three key requirements for any OCE-helper, which we outline and motivate through examples based on real incidents. Initial work on such helpers shows promise [1, 13], but to realize their full potential, we have to take a holistic view of IM and identify how helpers can meet the fundamental requirements described in this work.

The core requirements we found for an OCE-helper are that it should be: *iterative*, *reliable and safe*, and *adaptive*. An iterative helper forms and tests hypotheses and recommends mitigation strategies through a multi-step process. A reliable and safe helper does not cause extra harm to the network, and *adaptive* helper can evolve and adjust as operators deploy new components, make configuration changes, and discover new issues. We discuss these requirements in depth.

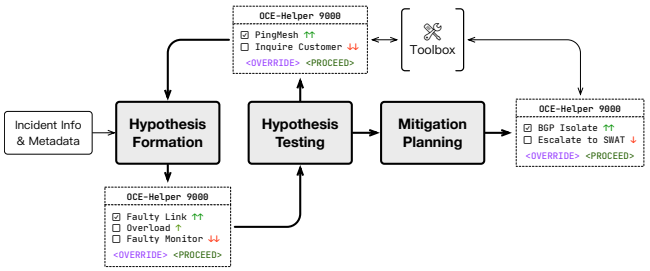

Figure 1: Our framework consists of three modules: hypothesis formation, hypothesis testing, and mitigation planning.

We propose a framework that meets these key requirements (§4) and explain the practical considerations and insights that led us to it. This framework (Figure 1) is analogous to OCE's natural thought process and includes three modules: hypothesis former, hypothesis tester, and mitigation planner. The hypothesis former offers plausible explanations for an incident, akin to forming a scientific hypothesis. The hypothesis tester aims to test these hypotheses, querying available monitors and tools and interpreting the results. The mitigation planner creates a mitigation plan based on the validated hypotheses, which the OCE can execute at their discretion.

This framework is ambitious, and the community needs further research to realize it in areas such as: (1) network-specialized LLMs to serve as backbones to these modules; (2) network-specialized embedding models that help knowledge-retrieval methods (*e.g.,* vector databases); and (3) structural approaches that breakdown complex tasks in network management (*e.g.,* isolate links) to LLM-digestible steps. The community needs to approach problems such as risk estimation and safety [2, 38] differently and think through how to: (1) analytically evaluate the end-to-end risk of the LLM proposals; (2) define abstractions between LLMs and existing tools to gracefully balance the trade-off between automation and safety; and (3) create verifiable LLM-based tools (*e.g.,* text-to-SQL [48]).

Our goal for this paper is to motivate the community to further explore this space and think about the implications such solutions have for network management and beyond. We hope that this work inspires novel research on one of the largest bottlenecks of modern cloud infrastructure.

2 Foundational Principles: Design

We analyzed real incidents across various teams, services, and time-frames and interviewed OCEs with several years of experience, with a focus on their day-to-day operations, pain points, and where they thought automation could help. We studied prior work, explored the different designs for OCE-helpers, and then ran through what-if scenarios where we took a specific design for an OCE-helper, replayed historical incidents and observed how this hypothetical helper would integrate into the IM workflow. Three main themes emerge which we believe are foundational principles any such design should meet: iterative prediction, reliable and safe, and adaptive.

In hindsight, these principles seem obvious; yet prior works do not fully meet them (Table 1).

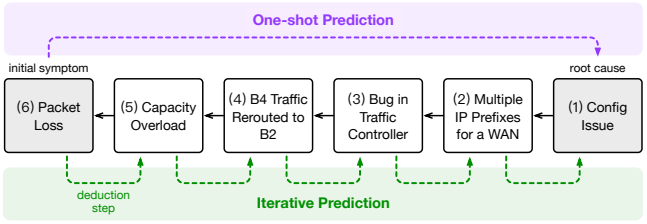

Figure 2: Iterative vs. one-shot design. One-shot predictors need to make logical leaps with no extra information. Iterative chaining mimics the natural thought process of OCEs.

Iterative Prediction. It is simple to formulate this problem as a Machine Learning (ML) prediction problem which entices us to train a one-shot feed-forward model that predicts root cause or mitigation, conditioned on past incidents and telemetry we carefully engineer: prior work [1, 13] takes the (predefined) incident information (*e.g.,* title, summary, and the auto-generated digest of system health and telemetry) and predicts the root cause or mitigation without any additional context. These approaches are promising: they work well with incidents similar to those resolved in the past, or where operators have fleshed out clean monitoring signals. But they fail with more complex (often more impacting[1]) or novel incidents.

