Cloud Atlas: Efficient Fault Localization for Cloud Systems using Language Models and Causal Insight

Zhiqiang Xie*
Stanford University
xiezhq@cs.stanford.edu

Yujia Zheng
Carnegie Mellon University
yujiazh@cmu.edu

Lizi Ottens
Stanford University
lottens@stanford.edu

Kun Zhang
Carnegie Mellon University
kunz1@cmu.edu

Christos Kozyrakis
Stanford University
christos@cs.stanford.edu

Jonathan Mace
Microsoft Research
jonathanmace@microsoft.com

Abstract

Runtime failure and performance degradation is commonplace in modern cloud systems. For cloud providers, automatically determining the root cause of incidents is paramount to ensuring high reliability and availability as prompt fault localization can enable faster diagnosis and triage for timely resolution. A compelling solution explored in recent work is causal reasoning using causal graphs to capture relationships between varied cloud system performance metrics. To be effective, however, systems developers must correctly define the causal graph of their system, which is a time-consuming, brittle, and challenging task that increases in difficulty for large and dynamic systems and requires domain expertise. Alternatively, automated data-driven approaches have limited efficacy for cloud systems due to the inherent rarity of incidents. In this work, we present Atlas, a novel approach to automatically synthesizing causal graphs for cloud systems. Atlas leverages large language models (LLMs) to generate causal graphs using system documentation, telemetry, and deployment feedback. Atlas is complementary to data-driven causal discovery techniques, and we further enhance Atlas with a data-driven validation step. We evaluate Atlas across a range of fault localization scenarios and demonstrate that Atlas is capable of generating causal graphs in a scalable and generalizable manner, with performance that far surpasses that of data-driven algorithms and is commensurate to the ground-truth baseline.

1 Introduction

Modern cloud systems are large-scale, complex, and dynamic, combining many inter-operating services. Runtime failure and performance degradation occurs frequently, arising for many reasons including software bugs, hardware failures, incorrect high-level design and unanticipated workload dynamics [10]. To combat such incidents, engineers continuously monitor the health of cloud systems [15, 26] using detailed telemetry of data emitted by the systems. An incident is triggered when a top-level health metric, such as end-to-end request latency or API error rate, exceeds an acceptable threshold; on-call engineers must then investigate, localize, and mitigate the incident

*Work partially done during internship at Microsoft Research.

Preprint. Under review.

as quickly as possible. This task requires domain-specific expertise supplemented by the on-call engineer's knowledge of the system to understand the root cause of the issue in need of triage.

Automatically localizing faults and determining the root cause of incidents is a goal of utmost importance for cloud providers. A solution to this problem must overcome two fundamental challenges: first, measurement data is diverse; systems emit an enormous amount of data at different layers, including both standardized measurements such as CPU utilization and application-specific measures; second, symptoms of an incident typically cascade from the root cause to all dependent components recursively, resulting in many changes to correlated metrics which may be obfuscated by fault-tolerance mechanisms that change the workload dynamics of the system.

Recent work in the systems community has identified a compelling solution to both challenges: reasoning on causal graphs [27, 12, 34, 20]. Yet despite the effectiveness of causal reasoning, it is critically dependent on the correctness of the causal graphs. Users are faced with an inherent trade-off between rules-based or data-driven construction of causal graphs: while rules-based construction often leads to better causal graphs, composing and updating rules manually is an arduous task; on the other hand, automated data-driven approaches rely heavily on the specific parametric form of the data distribution and in real systems these assumptions are often not testable or rarely satisfied. Moreover, anomalous system behavior is rare and some system behaviors may never have been observed before. In both cases, the challenge is exacerbated due to the scale of real systems.

We view the construction of causal graphs as a process of converting system knowledge into a structured representation. While some existing data such as dependency graphs can potentially be processed automatically, most critical semantics about system components and measurements only exist informally, i.e. high-level system descriptions are commonplace in the form of design and architecture documents, and primarily exist for the benefit of human consumption.

In this work we present Atlas, a tool that uses large language models (LLMs) to automate the understanding and processing of unstructured semantic information into the causal construction process. Despite their scale, cloud systems are inherently modular, with complexity arising due to composition, encapsulation, and abstraction. Atlas exploits these characteristics by decomposing a system into its constituent components, interpreting textual component descriptions and identifying pairwise causal relationships between measurements. Atlas contains a novel representation of system components and measurements and employs several LLM prompting strategies to exploit this representation. After constructing a candidate causal graph, Atlas applies a novel data-driven validation step to identify possible mistakes in the causal graph and to propose potentially missed connections through Markov blanket discovery.

We validate Atlas using real software system datasets and compare the generated causal graphs with those produced by traditional causal discovery algorithms. We further evaluate Atlas on several diverse fault localization case studies. Our results demonstrate that Atlas can produce significantly higher-quality causal graphs than existing data-driven techniques, and when used to localize faults, the graphs produced by Atlas have comparable effectiveness to the ground-truth causal graphs. To the best of our knowledge, this is the first study to employ LLMs in the construction of causal graphs capable of localizing faults for large-scale software systems. In addition to the system itself, we will also release the simulator code and generated measurement dataset as we believe these resources will benefit the broader community.

2 Background

Fault Localization. Most incident investigations today are a manual process overseen or driven by a human operator, with little automation [15, 26]. Human-driven investigations implicitly leverage the human's mental model of system behavior to navigate the space of recorded metrics and system components. From experience, code, and familiarity with system design, humans understand the relationships between different metrics and reason using their system knowledge to determine what to investigate via a chain of cause and effect.

Causal Reasoning. Despite promising results in controlled environments, little automation is deployed in practice: data-driven approaches are inhibited by high dimensionality, excessive correlations, and high integration overhead [25]. As a possible solution, causal reasoning is compelling, because it scopes measurements based on their direct causal influences [27, 12, 34, 20] and because

it incorporates missing cause-and-effect domain knowledge that human-driven investigations utilize. However, capturing and maintaining this knowledge in an up-to-date causal graph has high overhead; consequently causal reasoning is often limited to constrained scenarios where the semantics of a small set of measurements are predefined and combined with predefined rules.

