MoCE: A Mixture-of-Context-Aware Experts Framework for Troubleshooting Internet-scale Services

Vipul Harsh[1,3], *Sayan Sinha*[1,2], *Henry Milner*[1], *B Aditya Prakash*[1,2], *Vyas Sekar*[1,3], *Hui Zhang*[1,3]

[1]*Conviva* [2]*Georgia Tech* [3]*Carnegie Mellon University*

Abstract

Modern Internet-scale services need to rapidly identify root causes of customer-impacting incidents and remediate them. While there are many algorithms (including LLM-assisted solutions) for root cause analysis, these have significant limitations in terms of *coverage*, *extensibility*, and *scalability* due to the diversity of incidents that can occur at Internet-scale and the complexity of telemetry analysis. We argue the need for a paradigm shift in root cause analysis to depart from algorithm development to a systems approach.

To this end, we introduce a *mixture of context-aware experts* framework where each "expert" represents a root cause hypothesis exploration. To enable rapid development of new experts and allow computational reuse across the ensemble for scalability, we design an abstraction that allows us to express an expert as a dataflow DAG combining relational, stateful, and statistical operations. To ensure scalability and extensibility, we develop a *lazy DAG* runtime system that lazily schedules execution of DAG nodes. We implement this idea in MoCE and demonstrate its value using a mix of real-world incident data from four large application analytics providers and synthetically generated incidents. We show that many existing and novel approaches can be expressed succinctly in our framework. We find that MoCE achieves high RCA accuracy (>95%) across diverse incidents compared to 34% for the closest single expert (including prior works) achieving high coverage. We also show the value of the mixture paradigm and the lazy DAG runtime using controlled experiments.

1 Introduction

Providers running Internet-scale services (e.g., Netflix, Uber, DoorDash, Amazon) strive to deliver a good experience to their users. For instance, video providers may worry about startup latency or buffering [28]. Other e-commerce services care about login times, time to submit payments etc. [2]. Other mobile application providers may care about app loading times [22], and so on. As observed elsewhere, user-impacting incidents are the norm in a complex ecosystem with numerous third-party dependencies and client-side interactions [27, 53].

When user-impacting incidents occur, incident response teams today rely on a hodge-podge of tools, dashboards, manual effort, and root cause analysis (RCA) algorithms to identify *leads*[1]. Once identified, these leads are validated, and then remedial actions are taken (e.g., restarting a service, changing the load balancing policy, or rolling back an update). This process is challenging due to the *diversity* of incidents that manifest at Internet scale, the *complexity* of the Internet-scale services (e.g., third-party dependencies, software updates, device versions, ISP, geography) and the *scale* of data analysis requiring complex spatiotemporal and contextual correlations across multiple telemetry sources.

While there are many existing root cause analysis algorithms, they offer narrow or point solutions tailored for specific classes of failure scenarios and specific types of telemetry data. Industrial efforts in so-called "AIOps" have largely focused on "backend" services (e.g., microservices) [33, 40] and in general these solutions have failed to deliver the promise of AIOps [32]. While recent LLM-assisted approaches [24, 70] for troubleshooting have the potential to be generally applicable across multimodal data, but have poor accuracy [70].

Instead of proposing yet another point solution tailored to a specific incident or telemetry type, we argue for a systems approach for root cause analysis that addresses three key requirements: (1) *High coverage and accuracy* to produce accurate results that reduce analyst effort and cover diverse and possibly unforeseen incident classes; (2) *Extensibility* to support new types of incidents, analysis algorithms, and telemetry sources as the system evolves; and (3) *Scalability* to handle large volumes of data and sophisticated algorithms that run over the data.

To this end, we present the design and implementation of MoCE based on three interconnected ideas:

- First, to ensure high coverage, accuracy and extensibility, we propose a *mixture of context-aware experts* paradigm. That is, we explicitly embrace the diversity and argue for

[1]We intentionally call these leads rather than root causes because the result is a hypothesis that needs to be further explored and validated

an ensemble approach, where each "expert" can shed light on a smaller scope of possible leads. Context here refers to obtaining relevant slices of the telemetry data for the diagnostic process. It can be any type of temporal, spatial, or user-session state-level correlations and can manifest at multiple resolutions; e.g., fine-grained context for specific services contacted "during" a user session or the spatial context looking for aggregate patterns across users with a specific device version.

- Second, by analyzing existing proposed and potentially new root cause analysis algorithms, we identify a succinct abstraction for expressing a broad spectrum of such experts. More specifically, each "expert" is expressed as a declarative dataflow DAG combining three types of operations– relational analytics, stateful analytics, and ML/statistical computation. This abstraction aids in rapid prototyping of new experts, as they can be based on existing operators and DAGs, and improves scalability by enabling systematic reuse of intermediate sub-expressions across experts, thereby avoiding redundant computations– a key source of efficiency beyond accuracy improvements.
- Finally, to enable extensibility and efficiently run the mixture, we introduce a new *lazy DAG* runtime framework. Naively, the runtime of a mixture algorithm with several experts can become a performance bottleneck. The lazy DAG approach allows us to intelligently reuse and lazily trigger the computation of the dataflow nodes.

We envision MoCE running in concert with an existing anomaly detection or alerting framework. When a new incident alert is received, MoCE "routes" the analysis to a suitable subset of experts. The mixture of experts is then executed by the lazy DAG runtime. MoCE then collects their hypothesized RCA *leads*, and *combines* their results to present a candidate shortlist of leads to the analyst. By design, MoCE is extensible to support new telemetry sources, new expert algorithms, and new policy logic as needed.

We implement a library of operators to enable diverse experts in MoCE and a scalable implementation of the lazy DAG runtime by extending `Ray` [7]. We prototype a broad array of context-aware experts written in our framework. We evaluate MoCE on large-scale datasets obtained from a large data analytics provider supporting Internet-scale video and application services against a number of baselines from prior work, including emerging LLM-assisted RCA approaches [24, 70]. We systematically validate MoCE using synthetic but real-world inspired incidents and find that MoCE achieves high recall ($> 95\%$) across diverse classes of incidents whereas prior works achieve less than 25%. We also run a pilot study on one week of production data from four large-scale services and find that MoCE gets the ground truth root cause in its top-5 candidate leads 90% of the time.

Contributions: We summarize our contributions below:

- We present a novel MoCE paradigm for reformulating the root cause analysis problem as a systems-level framework that tackles diversity and extensibility.
- We develop a library of operators and workflows, and a number of end-to-end concrete experts inspired by prior work and novel ideas that can aid in troubleshooting multiple types of incidents involving diverse data sources.
- We present an end-to-end design and implementation of the MoCE framework that runs the ensemble at scale using the novel lazy DAG runtime.
- We design a simulator to generate user events for internet-scale services, inspired by patterns and data in the real world. We use this simulator to recreate multiple types of incidents seen in production.
- We present a real-world evaluation of incidents that were surfaced by our system.

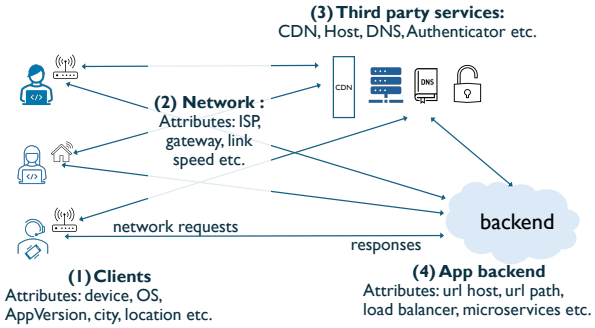


Figure 1: Internet-scale services entail complex interactions between (1) clients, (2) network, (3) third-party services, and (4) backend components and can fail in myriad ways that impact the user experience.

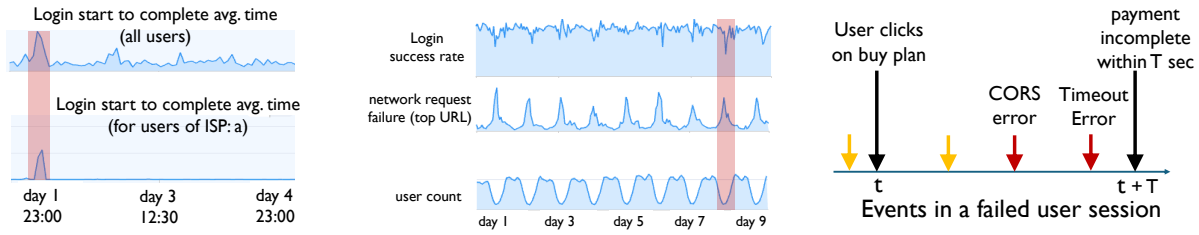
2 Background and motivation

In this section, we begin with some real-world incidents to highlight the limitations of existing approaches in tackling the coverage, extensibility, and cost-effectiveness requirements.

2.1 Motivating real-world scenarios

Background and Setting: We consider the setting of modern Internet-scale services such as e-commerce marketplaces, media platforms, and web/mobile applications. These applications have complex dependencies on numerous components, including client-side entities, third-party services, backend infrastructure (see Figure 1). Consequently, these services can have diverse failure modes that impact user-centric experience and performance KPIs (e.g., page load time, time to login, payment failure rate).

To this end, such services use data collection and analytics tools to: (1) measure user-impacting KPIs of interest; (2) detect and alert analysts when KPIs are impacted; and (3) diagnose the root cause of the incident to suggest mitigations (e.g., [24, 25]). Our focus in this paper is on the third step of *root cause diagnosis* and we assume there are suitable mechanisms for computing KPIs (e.g., [29]) and alerting (e.g., [62]).



(a) Example 1 **(b) Example 2** **(c) Example 3**

Figure 2: Illustrative failure incidents from production services: (a) Users of a particular ISP experienced high login completion times; (b) Users experienced high request failure rates for a service every night during periods of low load in a particular region highlighted in red. (c) Event timeline for a user session ending in a payment conversion failure.

Illustrative Examples: We describe real incidents in production from a large application observability and analytics provider serving leading mobile, web, and video streaming applications. We use these examples to highlight the diversity of failures that can occur, and why it is challenging for any single root cause analysis (RCA) algorithm to effectively identify RCA leads.