Challenging incidents deviate from known patterns, are sparse, and often happen due to a complex chain of events. There is not enough information in the initial incident summary of many networking incidents to find the right mitigation or root cause in one-shot—operators need to make multiple informed attempts to test various hypotheses, *safely* mitigate the incident, zero in on the root cause, and resolve the problem. The one-shot approach falls short in these cases, and such a design is inherently too restrictive for the IM workflow.

Consider the `Casc-1` incident from a report on Google Cloud incidents [24] (Figure 2). At the time, Google's network was comprised of two Wide Area Networks (WANs), namely B2 and B4. During a network upgrade, a transient configuration inconsistency (event 1) caused more than one cluster to observe B4 with several IP prefixes (event 2). The traffic controller that managed traffic across B2 and B4 mistakenly interpreted that as B4's failure (event 3) and rerouted all B4 traffic through B2 (event 4). This led to an overload (event 5) and packet loss (event 6). A one-shot predictor has to infer event 1 after it observes event 6, and a naïve OCE-helper that only observes predefined information would decide it is either a device failure (*e.g.,* a switch) or a transient increase in traffic.

An OCE checks both of these suggestions first, but would quickly reassess after they gather more evidence that shows both are incorrect: high traffic utilization of B2 (event 5) and no traffic for B4. They would hypothesize and confirm that traffic is intentionally rerouted away from B4 (event 4). This cascade of hypothesis, testing, and reassessment would continue until they discover the inconsistency and the bug. We believe an

[1]These works don't report accuracy per severity of incidents they studied and this discussion is based on our experience in IM.

OCE-helper should behave the same, *i.e.,* only an OCE-helper that hypothesizes, tests, and re-evaluates its decisions in a feedback loop can successfully resolve such complex incidents.

Reliable & Safe. When we deploy the OCE-helper we need to be sure it won't make mistakes with $100M in damages [20] on its first day: unlike past research that applied ML to networking problems, risk assessment cannot be an afterthought—we cannot wait until the problems happen before we assess the safety of the model's suggestions. Even smaller mistakes can drive operators away—if they can't reliably show when drastic cases happen, they will bypass the helper with the mere threat of a mistake [22]: helpers need to be *safe* (should not worsen the situation) and *reliable* (should not have inconsistencies that introduce mistakes in the workflow).

Helpers need to include risk mitigating mechanisms both baked in—where it provides a reason for why it arrived at a particular response (these explanation mechanisms often improve LLM responses [52])—and on the outside, *e.g.,* a wrapper around the helper which statistically analyzes the mitigation outcome the helper proposes and its effect on the network [38]. We should not just analyze what helpers suggest: we should explicitly design the helper to take in feedback from internal and external risk assessors, and search for actions with lower risks.

If we intend to eventually automate OCE-helpers, confidence and risk measures are non-negotiable. We discuss why prior work does not enable such a solution and the research the community needs to fill the gap in §4.4.

Adaptive. A core challenge with IM is that network components' software and hardware evolve rapidly—often teams don't explicitly coordinate with each other when they update components they own. Such evolution invalidates prior incidents and mitigation strategies, and uncoordinated changes lead to new incidents [5, 24]. Such "new" incidents are complex and have the highest negative impact, but also where we see the most potential for helpers to improve the IM workflow and where we believe research in this space should focus.

Consider a failure reported in AWS Direct Connect Tokyo [44] (Figure 3). This service provides low-latency consistent tunneling to the Tokyo region. New software and a perfect storm of events led to this failure, where customers saw latency spikes and packet loss. Operators deployed a new protocol that reacted faster to network failures. This protocol had an unknown defect triggered by a specific set of packet headers and payloads. Months later, traffic from one customer happened to fit that pattern, and several network devices failed to forward traffic. Operators first removed these devices to mitigate the incident, but the same failure popped up again on other devices. Operators eventually realized the culprit, disabled the protocol, and resolved the defect. This was a unique incident, partly because the events that led to it involved new and untested protocols. No amount of historical incidents could supply a helper with the knowledge to mitigate such an incident.

