From Logs to Causal Inference: Diagnosing Large Systems

Markos Markakis
MIT CSAIL
Cambridge, MA
markakis@mit.edu

Brit Youngmann
Technion
Haifa, Israel
brity@technion.ac.il

Trinity Gao
MIT CSAIL
Cambridge, MA
trinityg@mit.edu

Ziyu Zhang
MIT CSAIL
Cambridge, MA
sylziyuz@mit.edu

Rana Shahout
Harvard University
Cambridge, MA
rana@seas.harvard.edu

Peter Baile Chen
MIT CSAIL
Cambridge, MA
peterbc@mit.edu

Chunwei Liu
MIT CSAIL
Cambridge, MA
chunwei@mit.edu

Ibrahim Sabek
University of
Southern California
Los Angeles, CA
sabek@usc.edu

Michael Cafarella
MIT CSAIL
Cambridge, MA
michjc@csail.mit.edu

ABSTRACT

Causal inference can quantify cause-effect relationships in domains as varied as medicine, economics and public policy. Production computer systems exhibit a similar level of complexity and a recurring need to diagnose phenomena quickly. However, they are only observed imperfectly, often via long, messy, semi-structured logs.

In this work, we want to accelerate large systems debugging by applying causal inference over logs, enabling engineers to diagnose problems and assess interventions in a principled manner. Our framework achieves this through two human-in-the-loop modules: (1) The **Candidate Cause Ranker**, through which one can determine the causes of a variable without running a full causal discovery algorithm; and (2) the **Interactive Causal Graph Refiner**, which helps engineers compute an unbiased estimation of their effect of interest without extensive manual causal graph verification. Both modules are powered by the insight that only part of the causal graph of the system is needed to correctly quantify an effect of interest. We also provide a data preparation pipeline, the **Log Converter**, which transforms raw, messy, real-world logs into an appropriate tabular input for causal inference, using methods drawn from data transformation, cleaning, and extraction.

We evaluate LOGos, a prototype implementation, on both real-world and synthetic logs and find that: (1) The **Candidate Cause Ranker** achieved an average precision 1.08×-18× higher than the baselines, in interactive time; (2) The **Interactive Causal Graph Refiner** required a number of causal judgments $1.61\times-16.83\times$ lower than the baselines; and (3) The latency of **Log Converter** scaled linearly with three measures of the complexity of a log: length, distinct templates, and fraction of tokens that are variables.

PVLDB Reference Format:
Markos Markakis, Brit Youngmann, Trinity Gao, Ziyu Zhang, Rana Shahout, Peter Baile Chen, Chunwei Liu, Ibrahim Sabek, and Michael Cafarella. From Logs to Causal Inference: Diagnosing Large Systems. PVLDB, 18(1): XXX-XXX, 2024.
doi:XX.XX/XXX.XX

Proceedings of the VLDB Endowment, Vol. 18, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/mitdbg/logos.

1 INTRODUCTION

The scale and complexity of today's computer systems make failures frequent [32, 45, 84, 112] and their diagnosis challenging, especially in production [47, 81, 99]. Traditional systematic debugging techniques, like testing [19, 61, 63, 69, 80], formal verification [13, 24, 40, 103] and simulation [46, 52, 104], often fall short for problems on a large complex production system [81]. Operations teams do not have the time and expertise (or often even the access permissions) to fully ascertain the correctness of the codebase [45, 87]. Instead, they have to work backward from each failure towards its cause using observational data collected from the system, most notably large volumes of logs, usually in text format.

Episode 1. *Alex is an on-call engineer at a startup that offering a cloud-based data service. Several users are complaining that a certain feature, which triggers a query on a user-specific PostgreSQL back end database, is very slow. Alex's manager has tasked Alex with discovering why the feature is so slow for these specific users and the best way to fix it quickly. But all Alex has to go on is text logs!*

User Goal. Informally, we have heard reports from operations teams spending tens of human-hours with tools such as Splunk [94] or Datadog [21] in order to diagnose a poorly-understood problem. Such log analysis tools offer a good starting point, given their broad coverage of software and hardware components through existing log management infrastructure. However, if users could simply *ask* such a tool *why* an observed problem took place and *by how much* we expect each of a collection of remedies to help, then we might dramatically lower the Mean Time to Repair (MTTR).

To answer such questions in a principled manner, we turn to the growing field of causal reasoning [74], which aims to quantitatively describe cause-effect mechanisms. Causality has provided scientists with a common language to express and evaluate hypotheses about complex systems across diverse domains [11, 16, 78, 90, 100]. In this work, we adopt the standard theory of causality developed by Pearl [74], where the goal is to correctly calculate an *Average Treatment Effect (ATE)* corresponding to a hypothesis about the observed phenomenon - for example, in Alex's case, if one more worker is added to PostgreSQL, how much will mean query latency change? What if the working memory is increased by 1MB?

To calculate such ATEs, Pearl's theory requires not only a set of observations, but also another input called a *causal graph.* The causal graph represents each observed variable as a node and each potential direct causal relationship as a directed edge, and can be used to infer which variables to *adjust for* (the *adjustment set*) in order to calculate an ATE correctly. Without a proper adjustment set, an ATE calculation can be rendered inaccurate due to *confounding bias.* We aim for a causal diagnosis tool that can help address a wide range of software and data systems problems, enabling a user experience like the following:

Episode 2. *Alex uses our framework to rapidly find that the maximum allowed number of parallel workers exerts a high absolute ATE on mean query latency. Alex forwards this to the database team to ensure that this setting is set appropriately, restoring user performance.*

Technical Problem. As described, correctly estimating an ATE from observational data requires a valid adjustment set; and finding a valid adjustment set is in turn made possible by an accurate causal graph. Obtaining such a causal graph is therefore our main concern.

For problem instances with few variables, a causal graph is usually assumed to be manually curated by a domain expert [60, 101]. However, manual curation is a daunting and error-prone task over the possibly hundreds or thousands of log-derived variables [68], since the number of edges to consider for inclusion is quadratic in the number of variables. Instead, it is often possible to obtain a (partial) causal graph from the data itself, using one of several available *causal discovery* algorithms [29, 89, 93, 102, 113, 121]. However, as we will show in Section 2.2, these methods fall short for log datasets because of three **challenges of log data**:

(1) **Functional Dependencies**: Logging aims to capture as much information about the system as possible, even if redundant, since reproducing a sequence of events may not be an option. This leads to widespread functional dependencies in the resulting datasets, which cause issues with the conditional independence tests used by many causal discovery algorithms.
(2) **Large Number of Variables**: The sophistication and modularity of modern systems leads to logs capturing hundreds or thousands of variables. The exponential complexity of causal discovery algorithms can be prohibitive in this setting.
(3) **Biased Data Collection**: Capturing a large number of variables at a high frequency can mean that an interesting log message is drowned out by strong "common case" message sequences, which will be more readily identified during causal discovery.

The key difficulty we tackle in this paper is efficiently obtaining a causal graph from log data despite these challenges.

Our Approach. To combat the three challenges above, we use a key insight: **correctly calculating an ATE only requires part of the causal graph**. As long as we focus on this part, we can solicit human input efficiently, without overwhelming the user. We therefore approach causal inference over log data using a *human-in-the-loop* framework, combining the intelligent use of available data and a judicious solicitation of user input. This leads to a causal graph sufficient for calculating the ATE of interest correctly, with significantly less user input than a naive approach. We achieve this through two human-in-the-loop modules: the **Candidate Cause Ranker** and the **Interactive Causal Graph Refiner**, together with a data preparation pipeline, the **Log Converter**. Our framework can be an important tool for engineers managing complex systems.

The **Candidate Cause Ranker** helps users quickly discover causes of their phenomenon of interest. In each call, the user specifies a variable E and is presented with a ranked list of candidate causes $\{C_i\}$. The user can then tap their expertise to evaluate the plausibility of each candidate C_i and possibly add the edge $C_i \longrightarrow E$ to their causal graph. Crucially, if some candidates are functionally dependent, the user has the freedom to include the most appropriate one for their analysis into the causal graph. Thus this module addresses **Challenge 1**. By iteratively applying this process, users can navigate log data efficiently to find a "root cause" T of their original outcome variable of interest O (e.g. mean latency, for Alex).

However, discovering *one* causal path from T to O is not enough to correctly calculate the ATE, since there may be additional variables confounding this effect. Such variables should be added to the graph and used to derive the adjustment set. For example, Alex may mis-estimate the impact of increased parallelism on mean query latency if the changes made to *other* settings (e.g. the working memory) in order to accommodate more parallelism are not considered. The **Interactive Causal Graph Refiner** addresses this problem through another human-in-the-loop process. In particular, given T, O and the user's current causal graph, it presents to the user graph edits that would *maximally affect* the ATE of T on O. The user can then consider the plausibility of such edits and decide whether to apply them, progressively increasing the accuracy of their graph and the robustness of their ATE calculation. This frees the user from having to consider the possible influence of *every* variable in the dataset, addressing **Challenge 2**.

Of course, even before a causal graph can be built, one must identify the problem variables and extract them from the textual log. Therefore, alongside our human-in-the-loop approach to causal graph construction we present the **Log Converter**, which implements a pipeline transforming raw log inputs into a tabular dataset appropriate for causal analysis. This dataset both serves as the input to our human-in-the-loop modules, and is the second input to Pearl's model for ATE calculation (alongside the graph itself). After parsing the logs, our pipeline uses Large Language Models to derive human-understandable variable tags, before distilling log information around the user's desired causal units and generating appropriate aggregated variables for every causal unit to maximize empirical entropy. This approach to aggregation allows interesting observations to cut through the noise of uninteresting log messages, combating **Challenge 3**.

We speculate that our approach may also work well in other domains involving challenges similar to those of logs. We will explore this idea in future work.

Overview of Related Work and Novelty. Large system debugging is an active research area with serious commercial implications. However, we are the first to simultaneously tap *principled Pearl-style causality* and the *wealth of information present in text logs.*

On one hand, some works leverage causality but do not tap the ubiquitous textual log data. ExplainIt! [44] draws inspiration from causal techniques to build a root cause analysis engine over diverse time series, while Sage [28] uses modular causal graphs based on telemetry data to localize failures in microservice architectures.

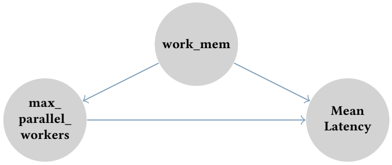


Figure 1: An example partial causal graph for Alex's problem.

CausalSim [4] leverages causality to make the most out of existing tabular RCT data, avoiding further trials.

On the other, there are works that operate on logs but model causality informally at best. Aggarwal et al. only model causality among the *counts* of error-level log messages [1], while we consider the actual *values* embedded in them. Falcon [66] and Horus [67] use simple *temporal precedence* as a proxy for causality, whereas we model causality using the probabilistic Pearl model instead. Additional related work is discussed in Section 9.

Our recent demonstration of a prototype of this work [59] allowed us to gather invaluable feedback on the underlying technical ideas, as well as on practical considerations regarding our system. Moreover, the **Interactive Causal Graph Refiner** is built on ideas from another recently presented work of ours [60], which captured community interest leading to a Best Paper award at GUIDE-AI '24.

Contributions. In summary, we make the following contributions:

- We propose a human-in-the-loop framework for efficiently applying causal inference on data derived from logs.
- We introduce the **Candidate Cause Ranker**, a module for uncovering candidate causes using curated data-driven suggestions.
- We present the **Interactive Causal Graph Refiner**, a module for refining causal graphs for accurate calculation of an ATE.
- We describe the **Log Converter**, a data transformation pipeline from logs into a tabular representation for causal inference.
- We evaluate LOGos, an open-source implementation of our proposed framework [58] and the first end-to-end system that can go from log files to ATEs, and find that:
  - The average precision achieved by the **Candidate Cause Ranker** was 1.08×-1.66× higher than Regression and 1.54×-18× higher than LangModel, while remaining interactive.
  - The number of judgments required by the **Interactive Causal Graph Refiner** was 1.61×-2.50× lower than Regression and 5.50×-16.83× lower than LangModel.
  - The **Log Converter** scaled linearly with each of three measures of the complexity of a log: length, distinct templates, and fraction of tokens that are variables.

2 BACKGROUND

2.1 Background on Causality

We now introduce some notation and discuss elements of the conventional causal inference framework developed by Pearl [74, 75].

Notation. We use uppercase letters to represent variables (e.g. V) and lowercase letters to denote their values (e.g. $V = v$). $V \longrightarrow W$ indicates a directed edge from V to W. Bold font denotes sets of variables, edges or edge edits. Calligraphic font denotes causal graphs (e.g. $\mathcal{G}$). For directed graphs, we distinguish a *path* (where edges need not share the same orientation) from a *directed path*.