- *Incident 1: High response times for a particular ISP (Figure 2a).* Users of a video OTT platform experienced sharp spikes in login completion duration, taking 5-10× the usual time. The likely source of the problem was a particular ISP whose users were experiencing 1-2 orders of magnitude higher completion times during the anomalous period.
- *Incident 2: Daily pattern of high failure rates for a network request service (Figure 2b).* Users of a media content platform experienced lower login success rates every night, counterintuitively during periods of low load. The likely source of the problem was a specific service endpoint (identified by URL), involved in the Login operation, which had a high failure rate (up to 40%) during these periods.
- *Incident 3: A sequence of events leads to an error (Figure 2c).* To monitor how quickly new users are able to subscribe, a content platform created a binary metric to track whether a prospective subscriber who clicked the "buy plan" button was able to complete "payment" within a specific time. When debugging an unrelated problem, our analysts discovered that many sessions where this failure metric was flagged showed a specific event sequence: users who clicked buy_plan, then encountered a web_cors_error, and subsequently a timeout_error, did not complete the payment (Figure 2c). The initial CORS error (a browser security feature) triggered retries that led to the timeout.[2]

2.2 Requirements and Prior Work

Based on these examples, we derive qualitative requirements for RCA systems and discuss why existing solutions fall short.

Qualitative Requirements: First, these incidents represent diverse failure modes (e.g., in the networking layer, backend service, a pattern of client events). In practice, there is likely an even larger set of possible failure modes; e.g. network bandwidth, resource contention in microservices, and so on. Thus, an effective RCA system must handle *diverse classes of incidents* to achieve high coverage.

Second, to zoom in to the possible hypothesis that explains the incident it may be required to look at a large volume of diverse telemetry data (e.g. metrics, logs, events) collected from various vantage points (e.g., client-side vs backend metrics). For instance, incident 1 can entail analyzing millions of possible user-attribute-cohort timeseries, incident 2 requires us to look at correlations across multiple sources of data, and incident 3 requires us to mine event sequence patterns to get to the root cause. Even a "medium scale" provider shown here can generate up to 1 TB of just client-side raw event data in a day. Identifying the contextually relevant data across space and time involves complex relational and stateful data processing. Then, we need to run statistical machinery to specify and check hypotheses. This means that any RCA solution should handle complex, stateful data processing at *scale*.

Finally, the only constant thing in Internet-scale services is change and there is an inherent churn in the deployment, the types of failure modes (e.g., some problems get resolved with updates), the types of algorithms that analysts want to adopt, and the set of telemetry sources that become available. This means that an ideal RCA system should be *extensible*. For instance, if we had a rigid system that cannot handle raw event data or correlate metrics across client- and backend services, that would limit the expressivity and lead to false leads and/or blind spots.

Existing approaches and limitations: At a high level, we can taxonomize prior RCA work along two dimensions: (1) Type of telemetry data that they can handle and (2) Type of analytics algorithms that they use. Table 1 provides a high level summary and exemplars from prior work.

Some solutions (e.g., Adtributor [18]) identify problematic user subpopulations based on attributes (e.g., device, ISP) but cannot model backend request failures (example 2), client-side patterns (example 3), network degradations, or software bugs. Other works such as Minesweeper [53] can find problematic event patterns but cannot detect cohort-level issues

[2]Note that occasional CORS errors alone are not necessarily an issue since the application would usually recover from the error by retrying.

Type of data in scope	Algorithmic Approach	Exemplars	Coverage	Scalability	Extensibility	Limitations
Client-side metrics	Causal inference, statistical tests	CFA, Adtributor [18]	X	✓	X	cannot be extended to events or backend-side logs, traces
Backend metrics, traces, logs	Dependency graph analysis, log mining	Sage [33], Murphy [40]	X	✓	X	cannot be extended to events or client-side metrics
Client-side events	Statistical tests	Minesweeper [53]	X	✓	X	cannot be extended to logs, metrics or traces
Multimodal data	LLM-assisted	OpenRCA [70] RcaCopilot [24]	X	✓	✓	generally applicable but has poor RCA accuracy

Table 1: Exemplar approaches in the RCA literature from academia and industry and their limitations in meeting the coverage, scalability, and extensibility requirements for troubleshooting Internet-scale services.

or backend problems. Many recent efforts like Sage [33] and Murphy [40] focus on backend infrastructure, but network failures or client-side patterns are outside their scope. In essence, these efforts are valuable in specific contexts but do not meet our goals of achieving high-coverage of diverse incident classes. While some solutions aim to be more general, they too have limitations. Scouts [35] uses domain-specific classifiers written by individual teams to route an incident to the right team but it too doesn't provide a path for achieving high-coverage and extensibility. Emerging LLM-assisted methods [24, 37, 70] can be more general, but have poor accuracy [70] as we will also see later.

Takeaways: Troubleshooting user-impacting incidents in modern Internet-scale services is challenging due to the diversity of incidents, scale of telemetry data and processing needs, and the constant change in deployments. Ideally, we want an RCA system that can satisfy: (1) *high-coverage* of diverse classes of failure incidents (2) *extensibility* to quickly incorporate new failure types and telemetry sources and (3) *scalability* to analyze large datasets efficiently. While existing approaches have made significant algorithmic strides in specific scenarios, they fall short on delivering a systems solution to meet these requirements.

3 MoCE Design Overview

To meet the goals outlined earlier, we present MoCE a system for enabling root cause diagnosis for Internet-scale services (Figure 3). MoCE can handle diverse sources of telemetry with different modalities (e.g., client-side vs. backend telemetry, software update logs) that can vary in their arrival patterns (e.g., real time, asynchronous), nature of access (e.g., push vs. pull), and location (e.g., client vs. backend).

MoCE is invoked by some alerting workflows. At run time, the input to MoCE is some identified anomaly in a user experience metric. The design of such an alerting framework (e.g., [6]) is outside the scope of this paper and MoCE is agnostic to the design of the detection algorithms. As such, MoCE receives a specification of a customer-impacting incident (e.g., the subpopulation of users impacted and the incident time period).

Given available telemetry and the incident specification,

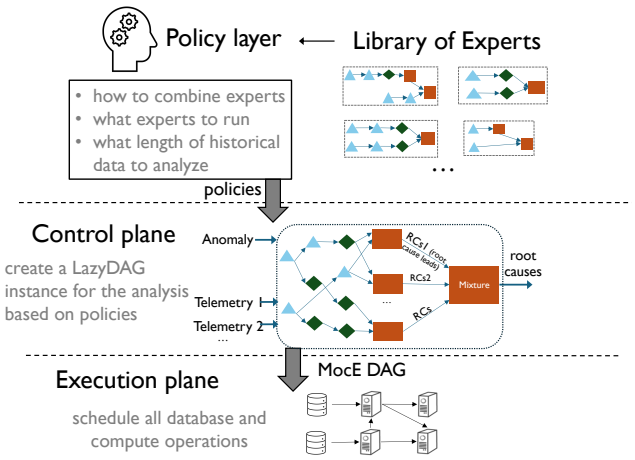


Figure 3: The MoCE system has four logical components: (a) the policy layer (b) a library of experts (c) the control plane and (d) the execution plane.

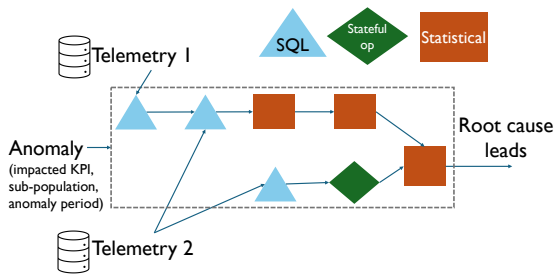
MoCE produces a succinct list of *RCA leads and evidence trails* to the human analyst, automating most of the manual triage and effort that analysts need to do today. Our goal is to significantly reduce the human effort in incident response rather than a fully autonomous system. The end goal of validation and mitigation is still performed by the human analysts.

Scope: In this work, we focus primarily on client-side telemetry. We note, however, that our system can be extended for other telemetry sources, and multi-telemetry analysis.

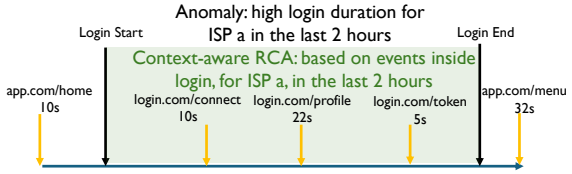
3.1 Key Ideas

Next, we describe the key design ideas in MoCE to tackle the coverage, scalability, and extensibility challenges.

Mixture of Context-aware Experts: We argue that we need to rethink RCA not as an algorithmic problem but as a *systems* problem for enabling analysts and domain experts to express and iterate on RCA approaches. We posit that there will not be a *one size fits all* RCA algorithm that will cover all possible current and future incident patterns for all telemetry sources. To this end, we make a case for moving away from a single "algorithm" to support a *ensemble* approach where we can have a collection of *context-aware expert algorithms*



(a) Anatomy of an expert



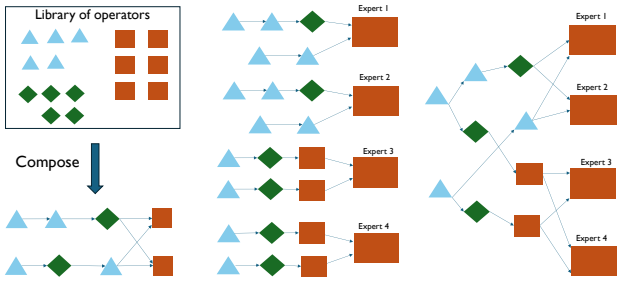
(b) Context-aware RCA example

Figure 4: (a) An expert is a DAG of 3 types of operators: relational, stateful, and statistical/algorithmic. (b) Context-aware RCA example: RCA is focused on the session events (i) inside the Login-window (ii) during the time of the anomaly (iii) for the anomalous sub-population.

that can run together. A mixture of experts scheme is better equipped to handle diverse incident classes, tackling the issue of *coverage* and *extensibility*. Having experts be context-aware focuses the analysis on relevant data, leading to more accurate RCA.

A declarative DAG framework to write experts: From analyzing previous RCA solutions and our own workflows, we observe a common pattern. First, temporal and spatial cohort filters based on the anomaly time period and the impacted user-subpopulations are applied. Then, *stateful context-aware* filtering to obtain relevant sequences of events e.g. events within a relevant session flow, such as Login, defined by the start and end events (see Figure 4b). In many cases, this is followed by aggregation, where metric values are grouped by cohort attributes (e.g., device type, location). Finally, we run *statistical models* on these datasets to generate output RCA leads. Based on this pattern, we identify a succinct and expressive abstraction to declaratively express RCA experts. Specifically, many RCA workflows can be expressed as a DAG of the following 3 types of operators (Figure 4a):

- Relational operators: these are SQL queries for manipulating relational data and are based on the familiar SQL primitives: SELECT, PROJECT, JOIN, GROUP_BY etc.
- Stateful operators over a sequence of events in a user-session: these operators are designed to work on ordered sequences of events, typically grouped into "sessions" based on some identifier. They are essential for understanding behavior over time. Expressing such stateful computation using only relational primitives can be tedious (e.g. [51]); thus we elevate these as first-class primitives in MoCE.