Rule-Based Construction. In this approach, a domain expert (e.g. a system developer or operator) provides the causal graph by enumerating the causal relationships between variables using their domain knowledge. In computer systems, the breadth of system diversity and measurements is too large to expect humans to provide the full, correct graph manually. Though some work [4, 18] has explored easing this burden for constrained scenarios, it remains a significant obstacle in practice.

Data-Driven Causal Discovery. Causal discovery algorithms are a data-driven approach to extracting causal relationships from observed measurements, potentially overcoming the need for time-consuming input from domain experts. Causal discovery provably cannot produce a single correct result, but can produce viable candidates for evaluation by a human. Causal discovery is a poor fit for identifying a cloud system's causal structure, because incidents are edge-case system behaviors that are not exercised in the steady-state; thus steady-state telemetry is insufficient for identifying the full diversity of system behaviors. Acquiring such a dataset is extraordinarily challenging, requiring both the ability to anticipate edge-case behavior that may never have been seen before, as well as the ability to perform interventional testing by running the system at scale and injecting anomalies.

3 Observations

Atlas automates *rules-based construction* of causal graphs for computer system telemetry. This section describes the key intuitions behind Atlas; in §4, we present Atlas's end-to-end design.

Domain Knowledge from System Documentation. Causal relationships between system telemetry derive from a high-level model of system components and interactions. In practice, it is exceedingly rare for such a model to be explicitly formalized. Instead, developers predominantly communicate the high-level behavior of a system through design and architecture documents, describing in text the system components, topology, and interactions. Examples can be found in many open-source software projects [2, 29]. System descriptions use a high level of abstraction and make heavy reference to well-known systems concepts such as replication, load balancing, containers, processes, etc. Telemetry can be similarly related to system concepts: measurements use descriptive names and are typically accompanied by descriptions of what is being measured. Atlas leverages this observation by utilizing LLMs to interpret text-based system descriptions to extract causal relationships. LLMs have knowledge of common systems concepts and are able to interpret design documents.

Causal Relationships from System Interactions. Although systems can expose a large amount of telemetry, the relationships between measurements are not arbitrary. We observe that direct causal relationships between telemetry only arise when there is an interaction between corresponding components. For example, the throughput of service A running on server X can only have a direct causal influence on CPU_X utilization because A is directly interacting with the CPU by running instructions on it. By contrast, there can be no *direct* influence on server Y's CPU_Y utilization; that dependency can only be established indirectly through other intermediate components, such as some service B running on Y. Moreover, software systems are designed in a modular, hierarchical manner to mitigate complexity; it is universally discouraged to tightly couple software components. Therefore, we only need to consider causal relationships between measurements from components that directly interact with one another. Atlas leverages this observation from system documentation, which typically describes directly which components interact with others, as well as distributed trace data [26] to directly observe runtime interactions.

Scalability from Decomposition. Thanks to the locality of interaction and causal relationships, Atlas does not need to present an LLM with wide-ranging information about the system to identify causal relationships. Instead, it can decompose the task into smaller, manageable pieces, inspecting locally connected components and their relationships independently. Atlas is thus scalable by design; as the size of a system scales up, it increases the number of LLM interactions, but not the complexity of inputs or outputs to and from the LLM for each iteration.

Reconciling with the Ground Truth. Systems do not measure everything that can truly be measured. For example, RPC servers in general might measure throughput, queueing time, latency, and request

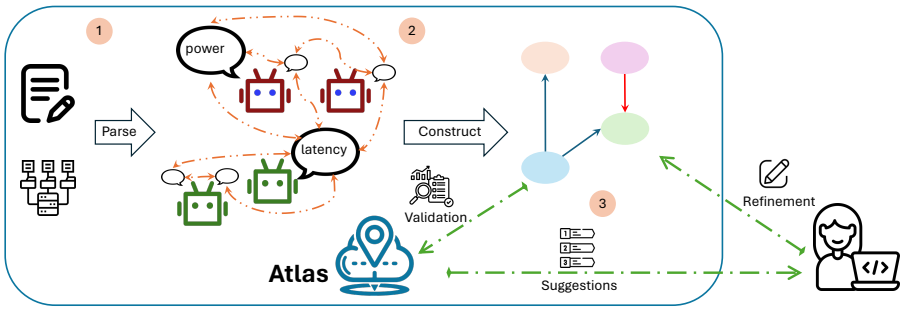


Fig. 1: Overview of Atlas: (1) Instantiating agents for each system component by parsing system documentation and telemetry; (2) Enumerating causal relationships between metrics; (3) Constructing the causal graph and refining with human-in-the-loop feedback.

success; yet a specific RPC server implementation might only measure a subset of those. This poses a potential challenge to correctly identifying causal relationships: two measurements that have an indirect causal relationship in the idealized version of the system might now have a direct causal relationship in a particular system implementation if the intermediate variables aren't observed. Atlas is robust to this by construction: it does not initially work with the actual measurements available in the system, but instead leverages general knowledge of systems concepts to construct the ideal causal graph which contains a superset of the actual measurements available. Atlas then reconciles the ideal causal graph with the actual measurements that are available.

Complementary Data-driven Validation Unlike data-driven causal discovery, LLMs lack useful indicators like confidence levels, making it difficult for users to identify, localize, and correct any potential errors in the generated causal graph. Our intuition to address this is that Atlas is compatible with data-driven techniques: traditional statistical methods can optionally be leveraged to validate the output of Atlas, if desired. These methods can use measurement data to negate proposed connections that are contradicted by data, and suggest potentially missed connections with confidence scores. Atlas leverages this as an optional additional validation step after generating a causal graph.