It is inherently difficult to adapt to new incidents because there is little "learn-able" data in such sparse novel scenarios: we know the changes, but are unaware what impact these

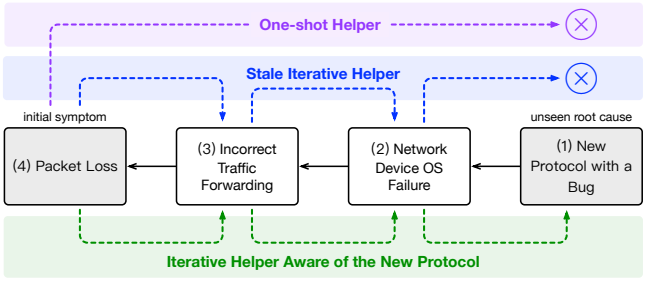

Figure 3: Helper design affects how quickly we can adapt. One-shot methods need to learn similar incidents (fine-tune or in-context) before it can help. Iterative methods only need to learn incremental changes.

changes may cause until they happen. We cannot use end-to-end approaches (*e.g.,* one-shot helpers) for new incidents since they need end-to-end samples the change causes. An iterative helper that explicitly *reasons* about individual components and how operators change them can gradually build towards the right mitigation. Operators only need to update this helper with the new behavior of the system (*e.g.,* either fine-tune on new protocol documents, or add the relevant new protocols to context), and not its impact. This helper could evolve almost as quickly as OCEs to any changes in the infrastructure, guidelines, and incidents, and does not need end-to-end samples.

An OCE-helper that safely assists with critical incidents needs to satisfy these three principles. While they are stringent, and there is no current materialized automated OCE-helper that satisfies them, we have designed a framework that addresses all of them systematically, which we explore in §4. But first, we describe how we can verify and measure whether any such framework can satisfy these requirements.

3 Foundational Principles: Evaluation

It is hard to evaluate solutions in this space. Prior works' metrics (*e.g.,* F_1 score, semantic similarity) paint an incomplete picture of the impact these helpers have on metrics operators care about: how fast they resolve the incident and whether they can do so safely—alone, they will not reveal the helper's ability to meet our requirements. Instead, we need to track: (1) TTM; and (2) the overheads the helper's mistakes induce (new incidents caused by wrong mitigation plans, SLA violations, etc).

We can A/B test a helper to conduct an end-to-end evaluation of the target system, *i.e.,* randomly assign incidents to either a helper-enhanced intervention or to a helper-free control group. This is the most robust evaluation we can get and with enough cases and statistical tests, we can compare the two groups at any statistic (e.g., average TTM). This is nonetheless challenging, because of the high variability in the nature of each incident, among OCE's expertise, and other confounding factors.

It is hard to estimate the overhead of the helper's mistakes—the complex interactions between the system state and the mitigation make it hard to model; A/B tests are one mechanism to do so but they are also costly and invasive. Many incidents (*e.g.,* customer-reported ones) are diverse and heavy-tailed:

Table 1: Foundational principles in prior work.

	Iterative	Reliable & Safe	Adaptable	End-to-end Evaluation
Ahmed *et al.* [1]	✘	✘	✘	✘[2]
RCACopilot [13]	✘	✘	✓	✘

we need a long test span to collect enough measurements.

We can instead 'replay' historical incidents and compare the replayed TTM to the original; we can use this to scale up the measurements. We can query telemetry retroactively, but this will not work if the mitigation the helper suggests differs from the one the operator used in the original incident.

We can naïvely solve this problem and report the TTM savings for incidents where the mitigations match and the fraction where this was not the case. Can we do better? We can find past incidents where operators used the same mitigation the helper proposed and estimate the impact of that mitigation on the TTM distribution—this measure will be, by definition, approximate: the semantics of the specific incidents may be different which will impact the TTM. We need further research to formalize how we derive this estimate, and what to condition it on.

Besides TTM savings, operators need *efficient* OCE-helpers—they cannot cost more than the SLA violations the incidents induce. The infrastructure cost to train (or fine-tune) and run inference constitute the *system cost* of the helpers—research such as [17, 45, 50] which make ML training and inference more efficient can help reduce this cost. Work on OCE-helpers should report these costs.

Research on OCE-helpers should also report management costs. Each time operators have to research how to adapt to new incidents or maintain a reliable and performant solution they add to the management cost of the OCE-helper. Prior to this work, we considered an OCE-helper that automates well-structured incidents that have a clear resolution strategy—in these cases OCEs follow a detailed Troubleshooting Guide (TSG) to mitigate and resolve the incident and the LLM can do the same—but the cost outweighs the benefit: to make this (seemingly simple) solution work we had to integrate LLMs with monitoring APIs, put guard-rails to minimize damages that results from mis-predictions, and even carefully design prompts to make sure the LLM exactly follows the TSG. We can achieve the same goal with a hard-coded Python script (in fact, operators already do)! A change in the TSG necessitates a change in both solutions, and the cost would not amortize.