The ATE. Causal inference estimates the *Average Treatment Effect (ATE)* of a treatment T on an outcome O over a population of *causal units* - e.g. PostgreSQL sessions in Alex's case. The ATE may be distorted by *confounding variables* (or *confounders*) $\mathbf{Z}$ which influence both T and O [74, 75]. For example, as shown in Figure 1, if Alex estimates the ATE of `max_parallel_workers` on mean latency, a confounder may be `work_mem` - more working memory can accelerate individual operations and reduce latency directly, but it may also mean a lower allowable level of parallelism, in the interest of managing the total available memory in the system.

To avoid confounders, one can collect *interventional* data, where T no longer depends on $\mathbf{Z}$ because it is externally assigned - e.g. through a randomized controlled trial (RCT). However, RCTs can be expensive and time-consuming - for example, Alex would have to subject users to randomly selected settings. Depending on the domain, RCTs may also be unethical or completely infeasible. Thankfully, we can also compute the ATE over *observational* data, such as the log data we focus on, as long as we *adjust for the confounders*:

Definition 1 (ATE from Observational Data [74]). *Given a treatment T, an outcome O, and a valid adjustment set $\mathbf{Z}$,*

$$ATE(T, O) = \mathbb{E}_Z \left[\mathbb{E}\left[O \mid T = 1, \mathbf{Z} = z\right] - \mathbb{E}\left[O \mid T = 0, \mathbf{Z} = z\right] \right]$$

For the obtained ATE to be accurate, $\mathbf{Z}$ must be selected to correctly adjust for confounding bias. One method for achieving this is the back-door criterion [74], which we will now build towards.

Causal Model. A causal model [74] captures causal relationships among the problem variables. These relationships can be arbitrary, but they are most often assumed to be linear [74, 93]:

Definition 2 (Linear Causal Model). *Given variables V_i for $i = 1, \ldots, n$, with associated sets of direct causes (*parents*) $\mathbf{P}_i$ and error terms E_i, a **linear causal model** is a set of equations:*

$$V_i = E_i + \sum_{P \in \mathbf{P}_i} \lambda_P P, \quad \lambda_P \in \mathbb{R}, \quad i = 1, \ldots, n$$

Causal Graph. A causal model can be visualized as a *causal graph* [73], like Figure 1: a directed acyclic graph (DAG) with one node per variable, where a directed edge $V_i \longrightarrow V_j$ implies $V_i \in \mathcal{P}_j$. In this context, we may use "variable" to refer to the corresponding node in the causal graph, and we can extend the notion of a parent:

Definition 3 (Ancestors and Descendants). *Each* ancestor *W of V_i has a directed path* to *V_i ($W \in \mathbf{An}(V_i)$). Similarly, V_i has a directed path to each W among its* descendants *($W \in \mathbf{De}(V_i)$).*

Path Blocking and the Back-Door Criterion. A valid adjustment set is one that eliminates all sources of "indirect" causal influence. This is formalized through the concept of path blocking:

Definition 4 (Path Blocking [74]). ***Path p is blocked** by nodes $\mathbf{B}$ if and only if:*

(1) p has a chain $i \longrightarrow m \longrightarrow j$ or fork $i \longleftarrow m \longrightarrow j$ s.t. $m \in \mathbf{B}$; or
(2) p has a collider $i \longrightarrow m \longleftarrow j$ s.t. $(\{m\} \cup \mathbf{De}(m)) \cap \mathbf{B} = \emptyset$.

Definition 5. *A path p between T and O is a **T-backdoor path** if some directed edge on p has T as its destination.*

Definition 6 (Back-Door Criterion [74]). *A set $\mathbf{Z}$ satisfies the **backdoor criterion** relative to variables (T, O) in a DAG $\mathcal{G}$ if:*

(i) $\mathbf{Z} \cap \mathbf{De}(T) = \emptyset$; and
(ii) $\mathbf{Z}$ blocks every T-backdoor path between T and O.

In this case, $\mathbf{Z}$ is a sufficient adjustment set for $ATE(T, O)$.

Table 1: Causal discovery on the PostgreSQL dataset. ✓ and ● indicate a non-empty and empty causal graph, respectively; ▲ indicates a 30-minute timeout; and ✗ indicates an error.

Alg.	Independence Test / Method	Result	Alg.	Scoring Function	Result
PC [92]	fisherz	✗	GIN [106]	kci	▲
	mv_fisherz	✗		hsic	▲
	mc_fisherz	✗	GRaSP [51]	CV_general	✓
	kci	▲		marginal_general	✗
	chisq	✓		CV_multi	✗
	gsq	●		marginal_multi	✗
	d_separation	✗		BIC	✗
FCI [93]	fisherz	✗		BDeu	✓
	kci	▲	GES [14]	CV_general	▲
	chisq	✓		marginal_general	✗
	gsq	●		CV_multi	▲
LiNGAM [89]		✗		marginal_multi	✗
				BIC	✗
Exact	dp [91]	✗		BIC_from_cov	✗
	astar [110]	✗		BDeu	✓

2.2 How Existing Approaches Fall Short

We will next showcase why existing approaches to causal discovery are ill-suited for our use case through a motivating example.

The PostgreSQL Dataset. This 20MB dataset, presented fully in Section 8.1.4, mimics Alex's predicament. It includes collated logs from 96 workload runs of queries from the TPC-DS benchmark [77], issued against PostgreSQL. Before each run, we modified 6 PostgreSQL configuration knobs, each of which has a known impact on query latency. We selected the combinations of knob settings to introduce confounding between work_mem and max_parallel_workers. This is a small representative example of a broader class of confounding related to resource sharing that can come up often in the real world, albeit with much larger volumes of data. Using the **Log Converter**, we converted this log dataset into a tabular format with 172 variables. We then attempted to derive a causal graph over these variables. **A useful causal graph would, at a minimum, capture the hand-constructed structure shown in Figure 1: work_mem and max_parallel_workers each affect query latency and they confound each other's effect.**

Causal Discovery. We tested 29 different combinations of causal discovery algorithms and configuration parameters from Causal-Learn [116] on the PostgreSQL dataset. As shown in Table 1, 16 produced errors, 6 timed out after 30 minutes and 2 only produced an empty graph. Of the 5 remaining cases:

- PC-chisq and FCI-chisq: The variable capturing query latency was absent from the 4-node graph produced.
- GRaSP-BDeu and GES-BDeu: One 5-node connected component included the variable capturing query latency and no configuration-related variables. A different 4-node connected component included the variables for work_mem and max_parallel_workers.
- GRaSP - CV_general: The graph showed query duration as *causing* work_mem and totally lacked max_parallel_workers.

Analysis The 3 problematic features of log data from Section 1 directly lead to the failures of existing causal discovery algorithms:

(1) **Functional Dependencies → Violated Assumptions**: Many of the 16 errors produced relate to violated algorithm assumptions, most notably due to a singular correlation matrix among the variables, which leads to errors for some conditional independence tests. This singularity is an expression of the functional dependencies in log data. While such dependencies could be automatically eliminated, this could unintentionally eliminate variables crucial to system understanding in favor of less interpretable alternatives. For example, in PostgreSQL, removing all the variables associated with the configuration knobs from the causal graph, because the session ID is sufficient to determine them, would clearly be unhelpful for Alex.

(2) **Large Number of Variables → Prohibitive Running Time**: Even when algorithmic assumptions are met, deriving a full causal graph can be time-consuming, since many of the algorithms offer exponential complexity, and log-derived datasets can contain hundreds to thousands of variables. When diagnosing a production problem live, such complexity is unacceptable.

(3) **Biased Data Collection → Non-Actionable Graphs**: Even when the algorithm finishes quickly, the resulting graph may be unhelpful, because it lacks some variables of interest. As seen above, no algorithm places query latency, work_mem, and max_parallel_workers in the same connected component, with some not even including all of these variables in the graph. This is because causal discovery algorithms capture the causal relationships *most salient in the data*, but the biased logging may mean that these relationships are not the *most interesting* ones, of which there may only be weaker evidence.

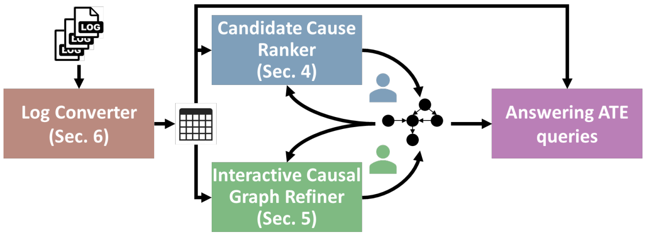

LOG
Log Converter (Sec. 6)
Candidate Cause Ranker (Sec. 4)
Interactive Causal Graph Refiner (Sec. 5)
Answering ATE queries

Figure 2: An architectural overview of our framework.

Since fully automatic causal discovery falls short, we develop a human-in-the-loop framework that combines data-driven causal hypotheses with the user's expertise.

3 PROBLEM DEFINITION AND OVERVIEW

We now define the problems we address and present our framework, illustrated in Figure 2. Input logs are transformed by the **Log Converter** into a tabular dataset, from which the **Candidate Cause Ranker** and the **Interactive Causal Graph Refiner** help derive a partial causal graph, using a human in the loop. The tabular dataset and the causal graph can be then used to answer ATE queries.

3.1 Problem Definitions

As seen in Section 2.2, causal discovery algorithms fall short when applied to log-derived data. At the same time, although the user *could* fully specify the causal graph $\mathcal{G}_{real}$ by hand, this would be impractical for large $\mathbf{V}$ as it would require $O(|\mathbf{V}|^2)$ causal *judgments*:

Definition 7 (Judgment). *A user judgment for (A, B) outputs whether the causal graph should include $A \longrightarrow B$, $A \longleftarrow B$, or neither.*

However, many of the judgments produced in such an exhaustive pass would neither help discover the root cause T of a variable of interest O, nor significantly impact the value of $ATE(T, O)$ once T is specified. The latter is true because many edges would be downright irrelevant when assessing the backdoor criterion (Definition 6).

This observation reveals a path to reducing the work required by the user, bridging manual graph construction and automatic causal discovery: using the log data to *solicit user judgments in the most impactful order*. Ensuring a high-quality ordering directly translates to reduced user effort for a given target downstream quality.

Given this approach, we present two problems, each addressed by one of our framework's human-in-the-loop modules:

Problem 1 (Candidate Cause Ranking). *Let O be a variable of interest. Rank highly relevant candidate causes for O based on the available observations and solicit user judgments in this order, aiming to maximize* average precision (AP)*, defined as:*

$$AP = \frac{\sum_{i=1}^{ranking\_length} \textsc{PrecisionAmongTop}(i) \cdot \textsc{IsPositive}(i)}{\#\ Ground\ Truth\ Positive\ Elements}$$

Problem 2 (Interactive Causal Graph Refinement). *Let an ATE of interest, $ATE(T, O)$, and an initial causal graph $\mathcal{G}_{init}$. Minimize the number of user judgment solicitations needed to converge to the ground truth value of $ATE(T, O)$.*

3.2 Candidate Cause Ranker

The **Candidate Cause Ranker** (Section 4) addresses Problem 1. It operates on a tabular dataset D with variables **V**, derived from the log. We will explain how this dataset is obtained using the **Log Converter** in Section 3.4. The user can invoke the **Candidate Cause Ranker** on an variable E, which we will refer to as the *effect variable*, yielding a ranked list of candidate causes. By evaluating the plausibility of each of these causes C_i, the user can decide on the appropriate state of the corresponding edge $C_i \longrightarrow E$, possibly adding it to the causal graph. The user can then apply this process recursively by first invoking invoking **Candidate Cause Ranker** on O, then on one of the C_i for which $C_i \longrightarrow O$ was added to the graph, and so forth, until a satisfactory root cause T is discovered.

3.3 Interactive Causal Graph Refiner

The **Candidate Cause Ranker** lets the user find T and pose their ATE query of interest: $ATE(T, O)$. However, the user has so far only unveiled part of the causal graph around T and O, which means that computing $ATE(T, O)$ may be erroneous. Per Section 2, correctly computing $ATE(T, O)$ from an observational dataset like D requires a valid adjustment set, which can be derived from a causal graph using the backdoor criterion. But to apply the backdoor criterion correctly, one must have access to a sufficiently refined graph, which includes all the T-backdoor paths between T and O.