(a) Authoring experts (b) Separate experts (c) Unified DAG

Figure 5: The DAG abstraction helps with accelerating expert development via operator and workflow reuse, and avoiding duplicate computation of intermediate nodes, enabling extensibility and scalability. (a) Operators from the library can be composed together to author new experts (b)→ (c) All expert DAGs can be merged into one unified DAG to avoid duplicate computation.

- Statistical and ML operators: These operators apply complex statistical transformations, featurization, and modeling primitives to intermediate data, typically the output of some sequence of relational and stateful operators. These are often expressed in a high-level programming language like Pythonic libraries (e.g., scikit-learn), to generate new features. These operators express the algorithmic or machine learning logic required for the RCA tasks.

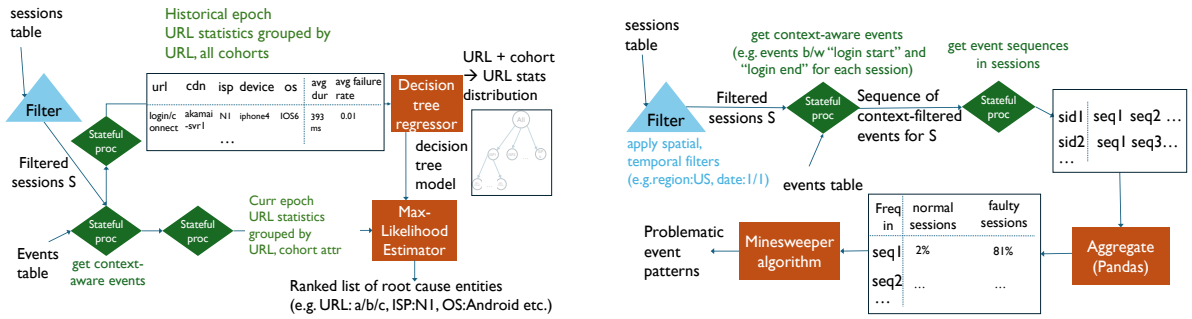
While the notion of dataflow DAGs is not new, our insight here is in identifying the building blocks (§ 4.1) relevant for expressing RCA experts. This abstraction and library makes it possible for sub-expression reuse and to avoid duplicate intermediate computation across experts. MoCE's ensemble approach subsumes existing point solutions, as any specific algorithm is one more expert in the ensemble. This framework is extensible as new experts can be easily written, reusing existing DAG operators and workflows.

Efficient lazy DAG execution: On one hand, for higher coverage and extensibility, we want to enable domain scientists to write more context-aware experts. On the other hand, to control the compute cost, we may want to limit the set of experts to run. We address this trade-off by introducing a *lazy DAG* runtime framework to balance extensibility and efficiency. With this runtime model, the system can use a large library of experts but only invoke the relevant DAG nodes to be computed and reuse intermediate results across experts.

3.2 End-to-End View

Putting these ideas together, MoCE is organized across four logical layers as shown in Figure 3:

- *Library of experts:* Each expert will produce a list of plausible root causes, ranked by confidence scores.
- *Policy Layer:* The policy layer defines the logic for *routing* incidents to specific experts and *merging* the hypotheses



(a) MLE based inference of faulty URLs **(b) Minesweeper**

Figure 6: Illustrative expert DAGs: (a) An expert that checks for significant changes in URL failures during historical and anomalous periods (b) A previous work Minesweeper [53] expressed in our framework that finds event patterns as root causes. Note how the initial operators are common across these two experts.

across the chosen experts. MoCE allows analysts to configure different policies for routing and combining; e.g., based on confidence level, history, agreement, and so on.

- *Control plane:* This layer is set up with the ensemble DAG and access to telemetry sources that experts will access. At run time, on receiving a new incident, it triggers the computation of the lazy DAG in the execution layer. MoCE can be invoked on demand, when an analyst manually detects an anomaly, or run periodically in conjunction with an anomaly detection module.
- *Execution plane:* These are the "workers" that run the ensemble of experts lazily, cache intermediate results, and notify the control plane when they complete a job. MoCE can accommodate asynchronous telemetry sources, such as those updated periodically, by running the relevant experts as soon as new data arrives and incrementally updating its final result as each expert finishes execution.

4 MoCE Detailed Design

We describe the detailed designs for the following subcomponents of our system (Figure 3): (1) the DAG framework to author new context-aware experts (for the library of experts); (2) the lazy DAG runtime system to run the experts efficiently (in the execution plane); and (3) the mixture policy for routing incidents and combining results from experts (in the policy layer).

4.1 Expressing context-aware experts

In MoCE, we want the developer to be able to quickly author new experts to add more algorithms, telemetry sources, and incident types. As described earlier, each RCA expert can be expressed as a dataflow DAG composed of three types of operators (Figure 4a). These operators help to obtain the context-relevant data for RCA for a given incident. For instance, if a spike is observed in the login authentication latency metric for a sub-population of users (aka cohort), the context for the corresponding RCA can focus on the cohort, the anomaly time period and events between a "login-start" event and before a "login-end" event in each session (see Figure 4b). We will formally describe these operators in detail later.

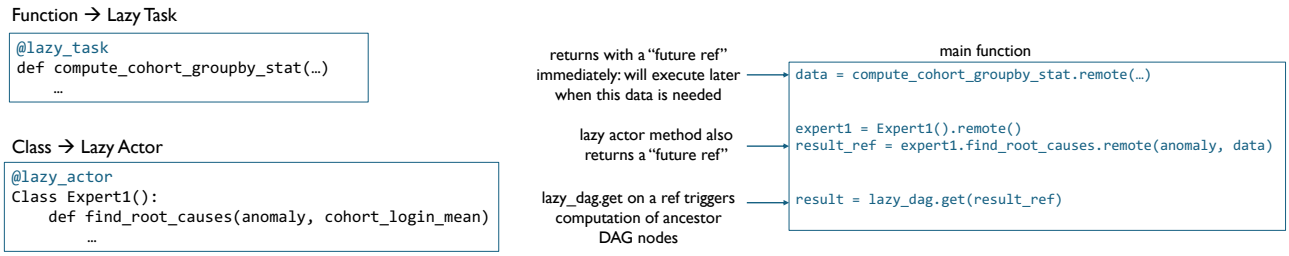
Authoring experts: Figure 5a illustrates how developers and analysts can write new experts based on operators in the library. We consider two examples one from prior work and a novel RCA approach we developed (see Figure 6). The first expert checks for URLs (i.e., service endpoints) that showed significant changes in request failure rate between the normal and anomalous periods as candidate leads. The second expert implements the prior Minesweeper approach [53] to uncover event patterns as candidate root causes.

While authoring an expert, the developer can use existing operators to define their DAG as shown in Figure 5a. Additionally, MoCE allows the developer to publish nodes in the dataflow DAG as *predefined workflows*. Other developers writing experts can base their extensions using these published workflows. For instance, if the developer wants to write a new event pattern expert, they can reuse much of the DAG in Figure 6b. This reduces the development overhead for writing new experts, as often developers only have to write a final statistical logic, if the workflow for computing the relevant contextual data already exists. Finally, expressing the RCA as DAGs allows the runtime system to compile all experts into one unified DAG and schedule them efficiently (Figure 5c).

4.1.1 Computational tools for context-aware experts

As described earlier, we need three types of computational tools for expressing context-aware experts: corresponding to SQL, Stateful and Statistical operations . Each operator in the MoCE library is internally implemented using one of these tools. Before formally describing these computational engines, we define some useful terminology.

Let D be a set of primitive domains (e.g., Integer, String, Timestamp). A tuple t is an ordered list of values $(v_1, v_2, ..., v_n)$ where for v_i, type$(v_i) \in D$. We define two types of primitive data types in MoCE: (1) a *Relation (R)* is a set of tuples with a fixed schema (as in SQL): $R = \{t_1, t_2, ..., t_m\}$; and (2) a *Sequence (E)* is an ordered list of tuples (e.g. events),



(a) Declaring a lazy task and a lazy actor **(b) Invoking a lazy task and a lazy actor method**

Figure 7: Lazy DAG programming model: Developers can decorate Python functions and classes as lazy_task and lazy_actor respectively. When these methods are invoked, the backend intercepts the task and returns with a reference without starting computation, allowing implicit DAG creation. The task gets scheduled only when the results are required.

partitioned by a key K (e.g. session ids). A sequence is a function from a key $k \in K = \text{dom}(E)$ to an ordered list of tuples $E : K \rightarrow [t_1, t_2, ..., t_p]$.

Next we describe the primitive operations that can be used to express SQL, stateful or statistical operators:

- *Primitives for Relational DB (Φ)*: These include the SQL-like primitives: SELECT, PROJECT, JOIN, GROUP_BY and AGGREGATE, that map one or more relations to a new relation: $\Phi : R \times \cdots \rightarrow R$.
- *Primitives for Stateful Operators (Ψ)*: Stateful primitives are essential to express computation that depends on event ordering e.g. obtaining event sequences between login start and login end. They transform data between relations and sequences and allow the programmer to conveniently express stateful logic that is difficult to express using relational algebra [51]. We describe a non-exhaustive list of primitives that can be used to express a stateful operator in the MoCE operator library:
  - Sequencify ($\Psi_{k,o}(R)$): Converts a relation to a sequence partitioned by key k and ordered by o. Let R_k be the subset of tuples in R with key k. Let $\text{sort}_o(R_k)$ be that subset ordered by attribute o.
    $\Psi_{k,o}(R) = E$ such that for each key $k', E(k') = \text{sort}_o(R_{k'})$
  - Filter Sequence ($\Psi_p(E)$): This operator takes a sequence and produces a new sequence containing only the tuples that satisfy a predicate p. $\Psi_p(E) = E'$ such that,
    $$\forall \text{ key } k \in \text{dom}(E), E'(k) = [t \mid t \in E(k) \wedge p(t)]$$
  - Merge Sequences ($\Psi_{\text{merge}}(E_1, E_2)$): This operator combines two sequences that share the same key domain into a single sequence, merging the ordered lists of tuples for each key. $\Psi_{\text{merge}}(E_1, E_2) = E_{\text{merged}}$ such that,
    $$\forall \text{ key } k \in \text{dom}(E), E_{\text{merged}}(k) = \text{merge}_o(E_1(k), E_2(k))$$
    Where $\text{merge}_o(list_1, list_2)$ be a function that merges two lists based on an ordering attribute o.
  - Stateful Map ($\Psi_f(E)$): This primitive produces a new sequence by applying a stateful transition function f to each tuple in the original sequence in order. $\Psi_f(E) = E_{\text{new}}$ such that with the following semantics: let S be the domain of possible states, and $s_0 \in S$ be the initial state. The function f takes a state and a tuple $\in T$, and returns a new state and a list of new output tuples, possibly from a new set T'. For a single sequence $E(k) = [t_1, t_2, \ldots, t_n]$, the transformation is defined recursively:
    $$(s_k, T'_k) = f(s_{k-1}, t_k) \forall k \in [1, n]$$
    The resulting sequence is then $E_{\text{new}}(k) = [T'_1 \circ T'_2 \circ ... T'_n]$, where $\circ$ denotes the list concatenation operator.
  - Relationalize ($\Psi_{\text{rel}}(E)$): This operator converts a sequence back into a relation, flattening the ordered, grouped structure. The resulting relation is the union of all tuples from all sequences. Let $\text{concat}(k, t)$ be the operation that prepends key k to tuple t.
    $$\Psi_{\text{rel}}(E) = \{\text{concat}(k, t) \mid k \in \text{dom}(E), t \in E(k)\}$$
- *Statistical Operators (Ω)*: Once we obtain the relevant contextual inputs (and features) using the relational and stateful operators, these final set of operators directly apply an external function f to either a relation or a sequence (e.g. applying a ML model over a relation) and produce a relation or a sequence. For instance, the decision tree regressor in Figure 6 is a statistical operator.