4 Framework

We present a framework that meets the foundational principles we discussed. It consists of three modules: a hypothesis former, a hypothesis tester, and a mitigation planner. Together, these modules replicate an OCE's thought process when resolving an incident. We discuss where and why LLMs fits within this framework. This framework needs further research to materialize, which we will discuss. We invite the community to also propose other viable options in this space.

4.1 Our Perspective

To provide context for certain design choices, we discuss our design perspective. While these choices are not fundamentally necessary (unlike the principles in §2), they are well-motivated based on firsthand experience.

OCE-centric design. IM workflows are too chaotic and intricate to fully automate. We believe the helper should act as a "copilot" and suggest the next steps but keep the OCE in the driver's seat. The OCE remains responsible for the incident and can decide to use helpers only when they deem it useful. This makes it easier for operators to deploy the solution and helps prevent the negative impacts of the helper on the IM workflow when it inevitably makes mistakes.

Decentralized extensibility. 100+ independent networking teams control and execute IM—each team has its own categories of incidents and root causes, guidelines, tools, and documentation. It is impractical for all teams to follow the same formula [34] (especially so if they already have legacy workflows which they have trained their OCE to follow). We instead believe in a design where each team can modify how the helper impacts their workflow but not interfere with other teams (and their preferences) in the process—we want to enable operators to distributedly manage the helper.

Modular design. We split the helper into independent modules—similar to how, early on, researchers split the networking stack into separate layers (*i.e.,* OSI [18]). A well-designed monolithic system can outperform a modular one—e.g., co-optimized application and transport layers outperform the modularized stack [29]—but a modularized design accelerates early research and helps find solutions quickly.

Asset reuse. OCEs use many tools: device health monitors, packet loss detectors, link/device isolation tools, etc. It took operators many years of research and experience to create them, and we designed our framework to leverage them.

4.2 The Case for LLMs

LLMs have a unique ability to help us achieve a holistic OCE-helper design, and perhaps a fully automated solution:

They can reason (kind of). Resolving incidents (or in general, debugging faults) boils down to inferring the *chain of causal relationships* that lead to the incident (*e.g.,* Figure 2). Operators logically understand the components—*e.g.,*, switch failures lead to packet loss—which they use to make these inferences—and backtrack through the possible chain and confirm or reject prior states in the chain to hone in on the correct cause.

We see a similar causal deduction in LLMs [52]: they *parrot* the cause and effect relationships they learned through training on vast corpora of text; *e.g.,* we asked GPT-4 why a VM experienced packet loss, and it provided an exhaustive list including congestion, device failure, misconfiguration, bugs, *etc.* Since OCEs also learn such deduction through training, we can get

[2] Authors ask OCEs to rate how useful responses are, but, as they also mention, this metric is subjective and the scale is small [1].

more detailed responses if we fine-tune LLMs on the same documents and if we look at the mitigation history of previous incidents—this is in line with prior works' observations [1].

We can embed causal deductions statistically in a black-box [3, 4, 22], *e.g.,* statistically model the causal link between device failures and packet loss. But such models are usually not tractable (operators make them explicit and narrow to make them tractable): a monolithic black-box that models all causal relationships needs to observe all telemetry and predict all possibilities. We have too much telemetry to collect or feed to a black-box and the larger input/output sizes increase sample complexities—they need too many training samples. IM involves rare incidents and is structurally sample-starved.

They are technically well-suited to the problem. Unlike prior ML solutions which have fixed input spaces, we can input anything from normal text, to numbers, to system logs, to loose descriptions, or instructions to an LLM. LLMs also quickly and cheaply instance-optimize, *i.e.,* specialize to a specific instant and problem, with in-context learning and can integrate with external tools [43] and plugins [40].

But LLMs are not magic. Computational complexity grows quadratically with token count, and we have input limits (currently 32K tokens $\approx$ 24K words in GPT-4).[3] Mainstream LLMs only accept text, and we need clever tricks for multi-modal inputs (*e.g.,* images, graphs, time series). We still can't use them for long-term planning and reasoning [8, 41]. LLMs are unreliable—they are powerful enough to enable OCE-helpers, but imperfect on their own: they cannot replace the OCE. As researchers, we have to find how best to use them, and we discuss our initial work on this in the next section.

4.3 Overview

The framework we propose has three "LLM agents" (Figure 1): **(1)** the hypothesis former produces bite-sized hypotheses and describes possible root causes or mitigations in each step; **(2)** the hypothesis tester takes a hypothesis as input and generates a procedure to verify it; and **(3)** the mitigation planner creates a detailed mitigation plan. These modules need not use the same techniques: some may use a simple LLM while others more complex goal-driven LLM-agents [47, 55]. Each module presents several suggestions to the OCE who then approves a select few at their discretion. OCEs can pre-approve certain suggestions that have high confidence and low risk. This strict approval process avoids compounding mistakes—one early irrelevant hypothesis can lead the LLM astray for long durations.