The **Interactive Causal Graph Refiner** (Section 5) helps achieve this efficiently. It finds promising adjustment set edits and transforms them to sets of causal graph edits, soliciting the corresponding judgments from the user. This helps the user progressively narrow the possible range of values for $ATE(T, O)$.

3.4 Log Converter

The two human-in-the-loop modules presented so far both operate on a tabular dataset D. Although deriving *some* tabular dataset from the log can be easily achieved using a log parsing algorithm [36, 114, 120], there are two usability issues not covered by this approach:

- **Opaque Variables**: Most log parsing algorithms *identify* variables but do not assign them meaningful names, which users need in order to reason about them interactively.
- **Per-Message Granularity**: Log parsing creates a record per log message, but users are interested in questions about larger units: sessions, machines etc.

Algorithm 1 Rank candidate causes for an effect variable E.

```
function ExploreCandidateCauses(D, $E$)
    X ← D \ $E$
    X, $scale$ ← StandardScaleColumns(X)
    $coefs$ ← LASSO(X, $E$)
    $unscaled\_coefs$ ← Unscale($coefs$, $scale$)
    $res = \emptyset$
    for $V \in$ X do
        if $unscaled\_coefs(V) \neq 0$ then
            $slope, p\_value$ ← OLSLinearRegression($V$, $E$)
            $res \leftarrow res \cup \{V, slope, p\_value\}$
    return SortAscending($res$, by=$p\_value$)
```

The **Log Converter** (Section 6) provides a data preparation pipeline that addresses these issues. It utilizes Large Language Models to derive meaningful tags for each log-derived variable, provides the ability to aggregate log information over meaningful causal units, and computes entropy-maximizing aggregates of each variable. The resulting dataset both serves as the input to our human-in-the-loop modules, and is the second input to Pearl's model for ATE calculation (alongside the graph itself).

4 CANDIDATE CAUSE RANKER

The **Candidate Cause Ranker** helps discover a directed path from a sufficiently informative root cause T to the outcome variable of interest O "backwards", letting the user eventually specify the *causal question* they want to answer quantitatively - the $ATE(T, O)$.

4.1 Pruning and Ranking Candidate Causes

While using this module, the user will need to make a series of judgments based on the returned ranking. A high-relevance ranking is therefore clearly desirable. We achieve this through *edge pruning*. The guiding principle is as follows: the user's knowledge of the system is invaluable for distinguishing *which quantitative relationships* between pairs of variables are indeed manifestations of system-related causal mechanisms, as opposed to spurious correlations; however, the *absence of a quantitative relationship* can be a reliable indicator that a causal relationship does not exist, as long as the dataset is fairly representative of practical cases [44, 74].

In particular, note from Definition 2 that each variable is linearly related to its parents, up to its own error term. As such, if we run a multivariate regression of variable E on the remaining variables, and some variable V gets assigned a zero coefficient, we can disregard $V \longrightarrow E$: V does not impact E with its fluctuations, given the other variables. If we further assume that the causal graph is sparse, a common assumption in causal discovery literature [15, 92], we can prune more edges by using a *sparse regression* algorithm. This will drive very small coefficients, likely to be an artifact of data availability rather than indicative of causality, to zero. This process requires no user input and typically prunes the vast majority of edges, since most variables are pairwise unrelated.

Concretely, we use LASSO [97], as presented in Algorithm 1. First, we apply LASSO on the variables, normalized by subtracting the mean and scaling the variance (lines 3-5). We only perform further work for each variable V selected by LASSO (lines 7-8). In particular, we apply an ordinary least-squares (OLS) linear regression to each candidate causal relationship, recording the slope and p-value (lines 6, 9-10). The results are sorted by increasing p-value (line 11), placing the candidates with the most reliable relationships to E (lowest p-value) at the top. The user can then inspect each returned candidate V and decide whether to add $V \rightarrow E$ to the causal graph. By iteratively calling ExploreCandidateCauses(E), the user can arrive at a satisfactory "root cause" variable T.

4.2 Computational Complexity

If the threshold for convergence of the objective function is σ, a LASSO solution can be found in $O(|\mathrm{D}||\mathbf{V}|\sigma^{-1/2})$ [10, 115], where $|\mathrm{D}|$ is the number of data points and $\mathbf{V}$ is the set of variables. Each regression requires $O(|\mathrm{D}||\mathbf{V}|^2+|\mathbf{V}|^3)$ time, so the overall complexity is $O(|\mathrm{D}||\mathbf{V}|\sigma^{-1/2} + m|\mathrm{D}||\mathbf{V}|^2 + m|\mathbf{V}|^3)$, if LASSO finds m candidates.

5 INTERACTIVE CAUSAL GRAPH REFINER

Having found which ATE to calculate ($ATE(T, O)$), the user must now focus on finding an appropriate adjustment set. The **Interactive Causal Graph Refiner** helps the user achieve this using a "sensitivity" approach: it finds the single-variable adjustment set edit which will yield the *maximum absolute change* in $ATE(T, O)$. It then maps this change to causal graph edits and solicits user judgments for each of the edges involved in these edits.

Over time, this helps converge to the correct value of $ATE(T, O)$, by having the user consider graph changes in decreasing order of impact and therefore tightening the region in which $ATE(T, O)$ lies. We first presented a version of this approach in previous work [60]. We present here an algorithmically refined version.

The next judgment to be solicited from the user is provided by SolicitJudgment, presented in Algorithm 2. It takes as inputs the current causal graph $\mathcal{G}$ and dataset D, alongside the variables T and O. The user can also provide a set of *FIXED* edges $\mathbf{F}$, as well as a set of *BANNED* edges $\mathbf{B}$, which should not be added to the causal graph, or removed from it, respectively. The **Interactive Causal Graph Refiner** maintains two variables across invocations of this algorithm (line 1): *edits_cache* contains cached suggested edits from previous invocations of the algorithm, and $\mathcal{G}_{next}$ contains the causal graph that would result if the most recent user judgment agreed with the most recent suggested edit.

The algorithm first checks if the cache is non-empty and valid (line 3). The cache is invalid if the user did not judge according to the most recent suggested edit, since the causal graph edits must *all* be applied to achieve the desired effect on the adjustment set.

In that case, it proceeds by calculating the ATE(T, O) given $\mathcal{G}$ and D and finding a valid adjustment set using existing algorithms [96] (lines 4-5). It then considers each of the variables in D (line 8) and calls MapRemoval (Section 5.1) or MapAddition (Section 5.2) to find edge edits that would change that variable's adjustment set membership (lines 9-13). Next, it applies these edits to a copy of $\mathcal{G}$, and calculates the resulting ATE(T, O) and the associated ATE impact (lines 14-16). It keeps track of the edits that produce the maximum ATE impact and caches them (lines 6-7, 17-20). At this point a valid cache exits, so the algorithm removes its first element and computes the resulting graph, if the edit is accepted (lines 21-22). The edit is then returned for the user to judge (line 23).

Algorithm 2 Return the next judgment to be solicited, based on the adjustment set edit that would impact $ATE(T, O)$ maximally.

```
 1: State across invocations: edits_cache, $\mathcal{G}_{next}$
 2: function SolicitJudgment($\mathcal{G}$, D, $T$, $O$, F, B)
 3:     if edits_cache = ∅ or $\mathcal{G} \neq \mathcal{G}_{next}$ then
 4:         ate ← ATE($\mathcal{G}$, D, $T$, $O$)
 5:         base_adj_set ← AdjSet($\mathcal{G}$, $T$, $O$)
 6:         best_edits ← ∅
 7:         best_ate_impact ← 0
 8:         for $V \in$ (Nodes($\mathcal{G}$) \ {$T, O$}) do
 9:             S ← ∅
10:             if $V \in$ base_adj_set then
11:                 S ← MapRemoval($\mathcal{G}$, D, $T$, $O$, F, B, $V$)
12:             else
13:                 S ← MapAddition($\mathcal{G}$, D, $T$, $O$, F, B, $V$)
14:             $\mathcal{G}'$ ← ApplyEdits($\mathcal{G}$, S)
15:             ate' ← ATE($\mathcal{G}'$, D, $T$, $O$)
16:             ate_impact ← $\frac{|ate' - ate|}{|ate|}$
17:             if ate_impact > best_ate_impact then
18:                 best_edits ← S
19:                 best_ate_impact ← ate_impact
20:         edits_cache ← best_edits
21:     edit ← PopHead(edits_cache)
22:     $\mathcal{G}_{next}$ ← ApplyEdits($\mathcal{G}$, edit)
23:     return edit
```

Algorithm 3 Suggest causal graph edits that would remove a variable from the adjustment set.

```
 1: function MapRemoval($\mathcal{G}$, D, $T$, $O$, F, B, $V$)
 2:     S ← ∅
 3:     if $V \in$ An($T$) then
 4:         B, success ← BreakPaths($\mathcal{G}$, F, {$V$}, $T$ )
 5:         if not success then
 6:             return ∅
 7:         S ← B
 8:     T ← De($T$) ∪ {$T$}
 9:     T ← SortAscending(T, by= DirPathLengthFrom($T$))
10:     for $W \in$ T do
11:         if $W \rightarrow V$ is not BANNED then
12:             if ($V \rightarrow W$) ∉ $\mathcal{G}$ then
13:                 S ← S∪ ($W \rightarrow V$, Add)
14:             else
15:                 S ← S∪ ($V \rightarrow W$, Flip)
16:             return S
17:     return ∅
```

The hardest part is deriving a set of causal graph edits that corresponds to the desired effect on the adjustment set - i.e. the functionality implemented by MapRemoval and MapAddition. In general, multiple such sets may exist that achieve the same effect - e.g. removing a certain variable from the adjustment set. We present one possible approach to finding such a set, based on Definition 6.

5.1 The MapRemoval Algorithm

MapRemoval (Algorithm 3) finds a set of edge edits that ensures that V is not included in the adjustment set. It achieves this by making V a descendant of T, which means that V violates the first condition of Definition 6. We consider two cases:

Case 1: $V \notin \mathbf{An}(T)$. V can be easily made a descendant of T by adding a directed edge from T or one of its descendants (lines 8, 12-16). We suggest the option that minimizes the directed distance from T (lines 9-10) while not being *BANNED* (line 11).

Algorithm 4 Suggest causal graph edits that would include a variable in the adjustment set.

```
1: function MapAddition($\mathcal{G}$, D, $T$, $O$, F, B, $V$)
2:     S ← $\emptyset$
3:     if $V \in \mathbf{De}(T) \vee V \in \mathbf{De}(Y)$ then
4:         B, $success$ ← BreakPaths($\mathcal{G}$, F, $\{T, Y\}$, $V$)
5:         if not $success$ then
6:             return $\emptyset$
7:         S ← B
8:     S ← S $\cup$ {($V \rightarrow T$, Add), ($V \rightarrow Y$, Add)}
9:     return S
```

Algorithm 5 Break each directed path from one of **sources** to *sink*.

```
1: function BreakPaths($\mathcal{G}$, F, sources, $sink$)
2:     reachable_from_sources ← BFS($\mathcal{G}$, from=sources)
3:     sink_reachable_from ← BFS(ReverseGraph($\mathcal{G}$), from=$sink$)
4:     nodes_to_keep ← reachable_from_sources $\cap$ sink_reachable_from
5:     $\mathcal{G}_{filtered}$ ← RemoveNodes($\mathcal{G}$, Nodes($\mathcal{G}$) - nodes_to_keep)
6:     B ← $\emptyset$
7:     $queue$ ← $sources$
8:     $visited$ ← $sources$
9:     while $queue \neq \emptyset$ do
10:        $V$ ← Dequeue($queue$)
11:        if $V = sink$ then
12:            return $\emptyset$, False
13:        for $W \in$ Children($\mathcal{G}_{filtered}$, $V$) do
14:            if $(V \rightarrow W) \notin$ F then
15:                B ← B $\cup$ $\{V \rightarrow W$, Remove$\}$
16:            else if $W \notin visited$ then
17:                $visited$ ← $visited \cup \{V\}$
18:                Enqueue($queue$, $W$)
19:    return B, True
```

Case 2: $V \in \mathbf{An}(T)$. In this case, making V a descendant of T would create a cycle. To fix this, we must first remove V from $\mathbf{An}(T)$ by calling BreakPaths to find edits breaking all directed paths from V to T (lines 3-4, 7). If that is impossible, the second return value of BreakPaths is False prompting MapRemoval to return, since we cannot include V in $\mathbf{De}(T)$ without creating a cycle (lines 5-6).