4.2 Lazy DAG runtime system

With our DAG framework, developers can publish many predefined and preconfigured workflows (PWs) to ease development of new experts. Intuitively, these are concrete subDAGs of interest. However, executing a large number of these workflows can pose a heavy computational cost. To manage the cost of the DAG framework, we design a runtime system that *lazily* executes the nodes in the DAG, only if they're needed.

We develop a parallel processing framework called lazy DAG. Concretely, this framework exposes decorator interfaces for marking a Python function as `lazy_task` and a class as `lazy_actor` (see Figure 7). When a lazy task or a method of a lazy actor is invoked, the method does not begin execution. Instead, the invocation returns immediately with a

future reference for result of the computation. This implicitly instantiates a DAG with minimal changes to existing code.

When a program explicitly calls `lazy_dag.get` on a reference, the lazy DAG backend begins its execution, recursively resolving arguments that are also references. Lazy executions allows developers to develop and publish an unlimited number of predefined workflows, without incurring runtime costs for unused node operations. Further, lazy execution allows us to support different data fetching modes such as push vs pull data model or asynchronous data updates. As an added benefit, existing monolithic RCA programs can also be expressed in MoCE. Taken together, this lazy DAG runtime is key for extensibility and scalability of MoCE.

The resultant data for each node in the DAG is cached by the runtime system so that subsequent accesses avoid duplicate computation. The user can optionally specify an upper limit on the size of the cache and the data eviction policy (Least Recently Used or Last In First Out).

4.3 Mixture policy

Given an anomaly incident and our library of experts (§ 5.2), we have two policy tasks (Figure 3):

- The *expert routing algorithm* "routes" a given anomaly incident to a relevant subset of experts. The selection of the experts can be based on the anomaly type (e.g. real-valued metric, success/failure metric), preliminary statistical analysis or user-provided prompts. In our current implementation, we employ a simple routing algorithm that invokes relevant experts based on the anomalous metric type (real-valued, binary success/failure), but note that more advanced methods can be employed.
- The *expert combining algorithm* combines the output of each expert to generate the final RCA leads. While there are many choices here such as voting, stacking [20] or a gating network [43], our current implementation uses a simple weighted voting scheme: each expert analyzes an incident and outputs a ranked list of potential root causes with confidence scores. The final score for each root cause is the normalized weighted sum of the confidence scores from all experts that identified it. The system then outputs the top-K root causes based on this aggregated score.

Calibrating experts in the mixture: Ideally, we want to report a small top-K set of leads to an analyst for validation. Expert false positives can harm the ensemble's precision at small K. Hence, we calibrate each expert's hyperparameters offline (using simulated examples) to prioritize a low false positive rate over a low false negative rate. The rationale here is that in an ensemble, other experts can still find the correct lead even if one expert misses it due to stricter calibration.

5 Implementation and setup

We built a prototype for MoCE using 6K lines of Python code, which includes (1) the DAG operators and workflows (2) library of experts and (3) the Lazy DAG runtime system that lazily executes the tasks and caches the results. We describe these components next.

5.1 DAG operators and workflows

For relational operators, we use standard SQL atop ClickHouse [1]. We develop our statistical operators using Python functions (e.g. via data transformations in Pandas). Since stateful operators can be expensive, we use sampled operations. For instance, in `Sequencify`, for converting a relation to a sequence, we sample n=5000 sessions that are context-aware (e.g. in the anomalous cohort, during the anomaly period, performed login). We choose n=5000 as we can handle that many sessions on a single machine while still yielding a reasonably accurate analysis. For stateful map, merge, filter, since the sequencify step outputs sampled sessions that can fit in-memory of a single server, our implementation of these operators is based on Python functions that operate directly on the input sequence. While sampling is a simplification, it suffices for our RCA experts as the computed statistics come close to their true values. As future work, we plan to incorporate more capable stateful event processing [3, 51].

Using these operators, we develop several reusable predefined workflows (§ 4.1): (1) *cohort statistics* (e.g. payment success rate) grouped by leaf-cohort attributes (recall that a leaf cohort is one which has values for all possible cohort attributes); (2) *url statistics in context filtered sessions*: feature vector for each sampled session comprising URL statistics (response times, failure rates) for all URLs used more than once; (3) *context-aware events grouped by sessions*: events during the relevant application flow (e.g. Login, Payment) in the session for each of the sampled sessions; and (4) *event patterns in context-filtered sessions*: user event sequences during the relevant flow of the session for each of the sampled sessions. All of our experts are based on one or more of these types of workflows.

5.2 Library of experts

We authored a diverse set of experts (Table 3 in appendix) to cover three different root cause types: services identified by URL endpoints, user sub-population cohort attributes, and event pattern sequences. We focus on these three since they cover most of the real-world incidents, based on feedback from field teams monitoring production services.

- **URL-centric experts (Experts 1-10)**: This category of experts focuses on the set of URL services for finding the root cause leads. These can reveal performance degradations in backend services, third party APIs and network endpoints which may have caused the anomalous KPI metric.
- **Cohort-centric experts (Experts 11, 12, 13)**: These experts target non-URL, cohort attributes like ISP, app version, or device type that define a user sub-population. These can reveal various kinds of issues that disproportionately affect a specific sub-population of users such as a network

issue in an ISP, a software bug in an app version, a backend infrastructure issue affecting an entire region etc.

- **Event pattern experts (Experts 14, 15)**: These experts analyze event sequences inside sessions to mine problematic user behavior sequences. This is useful for issues like broken user flows that are not visible in aggregated metrics.

The above experts use different algorithms: (1) FlowModel models the anomalous metric as a function of a pattern of URL accesses (e.g. login success rate as a function of success/failure of constituent URLs accessed for login) and uses regression to localize root causes; (2) Single entity maximum likelihood (MLE) models the distribution of metric values using a decision tree regressor and infers root causes via MLE; and (3) Differential diagnosis runs statistical tests comparing deviations in metric values between historical and current epochs. Together, these experts analyze incidents from multiple complementary perspectives, ranging from network and backend degradation to cohort-specific failures and event sequence patterns (see Appendix A for more details).

Baselines: Using our framework, we also implement state-of-the-art RCA baselines that cater to client-side telemetry:

- *Adtributor* [19] generates ranked cohorts (e.g. ISP A) that explain anomalies in aggregate metrics. To align with our setting, we restrict its output to single-entity cohorts and report the top-K candidates.
- *Minesweeper* [53] outputs ranked sequential patterns mined from exemplar traces. In our setup, we enumerate all subsequences of length up to 4 and report the top-K patterns based on scores, computed as in [53].
- *OpenRCA* [70], an LLM-based tool, employs two prompting modes: (1) *sampling-based*, where representative data samples are included in the prompt, and (2) *agent-based*, where the LLM is given only metadata and iteratively generates Python code to access the data during execution. In our setting, running the agent mode given data samples gave the best results, hence we use it as the baseline.

5.3 Lazy DAG implementation

The lazy DAG runtime system is implemented as a layer on top of the open-source Ray [7] framework. We chose Ray for its ability to schedule tasks and manage global objects using a distributed store.

The lazy DAG backend intercepts a LazyTask and defers its execution until a result is requested, which allows us to build the entire computation DAG before execution. Computation is triggered by a call to `lazy_dag.get(future_ref)`. The lazy DAG backend traverses the DAG from the target reference, recursively resolving dependencies. The runtime materializes (potentially) nested dependencies into concrete inputs before submitting a task to Ray.

Once resolved, independent tasks are scheduled in parallel across worker nodes. The result of each task is cached by Ray in its distributed in-memory object store and the reference

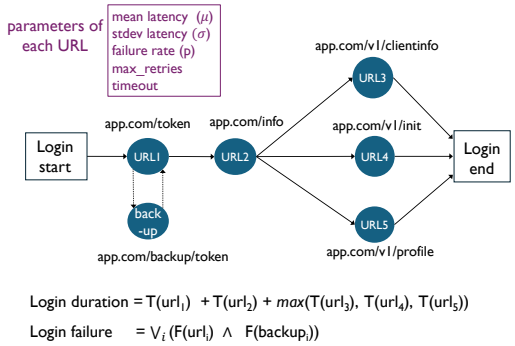


Figure 8: Modeling a "Login" process in the simulator

is returned to the lazy DAG layer. When another part of the DAG needs the same result, the lazy DAG reuses the reference to retrieve it from the object store instead of recomputing. A memory limit and eviction policy can be configured via Ray's APIs. As future work, we can add more sophisticated policies to trade off compute versus storage.

5.4 Incident Simulator

It is challenging to obtain large real-world datasets with labeled failure examples. To systematically evaluate MoCE and baselines, we build a simulator to generate events; this simulator mimics interactions in Internet-scale services (Figure 1) by modeling the four components (Client, third party services, network and backend):

- Client: We simulate multiple clients, each running some sequence of application flows (e.g. Login → Profile → Search → Play → Search), generated randomly via a state machine. Each flow is represented by a pattern graph of URL requests, as shown in Figure 8, where each request is defined by its URL, mean (latency), variance (latency), failure rate, max_retries and timeout_sec. We parameterize these by analyzing production data. A service is optionally assigned a backup service in case the primary URL fails.
- Network, third party services and backend: We model these components using a module that takes in all cohort attribute values, the time of day (to accommodate seasonality) and the URL request and returns a response latency and response success/failure. This allows us to inject failures and performance degradations in the simulator.