4 Design

Overview. Fig. 1 depicts the end-to-end pipeline of Atlas. Atlas processes descriptions of system components, metrics, and caller-callee relationships, as well as an optional corpus of common component and metric descriptions. Using an agent-based approach, Atlas represents each system component as a separate LLM agent with relevant descriptions. Atlas iteratively enumerates all possible metrics for each component and prompts the LLM accordingly to identify causal relationships and their directionality between interacting agents. It then combines these outputs into a single graph, collapsing indirect causal relationships through unmeasured nodes. Finally, Atlas presents the candidate causal graphs to users and provides an interface for interactively refining the candidate graph through review and feedback on a small set of proposed causal relationships.

4.1 Inputs

The inputs to Atlas are organized as follows:

- A collection of named system components (e.g. services, processes, etc.) and corresponding textual descriptions.
- A collection of named measurements and corresponding text descriptions where each measurement is associated with a system component.
- A list of caller-callee relationships between components.
- A list of computational resources available to components with corresponding text descriptions.

Descriptions can be concise or verbose; for example, the description of a gateway server may simply be, *"Website gateway receives the client request."* See Appendix A for detailed examples. Note most

of these inputs can be automatically extracted from request traces, system deployment configurations files, and developer documentation.

4.2 Instantiating Agents

Atlas conceptualizes each component in the system as an agent. Agents are categorized as one of the following classes: *request*, *service*, or *resource*. Each agent is assigned relevant information about its role: component descriptions, metric descriptions, and resource descriptions. Each agent is also assigned the information about any components it directly interacts with; only the following kinds of interaction are considered: (a) a request invokes a service; (b) a request utilizes a resource; (c) a service utilizes a resource; (d) a service invokes another service. Each agent corresponds to a bounding box in Fig. 2.

4.3 Metrics Enumeration: Creating Nodes for the Causal Graph

As motivated in §3, in addition to having all available metrics of system components as nodes within the causal graph (blue nodes in Fig. 2a), we also enumerate typical measurements associated with those components and convert them to be nodes as indicated in orange in Fig. 2a.

4.4 Causal Relationship Examination: Connecting the Nodes

Atlas iteratively evaluates whether a causal relationship exists between pairs of measurements. Atlas only considers measurements that either (a) exist within the same component; or (b) exist on components that directly interact with one another. Atlas prompts the LLM from the perspective of one agent: it furnishes the LLM with the information known by that agent about its own role and the pair of measurements, asks whether there exists a causal relationship between the two measurements, and if so, asks what the direction of the causal influence is. Atlas uses common techniques to construct the prompt: chain of thoughts [35] and alternating order of options with repeating queries for consistent output [37]. We provide concrete prompt examples in Appendix A.

Atlas employs a semantic cache to store the full semantics of each query and its result. This optimization greatly reduces the number of times Atlas prompts the LLM because the same set of components can be instantiated in multiple places in large-scale systems, with the same semantics and thus same causal relationships between measurements. By utilizing cached responses, the execution time and token cost of Atlas is greatly reduced.

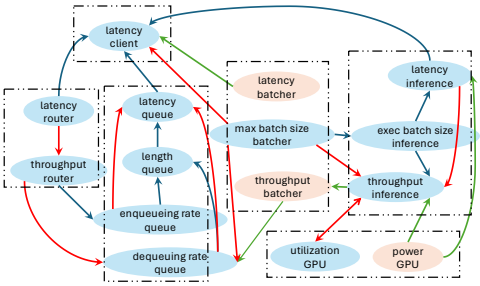
4.5 Graph Construction

Atlas combines the individual causal relationships discovered during the previous step into a causal graph. The resulting causal graph is over-complete: it includes nodes for all measurements recorded by the system, and can also include nodes for common measurements that *could* be recorded by the system but are not (i.e., unobserved measurements). We call this graph the *confounder* graph because it exposes potential unobserved confounders; we do not discard the confounder graph because knowledge of potential confounders may be useful for users in some future causal reasoning task or in revisiting the choice of measurements made by the system.

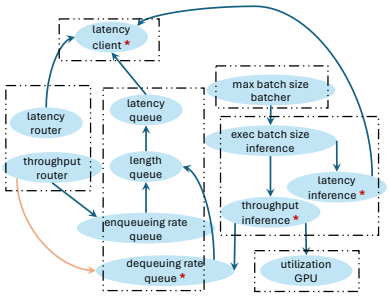
To produce the final causal graph, Atlas duplicates the confounder graph and deletes all unobserved nodes. To delete an unobserved node, Atlas removes the node while preserving any transitive causal influence that may exist, for example, if A and B are observed, but U is unobserved, then $A \to U \to B$ collapses to $A \to B$. The locations of these collapsed unobserved nodes are recorded, which will be useful for fault localization as explained in §5.3. By removing unobserved nodes, causal relationships that were previously indirect may now be direct. Fig. 2 illustrates this process.

4.6 Data-driven Human-in-the-loop Pareto Refinement

Atlas lacks guarantees that the resulting causal graph is correct, so it employs an optional human-in-the-loop refinement step to identify and correct four kinds of errors that may exist in the graph: cycles, false positives, false negatives, and reversed directionality. As motivated in §3, we employ data-driven approaches to filter out a small set of candidates for feedback and refinement. Specifically, we: 1) Apply Markov blanket discovery to examine all proposed causal relationships. We sample five



(a) Confounder graph where unobserved nodes are indicated in brown, incorrect causal relationships are indicated with red edges, and reversed directionality is indicated with double arrows.



(b) Causal graph after post-processing where the red asterisks indicate that collapsed nodes are recorded. Incorrect edges remaining after refinement are indicated with yellow edges.