5.2 The MapAddition Algorithm

MapAddition (Algorithm 4) finds a set of edge edits that ensures that V is included in the adjustment set. It achieves this by creating a new path $T \leftarrow V \rightarrow O$, which must be blocked in order to satisfy the second condition of Definition 6 and can only be blocked by including V in the adjustment set. There are, again, two cases:

Case 1. $V \notin (\mathbf{De}(T) \cup \mathbf{De}(Y))$. We can directly create the path described above by adding edges $V \rightarrow T$ and $V \rightarrow Y$ (lines 8-9).

Case 2. $V \in (\mathbf{De}(T) \cup \mathbf{De}(Y))$. We first call BreakPaths to break these descendant relationships, if possible (lines 3-7).

5.3 The BreakPaths Algorithm

Both MapRemoval and MapAddition may require identifying a set of graph edits that breaks each directed path in a collection of directed paths. In principle, this can be achieved by finding a minimum cut in the subgraph consisting only of these paths, achievable in time $O(|\mathbf{V}|^2|\mathbf{E}|)$ using Dinitz's algorithm [23]. However, since we do not need the cut to be minimal, we present an alternative, more efficient algorithm, shown in Algorithm 5.

In particular, we start by identifying the sub-graph $\mathcal{G}_{filtered}$, consisting only of directed paths from any of the nodes in **sources** to *sink*. We detect all the nodes that both are reachable from the **sources** and from which *sink* is reachable, and remove all other nodes (and their incident edges) from $\mathcal{G}$ (lines 2-5). We then apply a variant of breadth-first search: we initialize a queue and visited list based on **sources** (lines 7-8) and start proecssing elements from the queue until it is empty (lines 9-10). For each processed element, we attempt to remove all directed edges to its children, if they are not *FIXED* (lines 6, 13-15). If some edge among them is indeed fixed, we defer the breaking of that path by adding its children to the queue (lines 16-18). If we reach the sink node, it means that some directed path only contained *FIXED* edges, so breaking it is infeasible; we therefore return an empty set of edits and a boolean False value (lines 11-12). Otherwise, once the queue is empty, we return the identified edits and a True value (line 19).

5.4 Computational Complexity

When not using the cache, SolicitJudgment loops over $O(|\mathbf{V}|)$ variables, calling MapRemoval or MapAddition in each iteration. Each of those algorithms calls BreakPaths, which involves three breadth-first searches (lines 2, 3 and 7-18) of complexity $O(|\mathbf{V}| + |\mathbf{E}|) = O(|\mathbf{V}|^2)$. MapRemoval also includes a sort by distance, also implementable using a breadth-first search in $O(|\mathbf{V}|^2)$, and a loop of complexity $O(|\mathbf{V}|)$. Each of MapRemoval and MapAddition therefore have complexity $O(|\mathbf{V}|^2)$. SolicitJudgment then calls ATE, which involves a linear regression with $O(|\mathrm{D}||\mathbf{V}|^2 + |\mathbf{V}|^3)$. The full complexity of SolicitJudgment is then $O(|\mathrm{D}||\mathbf{V}|^3 + |\mathbf{V}|^4)$.

6 LOG CONVERTER

The two human-in-the-loop modules operate on a tabular dataset D. We will now describe a pipeline to transform a log into such a dataset efficiently. This dataset also serves as the second input to Pearl's model for ATE calculation (alongside the causal graph).

6.1 Log Parsing

Text logs must first be *parsed* into a table of variable observations:

Definition 8 (Parsed Table). *The parsed table includes a row per log message and a column per log-derived parsed variable.*

To generate the parsed table, we go through five steps:

(1) Users can specify a log message prefix (e.g., a timestamp regular expression), to collate multi-line log messages.
(2) Users can pass zero or more regular expressions to parse formatted variables (e.g IDs) from each log message.
(3) We use an off-the-shelf parsing algorithm [36, 114, 120] to extract the rest of the parsed variables from their *log templates*. Any "classical" [25, 37, 38, 41] or learning-based [7, 18, 22, 55, 62, 65, 95, 105] log parsing algorithm is acceptable, as long as it leads to a parsed table that conforms with Definition 8. Occasionally, log parsing algorithms may map semantically different variables to the same parsed variable. For such cases, our implementation provides functionality for the user to separate the variables involved.
(4) We also add a binary indicator variable per log template, to count the rate of incidence of each template.
(5) Some parsed variables are uninteresting for causal inference - e.g. identifiers. We discard any categorical parsed variable where the number of distinct values exceeds 15% of the variable's occurrences, similar to past work [107].

6.2 Variable Tagging

Working with variables to perform causal inference requires *naming* the variables in a human-understandable way, so that users can reason about them. We will call these names *tags*. For variables parsed using a regular expression, the user provides a tag together with the regular expression. For each of the remaining parsed variables, we consult three sources to generate a tag:

(1) The three tokens preceding the variable in the log template, if the variable is preceded by ':' or '=' (possibly with an interspersed opening quote).
(2) GPT-3.5-Turbo [72], providing (a) an example message where the variable appears, (b) the 3 tokens that precede the variable (c) 4 additional example values for the variable from other log messages of the same template. While our prompt (available online [58]) delivers acceptable results, we acknowledge that changing the prompting technique and/or model can affect output quality [48, 49, 53]. Fully exploring this problem is not a focus of our work.
(3) GPT-4 [70], using the same prompt as above.

As soon as one source produces a tag, we move on to the next parsed variable. If no source produces a tag, we assign a unique string of symbols. Users can later edit these tags manually.

Sometimes, distinct parsed variables may be assigned the same tag - e.g., several may be called "port", from different contexts. We use the tag only for the first parsed variable to obtain it and use a unique system-generated variable name for the rest.

6.3 Causal Unit Definition

The rows of the parsed table, one per log message, are not ideal for causal inference. We instead need a row per "data point" as defined for the question at hand (e.g. per session). We achieve this by bundling groups of log messages together into *causal units*.

Since the right choice of causal unit inherently depends on the question the user is interested in, we defer to them for a causal unit definition. Such a definition consists of a parsed variable W, on the value of which the units will be based, and optionally a discretization function (e.g., a binning function) if W is continuous.

However, not *every* choice of causal unit is permissible, because Definition 1 requires the Stable Unit Treatment Value Assumption (SUTVA) [79]: the outcome of each unit should not depend on the treatments of *other units*. For example, processes may be unsuitable causal units if they share hardware: process A's work could impact process B's latency. We defer to the user's knowledge of the system to ensure that this condition is satisfied.

6.4 Aggregation

There can be a varying number of values for each parsed variable in each causal unit. For example, if Alex defines a causal unit per session, the number of query latency readings for each session vary. To make causal units comparable, we must replace varying-size sets of values with *the same value(s) per causal unit*:

Definition 9 (Prepared Table). *The prepared table includes one row per causal unit and one column per prepared variable.*

Each prepared variable is derived from some parsed variable, its *base variable*, by using an aggregation function (e.g. the mean) to reconcile the values within each causal unit. If the user is confident that a certain aggregation function is best for a particular parsed variable, they can explicitly specify it. Otherwise, one cannot find the most useful aggregation function for each parsed variable a priori, since the downstream ATE question is not yet known. Even if it were, optimizing for it could lead to overfitting, generating variables that would be less informative for adjacent questions the user may later pose. We therefore approximate this objective by trying to maximize the information captured by each prepared variable. We achieve this in two steps.

First, we compute multiple aggregates of each parsed variable $W \in \mathbf{W}$, based on its type. For numerical variables, we consider $\mathbf{A}_W = [max, min, mean]$. For categorical variables, we consider $\mathbf{A}_W = [first, last, mode]$, where $first(W)$ and $last(W)$ return the value of W that appeared earliest or latest within each causal unit, respectively. When calculating these functions, we ignore all missing values in their inputs.

Then, for a parsed variable W, we let $R(W, a)$ be the range of $a(W)$, $a \in \mathbf{A}_W$. Among the prepared variables with W as their base variable, we keep the one that maximizes empirical entropy; that is, we keep the one resulting from $a^*_W(W)$, $a^*_W \in \mathbf{A}_W$, where:

$$a^*_W = \arg\max_{a \in \mathbf{A}_W} \sum_{x \in R(W,a)} -p(x) log_2(p(x))$$

We finally one-hot encode any categorical variables in this set.

6.5 Imputation

In each causal unit, there may be parsed variables with no observations. Ignoring causal units with missing values can result in *selection bias*. Luckily, imputing missing values in the prepared table is often possible, since the missing values are interpretable. We can impute a default based on domain knowledge, which is easy to tap now that log information is organized along causal units. Since the default may differ per prepared variable, we let the user specify it if they so wish, performing no imputation by default.

6.6 Computational Complexity

Log Parsing Depends on the chosen existing parsing algorithm.

Variable Tagging For tagging, we require $O(M)$ time and $O(|\mathbf{W}|)$ space for the metadata fed into the GPT prompts, where M is the number of log messages and $\mathbf{W}$ is the set of parsed variables. Depending on the choice of log parsing algorithm, this information can also be collected during parsing. We then require an additional $O(|\mathbf{W}|)$ time and up to $2|\mathbf{W}|$ GPT calls to compute the tags, which take $O(|\mathbf{W}|)$ space to store. Since tagging is a per-variable operation, linear complexity is optimal for this problem.

Causal Unit Definition Defining causal units takes $O(1)$ time.

Aggregation Aggregation takes $O(|\mathbf{C}||\mathbf{W}|T_{max})$ time, where $\mathbf{C}$ is the set of causal units, $\mathbf{W}$ is the set of parsed variables and T_{max} is the dominant time complexity among the aggregation functions. Among the defaults, *mode* dominates with $O(|C|log|C|)$ for a causal unit with $|C|$ elements. The time complexity of one-hot encoding is $O(|\mathbf{C}||\mathbf{W}_{cat}|k_{max})$, where $\mathbf{W}_{cat}$ is the set of categorical parsed variables and k_{max} is the maximum number of categories.

Imputation Imputation with fixed values takes $O((|\mathbf{C}||\mathbf{W}|)$ time.

7 PROTOTYPE SYSTEM: LOGOS

We implemented our framework in LOGos, using ~2850 lines of Python. Our code, including LLM prompts, is available online [58]. An earlier prototype was presented as Sawmill [59], but we revised the name because of a name clash with a different product [42]. For log parsing, we used Drain [38], based on an implementation by the authors of LogBERT [31]. Drain uses a fixed-depth parse tree in an online streaming manner that requires a single pass over the log. For variable pruning, we used LASSO from scikit-learn [76]. For estimating ATEs in GetATE we used "backdoor.linear_regression" from DoWhy [86], an open-source Python library for causality.

8 EVALUATION

After presenting our experimental setting (Section 8.1), we evaluate three claims about our framework, as implemented in LOGos, and the performance of the **Log Converter**:

- LOGos's **Candidate Cause Ranker** ranks the ground truth root causes higher than the baselines while remaining interactive, addressing Problem 1 (Section 8.2).
- LOGos's **Interactive Causal Graph Refiner** quickly leads the user to a graph on which $ATE(T, O)$ matches the ground truth, addressing Problem 2 (Section 8.3).
- The **Log Converter** scales gracefully (Section 8.4).

8.1 Experimental Setting

8.1.1 Environment. All experiments were run on a machine equipped with two 20-core 2.10 GHz Intel Xeon Gold 6230 CPUs [43] and 256 GiB of memory, running Linux 6.9.7-arch1-1.

8.1.2 Metrics. Our evaluation focuses on the following metrics:

Average Precision (Candidate Cause Ranker, higher is better) Since this module addresses Problem 1, we evaluate it using the average precision of its output ranking r, defined as:

$$AP(r) = \frac{\sum_{i=1}^{\textsc{Length}(r)} \textsc{PrecisionAmongTop}(i) \cdot \textsc{IsPositive}(i)}{\text{\# Ground Truth Positive Elements}}$$

Number of Judgments (Interactive Causal Graph Refiner, lower is better) Leveraging the Absolute Relative Error [2] in ATE, we report the number n of judgments to reach $\mathcal{G}_n$ for which:

$$ARE\_ATE(n) = \left| \frac{ATE_{\mathcal{G}_n}(T, O) - ATE_{\mathcal{G}_{\text{Real}}}(T, O)}{ATE_{\mathcal{G}_{\text{Real}}}(T, O)} \right| \leq 10^{-5}$$

We selected this metric instead of measuring the structural accuracy of the causal graph because a fully correct graph is not necessary to estimate the ATE correctly, as explained in Section 1.

Latency (lower is better) We evaluate whether the latency of our two human-in-the-loop modules is indeed interactive, as well as how the **Log Converter** scales to more complex logs.

8.1.3 Baselines. We compare LOGos to two baselines:

Regression Produce suggestions using linear regressions.