The telemetry data comprises user action, request, response, and timeout events from each client session. We recreated a diverse set of incidents in this simulator: (1) service failures by injecting high latency or request errors in a randomly chosen URL service, including failures in the backup service after the primary fails. (2) Cohort failures: by injecting high latency and request errors in a randomly chosen single-entity cohort (e.g. in sessions for ISP A) and (3) failures in the backend for a specific user-action sequence (e.g. repeated clicks). We generated 36 incidents in each category by varying the fraction of sessions affected by the failure (chosen uniformly at random between 0.1 and 1) and varying failure parameters

such as injected delay.

6 Evaluation

Evaluation metrics: We report **recall in top-K** (with K=5), the fraction of times the true root cause is among the top-K leads of a scheme. We also report **precision**, defined as $1/r$, where r is the rank of the true root cause in the ordered list of root cause leads. The rank r reflects the effort of the analysts before they arrive at the true root cause if they inspect the list of leads in rank order. If the true root cause does not appear in the list, we set precision to 0.

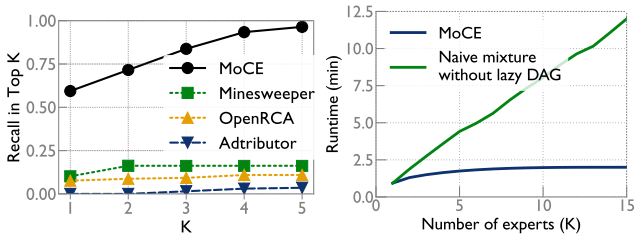
6.1 End-to-end evaluation

Evaluation goals: We want to show overall benefits of MoCE on our goals of coverage, extensibility and scalability. We provide detailed results on synthetic incidents and a pilot study on four large-scale internet media service applications. We will also break down the benefits for our three key ideas: Mixture, Context-aware RCA and the lazy DAG framework.

Coverage over diverse incidents: To evaluate coverage of incidents, we consider the diverse classes of incidents generated by our simulator encompassing diverse failure scenarios such as a faulty ISP, a slow backend service, a problematic pattern of user-events, a bug in an app version, etc. (§ 5.4). Figure 9a shows that MoCE achieves high recall (>95%), outperforming previous solutions (<35%) achieving high-coverage across all types of incidents. Adtributor and Minesweeper are good at finding certain classes of incidents but have poor overall recall across all incidents. The LLM-assisted approach has poor accuracy, consistent with earlier findings [70], primarily because RCA is a complex task that involves complex SQL queries, stateful processing and intricate algorithms. MoCE's recall on any individual incident class is within 6% of the best "hindsight optimal" scheme for that incident class. MoCE also achieved high precision ($> 80\%$) whereas all other schemes had poor precision $< 30\%$. Despite MoCE's good accuracy, we believe that there is room for improvement with better ensemble algorithms than our current weighted score (§ 4.3). We leave this for future work.

Upon inspecting incidents where MoCE's top 5 list did not include the true root cause, we identified two reasons. For some incidents, MoCE produced root causes entities on the causal chain from the ground truth to the symptom; e.g., in one scenario, a faulty app version resulted in high response latencies for all URL requests, MoCE's top choices included most used URLs. For some other incidents, MoCE's root cause was highly correlated with the ground truth; e.g., in one case, a certain device type was the true root cause but MoCE's top choices included a device OS that was primarily used by the faulty device. These examples suggest room for algorithmic improvements in our current experts and in MoCE's ensemble algorithm. We leave this for future work.

Extensibility of MoCE: To demonstrate extensibility, we introduce a new incident class, a new telemetry source, and a new expert for MoCE. While our implementation of MoCE and the simulator was designed for client-side and network-side telemetry, we introduce a new failure mode in the simulator– in the front-end Apache web-server. We integrated a real-world server log [4] as a new telemetry source, using its error spikes to trigger the incident. We develop a corresponding new expert to correlate these log errors with KPI degradations and added it to the existing DAG. MoCE successfully identified this new class of incidents ("log error" incident type in Figure 10). This experiment illustrates how easily MoCE can be extended to handle (1) a new, previously unforeseen incident class, (2) a new telemetry source, and (3) a new expert type based on the new telemetry.



(a) Accuracy vs other schemes **(b) Running time**

Figure 9: (a) MoCE achieves high Top-K recall ($> 95\%$) across diverse incidents far outperforming prior work ($< 25\%$). (b) As more experts are added, MoCE's Lazy DAG avoids duplicate computation reducing execution time by up to $6\times$ vs. naive implementation (estimated)

Scalability: We estimated compute costs for analyzing 30 TB of uncompressed telemetry data from a large-scale media application, representative of a large enterprise [9]. Using Snowflake's credit-based pricing [10], running MoCE once every 15 minutes on this data would result in an estimated yearly cost of approximately $7,000, a small fraction of the total observability cost itself assuming 30TB corresponds to one month of data. Overall, MoCE's analysis for this data took less than 2 minutes.

We also compare MoCE to a mixture-of-experts implementation that does not use our lazy DAG framework. First, we measure the actual runtime of every node in the DAG on production data. Then, using these per-node runtimes, we estimate the average runtime when using k experts by considering all subsets of experts of size k, and computing the average ensemble runtime over these subsets (including repeated work for the naive baseline). Overall, MoCE is 6× faster than the naive mixture-of-experts implementation without MoCE 's DAG framework (see Figure 9b).

6.2 Pilot study on production data

We conducted a pilot study on production data from a large application-level monitoring and analytics service provider for four large-scale media content applications. The provider gives its customers (who run internet-scale applications) fine-

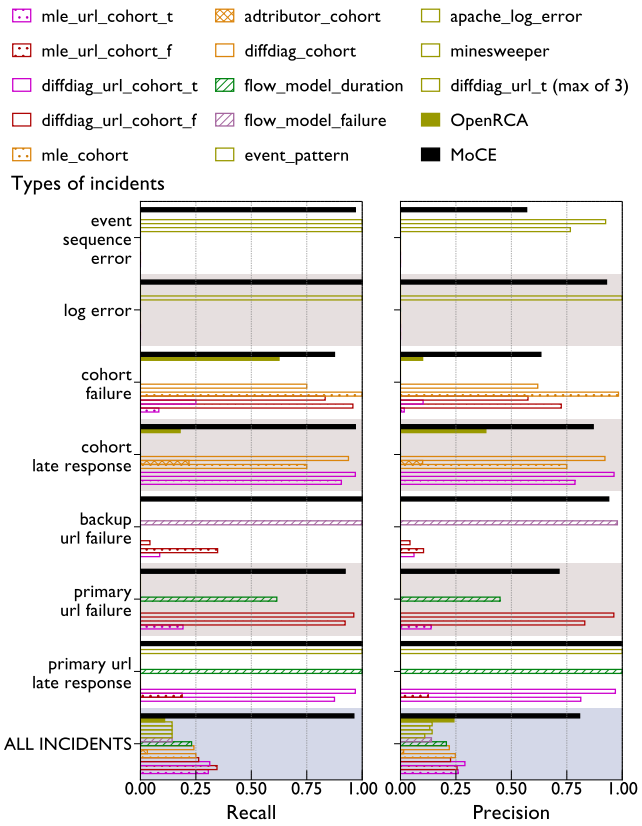


Figure 10: Evaluation on synthetic incidents: the y-axis lists the different categories of incidents generated. Each bar represents recall/precision for a RCA method, with the black bar representing MoCE. MoCE achieved $> 95\%$ recall across all incident types while individual schemes had $< 35\%$ recall. MoCE's recall was within 6% of the best scheme in hindsight for each incident category. MoCE also achieved high precision (80%) across all incidents whereas all other schemes achieved $< 30\%$.

grained visibility into their end users' digital experience allowing them to create custom KPIs/metrics (e.g., error, conversion, timing) to track business outcomes. The telemetry data comprise KPIs for experience metrics, events for URL request-responses, user actions and client-side events, along with client-side cohort information (e.g. device, region).

We ran a popular anomaly detection library [62] to flag anomalies in various KPIs for one week of production data for these applications. This resulted in 150 anomalies (avg. 2-3 per metric). Then we sampled 10 out of these anomalous incidents, discarding anomalies which affected only few users (<10). We ran our system for these 10 anomalies. For each incident, we manually identified the ground truth and asked a field team specialist to validate them using their existing tools and workflows. Table 2 shows these 10 incidents. We note that the identified ground truths for these incidents are of different types– URL, app version, device, region, etc. Overall, in 9/10 incidents, MoCE found the likely ground truth in its top 5 list.

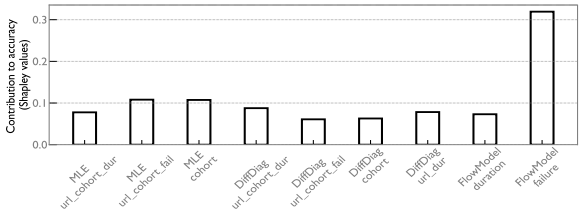


Figure 11: Breakdown of contribution by experts: the y-axis shows the value-add of an expert as measured by its Shapley contribution to overall accuracy (some experts are grouped into one category). As can be seen, each expert contributes to the overall accuracy of the ensemble.

6.3 Value of MoCE's design ideas

Value of mixture: We already saw the value of the mixture idea in MoCE across diverse incident types vs. individual experts (including prior work) that are good only for a narrow set of incident classes (Figure 10). Figure 11 shows the Shapley values of each expert on the simulated incidents dataset. The Shapley value of an expert E can be interpreted as the average marginal increase in RCA accuracy when E is added to an existing mixture with a random subset of experts. The figure shows that each expert uniquely improves the accuracy when added to a mixture model with a random set of experts, thus demonstrating the marginal utility of each expert.

Cost benefits of lazy DAG: We compare MoCE to a naive mixture baseline that runs each expert independently and combines their outputs with the same mixture algorithm, but without the DAG framework. By reusing intermediate DAG nodes, MoCE supports 15 experts using only 10 shared internal nodes (excluding 15 statistical leaf nodes), instead of 37 intermediate tasks if experts were written as separate monolithic programs for the baseline. This reuse reduces the resource costs by $5\times$ (using the methodology in § 6.1).

Robustness of ensemble (sensitivity analysis): To evaluate the mixture's robustness to false positives in constituent experts, we randomly corrupted experts by altering (a) the predicted root cause and (b) the confidence score of the root cause. We observed that even after corrupting 16% of the leads, MoCE maintained high recall >90%, indicating that the ensemble strategy is resilient to individual mistakes (figure not shown for brevity).

7 Related Work

We discussed a few closely related works earlier in §2. Here, we discuss a broader range of related works.