Fig. 2: Causal graph of a model serving task (some nodes and edges omitted for readability).

suspicious candidates, prioritizing edges whose removal would change connectivity, and ask the user to correct any erroneous connections. This process is iterative until all five candidates are correct. 2) Apply the Additive Noise Model to examine the direction of established edges, following the same procedure. If there still exists a cycle in the directed acyclic graph, Atlas will suggest a minimal set of edges to cut to resolve the issue. 3) Apply Markov blanket discovery again to suggest potentially missed connections among edges that will create new connectivity. In this optional validation stage, the user only needs to manually examine a small set of causal relationships to obtain a significant Pareto refinement on the causal graph as shown in Table 1. Shared in Fig. 2b is a sample causal graph after this refinement procedure.

5 Evaluation

Our evaluation aims to answer the following key questions:

- Does Atlas generate superior causal graphs compared to traditional data-driven causal discovery algorithms? (§5.1)
- How effective are the causal graphs generated by Atlas for fault localization use cases? (§5.2)

In addition to addressing the questions above, our evaluation concludes with three Atlas case studies (§5.3).

Atlas Variants. To better understand the contribution of Atlas' design, our evaluation compares three different variants of Atlas: **NAÏVELLM** uses a one-shot LLM prompt to construct the graph. It provides all of the text descriptions used by Atlas in a single input, and prompts the LLM to generate all edges of the causal graph. **ATLAS+V** utilizes the full Atlas pipeline including the data-driven validation step described in §4.6. **ATLAS** utilizes the full pipeline, but omits the final data-driven validation step. All Atlas variants use GPT-4-turbo as the LLM. Due to non-deterministic LLM outputs, we repeat all Atlas experiments 5 times and report averaged results.

Data-driven Causal Discovery. We compare the causal graphs generated by Atlas to those produced by the following data-driven causal discovery algorithms: GES [6], GRaSP [17], and PC [32]. All experiments are also repeated 5 times and averaged results are reported.

Existing Scenarios. We make use of several pre-existing datasets from prior work that provide observational data and ground-truth causal graphs for systems scenarios: the Middleware Oriented Message activity (MoM) and Antivirus Activity (AA) scenarios from prior IT system monitoring work [1] and the Microservices latency scenario from the DoWhy causal reasoning library [8]. We supplement the observational data with brief textual descriptions of the system components.

New Scenarios. To evaluate Atlas at larger scale, we have developed a discrete event simulator that can simulate the dynamics of cloud environments including dynamic service topologies, execution flow, resource contention, workloads, and faults. Within this simulator we implement a model serving service and workload, following the design of DLIS [31]. Fig. 3 depicts the high-level execution flow: a client's request reaches a router and is load-balanced to a worker server; it enters a queue, and when

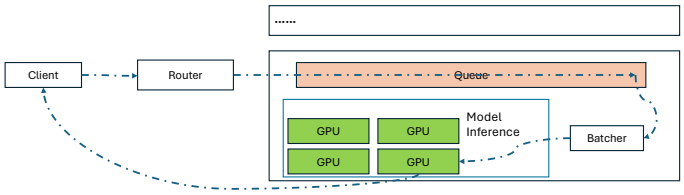


Fig. 3: Simulated Model Serving Diagram.

a GPU becomes available for inference, the request is dequeued and executed. Different requests can be batched together for GPU execution to achieve higher throughput. The system can be configured with a variable number of machines, each with a variable number of GPUs. We simulate workloads with exponentially distributed request interrarival times and permute the following dimensions to generate new datasets:

- ModelServingS configures one worker machine with one GPU; ModelServingM configures one worker machine with four GPUs; and ModelServingL configures two worker machines each with four GPUs.
- For fault localization scenarios we simulate both a normal operating state and several failure scenarios that are commonly observed in model serving systems [28], including workload spikes, network slowdown, batch size misconfiguration, and GPU throttling. We generate 7, 13, and 24 different failure datasets for ModelServing -S, -M, and -L respectively.

Each scenario varies in the size of the ground-truth causal graph and the number of observational samples. Table 1 summarizes these characteristics.

5.1 Comparison on Causal Graph Construction

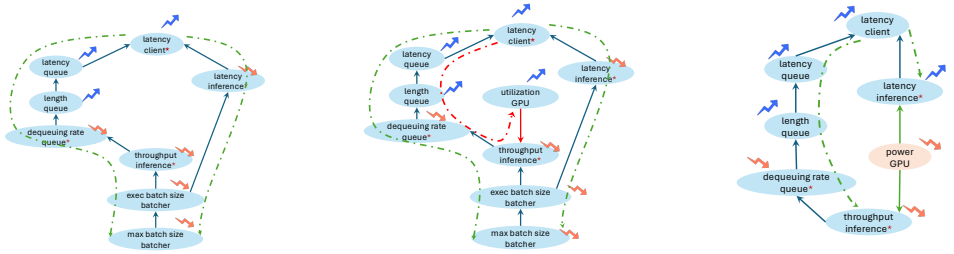
In this experiment, we apply all causal graph construction techniques to all datasets. We measure the effectiveness of each technique by calculating the F1 score of the edges of the constructed graphs with respect to a ground truth baseline. We present the results in Table 1.

In general, while different causal discovery algorithms (GES, GRaSP, PC) exhibit unique strengths depending on the data distribution, they generally perform poorly in system troubleshooting scenarios as indicated by the results in Table 1. This underperformance is primarily due to a fundamental bias in system measurement data: most data is collected during normal operating states as abnormal states are rare, diverse, and often fail to be recorded. Additionally, these algorithms tend to perform worse as the number of nodes increases.

By contrast, ATLAS attains the highest F1 scores across all datasets. On small datasets, ATLAS either generates the same causal graph as the ground-truth, or has additional superfluous edges. ATLAS maintains strong performance as the graph size increases. ATLAS+V demonstrates further improvements over the baseline ATLAS: the additional data validation and Pareto refinement steps further improve the quality of the causal graph. The average number of proposed and accepted graph modifications of ATLAS+V is included in Table 1.

Table 1: F1 scores of causal graphs compared to ground truth. We average Atlas results over 5 repetitions and report the number of changes proposed and accepted by ATLAS+V's validation step.