In this design, we generate hypotheses and tests in a loop and eventually produce a mitigation plan—we replicate how an OCE mitigates incidents (the design shadows the OCE). The end-to-end helper workflow is as follows:

Hypothesis former. The hypothesis former takes in the current context and produces a hypothesis small enough that we can easily verify—its only goal is to help the operator get one step closer to recognizing the root cause or mitigation.

[3] Ongoing trends suggest higher token lengths and specialized smaller models, which would benefit our applications.

Take an example incident where a link drops packets: the hypothesis former hypothesizes that either a link is faulty or overloaded, or the monitoring pipeline is broken. The OCE has to choose which hypothesis to test. The LLM must produce a confidence score and an explanation to help novice OCEs decide.

Hypothesis tester. The hypothesis tester receives these choices, and generates a plan to validate each: in our earlier example, the tester can query monitors to check the link utilizations across the network and validate whether any link is overloaded. The hypothesis tester can access the operator's monitoring infrastructure (e.g., link utilization dashboards), LLM-based tools (e.g., one that checks whether there is an ongoing incident with similar symptoms in the operator's network), or even produce manual steps the OCE can execute (e.g., it can tell the OCE to ask the customer for a packet trace). The role the LLM plays in this step is to identify what tools can test the hypothesis and how to interpret the results from the queries and either accept or reject the hypothesis. The OCE will verify both steps.

Mitigation planner. The OCE then invokes the mitigation planner which takes the *tested* hypothesis as input and produces a mitigation plan that the OCE then can trigger. We explicitly only allow the OCE (and not the other LLM modules) to start the mitigation process because: (1) mitigations can impose drastic costs on cloud tenants, and the responsibility of premature mitigation should fall on the OCE; (2) the hypothesis forming and testing is meant to decrease OCE uncertainty, not the helper's uncertainty (which is non-trivial to assess too). This adds an extra layer of risk protection without impacting TTM.

The mitigation planner generates a list of possible actions (e.g. de-isolate link and reroute traffic) and the associated risk. The OCE selects which to use. The OCE has to use various tools to execute each action (API, LLM or OCE) and then presents the results to the module, which parses it and decides whether the mitigation was successful.

Risk assessment is central to the third module. We highlight two important risk assessments in §2: **(1)** an *internal*, qualitative analysis, which the LLM produces based on thought chains and reason (e.g., if the de-isolated link is faulty, packet loss may increase); **(2)** an *external*, quantitative analysis, which a white-box algorithm produces based on network principles (e.g., 23% risk that de-isolating link A disrupts a microservice). We find these measures necessary as they are complementary views of risk, and vital to both experienced and novice OCEs.

Design benefits. Teams can independently contribute and develop the system. They can add new tools or remove deprecated ones from the hypothesis tester and mitigation planner. OCEs have first-hand experience and know best what tool or mitigation implementation is cost-effective and which tools to update to support new incident patterns.

While these modules work best in tandem, their tasks are separate. Operators benefit from each individual module, and OCEs can fill the gaps for modules in development. This allows a phased execution and lowers the high barrier to deployment.

A modular design eases adaptation. In an end-to-end model, each change requires a full prompt redesign or model fine-

tuning. With a modular design, we can surgically upgrade the relevant module. We can adapt LLM-based modules by: (1) fine-tuning [1] which pays a cost up-front but is the most accurate; (2) in context learning [13] which is faster to develop but cannot accept tasks with large contexts because of limited prompt size. Fine-tuning has a high infrastructure cost so operators should use it sparingly.

4.4 Research Directions

Our community needs to research several new directions to materialize this framework, which are:

Network expertise. We need to evaluate, and if necessary, improve the LLM's performance on complex systems and network problems (LLMs can diagnose simple problems but not complex ones [47, 55]). To fine-tune for networking knowledge, we can use IM data: **(1)** OCE training documents, slides, and videos (which are multi-modal, high-quality, and low volume); **(2)** TSGs (which are high-volume, detailed, and sometimes stale); **(3)** incident logs and OCE communications (which are high-volume, low-quality and unstructured); **(4)** direct LLM chatting with OCEs (which are high-quality and expensive); **(5)** IM tools and their documentation.