RCA techniques: Many approaches have been proposed including causal graph analysis (e.g., [16, 23, 46, 54]), counterfactual reasoning (e.g., [33, 40, 75]), provenance based event troubleshooting (e.g., [21, 38, 68, 69]), log mining (e.g., [71, 74]), as well as libraries of RCA algorithms (e.g., DoWhy [8] and PyRca [50]). While our current experts achieve high accuracy, our goal is not to develop better algorithms. Rather, MoCE provides a unifying abstraction and

Anomalous KPI	Root causes identified by MoCE and likely ground truth
Login success rate	MoCE's top root cause was a URL X/Y/Z/watch_history. We found that all URLs of the type X/Y/*/watch_history, in fact, had high failure rates during the anomalous period.
Login Verification failure rate	MoCE's one of the two (joint) top root causes was city A. On closer inspection, two Cities A and B had experienced sudden spikes in network request errors. MoCE's other top choice was a URL X/Y/otp which had errors coinciding with the anomaly, indicating it might be the URL service causing the issue.
Sessions with errors	MoCE's top root cause choice was an ISP which wasn't helpful. The next 3 root causes point to a vendor in some way (device, OS version, app) which was likely the true root cause.
Time till the user clicks on a button	MoCE's top choice, an ISP, wasn't helpful. The second root cause was device type A which was responsible for the spike. On further drilling, a specific device OS version and an app version were causing the issue. These were also the 3rd and 4th root causes produced by MoCE.
Authentication success rate	MoCE's top choice was City C which was experiencing high failure rates >85% and was primary cause of the spike.
Login verification failure rate	MoCE's top root cause was URL X/Y/otp had a high 400 error rate and was likely the true root cause. MoCE's second choice was one city which was much more affected than others.
Time from live to successful Digital Rights authentication	MoCE's top choice was TV + Android TV which was the cohort experiencing problems. It's second root cause was an app version which was the major app version rolled out for the Android TV users.
Spike in errors in displaying plans in the subscription page	MoCE's top root cause was an ISP that showed a high error rate but it was likely a coincidence. MoCE's second root cause was an app version which indeed was the cause of high error rates.
Search resulting in play conversion rate	MoCE's top root cause was a URL with higher response times during the anomaly period but the increase was insignificant (<0.2 sec). Other root causes were not helpful either (a state, a device OS, and a browser). Hence, this was deemed to be a false positive.

Table 2: Incidents from a pilot study in four large-scale media services. We asked a domain expert to validate these root causes. In 9/10 of these incidents, MoCE produced the likely true root causes in its Top 5 list.

extensible framework to accelerate the design and development of such RCA "experts".

Troubleshooting in other domains: Many domains need automated systems troubleshooting for detecting failures e.g. the network layer [13, 14, 16, 36, 39, 41, 42, 48, 52, 58, 61, 72], storage systems [73], backend infrastructure [33, 40, 45, 47, 75] and large software systems [26, 71]. MoCE's mixture framework is a promising approach for these fault diagnosis problems where point solutions lack sufficient coverage.

Data processing for observability: Modern observability data planes (e.g., Uber's M3 [66], Netflix's Mantis [17], TimescaleDB [56], VictoriaMetrics [65]) provide scalable streaming, efficient compression, and support for high-cardinality data. These can serve as the underlying data processing engines for our RCA-specific operators, complementary to MoCE.

Detection and alerting: Recent works on scalable alerting (e.g., [34,64]) are complementary: they can provide anomalies to MoCE, which can run RCA for those incidents. While we focus on RCA, the mixture-of-experts paradigm could also apply to anomaly detection. We leave this to future work.

Long-term analytics, remediation and mitigation: Existing work uses machine learning for incident routing (e.g., DeepTriage [59]), playbook construction (e.g., [49]), and automated remediation (e.g., [55, 63, 67, 76]). These capabilities can be modeled as downstream nodes in the DAG, enabling workflow reuse for automating tasks like ticketing or triggering remediation playbooks.

Telemetry for troubleshooting: There have been efforts to enable and standardize diverse telemetry sources such as distributed tracing (e.g., [12, 15, 30, 44, 57, 60]) and kernel-level events via eBPF (e.g., [31]). These can be readily plugged into MoCE for enhanced troubleshooting as MoCE is designed for extensibility.

8 Conclusion

Despite years of work, troubleshooting in Internet-scale services remains an open challenge. We revisit the problem from first principles and recast it from a systems lens: *enabling* analysts to express and run many RCA approaches at scale to capture the diversity of incidents, telemetry sources, and analysis techniques. To this end, we introduced an extensible mixture-of-context-aware expert framework, a succinct dataflow abstraction for expressing a broad array of experts, and the LazyDAG runtime to support this abstraction scalably. Our implementation is more expressive and performant than state-of-the-art approaches. That said, we are merely scratching the surface—we believe that MoCE's design ideas can serve as a *unifying* foundation for further innovation in RCA.

Acknowledgment

This work was partly supported by the NSF (Expeditions CCF-1918770, CAREER IIS-2028586, Medium IIS-1955883, Medium IIS-2403240, Medium IIS-2106961), NIH (1R01HL184139), CDC MInD program, Meta, and Dolby faculty gifts. We thank Klaudia Legutko and Jibin Zhan from Conviva for participating in our pilot study. We also thank Daying, Guangbin, Kailash, Gopal, Jose, and Chenchen at Conviva for various helps and their feedback.

References

[1] Clickhouse. https://clickhouse.com/.

[2] Conviva login experience. https://www.conviva.com/login-experience/.

[3] Kusto Query Language overview. https://learn.microsoft.com/en-us/kusto/query/?view=microsoft-fabric.

[4] Loghub log repositories. https://github.com/logpai/loghub.

[5] Permutation test. https://en.wikipedia.org/wiki/Permutation_test.

[6] Prophet. https://facebook.github.io/prophet/.

[7] Ray. https://www.ray.io/.

[8] Root Cause Analysis with DoWhy, an Open Source Python Library for Causal Machine Learning. https://www.pywhy.org/dowhy/v0.12/.

[9] Signoz price comparisons. https://github.com/SigNoz/signoz/wiki/Detailed-Pricing-comparison-of-observability-tools-with-a-calculator-spreadsheet.

[10] Snowflake warehouse pricing. https://docs.snowflake.com/en/user-guide/warehouses-overview.

[11] ttest. https://en.wikipedia.org/wiki/Student%27s_t-test.

[12] Zipkin. https://zipkin.io/.

[13] B. Arzani, S. Ciraci, L. Chamon, Y. Zhu, H. H. Liu, J. Padhye, B. T. Loo, and G. Outhred. 007: Democratically finding the cause of packet drops. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 419–435, Renton, WA, 2018. USENIX Association.

[14] B. Arzani, S. Ciraci, B. T. Loo, A. Schuster, and G. Outhred. Taking the blame game out of data centers operations with netpoirot. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 440–453, New York, NY, USA, 2016. ACM.

[15] S. Ashok, V. Harsh, B. Godfrey, R. Mittal, S. Parthasarathy, and L. Shwartz. Traceweaver: Distributed request tracing for microservices without application modification. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 828–842, New York, NY, USA, 2024. Association for Computing Machinery.

[16] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '07, pages 13–24, New York, NY, USA, 2007. ACM.

[17] A. Barker, I. Cox, et al. Mantis: Netflix's platform for building cost-effective, realtime, operations-focused stream processing applications. In *Proceedings of the VLDB Endowment (VLDB)*, 2020.

[18] R. Bhagwan, R. Kumar, R. Ramjee, G. Varghese, S. Mohapatra, H. Manoharan, and P. Shah. Adtributor: Revenue debugging in advertising systems. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 43–55, 2014.

[19] R. Bhagwan, R. Kumar, R. Ramjee, G. Varghese, S. Mohapatra, H. Manoharan, and P. Shah. Adtributor: Revenue debugging in advertising systems. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 43–55, 2014.

[20] N. Bosch, O. Shchur, N. Erickson, M. Bohlke-Schneider, and A. C. Turkmen. Multi-layer stack ensembles for time series forecasting. In *AutoML 2025 Methods Track*.

[21] A. Chen, Y. Wu, A. Haeberlen, W. Zhou, and B. T. Loo. The good, the bad, and the differences: Better network diagnostics with differential provenance. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 115–128, New York, NY, USA, 2016. Association for Computing Machinery.

[22] Q. A. Chen, H. Luo, S. Rosen, Z. M. Mao, K. Iyer, J. Hui, K. Sontineni, and K. Lau. Qoe doctor: Diagnosing mobile app qoe with automated ui control and cross-layer analysis. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, page 151–164, New York, NY, USA, 2014. Association for Computing Machinery.

[23] X. Chen, M. Zhang, Z. M. Mao, and V. Bahl. Automating network application dependency discovery: Experiences, limitations, and new solutions. In *OSDI*, January 2008.

[24] Y. Chen, H. Xie, M. Ma, Y. Kang, X. Gao, L. Shi, Y. Cao, X. Gao, H. Fan, M. Wen, et al. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 674–688, 2024.

[25] Y. Chen, H. Xie, M. Ma, Y. Kang, X. Gao, L. Shi, Y. Cao, X. Gao, H. Fan, M. Wen, J. Zeng, S. Ghosh, X. Zhang, C. Zhang, Q. Lin, S. Rajmohan, D. Zhang, and T. Xu. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the 2024 European Conference on Computer Systems (EuroSys '24)*, pages 674–688, 2024.

[26] W. Cui, X. Ge, B. Kasikci, B. Niu, U. Sharma, R. Wang, and I. Yun. REPT: Reverse debugging of failures in deployed software. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 17–32, Carlsbad, CA, Oct. 2018. USENIX Association.

[27] J. Dean and L. A. Barroso. The tail at scale. *Communications of the ACM*, 56:74–80, 2013.

[28] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the impact of video quality on user engagement. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, page 362–373, New York, NY, USA, 2011. Association for Computing Machinery.

[29] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the impact of video quality on user engagement. *ACM SIGCOMM computer communication review*, 41(4):362–373, 2011.

[30] DynaTrace. DynaTrace. https://www.dynatrace.com/.

[31] eBPF. ebpf. https://ebpf.io/case-studies/.

[32] D. Fisher. Observability and the misleading promise of aiops. https://thenewstack.io/observability-and-the-misleading-promise-of-aiops/.

[33] Y. Gan, M. Liang, S. Dev, D. Lo, and C. Delimitrou. Sage: practical and scalable ml-driven performance debugging in microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 135–151, 2021.

[34] J. Gao, X. Song, Q. Wen, P. Wang, L. Sun, and H. Xu. Robusttad: Robust time series anomaly detection via decomposition and convolutional neural networks. *arXiv preprint arXiv:2002.09545*, 2020.