Method	MoM	AA	Microservice	ModelServingS	ModelServingM	ModelServingL
# Nodes	7	13	11	15	30	56
# Edges	10	16	13	15	36	70
# Samples	654	2643	10001	7087	6966	13601
GES	0.14	0.19	0.28	0.22	0.30	*timeout*
GRaSP	0.17	0.26	0.18	0.28	0.24	0.14
PC	0.29	0.20	0.37	0.28	0.29	0.12
NAÏVELLM	0.65	0.44	1.0	0.06	0.0	0.03
ATLAS	1.0	0.86	1.0	0.65	0.68	0.69
ATLAS+V	**1.0**	**0.86**	**1.0**	**0.89**	**0.79**	**0.73**
Accepted/Proposed	–	0/5	–	11.8/14.8	11.8/16.5	4.4/7.6



(a) Chain of causal effect for a wrong batch size setting

(b) Same setting with a reversed edge (in red)

(c) Fault localization for the case of missing metric

Fig. 4: Case study on localizing a fault in ModelServing

5.2 End-to-end Fault Localization on Model Serving

We evaluate the impact of Atlas for fault localization use cases, and demonstrate that Atlas' superior causal graphs result in significantly improved fault localization performance. We focus on the three ModelServing scenarios for which we have separate normal-state and failure-state observational datasets.

We utilize an off-the-shelf fault localization algorithm (`distribution_change`) from the DoWhy causal reasoning library [3]. This algorithm takes as input a causal graph, a dataset of normal-state observations, and a dataset of failure-state observations. The algorithm calculates and outputs an attribution score for each dataset feature, representing the likelihood that the feature is the root cause.

We apply the algorithm separately using the causal graphs generated by ATLAS and ATLAS+V, and to the ground-truth causal graph. To serve as comparison, we also apply the algorithm to the highest scoring causal graph generated by GES, GRaSP, or PC (as reported in §5.1). We consider a failure to be correctly identified if the true root cause appears in the output and evaluate the top-1 and top-3 ranked features. We repeat this process for all failure datasets.

Table 2 reports results averaged across the 7, 13, and 24 failure scenarios for ModelServing -S, -M, and -L respectively. The results illustrate that the ATLAS and ATLAS+V graphs are sufficient for identifying root causes across most failure scenarios. Moreover, they do not perform significantly worse than the ground-truth causal graph. Upon inspection, we found that Atlas graphs were succeeding and failing for the same scenarios as the ground-truth causal graph. By contrast to Atlas, the graphs generated by causal discovery algorithms perform poorly and do not reliably localize faults.

These results indicate that F1 scores are not a sufficient indicator that a causal graph is effective for fault localization. We highlight two reasons for this. First, counterfactual reasoning is only conducted on the subgraph of causal predecessors of the symptom node, whereas the F1 score measures the relative correctness of the entire graph. Second, a single incorrect edge or inverted directionality can have a minor effect on the F1 score, but a significant effect on the efficacy of causal reasoning. When examining the causal graphs produced by Atlas, we found that most incorrect edges do not change the connectivity of the causal graph and do not break the chain of causality from symptom to root cause, enabling comparable performance to the ground truth. We examine this effect in more detail in §5.3.

Table 2: Percentage of faults successfully localized using different causal graphs.

	ModelServingS		ModelServingM		ModelServingL	
Causal Graph	Top 1	Top 3	Top 1	Top 3	Top 1	Top 3
Best Causal Discovery Algorithm	0%	14%	0%	8%	25%	29%
ATLAS	60%	97%	90%	92%	96%	96%
ATLAS+V	60%	100%	90%	92%	96%	96%
Ground Truth	57%	100%	92%	92%	96%	96%

5.3 Case Studies

We examine the causal graph in the ModelServing scenario to better understand the behavior of fault localization and its limitations with respect to causal graphs.

Localize the Real Fault Beyond Discrete Signals. A key strength of causal reasoning compared to other fault localization techniques is its robustness to diverse telemetry signals, including discrete,

categorical, and continuous data. By comparison, many techniques discard useful information by reducing continuous signals to discrete indicators. For example, recent large-scale training works characterize execution time as "normal" and "too long" [13]; in general, techniques based on e.g. decision trees are often reduced to a threshold comparison such as "latency is above X".

To illustrate, consider a scenario where an incorrect max batch size is set for the batcher. As illustrated in Fig. 4a, a smaller batch size reduces effective throughput, causing more requests to queue and thereby increasing queuing latency. Although the reduced batch size decreases model inference latency, overall user latency is longer. Fault localization techniques might incorrectly report abnormally high queuing latency as the root cause, leaving the developer to diagnose the issue. With counterfactual reasoning, the fault can be traced to its root cause by the chain of causal relations.

How a Wrong Causal Relationship Degrades Fault Localization Performance. In the same setting, Fig. 4b depicts a false case generated by Atlas, introducing an inverted causal relationship between GPU utilization and inference service throughput compared to Fig. 2b. This incorrect direction creates a new causal effect chain, inhibiting counterfactual reasoning and lowering the attribution score of batch size. As a result, the max batch size is relegated from the top-3 to top-5 likely root cause. Atlas' validation process described in §4.6 helps identify and correct this erroneous direction, thereby enhancing the accuracy of fault localization.

Fault Localization When Metrics are Missing. In practice, a system might not report all possible measurements. Atlas is robust to this by generating a confounder graph alongside its output causal graph. In the Model Serving scenario, GPU power is an example of an unobserved measurement. Cloud systems often dynamically scale the power draw of GPUs, but we often lack direct measurements of power. In this case, the graph produced by Atlas enables causal reasoning to attribute a fault to latency or inference service throughput as illustrated in Fig. 4c. From the confounder graph, GPU power will also be reported to the developer as a possible culprit in this case.