- **For Candidate Cause Ranking:** Remove prepared variables with zero variance and normalize the rest to zero mean and unit variance. Regress the effect variable E on the normalized data and rank the regressors by decreasing (normalized) absolute regression coefficient.

Table 2: Statistics about our evaluation datasets.

Dataset		Size		Parsed Variables	Prepared Variables
		Lines	Bytes		
PostgreSQL		507,648	20,587,286	71	172
Proprietary		1,223,000	237,048,470	121	120
XYZ	V=10	1,000,000	62,458,855	12	21
	V=100	1,000,000	64,791,158	102	201
	V=1000	1,000,000	65,943,179	1002	2001

- **For Interactive Causal Graph Refinement:** For each variable V in the current causal graph, for each of its non-neighbors W, regress V on W. Solicit a user judgment for the pair with the highest absolute regression coefficient. Before running the regressions, normalize as above. Cache regression results and continue soliciting judgments in ranked order while the user makes no changes to the graph.

LangModel Produce suggestions using a large language model, GPT-4o mini (`gpt-4o-mini-2024-07-18`) [71].

- **For Candidate Cause Ranking:** Prompt the model with the tags of all prepared variables and up to 3 unique example values for each, and have it rank causes for E.
- **For Interactive Causal Graph Refinement:** Prompt the model with the tags of all prepared variables, the edges in the current causal graph, and the pair (T, O). Ask it to rank pairs of variables based on their relevance to $ATE(T, O)$. Solicit a user judgment for the pair ranked first and cache the rest. Continue soliciting judgments in ranked order as long as the user makes no changes to the graph.

8.1.4 Datasets. Open-source collections of logs exist [119], but to evaluate LOGos we also need access to the *ground-truth causal effects* in the log-producing system, which are generally not available. We therefore develop three new datasets, summarized in Table 2, ranging from instrumenting a real system, to augmenting a real-world log with a known causal relationship, to generating synthetic log data for stress-testing. This strategy was also used by other causality-in-systems works, e.g., CausalSim [4]. Where applicable, we provide these datasets with our code [58]. The last two columns of Table 2 refer to the variables produced at different stages by the **Log Converter**. The dataset we use as input for the human-in-the-loop modules consists of the prepared variables.

PostgreSQL (Real-world): As mentioned in Section 2.2, this dataset emulates the situation of a database administrator who wants to determine which parameter is causing queries to be slow. We set up PostgreSQL 14 on an t2.2xlarge instance on AWS EC2 [85] and configured it to log the latency of each query. We then loaded the TPC-DS benchmark [77] for scale factor 1 and executed a workload consisting of sequentially issuing one query per TPC-DS template, excluding 4 long-running templates (1, 4, 11 and 74).

We ran this workload 64 times, once per combination of the parameter settings in Table 3, each in a separate session. These parameters both impact query latency and do not require a Postgres service restart to reset. For each of the 32 parameter combinations where the product of `work_mem` and `max_parallel_workers` is 256, we then ran the workload once more, leading to 96 total runs. This led to a dataset with confounding: `work_mem` affects both query latency and the value of `max_parallel_workers`.

Table 3: Parameter settings for PostgreSQL.

Parameter	Values	Parameter	Values
`work_mem`	128, 256	`max_parallel_workers`	1,2
`seq_page_cost`	1, 1000	`maintenance_work_mem`	32768, 65536
`random_page_cost`	4, 1000	`effective_cache_size`	262144, 524288

Proprietary (Real-world with post-processing): We wanted to process a log from another complicated piece of software that was as realistic as possible, but for which we knew the ground truth. We thus post-processed a real-world log for an HTTP-based client/server application from a large company, with different messages in the log generated by different parts of the software stack. Clients in this application periodically contact the server to perform operations. We kept the real-world syntax of this file but synthetically introduced a bug. We generated logs for 1,000 "users", to whom we assigned unique IDs, included in each of their log messages. We designated a fraction of the users F as "faulty". For each non-faulty user, we generated a log identical to the original, except that HTTP responses have probability $p_n = 10\%$ of reporting error code 401 instead of success code 200. There are 36 such messages in the original 1223-line log. For each faulty user, we equivalently defined a probability $p_f > p_n$ and we indicated their OS as iOS 15.0 in the log message that reports it, compared to iOS 14.3 in the original log. We generated 9 experimental scenarios by setting F to 50%, 10% or 1%, while p_f is set to 100%, 50% or 20%.

XYZ (Synthetic): Finally, we created a family of synthetic logs to evaluate the limits of our framework's ability to recover and adjust for confounding. We wanted a synthetic log so we could test how well LOGos can work even in the face of a harder-to-discern causal effect. We did that by carefully adjusting the level of noise and the number of irrelevant variables. Each XYZ log contains 1000 log messages for each of 1000 machines. Each message includes an artificial timestamp, a machine identifier and one of V variables chosen uniformly at random. We ensure every variable is reported at least once. While the value of most variables is drawn uniformly at random between 0 and 100 each time the variable is reported, the values of 3 special variables (x, y and z) are not:

- z is drawn uniformly between 0 and 10 once per machine.
- x is equal to $x_m + z$, with added Gaussian noise with $\sigma = R$. x_m is drawn once per machine, uniformly at random.
- y is equal to $2x + 3z$, with added Gaussian noise with $\sigma = R$.

As such, ATE(x,y)=2 after adjusting for z. We generated 9 scenarios by setting V to 10, 100 or 1000, and R to 1, 5 or 10.

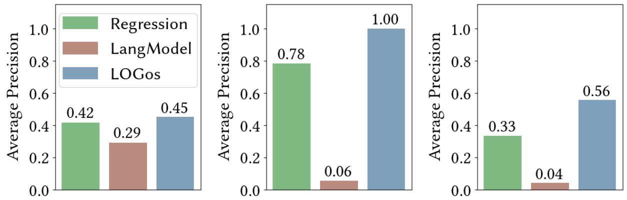
8.2 Candidate Cause Ranking

Candidate Cause Ranking: Claim

LOGos's **Candidate Cause Ranker** ranks the ground truth root causes higher than the baselines while remaining interactive.

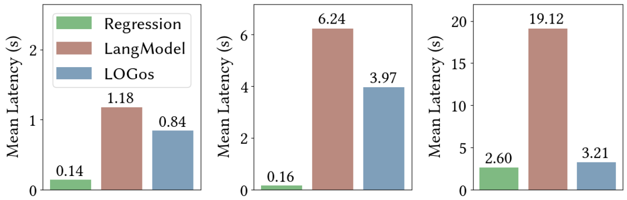
We begin our evaluation by assessing the ability of the **Candidate Cause Ranker** to produce high-quality rankings of candidate causes, therefore addressing Problem 1. Figure 3 presents the average precision results, while Figure 4 the mean latency results.

8.2.1 PostgreSQL. For this dataset, we inquired for candidate causes of query latency, represented by the prepared variable `duration mean`, and we aimed to recover the prepared variables associated with the two parameters involved in confounding: `work_mem mean` and `max_parallel_workers mean`. As shown in Figure 3a, the candidate cause ranking produced by LOGos achieved superior average precision in this task, beating Regression by 1.08× and LangModel by 1.54×. From Figure 4a we see that Regression offered a much lower latency, but the sub-second latency offered by LOGos was still firmly within the interactive range we are targeting.



(a) PostgreSQL (b) Proprietary (c) XYZ

Figure 3: Average precision for Candidate Cause Ranking (higher is better).



(a) PostgreSQL (b) Proprietary (c) XYZ

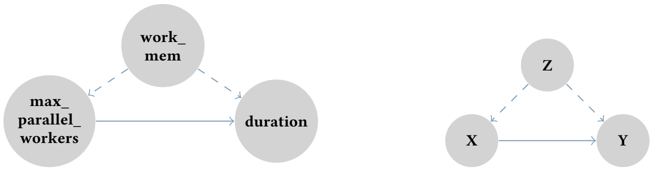
Figure 4: Candidate Cause Ranking latency (lower is better).

8.2.2 Proprietary. For this dataset, we inquired for candidate causes of failure codes in response to HTTP requests, represented by the prepared variable `code mean`, and we aimed to recover the prepared variable associated with the OS version, tagged `version mean`. We present mean results over the 9 experimental scenarios of this dataset. As shown in Figure 3b, LOGos outperformed Regression by 1.28× and LangModel by 18× in terms of average precision. The latency insight from Figure 4b is again that Regression offered very low latency, but LOGos remained interactive.

8.2.3 XYZ. Finally, for XYZ, we inquired for candidate causes of `y mean`, and we aimed to recover `x mean` as the candidate cause. We present mean results over the 9 experimental scenarios of this dataset. As shown in Figure 3c, LOGos once again led in average precision, beating Regression by 1.66× and LangModel by 13.47×. In terms of latency, Figure 4c once more shows that Regression produced results the fastest, but LOGos remained interactive.

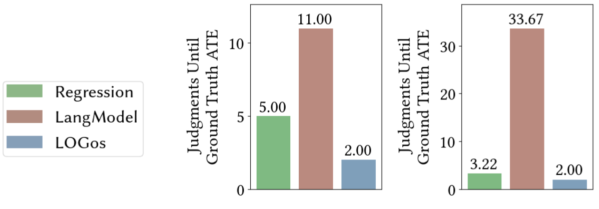
Candidate Cause Ranking: Summary

The average precision of the **Candidate Cause Ranker** was 1.08×-1.66× higher than Regression and 1.54×-18× higher than LangModel. A call to the **Candidate Cause Ranker** required on average under 4s, which remained interactive, although slower than Regression. LangModel was at least 40% slower than LOGos.



(a) PostgreSQL (b) XYZ

Figure 5: Ground truth causal graphs for Section 8.3. The corresponding starting graphs lack the dashed edges.



(a) PostgreSQL (b) XYZ

Figure 6: Number of judgments required to converge to ground truth ATE for Interactive Causal Graph Refinement (lower is better).

8.3 Interactive Causal Graph Refinement

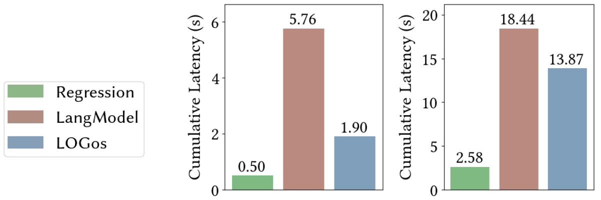
Interactive Causal Graph Refinement: Claim

LOGos's **Interactive Causal Graph Refiner** quickly leads the user to a graph on which $ATE(T, O)$ matches the ground truth.

8.3.1 PostgreSQL. We calculated the ground truth value of the ATE(`max_parallel_workers`, `duration`) on the graph shown in Figure 5a (including the dashed edges). Then, we removed the dashed edges to derive a starting graph. For each available method, we responded to each judgment it solicited with the appropriate edge status based on the ground truth causal graph, and measured how many judgments were solicited before the ATE converged to its ground truth value. As seen in Figure 6a, LOGos was able to immediately zero in on the two dashed edges and solicit judgments for them, reaching the ground truth graph (and thus the ground truth ATE) in only 2 judgments. Regression instead required 5 judgments (2.50× more) to achieve the same goal, while LangModel required 11 judgments (5.50× more).

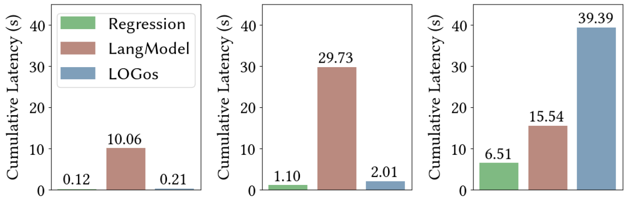
Moreover, as shown in Figure 7a, LOGos required only 1.90 s of total processing time for the 2 judgment solicitations, firmly within the interactive realm. While Regression was also interactive, LangModel required 5.76 s in total, 3.04× more than LOGos.

8.3.2 XYZ. We calculated the ground truth ATE(`X`, `Y`) on the graph of Figure 5b (incl. the dashed edges). Then, we removed the dashed edges to derive a starting graph and followed the same strategy as in Section 8.3.1. We present mean results over the 9 experimental scenarios of this dataset. As seen in Figure 6b, LOGos again outperformed the competition, requiring a perfect 2 judgment solicitations to identify the 2 dashed edges and help the user reconstruct the ground truth graph. Regression instead required 3.22 judgments on average (1.61× more) to achieve the same goal, while LangModel required 33.67 judgments (16.83× more).



(a) PostgreSQL (b) XYZ

Figure 7: Cumulative latency for Interactive Causal Graph Refinement (lower is better).