Network-focused Embeddings. We have a limited token size per query and have to carefully pick what information is relevant to the current prompt. Frameworks currently embed pieces of text to a vector, and store the tuple in a vector database. At runtime, the vector database runs an approximate nearest neighbour search on the database and appends top results to the prompt. But embeddings come from generic models trained on non-network specific data [23], and also not trained specifically to guide prompts. Future work can train network-specific embedding models that guide LLM prompts. Recent work may serve as a good starting point [57].

Intelligent planning. LLMs cannot plan long-term [8]. We require the framework to generate and test small hypotheses that are easy to plan and test but also benefit from better planning which requires more study. Methods such as chain-of-thought [28, 52, 54] show promise but rely on intuitive ways to structure the LLM response. There is no barrier to entry for this research, and in fact, the systematic mindset in our community may help us find better solutions. Recent work provides a good starting point for the design [37, 47].

Risk assessment. We propose three lines of research. First, automated qualitative reasoning with LLMs: LLMs can support their conclusions with logical [28, 52] or common-sense [6] arguments if we use them in a multi-step process with "short steps". But researchers have not yet built a reasoning engine for network and cloud systems. In the short-term, a network expert LLM that understands key network concepts (*e.g.,* routing, wired and wireless media, congestion, RDMA), cloud services (VMs, WANs, Clos topologies, SDN, NICs, Programmable switches), and can deduce the interaction between these different components, at least in steps can help address this.

Second, we can use reliable and white-box analytical models of risk which apply to specific scenarios. Prior work has looked at analytical risk assessment [2, 38, 53, 58] but they fall short: (1) they don't measure risk directly (*e.g.,* they measure the increase in link B's utilization if we isolate link A but the risk operators care about is the impact on the applications that used link A before isolation and link B after); (2) they consider a small set of mitigations compared to the full breadth of what operators can use (*e.g.,* they consider isolating the faulty link but do not model migrating the affected VMs or partially re-routing traffic); and (3) none consider whether the mitigation itself may cause an incident (*e.g.,* whether the mitigation may trigger a race condition that would cause further harm to the network).

Third, and perhaps the most exciting path, is how we can merge these two perspectives on confidence and risk; LLM-based qualitative and high-level assessments, and explicit models with quantified analysis. At a high level, an LLM-based agent decides what is at risk given some mitigation, generates a causal graph of components, and uses analytical tools to quantify the risks, given the causal model.

Toolbox abstraction. The boundary between modules and tools is unclear. There is a trade-off; tools can provide low-level telemetry and let the main module parse the information, or tools could parse low-level telemetry and provide the main module with high-level insights. With the former, we have to verify the LLM-based module parsed the data correctly, and with the latter, we have to manually design the tools.

Verifiable LLM-based tools. LLMs can help with code generation [12] and SQL queries [31, 48]. But we need to verify the outputs they generate if we want to use them in an automated pipeline. This requires research in: (1) formal verification techniques [21] to prove correctness; (2) repairing responses with consistency checks such as Haskell QuickCheck [15].

5 Related Works

LLMs. Prompt-engineering research has surged recently, and is necessary to utilize LLMs to their full potential. Some approaches attempt to improve the reasoning quality of LLMs through chain-of-though and scratch-padding [28, 39, 52, 54], some aim to keep the LLM self-consistent [27, 32, 42, 47, 51], some use debating among multiple models to improve results [19], some verify the response [16]. Some approaches provide general-purpose techniques that teach LLMs to use tools [43], some retrieve useful text from a large corpus and put in context when needed [25], whilst others provide a systematic architecture for tools to use LLMs [9, 33]. Prior work processes multi-modal data (e.g., image, audio, text, etc.) with LLMs without explicit training [56].

Incident Management. Prior works have tried to improve and understand the IM workflow either by describing examples from large production environments [24], monitoring solutions to help the diagnosis process [30, 35, 36, 49], or ML-based solutions to improve different parts of the OCE experience [22].

The work which focuses on using ML to improve the IM workflow itself includes but is not limited to [1, 10, 11, 13, 22, 26, 46]. From this set, those which involve LLMs [1, 13] laid

the initial foundation to show there is promise for LLMs to have an impact in this space. We take on the next step: we present an end-to-end view of how we can produce a reliable and deployable solution in practice.

6 References

[1] T. Ahmed, S. Ghosh, C. Bansal, T. Zimmermann, X. Zhang, and S. Rajmohan. Recommending root-cause and mitigation steps for cloud incidents using large language models, 2023. ICSE'23.

[2] O. Alipourfard, J. Gao, J. Koenig, C. Harshaw, A. Vahdat, and M. Yu. Risk-based planning of network changes in evolving data centers. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 414–429, 2019.