6 Related Work

LLMs for System Troubleshooting. RCACopilot [5] proposed a pipeline that leverages LLMs as powerful text processing tools to summarize historical incidents, match them with newly occurred incidents, and synthesize possible root causes and mitigations based on historical data. This exemplifies a line of research [30] utilizing LLMs as information processing and reasoning engines. However, the effectiveness of these approaches strongly depends on the historical data, as identical symptoms can arise from entirely different causes, and similar symptoms may have never occurred before. Atlas employs a two-stage approach to overcome this limitation: LLMs are used to process domain knowledge and assist in causal graph construction, while the reasoning is powered by a causal reasoning engine that utilizes abundant measurement data.

Causal Discovery. Causal discovery aims to uncover causal relations from observational data without any interventional experiments. Classical algorithms include constraint-based methods (e.g., PC [32]), score-based methods (e.g., GES [7]), and many others (e.g., GRaSP [17]). Recently, large language models (LLMs) have demonstrated impressive empirical performance in addressing some causality-related tasks [16, 36, 24, 22, 33, 14, 21]. Although LLM-based methods lack comprehensive theoretical guarantees, their notable successes in diverse scenarios highlight their potential applicability in complex real-world tasks, where assumptions for identifiability theory often do not hold perfectly in practice.

Causal Reasoning for System Troubleshooting. Several research works leverage the causal structure of service dependency graphs, where nodes are common service measurements such as latency [9, 19, 23, 34, 20]. This causal structure only relates a small number of externally-visible measurements, however it can serve as a useful starting point for constructing causal graphs, e.g. by including more metrics and their causal relationships [4, 18]. For more diverse measurements such as hardware utilization, service rate, or application-defined metrics, no universal rules or mapping exists. Causal discovery algorithms have been utilized for microservice metrics [11], and face the challenges outlined in this paper. Chaos engineering is a compelling approach to uncover the diversity of system execution behaviors in small scale [12]; however scaling such an approach to a full cloud system represents a significant challenge.

7 Limitations and Future Work

Constructing causal graphs is a key step for effective fault localization. However, end-to-end there remain several obstacles. First, causal reasoning algorithms scale poorly to large graphs; we have identified several optimization opportunities based on graph decomposition that we are leveraging in our ongoing work. Second, causal graphs require special handling for the temporal feedback cycles that exist in systems, which cannot be represented as a DAG. Third, the causal graph is insufficient for representing all semantics of a system, such as semantic equivalence between components that are represented by different subgraphs. Lastly, domain knowledge can provide hints as to the correct causal mechanism to assign to each node; this is not currently captured. Our future work on Atlas will be guided by user feedback from open-sourcing the project.

8 Acknowledgement

This research was partly supported by the Stanford Platform Lab and its affiliates, and by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. The work of Yujia Zheng and Kun Zhang is supported by NSF Grant 2229881, the National Institutes of Health (NIH) under Contract R01HL159805, and grants from Apple Inc., KDDI Research Inc., Quris AI, and Florin Court Capital.

References

[1] Ali Aït-Bachir, Charles K Assaad, Christophe de Bignicourt, Emilie Devijver, Simon Ferreira, Eric Gaussier, Hosein Mohanna, and Lei Zan. Case studies of causal discovery from it monitoring time series. *arXiv preprint arXiv:2307.15678*, 2023.

[2] Apache. HDFS architecture. `https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html`, March 2024.

[3] Patrick Blöbaum, Peter Götz, Kailash Budhathoki, Atalanti A. Mastakouri, and Dominik Janzing. Dowhy-gcm: An extension of dowhy for causal inference in graphical causal models. *arXiv preprint arXiv:2206.06821*, 2022.

[4] Sarthak Chakraborty, Shaddy Garg, Shubham Agarwal, Ayush Chauhan, and Shiv Kumar Saini. Causil: Causal graph for instance level microservice data. In *Proceedings of the ACM Web Conference 2023*, WWW '23, page 2905–2915, New York, NY, USA, 2023. Association for Computing Machinery.

[5] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. Automatic root cause analysis via large language models for cloud incidents, 2023.

[6] David Maxwell Chickering. Optimal structure identification with greedy search. *Journal of machine learning research*, 3(Nov):507–554, 2002.

[7] David Maxwell Chickering. Learning equivalence classes of bayesian networks structures, 2013.

[8] DoWhy. Root cause analysis (rca) of latencies in a microservice architecture. `https://www.pywhy.org/dowhy/v0.8/example_notebooks/rca_microservice_architecture.html`, 2024. Accessed: 2024-05-14.

[9] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. Sage: practical and scalable ml-driven performance debugging in microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 135–151, 2021.

[10] Supriyo GHOSH, Manish Shetty, Chetan Bansal, and Suman Nath. How to fight production incidents? an empirical study on a large-scale cloud service. In *SoCC 2022*. ACM, November 2022.

[11] Azam Ikram, Sarthak Chakraborty, Subrata Mitra, Shiv Saini, Saurabh Bagchi, and Murat Kocaoglu. Root cause analysis of failures in microservices through causal discovery. *Advances in Neural Information Processing Systems*, 35:31158–31170, 2022.

[12] Zhenlan Ji, Pingchuan Ma, and Shuai Wang. Perfce: Performance debugging on databases with chaos engineering-enhanced causality analysis. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1454–1466. IEEE, 2023.

[13] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, et al. Megascale: Scaling large language model training to more than 10,000 gpus. *arXiv preprint arXiv:2402.15627*, 2024.

[14] Thomas Jiralerspong, Xiaoyin Chen, Yash More, Vedant Shah, and Yoshua Bengio. Efficient causal graph discovery using large language models. *arXiv preprint arXiv:2402.01207*, 2024.

[15] Jonathan Kaldor, Jonathan Mace, Michał Bejda, Edison Gao, Wiktor Kuropatwa, Joe O'Neill, Kian Win Ong, Bill Schaller, Pingjia Shan, Brendan Viscomi, et al. Canopy: An end-to-end performance tracing and analysis system. In *Proceedings of the 26th symposium on operating systems principles*, pages 34–50, 2017.