[35] J. Gao, N. Yaseen, R. MacDavid, F. V. Frujeri, V. Liu, R. Bianchini, R. Aditya, X. Wang, H. Lee, D. Maltz, et al. Scouts: Improving the diagnosis process through domain-customized incident routing. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 253–269, 2020.

[36] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, and A. Vahdat. {SIMON}: A simple and scalable method for sensing, inference and measurement in data center networks. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pages 549–564, 2019.

[37] P. Hamadanian, B. Arzani, S. Fouladi, S. K. R. Kakarla, R. Fonseca, D. Billor, A. Cheema, E. Nkposong, and R. Chandra. A holistic view of ai-driven network incident management. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, HotNets '23, page 180–188, New York, NY, USA, 2023. Association for Computing Machinery.

[38] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown. I know what your packet did last hop: Using packet histories to troubleshoot networks. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 71–85, Seattle, WA, Apr. 2014. USENIX Association.

[39] V. Harsh, T. Meng, K. Agrawal, and P. B. Godfrey. Flock: Accurate network fault localization at scale. *Proc. ACM Netw.*, 1(CoNEXT1), July 2023.

[40] V. Harsh, W. Zhou, S. Ashok, R. N. Mysore, B. Godfrey, and S. Banerjee. Murphy: Performance diagnosis of distributed cloud applications. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 438–451, 2023.

[41] H. Herodotou, B. Ding, S. Balakrishnan, G. Outhred, and P. Fitter. Scalable near real-time failure localization of data center networks. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 1689–1698, New York, NY, USA, 2014. ACM.

[42] P. Huang, C. Guo, L. Zhou, J. R. Lorch, Y. Dang, M. Chintalapati, and R. Yao. Gray failure: The achilles' heel of cloud-scale systems. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, pages 150–155, 2017.

[43] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[44] Jaeger. Jaeger. https://www.jaegertracing.io/.

[45] V. Jeyakumar, O. Madani, A. Parandeh, A. Kulshreshtha, W. Zeng, and N. Yadav. Explainit! – a declarative root-cause analysis engine for time series data. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 333–348, New York, NY, USA, 2019. Association for Computing Machinery.

[46] S. Kandula, D. Katabi, and J.-P. Vasseur. Shrink: A tool for failure diagnosis in ip networks. In *Proceedings of the 2005 ACM SIGCOMM Workshop on Mining Network Data*, MineNet '05, pages 173–178, New York, NY, USA, 2005. ACM.

[47] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl. Detailed diagnosis in enterprise networks. *SIGCOMM Comput. Commun. Rev.*, 39(4):243–254, Aug. 2009.

[48] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren. Detection and localization of network black holes. In *Proceedings of the IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, pages 2180–2188, Washington, DC, USA, 2007. IEEE Computer Society.

[49] R. Kremer, P. N. Wudali, S. Momiyama, T. Araki, Y. Elovici, and S. Asaf. Ic-secure: Intelligent system for assisting security experts in generating playbooks for automated incident response. *arXiv preprint arXiv:2311.03825*, 2023.

[50] C. Liu, W. Yang, H. Mittal, M. Singh, D. Sahoo, and S. C. H. Hoi. Pyrca: A library for metric-based root cause analysis. *CoRR*, abs/2306.11417, 2023.

[51] H. Milner, Y. Cheng, J. Zhan, H. Zhang, V. Sekar, J. Jiang, and I. Stoica. Raising the level of abstraction for time-state analytics with the timeline framework. In *CIDR*, 2023.

[52] E. C. Molero, S. Vissicchio, and L. Vanbever. Fast in-network gray failure detection for isps. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 677–692, 2022.

[53] V. Murali, E. Yao, U. Mathur, and S. Chandra. Scalable statistical root cause analysis on app telemetry. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 288–297. IEEE, 2021.

[54] R. N. Mysore, R. Mahajan, A. Vahdat, and G. Varghese. Gestalt: Fast, unified fault localization for networked systems. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 255–267, Philadelphia, PA, 2014. USENIX Association.

[55] P. Namyar, A. Ghavidel, D. Crankshaw, D. S. Berger, K. Hsieh, S. Kandula, R. Govindan, and B. Arzani. Enhancing network failure mitigation with {Performance-Aware} ranking. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 335–357, 2025.

[56] M. Nugent, M. Freedman, and A. Kulkarni. Timescaledb: Sql made scalable for time-series data. In *Proceedings of the VLDB Endowment (VLDB)*, 2018.

[57] OpenTelemetry. OpenTelemetry. https://opentelemetry.io/.

[58] Y. Peng, J. Yang, C. Wu, C. Guo, C. Hu, and Z. Li. detector: a topology-aware monitoring system for data center networks. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 55–68, Santa Clara, CA, 2017. USENIX Association.

[59] P. Pham, V. Jain, L. Dauterman, J. Ormont, and N. Jain. Deeptriage: Automated transfer assistance for incidents in cloud services. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery Data Mining (KDD '20)*, pages 3281–3289, 2020.

[60] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. Technical report, Google, Inc., 2010.

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

[62] S. J. Taylor and B. Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

[63] M. S. Team. Automate threat response with playbooks in microsoft sentinel. https://learn.microsoft.com/en-us/azure/sentinel/automation/automate-responses-with-playbooks, 2025.

[64] U. O. Team. Observability at scale: Building uber's alerting ecosystem. https://www.uber.com/blog/observability-at-scale/, 2018.

[65] V. Team. Victoriametrics: Fast, cost-effective monitoring solution and time series database, 2018. Open source project.

[66] R. Wilkes et al. M3: Uber's open source, large-scale metrics platform for prometheus. In *Proceedings of the VLDB Endowment (VLDB)*, 2020. Uber Engineering Blog / Open Source Project.

[67] X. Wu, D. Turner, C.-C. Chen, D. A. Maltz, X. Yang, L. Yuan, and M. Zhang. Netpilot: automating datacenter network failure mitigation. *SIGCOMM Comput. Commun. Rev.*, 42(4):419–430, Aug. 2012.

[68] Y. Wu, A. Chen, and L. T. X. Phan. Zeno: Diagnosing performance problems with temporal provenance. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 395–420, Boston, MA, Feb. 2019. USENIX Association.

[69] Y. Wu, M. Zhao, A. Haeberlen, W. Zhou, and B. T. Loo. Diagnosing missing events in distributed systems with negative provenance. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 383–394, New York, NY, USA, 2014. Association for Computing Machinery.

[70] J. Xu, Q. Zhang, Z. Zhong, S. He, C. Zhang, Q. Lin, D. Pei, P. He, D. Zhang, and Q. Zhang. Openrca: Can large language models locate the root cause of software failures? In *The Thirteenth International Conference on Learning Representations*, 2025.

[71] D. Yuan, S. Park, P. Huang, Y. Liu, M. M. Lee, X. Tang, Y. Zhou, and S. Savage. Be conservative: enhancing failure diagnosis with proactive logging. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, page 293–306, USA, 2012. USENIX Association.

[72] H. Zeng, R. Mahajan, N. McKeown, G. Varghese, L. Yuan, and M. Zhang. Measuring and troubleshooting large operational multipath networks with gray box testing. Technical Report MSR-TR-2015-55, June 2015.

[73] Q. Zhang, G. Yu, C. Guo, Y. Dang, N. Swanson, X. Yang, R. Yao, M. Chintalapati, A. Krishnamurthy, and T. Anderson. Deepview: Virtual disk failure diagnosis and pattern detection for azure. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 519–532, Renton, WA, Apr. 2018. USENIX Association.

[74] S. Zhang, Y. Liu, W. Meng, Z. Luo, J. Bu, S. Yang, P. Liang, D. Pei, J. Xu, Y. Zhang, et al. Prefix: Switch failure prediction in datacenter networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(1):1–29, 2018.

[75] Y. Zhou, C. Sun, H. H. Liu, R. Miao, S. Bai, B. Li, Z. Zheng, L. Zhu, Z. Shen, Y. Xi, P. Zhang, D. Cai, M. Zhang, and M. Xu. Flow event telemetry on programmable data plane. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 76–89, New York, NY, USA, 2020. Association for Computing Machinery.

[76] D. Zhuo, M. Ghobadi, R. Mahajan, K.-T. Förster, A. Krishnamurthy, and T. Anderson. Understanding and mitigating packet corruption in data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 362–375, New York, NY, USA, 2017. ACM.

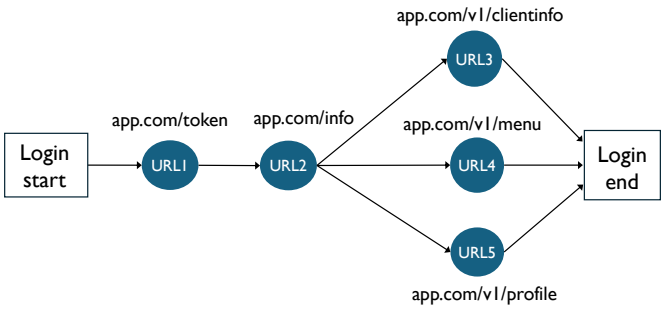


Figure 12: An application flow is modeled as a DAG of URL accesses

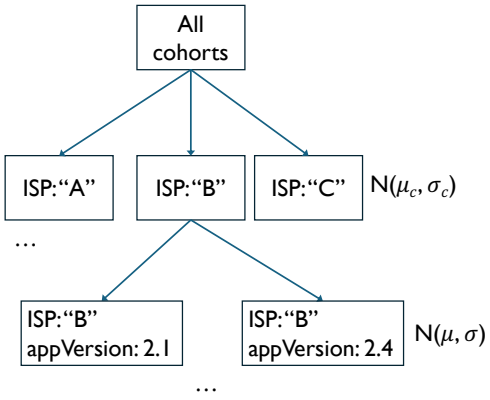


Figure 13: Decision Tree Regressor

A Appendix: Algorithms

All of our experts (Table 3) are based on one of the following algorithms–

A.1 Flow Model

The goal of this RCA algorithm is to predict the URL that can best explain the discrepancy in the metric of interest M between the anomaly time period and the historical time period.

We first model the application events within a *session-flow* relevant to M, such as Login, Payment, etc., as a pattern of network requests made to the backend application (see Figure 12). Specifically, we model the events in the flow as a DAG of network requests to the backend. The algorithm models the anomaly metric (M) as an (unknown) linear function of the network request URL u_i's statistic $f(u_i)$ (latency or failure rate depending on M). We have,

$$M = \beta_0 + \sum_i \alpha_i f(u_i) \tag{1}$$

Where the coefficients α_i and the constant β_0 are unknown.

The intuition behind the above model is that during a failure incident, network requests belonging to the failed URL will dominate the metric and hence a linear sum is a good approximation. We validated the same in our production data and found a high R^2 score for prediction using this model.