[3] A. Alomar, P. Hamadanian, A. Nasr-Esfahany, A. Agarwal, M. Alizadeh, and D. Shah. CausalSim: A causal framework for unbiased Trace-Driven simulation. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1115–1147, Boston, MA, Apr. 2023. USENIX Association.

[4] B. Arzani, K. Hsieh, and H. Chen. Interpret-able feedback for AutoML systems, 2021.

[5] M. Azure. Post incident review (pir) – azure networking – global wan issues, 2023. `https://azure.status.microsoft/en-us/status/history/`, Accessed: 2023-06-26.

[6] A. Bosselut, H. Rashkin, M. Sap, C. Malaviya, A. Celikyilmaz, and Y. Choi. COMET: Commonsense transformers for automatic knowledge graph construction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4762–4779, Florence, Italy, July 2019. Association for Computational Linguistics.

[7] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020.

[8] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang. Sparks of artificial general intelligence: Early experiments with GPT-4, 2023.

[9] H. Chase. LangChain, Oct. 2022. `https://github.com/hwchase17/langchain`, Accessed: 2023-06-26.

[10] J. Chen, X. He, Q. Lin, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang. Continuous incident triage for large-scale online service systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 364–375. IEEE, 2019.

[11] J. Chen, S. Zhang, X. He, Q. Lin, H. Zhang, D. Hao, Y. Kang, F. Gao, Z. Xu, Y. Dang, et al. How incidental are the incidents? characterizing and prioritizing incidents for large-scale online service systems. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 373–384, 2020.

[12] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code, 2021.

[13] Y. Chen, H. Xie, M. Ma, Y. Kang, X. Gao, L. Shi, Y. Cao, X. Gao, H. Fan, M. Wen, J. Zeng, S. Ghosh, X. Zhang, C. Zhang, Q. Lin, S. Rajmohan, and D. Zhang. Empowering practical root cause analysis by large language models for cloud incidents, 2023.

[14] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel. Palm: Scaling language modeling with pathways, 2022.

[15] K. Claessen and J. Hughes. QuickCheck: a lightweight tool for random testing of haskell programs. In *Proceedings of the fifth ACM SIGPLAN international conference on Functional programming*, pages 268–279, 2000.

[16] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training verifiers to solve math word problems, 2021.

[17] M. Cowan, S. Maleki, M. Musuvathi, O. Saarikivi, and Y. Xiong. Msccl: Microsoft collective communication library. *arXiv preprint arXiv:2201.11840*, 2022.

[18] J. D. Day and H. Zimmermann. The OSI reference model. *Proceedings of the IEEE*, 71(12):1334–1340, 1983.

[19] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2023.

[20] M. Fernandez. Prime day woes might have cost Amazon $72m-$99m in sales, 2018. `https://www.axios.com/2018/07/18/prime-day-woes-might-have-cost-amazon-from-72-99-million`, Accessed: 2023-06-26.

[21] E. First, M. N. Rabe, T. Ringer, and Y. Brun. Baldur: Whole-proof generation and repair with large language models, 2023.

[22] J. Gao, N. Yaseen, R. MacDavid, F. V. Frujeri, V. Liu, R. Bianchini, R. Aditya, X. Wang, H. Lee, D. Maltz, M. Yu, and B. Arzani. Scouts: Improving the diagnosis process through domain-customized incident routing. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 253–269, New York, NY, USA, 2020. Association for Computing Machinery.

[23] T. Gao, X. Yao, and D. Chen. SimCSE: Simple contrastive learning of sentence embeddings, 2022.

[24] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat. Evolve or die: High-availability design principles drawn from google's network infrastructure. In *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016.

[25] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M.-W. Chang. Realm: Retrieval-augmented language model pre-training, 2020.

[26] J. Jiang, W. Lu, J. Chen, Q. Lin, P. Zhao, Y. Kang, H. Zhang, Y. Xiong, F. Gao, Z. Xu, et al. How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1410–1420, 2020.

[27] G. Kim, P. Baldi, and S. McAleer. Language models can solve computer tasks, 2023.

[28] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners, 2023.

[29] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, et al. The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 183–196, 2017.

[30] Y. Li, R. Miao, C. Kim, and M. Yu. FlowRadar: A better NetFlow for data centers. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 311–324, Santa Clara, CA, Mar. 2016. USENIX Association.

[31] A. Liu, X. Hu, L. Wen, and P. S. Yu. A comprehensive evaluation of ChatGPT's zero-shot Text-to-SQL capability, 2023.

[32] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Gupta, B. P. Majumder, K. Hermann, S. Welleck, A. Yazdanbakhsh, and P. Clark. Self-refine: Iterative refinement with self-feedback, 2023.