[16] Emre Kıcıman, Robert Ness, Amit Sharma, and Chenhao Tan. Causal reasoning and large language models: Opening a new frontier for causality. *arXiv preprint arXiv:2305.00050*, 2023.

[17] Wai-Yin Lam, Bryan Andrews, and Joseph Ramsey. Greedy relaxations of the sparsest permutation algorithm. In *Uncertainty in Artificial Intelligence*, pages 1052–1062. PMLR, 2022.

[18] Mingjie Li, Zeyan Li, Kanglin Yin, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. Causal inference-based root cause analysis for online service systems with intervention recognition. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3230–3240, 2022.

[19] JinJin Lin, Pengfei Chen, and Zibin Zheng. Microscope: Pinpoint performance issues with causal graphs in micro-service environments. In *Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings 16*, pages 3–20. Springer, 2018.

[20] Chenghao Liu, Wenzhuo Yang, Himanshu Mittal, Manpreet Singh, Doyen Sahoo, and Steven CH Hoi. Pyrca: A library for metric-based root cause analysis. *arXiv preprint arXiv:2306.11417*, 2023.

[21] Chenxi Liu, Yongqiang Chen, Tongliang Liu, Mingming Gong, James Cheng, Bo Han, and Kun Zhang. Discovery of the hidden world with large language models. *arXiv preprint arXiv:2402.03941*, 2024.

[22] Stephanie Long, Alexandre Piché, Valentina Zantedeschi, Tibor Schuster, and Alexandre Drouin. Causal discovery with language models as imperfect experts. *arXiv preprint arXiv:2307.02390*, 2023.

[23] Yuan Meng, Shenglin Zhang, Yongqian Sun, Ruru Zhang, Zhilong Hu, Yiyin Zhang, Chenyang Jia, Zhaogang Wang, and Dan Pei. Localizing failure root causes in a microservice through causality inference. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2020.

[24] Narmada Naik, Ayush Khandelwal, Mohit Joshi, Madhusudan Atre, Hollis Wright, Kavya Kannan, Scott Hill, Giridhar Mamidipudi, Ganapati Srinivasa, Carlo Bifulco, et al. Applying large language models for causal structure learning in non small cell lung cancer. *arXiv preprint arXiv:2311.07191*, 2023.

[25] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. Use of formal methods at amazon web services. page 16, 2014.

[26] OpenTelemetry. Opentelemetry documentation. `https://opentelemetry.io/docs/`, 2024. Accessed: 2024-05-22.

[27] Judea Pearl. *Causality*. Cambridge university press, 2009.

[28] Haoran Qiu, Subho S. Banerjee, Saurabh Jha, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. FIRM: An intelligent fine-grained resource management framework for SLO-Oriented microservices. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 805–825. USENIX Association, November 2020.

[29] Redis. Redis enterprise cluster architecture. `https://redis.io/redis-enterprise/technology/redis-enterprise-cluster-architecture/`. Accessed: 2024-05-22.

[30] Devjeet Roy, Xuchao Zhang, Rashi Bhave, Chetan Bansal, Pedro Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. Exploring llm-based agents for root cause analysis. *arXiv preprint arXiv:2403.04123*, 2024.

[31] Jonathan Soifer, Jason Li, Mingqin Li, Jeffrey Zhu, Yingnan Li, Yuxiong He, Elton Zheng, Adi Oltean, Maya Mosyak, Chris Barnes, Thomas Liu, and Junhua Wang. Deep learning inference service at microsoft. In *2019 USENIX Conference on Operational Machine Learning (OpML '19)*, May 2019.

[32] Peter Spirtes, Clark N Glymour, and Richard Scheines. *Causation, prediction, and search*. MIT press, 2000.

[33] Masayuki Takayama, Tadahisa Okuda, Thong Pham, Tatsuyoshi Ikenoue, Shingo Fukuma, Shohei Shimizu, and Akiyoshi Sannai. Integrating large language models in causal discovery: A statistical causal approach. *arXiv preprint arXiv:2402.01454*, 2024.

[34] Hanzhang Wang, Phuong Nguyen, Jun Li, Selcuk Kopru, Gene Zhang, Sanjeev Katariya, and Sami Ben-Romdhane. Grano: Interactive graph-based root cause analysis for cloud-native distributed data platform. *Proceedings of the VLDB Endowment*, 12(12):1942–1945, 2019.

[35] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[36] Cheng Zhang, Stefan Bauer, Paul Bennett, Jiangfeng Gao, Wenbo Gong, Agrin Hilmkil, Joel Jennings, Chao Ma, Tom Minka, Nick Pawlowski, et al. Understanding causality with large language models: Feasibility and opportunities. *arXiv preprint arXiv:2304.05524*, 2023.

[37] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36, 2024.

A Sample Input Files

A.1 Trace File

Following is a sample trace file from ModelServingS dataset. This can be automatically extracted from traces of requests and system deployment configs.

```
{
    "request.Client": {
        "service_description": "client that sends request to the router",
        "resources": {},
        "callees": [
            "Client-Router"
        ]
    },
    "request.Client.Client-Router": {
        "service_description": "network communication that
            send request from client to router",
        "resources": {},
        "callees": [
            "Router"
        ]
    },
    "request.Client.Router": {
        "service_description": "router that processes and
            dispatches request to different servers",
        "resources": {},
        "callees": [
            "Router-Queue_0"
        ]
    },
    "request.Client.Router-Queue_0": {
        "service_description": "network communication that
            send request from router to servers",
        "resources": {},
        "callees": [
            "Queue_0"
        ]
    },
    "request.Client.Queue_0": {
        "service_description": "when requests are received at a server,
            they are buffered in the queue and waited to be executed.
            Requests will be dequeued and processed by the batcher
            when resources are available",
        "resources": {},
        "callees": []
    },
    "request.Client.Batcher_0": {
        "service_description": "when resources are available,
            the batcher will check the queue and create a batch of
            min(available requests, max batch size) requests
            and send it to the model inference service",
        "resources": {},
        "callees": [
            "Queue_0"
        ]
    },
    "request.Client.ModelInference_0": {
        "service_description": "model inference service that
```

```
            runs the model inference on the batched requests",
        "resources": {
            "GPU_0": "A A100 GPU"
        },
        "callees": [
            "Batcher_0",
            "ModelInference_0-Client"
        ]
    },
    "request.Client.ModelInference_0-Client": {
        "resources": {},
        "service_description": "network communication that
            send the model inference result back to the client",
        "callees": []
    }
}
```