(a) $V = 10$ (b) $V = 100$ (c) $V = 1000$

Figure 8: Figure 7b split by value of V (lower is better).

As shown in Figure 7b, LOGos required on average 13.87 seconds of total processing time across the 2 judgment solicitations. However, as the breakdown by value of V in Figure 8 shows, this was mainly caused by the deliberately computationally intensive experimental scenarios with $V = 1000$, which contain over 10× more prepared variables compared to PostgreSQL and Proprietary (see Table 2). LOGos offered interactive latency otherwise. While Regression was interactive across experimental settings, LangModel required 18.44 seconds in total on average, 1.33× more than LOGos.

Interactive Causal Graph Refinement: Summary

The **Interactive Causal Graph Refiner** required 1.61×-2.50× fewer judgments than Regression and 5.50×-16.83× fewer than LangModel. A call to the **Interactive Causal Graph Refiner** was interactive for most datasets, although slower than Regression. LangModel was at least 33% slower than LOGos.

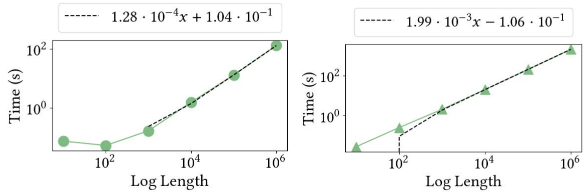
8.4 Log Converter Scaling

Log Converter Scaling: Claim

The **Log Converter** scales gracefully to more complex logs.

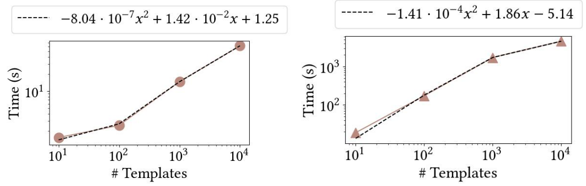
We close with an evaluation of the **Log Converter** as implemented in our prototype of LOGos. In particular, we evaluated the scalability of Log Parsing (Section 6.1) and Aggregation (Section 6.4), which present the greatest algorithmic interest. Recall that for Parsing we heavily rely on Drain [38], as explained in Section 7.

We used synthetic datasets defined based on 4 parameters: L, S, V and C. L represented the length of the log. Each log message included $2 + V + C$ space-separated tokens. The first token was a line ID of the form `line_`i, where i was the index of the message. The second token was a string token of the form `s_`j, where j was a uniformly random integer between 1 and S, inclusive. The next V tokens were numerical variables, each of which took the value i. The last C tokens were constants equal to 0. To sweep log lengths,



(a) Log Parsing **(b) Aggregation**

Figure 9: Log Converter scaling by log length.



(a) Log Parsing **(b) Aggregation**

Figure 10: Log Converter scaling by number of templates.

we fixed $S = 10$, $V = 1$ and $C = 10$, and swept L over powers of 10 from 10^1 to 10^6. To sweep the number of templates, we fixed $L = 10^4$, $V = 1$ and $C = 10$, and swept S over powers of 10 from 10^1 to 10^4. To sweep the fraction of variables, we fixed $L = 10^4$, $S = 10$ and $C = 100 - V$, and swept V over multiples of 10 from 10 to 100.

We parsed each log using a regular expression for `line_ID` and set the Drain similarity threshold so one log template per value of the string token in each experiment. As Figures 9a, 10a and 11a show, the time taken by log parsing scaled linearly with each of the swept measures of the complexity of a log. We then aggregated each log into causal units determined by `line_ID`. Figures 9b, 10b and 11b show the results, which again scale linearly.

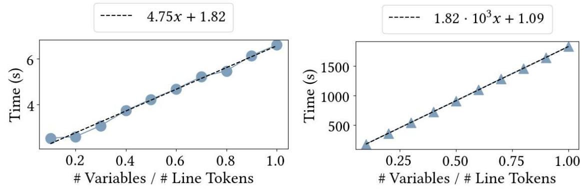
Log Converter Scaling: Summary

Log Parsing and Aggregation in the **Log Converter** scaled linearly with three measures of the complexity of a log: length, number of templates, and fraction of tokens that are variables.

9 RELATED WORK

9.1 Causality and Systems

Managing failures is crucial for large system operators. Past work has used formulations including failure classification [6], failing component detection [123], troubleshooting guide suggestion [45], job runtime impact analysis [122] and risk simulation [104]. Some existing work has also leveraged causality [1, 4, 28, 30, 33, 44, 66, 67]. Causal inference has also been used for other data management tasks more broadly [3, 27, 57, 83, 88, 108, 118]. However, logs have been under-explored as a data source, with most past work simply focusing on outlier detection [26, 31, 82]. One work does map outliers to their causes [12], but it requires a fully externally provided causal graph. The outlier detection framing also fails to flag longer-term problems (e.g. some users being consistently slower). Other works impose additional restrictions, like source code access [111], platform-specificity [117] or fine-grained hardware-supported logging [17]. On the commercial side, platforms like Splunk [94] and Datadog [21] provide log access interfaces, but no tailored support for causal inference. Datadog can identify some "root causes" for failures [20], but these verdicts appear driven just by temporal relationships, similar to Falcon [66] and Horus [67].



(a) Log Parsing **(b) Aggregation**

Figure 11: Log Converter scaling by fraction of variables.

9.2 Causal Discovery

Causal discovery has seen much prior research [29, 89, 93, 102, 113, 121]. Some existing algorithms are "constraint-based", like PC [92], FCI [93], GES [93] and variants of each. Others, like LiNGAM [89], estimate the parameters of Functional Causal Models (FCMs). Each algorithm makes assumptions about the data [29], some of which are false for logs - e.g. functional dependencies yield singular correlation matrices. A text-mining approach based on the variable *names* may also be possible if such names are informative enough [34, 35, 39], most recently by leveraging large language models (LLMs) [5, 8, 9, 50, 54, 56, 64, 98, 101, 109]. Despite often being impressive, LLMs still exhibit unpredictable failure modes that hinder their general adoption for causality [50]. Moreover, LLMs rely on publicly available textual information to derive their answers, which may be severely limited when dealing with log-derived variables particular to a specific architecture and deployment.

10 CONCLUSION AND LIMITATIONS

In this work, we presented a novel human-in-the-loop framework for applying causal inference on log data. This endeavor is challenging because log data suffer from three problems: Functional Dependencies, Large Number of Variables and Biased Data Collection. Our framework combines novel algorithmic approaches with judicious use of human input to mitigate these problems. Our prototype, LOGos, accurately and efficiently calculates Average Treatment Effects among log variables, in both real-world and synthetic logs, while scaling linearly with the complexity of a log. This work can spur further exploration of applying causality in systems.

Although powerful, our approach has two limitations. First, we currently rely on both the *correctness* and the *coverage* of the input log data. Still, our approach is flexible enough to integrate more data sources, like code or documentation, using preprocessing modules analogous to the **Log Converter**. Second, since we include a human in the loop, some user domain knowledge is necessary to derive a high-quality causal model from our framework's suggestions. We believe the required level of expertise is appropriate for our intended audience (engineers administering large systems).

ACKNOWLEDGMENTS

We are grateful for support from the DARPA ASKEM project (Award HR00112220042), the ARPA-H Biomedical Data Fabric project, and a grant from Liberty Mutual.

REFERENCES

[1] Pooja Aggarwal, Ajay Gupta, Prateeti Mohapatra, Seema Nagar, Atri Mandal, Qing Wang, and Amit Paradkar. 2021. Localization of Operational Faults in Cloud Applications by Mining Causal Dependencies in Logs Using Golden Signals. In *Service-Oriented Computing – ICSOC 2020 Workshops*, Hakim Hacid, Fatma Outay, Hye-young Paik, Amira Alloum, Marinella Petrocchi, Mohamed Reda Bouadjenek, Amin Beheshti, Xumin Liu, and Abderrahmane Maaradji (Eds.). Springer International Publishing, Cham, 137–149.

[2] OECD AI. 2024. Absolute Relative Error (ARE). Retrieved May 29, 2024 from https://oecd.ai/en/catalogue/metrics/absolute-relative-error-%28are%29

[3] Kamran Alipour, Aditya Lahiri, Ehsan Adeli, Babak Salimi, and Michael Pazzani. 2022. Explaining Image Classifiers Using Contrastive Counterfactuals in Generative Latent Spaces. arXiv:2206.05257 [cs.CV]

[4] Abdullah Alomar, Pouya Hamadanian, Arash Nasr-Esfahany, Anish Agarwal, Mohammad Alizadeh, and Devavrat Shah. 2023. CausalSim: A Causal Framework for Unbiased Trace-Driven Simulation. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 1115–1147. https://www.usenix.org/conference/nsdi23/presentation/alomar

[5] Alessandro Antonucci, Gregorio Piqué, and Marco Zaffalon. 2023. Zero-shot Causal Graph Extrapolation from Text via LLMs. arXiv:2312.14670 [cs.AI] https://arxiv.org/abs/2312.14670

[6] Behnaz Arzani, Selim Ciraci, Boon Thau Loo, Assaf Schuster, and Geoff Outhred. 2016. Taking the Blame Game out of Data Centers Operations with NetPoirot. In *Proceedings of the 2016 ACM SIGCOMM Conference* (Florianopolis, Brazil) *(SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 440–453. https://doi.org/10.1145/2934872.2934884

[7] Nicolas Aussel, Yohan Petetin, and Sophie Chabridon. 2018. Improving Performances of Log Mining for Anomaly Prediction Through NLP-Based Log Parsing. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, Piscataway, NJ, USA, 237–243. https://doi.org/10.1109/MASCOTS.2018.00031

[8] Taiyu Ban, Lyuzhou Chen, Derui Lyu, Xiangyu Wang, and Huanhuan Chen. 2023. Causal Structure Learning Supervised by Large Language Model. arXiv:2311.11689 [cs.AI] https://arxiv.org/abs/2311.11689

[9] Taiyu Ban, Lyvzhou Chen, Xiangyu Wang, and Huanhuan Chen. 2023. From Query Tools to Causal Architects: Harnessing Large Language Models for Advanced Causal Discovery from Data. arXiv:2306.16902 [cs.AI] https://arxiv.org/abs/2306.16902

[10] Amir Beck and Marc Teboulle. 2009. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. *SIAM Journal on Imaging Sciences* 2, 1 (2009), 183–202. https://doi.org/10.1137/080716542 arXiv:https://doi.org/10.1137/080716542

[11] Matthew Blackwell. 2013. A framework for dynamic causal inference in political science. *American Journal of Political Science* 57, 2 (2013), 504–520.

[12] Kailash Budhathoki, Lenon Minorics, Patrick Bloebaum, and Dominik Janzing. 2022. Causal structure-based root cause analysis of outliers. In *Proceedings of the 39th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (Eds.), Vol. 162. PMLR, Cambridge, MA, USA, 2357–2369. https://proceedings.mlr.press/v162/budhathoki22a.html

[13] Cristiano Calcagno, Dino Distefano, Jeremy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter O'Hearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez. 2015. Moving Fast with Software Verification. In *NASA Formal Methods*, Klaus Havelund, Gerard Holzmann, and Rajeev Joshi (Eds.). Springer International Publishing, Cham, 3–11.

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

[15] Tom Claassen, Joris Mooij, and Tom Heskes. 2013. Learning Sparse Causal Models is not NP-hard. arXiv:1309.6824 [cs.AI] https://arxiv.org/abs/1309.6824

[16] José M Cordero, Víctor Cristóbal, and Daniel Santín. 2018. Causal inference on education policies: A survey of empirical studies using PISA, TIMSS and PIRLS. *Journal of Economic Surveys* 32, 3 (2018), 878–915.

[17] Weidong Cui, Xinyang Ge, Baris Kasikci, Ben Niu, Upamanyu Sharma, Ruoyu Wang, and Insu Yun. 2018. REPT: Reverse Debugging of Failures in Deployed Software. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 17–32. https://www.usenix.org/conference/osdi18/presentation/weidong

[18] Hetong Dai, Heng Li, Che-Shao Chen, Weiyi Shang, and Tse-Hsun Chen. 2020. Logram: Efficient Log Parsing Using n n-Gram Dictionaries. *IEEE Transactions on Software Engineering* 48, 3 (2020), 879–892.

[19] Ermira Daka and Gordon Fraser. 2014. A Survey on Unit Testing Practices and Problems. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*. IEEE, Piscataway, NJ, USA, 201–211. https://doi.org/10.1109/ISSRE.2014.11

[20] Datadog. 2022. Automated root cause analysis with Watchdog RCA. https://www.datadoghq.com/blog/datadog-watchdog-automated-root-cause-analysis/

[21] Datadog. 2024. https://www.datadoghq.com/ [Accessed 31-Jul-2024].