[33] Microsoft. LangChain, Feb. 2023. `https://github.com/microsoft/semantic-kernel`, Accessed: 2023-06-26.

[34] J. C. Mogul, D. Goricanec, M. Pool, A. Shaikh, D. Turk, B. Koley, and X. Zhao. Experiences with modeling network topologies at multiple levels of abstraction. In *NSDI*, pages 403–418, 2020.

[35] M. Moshref, M. Yu, R. Govindan, and A. Vahdat. Dream: Dynamic resource allocation for software-defined measurement. In *Proceedings of the ACM SIGCOMM Conference*, 2014.

[36] M. Moshref, M. Yu, R. Govindan, and A. Vahdat. Trumpet: Timely and precise triggers in data centers. In *ACM SIGCOMM*, 2016.

[37] Y. Nakajima. Task-driven Autonomous Agent, Apr. 2023. `https://github.com/yoheinakajima/babyagi`, Accessed: 2023-06-28.

[38] P. Namyar, B. Arzani, D. Crankshaw, D. S. Berger, K. Hsieh, S. Kandula, and R. Govindan. Mitigating the performance impact of network failures in public clouds, 2023.

[39] M. Nye, A. J. Andreassen, G. Gur-Ari, H. Michalewski, J. Austin, D. Bieber, D. Dohan, A. Lewkowycz, M. Bosma, D. Luan, C. Sutton, and A. Odena. Show your work: Scratchpads for intermediate computation with language models, 2021.

[40] OpenAI. ChatGPT plugins, 2023. `https://openai.com/blog/chatgpt-plugins`, Accessed: 2023-06-26.

[41] OpenAI. GPT-4 technical report, 2023.

[42] D. Paul, M. Ismayilzada, M. Peyrard, B. Borges, A. Bosselut, R. West, and B. Faltings. Refiner: Reasoning feedback on intermediate representations, 2023.

[43] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language models can teach themselves to use tools, 2023.

[44] A. W. Services. Summary of AWS Direct Connect event in the Tokyo (ap-northeast-1) region, 2021.

[45] A. Shah, V. Chidambaram, M. Cowan, S. Maleki, M. Musuvathi, T. Mytkowicz, J. Nelson, and O. Saarikivi. TACCL: Guiding collective algorithm synthesis using communication sketches. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 593–612, 2023.

[46] M. Shetty, C. Bansal, S. P. Upadhyayula, A. Radhakrishna, and A. Gupta. Autotsg: Learning and synthesis for incident troubleshooting, 2022.

[47] N. Shinn, F. Cassano, B. Labash, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: Language agents with verbal reinforcement learning, 2023.

[48] R. Sun, S. O. Arik, H. Nakhost, H. Dai, R. Sinha, P. Yin, and T. Pfister. SQL-PaLM: Improved large language model adaptation for Text-to-SQL, 2023.

[49] C. Tan, Z. Jin, C. Guo, T. Zhang, H. Wu, K. Deng, D. Bi, and D. Xiang. NetBouncer: Active device and link failure localization in data center networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 599–614, Boston, MA, Feb. 2019. USENIX Association.

[50] W. Wang, M. Khazraee, Z. Zhong, M. Ghobadi, Z. Jia, D. Mudigere, Y. Zhang, and A. Kewitsch. TopoOpt: Co-optimizing network topology and parallelization strategy for distributed training jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 739–767, 2023.

[51] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou. Self-consistency improves chain of thought reasoning in language models, 2023.

[52] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[53] X. Wu, D. Turner, C.-C. Chen, D. A. Maltz, X. Yang, L. Yuan, and M. Zhang. NetPilot: Automating datacenter network failure mitigation. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 419–430, 2012.

[54] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023.

[55] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[56] A. Zeng, M. Attarian, B. Ichter, K. Choromanski, A. Wong, S. Welker, F. Tombari, A. Purohit, M. Ryoo, V. Sindhwani, J. Lee, V. Vanhoucke, and P. Florence. Socratic models: Composing zero-shot multimodal reasoning with language, 2022.

[57] S. Zhang, P. Jin, Z. Lin, Y. Sun, B. Zhang, S. Xia, Z. Li, Z. Zhong, M. Ma, W. Jin, D. Zhang, Z. Zhu, and D. Pei. Robust failure diagnosis of microservice system through multimodal data, 2023.

[58] D. Zhuo, M. Ghobadi, R. Mahajan, K.-T. Förster, A. Krishnamurthy, and T. Anderson. Understanding and mitigating packet corruption in data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 362–375, 2017.