A.2 Measurement Description

Following is a measurement description file explaining all the available metrics for different kinds of system components. These information can usually be found from documentation of observability framework and the observed system.

```
{
    "Client": {
        "request_level": {
            "latency": "The time taken for a request to be processed from
                the time it is sent to the time the response is received"
        }
    },
    "Client-Router": {
        "request_level": {
            "latency": "The time taken for an invoked service to
                process the request"
        }
    },
    "Router": {
        "request_level": {
            "latency": "The time taken for an invoked service to
                process the request"
        },
        "service_level": {
            "throughput": "The number of requests that
                a service successfully processes per second"
        }
    },
    "Router-Queue": {
        "request_level": {
            "latency": "The time taken for an invoked service to
                process the request"
        }
    },
    "Queue": {
        "request_level": {
            "latency": "The time waited by a request in the queue
                before it is dequeued",
            "queue_length": "The number of requests waiting in the queue
                when a request is enqueued"
        },
```

```
        "service_level": {
            "enqueueing_rate": "The rate at which requests  are being
                enqueued, in requests per second",
            "dequeueing_rate": "The rate at which requests are being
                dequeued, in requests per second"
        },
        "to_exclude": {
            "service_level": [
                "throughput"
            ]
        }
    },
    "Batcher": {
        "service_level": {
            "max_batch_size": "The maximum size of
                a batch that can be created"
        }
    },
    "ModelInference": {
        "request_level": {
            "latency": "The time taken for
                an invoked service to process the request"
        },
        "service_level": {
            "execution_batch_size": "The number of
                requests that are processed in a single batch",
            "throughput": "The number of requests that
                a service successfully processes per second"
        }
    },
    "ModelInference-Client": {
        "request_level": {
            "latency": "The time waited by a request in
                the queue before it is dequeued"
        }
    },
    "GPU": {
        "resource_level": {
            "utilization": "The percentage of time that
                a resource is busy processing requests"
        }
    }
}
```

Note that `to_exclude` is used to help Atlas generate a more accurate causal graph. In our context, throughput is not a relevant metric for the queue service since we already have the enqueueing rate and dequeueing rate as metrics.

A.3 Common Metrics

This file is used to realize metric enumeration for a more complete graph construction as discussion in §3.

```
{
    "service": {
        "request_level": {
            "latency": "The time taken for an invoked service
                to process the request"
        },
        "service_level": {
```

```
            "throughput": "The number of requests that
                a service successfully processes per second"
        }
    },
    "resource": {
        "resource_level": {
            "power": "A measure of capacity of
                a resource to process requests",
            "utilization": "The percentage of time that
                a resource is busy processing requests"
        }
    }
}
```

B Sample Prompt

Here is an sample prompt for the "Queue_0" to figure out the causal relationship between its "queue_length" and "dequeueing_rate".

> [
> {'role': 'system', 'content': 'You are a service named **Queue** in a software system and your job is: **when requests are received at a server, they are buffered in the queue and waited to be executed. Requests will be dequeued and processed by the batcher when resources are available.** You are about to figure out the causal relationship between a metric of you and a metric of another system component.'},
> { 'role': 'user', 'content': **queue_length** is a metric of you and it means the **number of requests waiting in the queue when a request is enqueued**. **dequeueing_rate** is another metric of you and it means **the rate at which requests are being dequeued, in requests per second.** If we can only choose one, which of the following cause-and-effect relationship is more likely?
> A. A change in "queue_length" of you directly causes a change in "dequeueing_rate" of Queue_0.
> B. These two metrics do not directly influence each other, even if they might be correlated through other components in the system.
> C. A change in "dequeueing_rate" of Queue_0 directly causes a change in queue_length of you.
> Please think step by step to make sure that you have the right answer. Please select from one of the following options: [A, B, C] as the final answer. Put it as the only content in the last line.}
>]

Here is another sample prompt for the "ModelInference" to figure out how its "execution_batch_size" influence the max_batch_size of "Batcher".

> ['role': 'system', 'content': 'You are a service named **ModelInference** in a software system and your job is: **model inference service that runs the model inference on the batched requests**.You are about to figure out the causal relationship between a metric of you and a metric of another system component.', 'role': 'user', 'content': '**Batcher_0** is the next service that requests will invoke after you finish processing them, whose job is: **when resources are available, the batcher will check the queue and create a batch of min(available requests, max batch size) requests and send it to the model inference service execution_batch_size** is a metric of you and it means **The number of requests that are processed in a single batch**. **max_batch_size** is a metric of the next service and it means **The maximum size of a batch that can be created**. If we can only choose one, which of the following cause-and-effect relationship is more likely?
> A. These two metrics do not directly influence each other, even if they might be correlated through other components in the system.

B. A change in "max_batch_size" of Batcher_0 directly causes a change in execution_batch_size of you.

C. A change in "execution_batch_size" of you directly causes a change in "max_batch_size" of Batcher_0.

Please think step by step to make sure that you have the right answer. Please select from one of the following options: [A, B, C], as the final answer. Put it as the only content in the last line.']

Note that each query are repeated multiple times with varied order of options until a majority vote agreement is reached.