[22] Biplob Debnath, Mohiuddin Solaimani, Muhammad Ali Gulzar Gulzar, Nipun Arora, Cristian Lumezanu, JianWu Xu, Bo Zong, Hui Zhang, Guofei Jiang, and Latifur Khan. 2018. LogLens: A Real-Time Log Analysis System. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, Piscataway, NJ, USA, 1052–1062. https://doi.org/10.1109/ICDCS.2018.00105

[23] Efim A Dinic. 1970. Algorithm for solution of a problem of maximum flow in networks with power estimation. In *Soviet Math. Doklady*, Vol. 11. American Mathematical Society, Providence, RI, USA, 1277–1280.

[24] Vijay D'silva, Daniel Kroening, and Georg Weissenbacher. 2008. A survey of automated techniques for formal software verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 7 (2008), 1165–1178.

[25] Min Du and Feifei Li. 2016. Spell: Streaming Parsing of System Event Logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, Piscataway, NJ, USA, 859–864. https://doi.org/10.1109/ICDM.2016.0103

[26] Ilenia Fronza, Alberto Sillitti, Giancarlo Succi, Mikko Terho, and Jelena Vlasenko. 2013. Failure prediction based on log files using random indexing and support vector machines. *Journal of Systems and Software* 86, 1 (2013), 2–11.

[27] Sainyam Galhotra, Amir Gilad, Sudeepa Roy, and Babak Salimi. 2022. HypeR: Hypothetical Reasoning With What-If and How-To Queries Using a Probabilistic Causal Approach. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1598–1611. https://doi.org/10.1145/3514221.3526149

[28] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. 2021. 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* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 135–151. https://doi.org/10.1145/3445814.3446700

[29] Clark Glymour, Kun Zhang, and Peter Spirtes. 2019. Review of causal discovery methods based on graphical models. *Frontiers in genetics* 10 (2019), 524.

[30] Helga Gudmundsdottir, Babak Salimi, Magdalena Balazinska, Dan R.K. Ports, and Dan Suciu. 2017. A Demonstration of Interactive Analysis of Performance Measurements with Viska. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 1707–1710. https://doi.org/10.1145/3035918.3056448

[31] Haixuan Guo, Shuhan Yuan, and Xintao Wu. 2021. LogBERT: Log Anomaly Detection via BERT. In *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, Piscataway, NJ, USA, 1–8. https://doi.org/10.1109/IJCNN52387.2021.9534113

[32] Saurabh Gupta, Tirthak Patel, Christian Engelmann, and Devesh Tiwari. 2017. Failures in large scale systems: long-term measurement, analysis, and implications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '17)*. Association for Computing Machinery, New York, NY, USA, Article 44, 12 pages. https://doi.org/10.1145/3126908.3126937

[33] Vipul Harsh, Wenxuan Zhou, Sachin Ashok, Radhika Niranjan Mysore, Brighten Godfrey, and Sujata Banerjee. 2023. Murphy: Performance Diagnosis of Distributed Cloud Applications. In *Proceedings of the ACM SIGCOMM 2023 Conference* (New York, NY, USA) *(ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 438–451. https://doi.org/10.1145/3603269.3604877

[34] Chikara Hashimoto. 2019. Weakly Supervised Multilingual Causality Extraction from Wikipedia. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China, 2988–2999. https://doi.org/10.18653/v1/D19-1296

[35] Oktie Hassanzadeh, Debarun Bhattacharjya, Mark Feblowitz, Kavitha Srinivas, Michael Perrone, Shirin Sohrabi, and Michael Katz. 2019. Answering Binary Causal Questions Through Large-Scale Text Mining: An Evaluation Using Cause-Effect Pairs from Human Experts. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, California, USA, 5003–5009. https://doi.org/10.24963/ijcai.2019/695

[36] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R. Lyu. 2016. An Evaluation Study on Log Parsing and Its Use in Log Mining. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, Piscataway, NJ, USA, 654–661. https://doi.org/10.1109/DSN.2016.66

[37] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R Lyu. 2017. Towards automated log parsing for large-scale log data analysis. *IEEE Transactions on Dependable and Secure Computing* 15, 6 (2017), 931–944.

[38] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. 2017. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, Piscataway, NJ, USA, 33–40. https:

//doi.org/10.1109/ICWS.2017.13

[39] Stefan Heindorf, Yan Scholten, Henning Wachsmuth, Axel-Cyrille Ngonga Ngomo, and Martin Potthast. 2020. CauseNet: Towards a Causality Graph Extracted from the Web. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (Virtual Event, Ireland) *(CIKM '20)*. Association for Computing Machinery, New York, NY, USA, 3023–3030. https://doi.org/10.1145/3340531.3412763

[40] Gerard J. Holzmann and Doron Peled. 1995. *An Improvement in Formal Verification.* Springer US, Boston, MA, 197–211. https://doi.org/10.1007/978-0-387-34878-0_13

[41] Shaohan Huang, Yi Liu, Carol Fung, Rong He, Yining Zhao, Hailong Yang, and Zhongzhi Luan. 2020. Paddy: An Event Log Parsing Approach using Dynamic Dictionary. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium.* IEEE, Piscataway, NJ, USA, 1–8. https://doi.org/10.1109/NOMS47738.2020.9110435

[42] Flowerfire Inc. 2024. Sawmill: Universal Log File Analysis and Reporting. Retrieved Aug 1, 2024 from http://www.sawmill.net/

[43] Intel Corporation. 2019. *Intel Xeon Gold 6230 CPU.* Intel Corporation. Retrieved July 31, 2024 from https://ark.intel.com/content/www/us/en/ark/products/192437/intel-xeon-gold-6230-processor-27-5m-cache-2-10-ghz.html

[44] Vimalkumar Jeyakumar, Omid Madani, Ali Parandeh, Ashutosh Kulshreshtha, Weifei Zeng, and Navindra Yadav. 2019. ExplainIt! – A Declarative Root-cause Analysis Engine for Time Series Data. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 333–348. https://doi.org/10.1145/3299869.3314048

[45] Jiajun Jiang, Weihai Lu, Junjie Chen, Qingwei Lin, Pu Zhao, Yu Kang, Hongyu Zhang, Yingfei Xiong, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2020. 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* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 1410–1420. https://doi.org/10.1145/3368089.3417054

[46] James Kapinski, Jyotirmoy V Deshmukh, Xiaoqing Jin, Hisahiro Ito, and Ken Butts. 2016. Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques. *IEEE Control Systems Magazine* 36, 6 (2016), 45–64.

[47] Baris Kasikci, Benjamin Schubert, Cristiano Pereira, Gilles Pokam, and George Candea. 2015. Failure sketching: a technique for automated root cause diagnosis of in-production failures. In *Proceedings of the 25th Symposium on Operating Systems Principles* (Monterey, California) *(SOSP '15)*. Association for Computing Machinery, New York, NY, USA, 344–360. https://doi.org/10.1145/2815400.2815412

[48] Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. 2023. Demonstrate-Search-Predict: Composing retrieval and language models for knowledge-intensive NLP. arXiv:2212.14024 [cs.CL] https://arxiv.org/abs/2212.14024

[49] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2023. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. arXiv:2310.03714 [cs.CL] https://arxiv.org/abs/2310.03714

[50] Emre Kıcıman, Robert Ness, Amit Sharma, and Chenhao Tan. 2023. Causal Reasoning and Large Language Models: Opening a New Frontier for Causality. arXiv:2305.00050 [cs.AI] https://arxiv.org/abs/2305.00050

[51] Wai-Yin Lam, Bryan Andrews, and Joseph Ramsey. 2022. Greedy relaxations of the sparsest permutation algorithm. In *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence (Proceedings of Machine Learning Research)*, James Cussens and Kun Zhang (Eds.), Vol. 180. PMLR, Cambridge, MA, USA, 1052–1062. https://proceedings.mlr.press/v180/lam22a.html

[52] Lj Lazić and D Velašević. 2004. Applying simulation and design of experiments to the embedded software testing process. *Software Testing, Verification and Reliability* 14, 4 (2004), 257–282.

[53] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baille Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, and Gerardo Vitagliano. 2024. A Declarative System for Optimizing AI Workloads. arXiv:2405.14696 [cs.CL] https://arxiv.org/abs/2405.14696

[54] Xiaoyu Liu, Paiheng Xu, Junda Wu, Jiaxin Yuan, Yifan Yang, Yuhang Zhou, Fuxiao Liu, Tianrui Guan, Haoliang Wang, Tong Yu, Julian McAuley, Wei Ai, and Furong Huang. 2024. Large Language Models and Causal Inference in Collaboration: A Comprehensive Survey. arXiv:2403.09606 [cs.CL] https://arxiv.org/abs/2403.09606

[55] Yudong Liu, Xu Zhang, Shilin He, Hongyu Zhang, Liqun Li, Yu Kang, Yong Xu, Minghua Ma, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. 2022. UniParser: A Unified Log Parser for Heterogeneous Log Data. In *Proceedings of the ACM Web Conference 2022* (Virtual Event, Lyon, France) *(WWW '22)*. Association for Computing Machinery, New York, NY, USA, 1893–1901. https://doi.org/10.1145/3485447.3511993

[56] Stephanie Long, Tibor Schuster, and Alexandre Piché. 2024. Can large language models build causal graphs? arXiv:2303.05279 [cs.CL] https://arxiv.org/abs/2303.05279

[57] Pingchuan Ma, Rui Ding, Shuai Wang, Shi Han, and Dongmei Zhang. 2023. XInsight: eXplainable Data Analysis Through The Lens of Causality. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–27.

[58] Markos Markakis, An Bo Chen, Trinity Gao, and Peter Baile Chen. 2024. LOGos Code. Retrieved Jul 25, 2024 from https://github.com/mitdbg/logos

[59] Markos Markakis, An Bo Chen, Brit Youngmann, Trinity Gao, Ziyu Zhang, Rana Shahout, Peter Baile Chen, Chunwei Liu, Ibrahim Sabek, and Michael Cafarella. 2024. Sawmill: From Logs to Causal Diagnosis of Large Systems. In *Companion of the 2024 International Conference on Management of Data* (Santiago AA, Chile) *(SIGMOD/PODS '24)*. Association for Computing Machinery, New York, NY, USA, 444–447. https://doi.org/10.1145/3626246.3654731

[60] Markos Markakis, Ziyu Zhang, Rana Shahout, Trinity Gao, Chunwei Liu, Ibrahim Sabek, and Michael Cafarella. 2024. Press ECCS to Doubt (Your Causal Graph). In *Proceedings of the Conference on Governance, Understanding and Integration of Data for Effective and Responsible AI* (Santiago, AA, Chile) *(GUIDE-AI '24)*. Association for Computing Machinery, New York, NY, USA, 6–15. https://doi.org/10.1145/3665601.3669842

[61] Themis Melissaris, Markos Markakis, Kelly Shaw, and Margaret Martonosi. 2020. PerpLE: Improving the Speed and Effectiveness of Memory Consistency Testing. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, Piscataway, NJ, USA, 329–341. https://doi.org/10.1109/MICRO50266.2020.00037

[62] Weibin Meng, Ying Liu, Federico Zaiter, Shenglin Zhang, Yihao Chen, Yuzhe Zhang, Yichen Zhu, En Wang, Ruizhi Zhang, Shimin Tao, Dian Yang, Rong Zhou, and Dan Pei. 2020. LogParse: Making Log Parsing Adaptive through Word Classification. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, Piscataway, NJ, USA, 1–9. https://doi.org/10.1109/ICCCN49398.2020.9209681

[63] Glenford J Myers, Corey Sandler, and Tom Badgett. 2011. *The Art of Software Testing.* Wiley, Hoboken, NJ, USA.

[64] Narmada Naik, Ayush Khandelwal, Mohit Joshi, Madhusudan Atre, Hollis Wright, Kavya Kannan, Scott Hill, Giridhar Mamidipudi, Ganapati Srinivasa, Carlo Bifulco, Brian Piening, and Kevin Matlock. 2023. Applying Large Language Models for Causal Structure Learning in Non Small Cell Lung Cancer. arXiv:2311.07191 [cs.AI] https://arxiv.org/abs/2311.07191

[65] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. 2021. Self-supervised Log Parsing. In *Machine Learning and Knowledge Discovery in Databases: Applied Data Science Track*, Yuxiao Dong, Dunja Mladenić, and Craig Saunders (Eds.). Springer International Publishing, Cham, 122–138.

[66] Francisco Neves, Nuno Machado, and JosÃ© Pereira. 2018. Falcon: A Practical Log-Based Analysis Tool for Distributed Systems. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, Piscataway, NJ, USA, 534–541. https://doi.org/10.1109/DSN.2018.00061

[67] Francisco Neves, Nuno Machado, Ricardo Vilaça, and José Pereira. 2021. Horus: Non-Intrusive Causal Analysis of Distributed Systems Logs. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, Piscataway, NJ, USA, 212–223. https://doi.org/10.1109/DSN48987.2021.00035

[68] Chris J Oates, Jessica Kasza, Julie A Simpson, and Andrew B Forbes. 2017. Repair of partly misspecified causal diagrams. *Epidemiology* 28, 4 (2017), 548–552.

[69] Michael Olan. 2003. Unit testing: test early, test often. *Journal of Computing Sciences in Colleges* 19, 2 (2003), 319–328.

[70] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[71] OpenAI. 2024. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/ [Accessed 26-Jul-2024].

[72] OpenAI. 2024. Models. https://platform.openai.com/docs/models/gpt-3-5 [Accessed 31-Jul-2024].

[73] Judea Pearl. 1985. Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proceedings of the 7th conference of the Cognitive Science Society.* Cognitive Science Society, Seattle, WA, USA, 15–17.

[74] Judea Pearl. 2009. *Causality: Models, Reasoning and Inference.* Cambridge University Press, Cambridge, UK.

[75] Judea Pearl and Dana Mackenzie. 2018. *The Book of Why: The New Science of Cause and Effect.* Basic Books, New York, NY, USA.

[76] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[77] Meikel Poess, Bryan Smith, Lubor Kollar, and Paul Larson. 2002. TPC-DS, taking decision support benchmarking to the next level. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data* (Madison, Wisconsin) *(SIGMOD '02)*. Association for Computing Machinery, New York,

NY, USA, 582–587. https://doi.org/10.1145/564691.564759

[78] James M Robins, Miguel Angel Hernan, and Babette Brumback. 2000. Marginal structural models and causal inference in epidemiology.

[79] Donald B. Rubin. 1980. Discussion of 'Randomization Analysis of Experimental Data in the Fisher Randomization Test'by Basu. *J. Amer. Statist. Assoc.* 75, 371 (1980), 591–93.

[80] Per Runeson. 2006. A survey of unit testing practices. *IEEE software* 23, 4 (2006), 22–29.

[81] Caitlin Sadowski and Jaeheon Yi. 2014. How Developers Use Data Race Detection Tools. In *Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools* (Portland, Oregon, USA) *(PLATEAU '14)*. Association for Computing Machinery, New York, NY, USA, 43–51. https://doi.org/10.1145/2688204.2688205

[82] Felix Salfner and Steffen Tschirpke. 2008. Error Log Processing for Accurate Failure Prediction. *WASL* 8 (2008), 4.

[83] Babak Salimi, Johannes Gehrke, and Dan Suciu. 2018. Bias in OLAP Queries: Detection, Explanation, and Removal. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 1021–1035. https://doi.org/10.1145/3183713.3196914

[84] Bianca Schroeder and Garth A Gibson. 2009. A large-scale study of failures in high-performance computing systems. *IEEE transactions on Dependable and Secure Computing* 7, 4 (2009), 337–350.

[85] Amazon Web Services. 2024. Cloud Compute Capacity - Amazon EC2 - AWS. https://aws.amazon.com/ec2/ [Accessed 31-Jul-2024].

[86] Amit Sharma, Emre Kiciman, et al. 2019. DoWhy: A Python package for causal inference. https://github.com/microsoft/dowhy.

[87] Manish Shetty, Chetan Bansal, Sai Pramod Upadhyayula, Arjun Radhakrishna, and Anurag Gupta. 2022. AutoTSG: learning and synthesis for incident troubleshooting. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Singapore, Singapore) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 1477–1488. https://doi.org/10.1145/3540250.3558958

[88] Xu Shi, Ziyang Pan, and Wang Miao. 2023. Data integration in causal inference. *Wiley Interdisciplinary Reviews: Computational Statistics* 15, 1 (2023), e1581.

[89] Shohei Shimizu, Patrik O Hoyer, Aapo Hyvärinen, Antti Kerminen, and Michael Jordan. 2006. A linear non-Gaussian acyclic model for causal discovery. *Journal of Machine Learning Research* 7, 72 (2006), 2003–2030. http://jmlr.org/papers/v7/shimizu06a.html

[90] Bill Shipley. 2016. *Cause and correlation in biology: a user's guide to path analysis, structural equations and causal inference with R.* Cambridge University Press, Cambridge, UK.

[91] Tomi Silander and Petri Myllymäki. 2006. A simple approach for finding the globally optimal Bayesian network structure. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence* (Cambridge, MA, USA) *(UAI'06)*. AUAI Press, Arlington, Virginia, USA, 445–452.

[92] Peter Spirtes and Clark Glymour. 1991. An algorithm for fast recovery of sparse causal graphs. *Social science computer review* 9, 1 (1991), 62–72.

[93] Peter Spirtes, Clark N Glymour, and Richard Scheines. 2000. *Causation, prediction, and search.* MIT press, Cambridge, MA, USA.

[94] Splunk. 2024. https://www.splunk.com/ [Accessed 31-Jul-2024].

[95] Abraham Starosta. 2021. How Splunk Is Parsing Machine Logs With Machine Learning On NVIDIA's Triton and Morpheus. https://www.splunk.com/en_us/blog/it/how-splunk-is-parsing-machine-logs-with-machine-learning-on-nvidia-s-triton-and-morpheus.html [Accessed 31-Jul-2024].

[96] Jin Tian, Azaria Paz, and Judea Pearl. 1998. *Finding Minimal D-separators.* Citeseer, University Park, PA, USA.

[97] Robert Tibshirani. 1996. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)* 58, 1 (1996), 267–288.

[98] Ruibo Tu, Chao Ma, and Cheng Zhang. 2023. Causal-Discovery Performance of ChatGPT in the context of Neuropathic Pain Diagnosis. arXiv:2301.13819 [cs.CL] https://arxiv.org/abs/2301.13819

[99] Joseph Tucek, Shan Lu, Chengdu Huang, Spiros Xanthos, and Yuanyuan Zhou. 2007. Triage: diagnosing production run failures at the user's site. *ACM SIGOPS Operating Systems Review* 41, 6 (2007), 131–144.

[100] Hal R Varian. 2016. Causal inference in economics and marketing. *Proceedings of the National Academy of Sciences* 113, 27 (2016), 7310–7315.

[101] Aniket Vashishtha, Abbavaram Gowtham Reddy, Abhinav Kumar, Saketh Bachu, Vineeth N Balasubramanian, and Amit Sharma. 2023. Causal Inference Using LLM-Guided Discovery. arXiv:2310.15117 [cs.AI] https://arxiv.org/abs/2310.15117

[102] Thomas Verma and Judea Pearl. 1990. Equivalence and Synthesis of Causal Models. In *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence (UAI '90)*. Elsevier Science Inc., USA, 255–270.

[103] Dolores R Wallace and Roger U Fujii. 1989. Software verification and validation: an overview. *Ieee Software* 6, 3 (1989), 10–17.

[104] Yiting Xia, Ying Zhang, Zhizhen Zhong, Guanqing Yan, Chiun Lin Lim, Satyajeet Singh Ahuja, Soshant Bali, Alexander Nikolaidis, Kimia Ghobadi, and Manya Ghobadi. 2021. A Social Network Under Social Distancing: Risk-Driven Backbone Management During COVID-19 and Beyond. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Berkeley, CA, USA, 217–231. https://www.usenix.org/conference/nsdi21/presentation/xia

[105] Tong Xiao, Zhe Quan, Zhi-Jie Wang, Kaiqi Zhao, and Xiangke Liao. 2020. LPV: A Log Parser Based on Vectorization for Offline and Online Log Parsing. In *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, Piscataway, NJ, USA, 1346–1351. https://doi.org/10.1109/ICDM50108.2020.00175

[106] Feng Xie, Ruichu Cai, Biwei Huang, Clark Glymour, Zhifeng Hao, and Kun Zhang. 2020. Generalized independent noise condition for estimating latent variable causal graphs. *Advances in neural information processing systems* 33 (2020), 14891–14902.

[107] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles* (Big Sky, Montana, USA) *(SOSP '09)*. Association for Computing Machinery, New York, NY, USA, 117–132. https://doi.org/10.1145/1629575.1629587

[108] Brit Youngmann, Michael Cafarella, Babak Salimi, and Anna Zeng. 2023. Causal Data Integration. *Proc. VLDB Endow.* 16, 10 (jun 2023), 2659–2665. https://doi.org/10.14778/3603581.3603602

[109] Brit Youngmann, Michael J. Cafarella, Babak Salimi, and Anna Zeng. 2023. Causal Data Integration. *Proc. VLDB Endow.* 16, 10 (2023), 2659–2665.

[110] Changhe Yuan and Brandon Malone. 2013. Learning optimal Bayesian networks: A shortest path perspective. *Journal of Artificial Intelligence Research* 48 (2013), 23–65.

[111] Ding Yuan, Haohui Mai, Weiwei Xiong, Lin Tan, Yuanyuan Zhou, and Shankar Pasupathy. 2010. SherLog: error diagnosis by connecting clues from run-time logs. *SIGARCH Comput. Archit. News* 38, 1 (mar 2010), 143–154. https://doi.org/10.1145/1735970.1736038

[112] Yulai Yuan, Yongwei Wu, Qiuping Wang, Guangwen Yang, and Weimin Zheng. 2012. Job failures in high performance computing systems: A large-scale empirical study. *Computers & Mathematics with Applications* 63, 2 (2012), 365–377.

[113] Jiaming Zeng, Michael F Gensheimer, Daniel L Rubin, Susan Athey, and Ross D Shachter. 2022. Uncovering interpretable potential confounders in electronic medical records. *Nature Communications* 13, 1 (2022), 1014.

[114] Tianzhu Zhang, Han Qiu, Gabriele Castellano, Myriana Rifai, Chung Shue Chen, and Fabio Pianese. 2023. System Log Parsing: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 35, 8 (2023), 8596–8614. https://doi.org/10.1109/TKDE.2022.3222417

[115] Yujie Zhao and Xiaoming Huo. 2023. A survey of numerical algorithms that can solve the Lasso problems. *WIREs Computational Statistics* 15, 4 (2023), e1602. https://doi.org/10.1002/wics.1602 arXiv:https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/wics.1602

[116] Yujia Zheng, Biwei Huang, Wei Chen, Joseph Ramsey, Mingming Gong, Ruichu Cai, Shohei Shimizu, Peter Spirtes, and Kun Zhang. 2024. Causal-learn: Causal Discovery in Python. *Journal of Machine Learning Research* 25, 60 (2024), 1–8. http://jmlr.org/papers/v25/23-0970.html

[117] Ziming Zheng, Zhiling Lan, Byung H. Park, and Al Geist. 2009. System log pre-processing to improve failure prediction. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*. IEEE, Piscataway, NJ, USA, 572–577. https://doi.org/10.1109/DSN.2009.5270289

[118] Jiongli Zhu, Sainyam Galhotra, Nazanin Sabri, and Babak Salimi. 2023. Consistent Range Approximation for Fair Predictive Modeling. arXiv:2212.10839 [cs.LG]

[119] Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, and Michael R. Lyu. 2023. Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics. arXiv:2008.06448 [cs.SE] https://arxiv.org/abs/2008.06448

[120] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R. Lyu. 2019. Tools and benchmarks for automated log parsing. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice* (Montreal, Quebec, Canada) *(ICSE-SEIP '19)*. IEEE, Piscataway, NJ, USA, 121–130. https://doi.org/10.1109/ICSE-SEIP.2019.00021

[121] Shengyu Zhu, Ignavier Ng, and Zhitang Chen. 2020. Causal Discovery with Reinforcement Learning. arXiv:1906.04477 [cs.LG] https://arxiv.org/abs/1906.04477

[122] Yiwen Zhu, Rathijit Sen, Robert Horton, and John Mark Agosta. 2023. Runtime Variation in Big Data Analytics. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–20.

[123] Danyang Zhuo, Monia Ghobadi, Ratul Mahajan, Klaus-Tycho Förster, Arvind Krishnamurthy, and Thomas Anderson. 2017. Understanding and Mitigating Packet Corruption in Data Center Networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 362–375. https://doi.org/10.1145/3098822.3098849