RCA algorithm: we first learn the coefficients α_i's using data from during the (anomaly + historical) time period using lasso regression with L1 penalty. We define the contribution of a URL u_i in the anomaly time period as $\alpha_i(\overline{f_A}(u_i) - \overline{f_H}(u_i))$, where $\overline{f_A}(u_i)$ and $\overline{f_H}(u_i)$ is the average value of $f(u_i)$ in the anomaly time period and historical time interval respectively. We then find the URL whose contribution explains most of the increase in M. That is,

$$\alpha_i(\overline{f_A}(u_i) - \overline{f_H}(u_i)) > \alpha * (\overline{M_A} - \overline{M_H}) \tag{2}$$

where $\overline{M_A}$ and $\overline{M_H}$ are the average value of M during the anomaly time period and historical time period respectively. α is a constant < 1. Intuitively, $\sim 1/\alpha$ URLs can satisfy Eq (2) assuming $f(u_i)'s$ does not decrease during the anomaly period. Hence, we set $\alpha = 0.2$ for getting $K = 5$ root cause leads. We set the confidence scores to be the contribution scores for each URL, as defined above, normalized so that the confidence scores of the root cause leads sum to 1. Finally, if the R^2 score for the linear model (Eq (1)) < 0.5, we return no root causes as the linear model is not deemed to be reliable.

We have described the case for metrics where lower value is better. The case for metrics where higher value is better (e.g. play time) is handled analogously with the difference that we would be looking to explain decrease in M.

A.2 Single entity diagnosis via Decision Tree MLE

The goal of this algorithm is to output a single entity (e.g. ISP A, device model B, application version 1.1.1) that best explains the discrepancy in a metric value of interest M in the current anomaly time period compared to the historical period before the anomaly. We will use this algorithm to write multiple experts by varying (a) set of entities to analyze– cohort attributes (ISP, appVersion etc.) or backend URLs and (b) varying the metric of interest– an experience metric or the average latency/failure rate of a backend URL service.

Choosing a predictive model: The algorithm first builds a predictive model $P[M|C]$ to model the distribution of the metric of interest M for each leaf cohort C using data from recent history (recall a leaf cohort has all attributes present§ **??**). We choose a historical period of t=5 hours before the anomaly which we found to be a good tradeoff to get enough sample points but also being temporally close to the anomaly period. Ideally, we could learn a separate model for every leaf cohort using past data. However, many leaf cohorts will have few sessions or even no sessions in the historical time period which make learning distributions separately for each leaf cohort

Expert	Input data	Hypothesis space	Algorithm
Experts 1,2 (Figure 14a)	Aggregate URL duration (1)/failure rate (2) for context-filtered URLs in the anomalous cohort during the (historical + anomaly) period	network request URLs to the backend or third-party services	Flow Model
Experts 3-8 (Figure 14b)	Average URL statistics (3-5: duration, 6-8: failure rate) for context-filtered URLs in the anomalous cohort during the (historical + anomaly) period	network request URLs to the backend or third-party services	Differential diagnosis (for mean, 75%ile, 95%ile statistics)
Expert 9, 10 (Figure 14c)	Average URL statistics (2: duration, 3: failure rate) for context-filtered URLs for all leaf cohorts during the (historical + anomaly) period	network request URLs to the backend or third-party services or single non-URL attribute	Single entity MLE
Expert 11 (Figure 14d)	Aggregate metric values for leaf cohorts during the (historical + anomaly) period	single non-URL attribute (e.g. ISP 'A')	Differential diagnosis
Expert 12 (Figure 14e)	Aggregate metric values for leaf cohorts during the (historical + anomaly) period	single non-URL attribute	Single entity MLE
Expert 13 (Figure 14f)	Aggregate metric values for leaf cohorts during the (historical + anomaly) period	single non-URL attribute	Adtributor
Expert 14 (Figure 14g)	ngram event sequences in the anomaly period for normal and faulty sessions	contiguous event sequences	Statistical
Expert 15 (Figure 14h)	(conditional: conversion metrics) Session event sequences for the anomalous cohort during the (historical + anomaly) period	event-name patterns (ordered subsequences)	Minesweeper

Table 3: Expert library

hard. Hence, we employ a decision tree regressor to estimate $P[M|C]$ (see Figure 13). The decision tree allows the model to work with coarser-grained cohorts (i.e. non-leaf cohorts grouped by a strict subset of all attributes) if leaf cohorts don't have enough data points.

Learning algorithm: Each node in the decision-tree represents a cohort C. For each cohort node, the algorithm finds the best fit distribution from a suitable distribution family for metric values filtered for that cohort. We use the Normal distribution family for continuous duration metrics and Binomial distribution family for failure metrics. The decision tree is learned recursively, finding the cohort attribute to split a node on that results in the best fit after splitting. When a max depth is reached, that node is not split further (max depth=2 in our implementation).

Prediction algorithm for a cohort: To predict probability for a metric value v for a fine-grained cohort C, we simply find the leaf node l that is a superset of the attribute values in C and sample from the distribution of the node l. For instance, in Figure 13, the probability of a cohort $C = \{ISP : B, appVersion : 2.4, state : IL\}$ having a metric value v is given by $P[M = v|l]$ where $P[M|l]$ is the distribution learned by the leaf node $l = \{ISP : B, appVersion : 2.4\}$.

RCA algorithm: the goal of the RCA algorithm is to produce entity C^* for a single cohort attribute that can best explain the discrepancy between the metric values in the anomaly period compared to recent history. By a slight abuse of notation, we denote the single entity cohort represented by C also as C. For instance, C could be the pair (ISP, B).

In our decision tree model, to find the root cause entity, we want to find the single entity cohort for which m deviates the most from its learned distribution. We formulate this as a Maximum Likelihood Estimator (MLE) problem, writing out the probabilities using a biased distribution P_c for candidate hypothesis $H_C = \{C\}$. Let $D = (m_1, m_2 ... m_n)$ be individual metric values observed in the anomalous period for the anomalous cohort. Define

$$P[D|H_C] = P[m_1|H_C]P[m_2|H_C]...P[m_n|H_C]$$

For a Hypothesis H_C, for each node v in the decision tree that is a subset of C, we define a biased distribution P'_v for v representing a distribution of metric values assuming C has failed. At a high level, the above probability should change according to the biased distribution when C is faulty (that is, if H_C is the true root cause hypothesis). For Normal distributions, we bias the distribution by setting the biased mean 3 standard deviations away from the unbiased mean, while keeping the standard deviation unchanged. For Binomial distributions, we set the biased failure probability as the midpoint of p and 1 if p is the unbiased failure probability.

$$P[m_i|H_C] = \begin{cases} P'[m_i|C] & \text{if } m_i \in C \\ P[m_i|C] & \text{otherwise} \end{cases}$$

Where $m_i \in C$ implies that the value m_i comes from a session in C. The MLE is then given by,

$$C^* = \underset{C \subseteq \text{1-entity-cohorts}}{\arg\max} P[D|H_C]$$

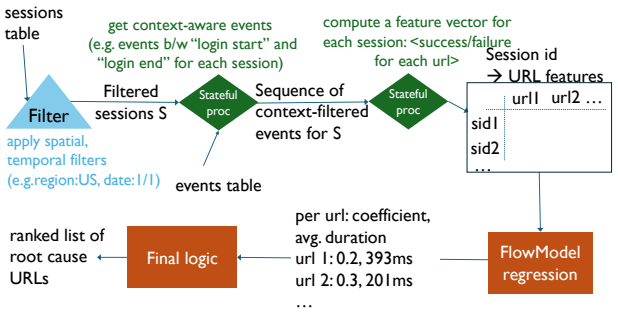
1-entity-cohorts is the set of all cohorts with a single attribute present. Finally, we employ a small optimization to the above algorithm. Since D could potentially represent a large set of values, we reduce D into a single weighted value (v_l, w_l) for each leaf cohort l, where v_l is the average value of the metric in cohort l and w_l is the total number of metric samples in l, representing weight of v_l. We use this reduced

D, which can be obtained using standard group by database queries, to perform the MLE.

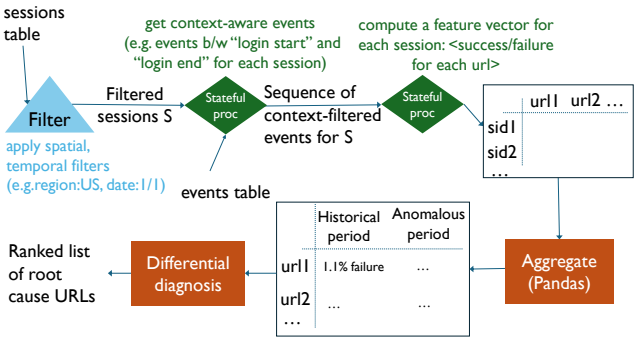
A.3 Differential Diagnosis

The goal of differential diagnosis is to determine if two sequences of random samples come from the same distribution. The samples can be optionally weighted by their frequencies. In the context of RCA, we will define experts based on differential diagnosis that check whether a certain metric is higher or lower in the anomaly time period compared to previous epochs or whether a metric is higher in one cohort and lower in another.

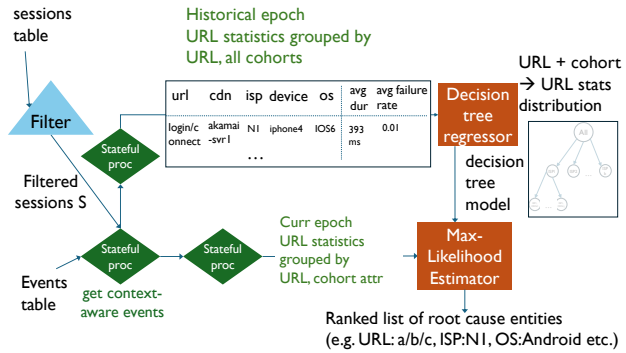
To perform differential diagnosis, we employ standard statistical procedures– (a) testing mean and variance using tTests [11] and (b) testing higher percentiles of the distribution using permutation tests [5].



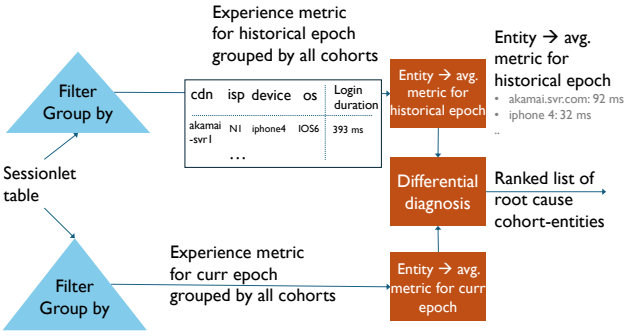
(a) Expert 1,2: URL flow model



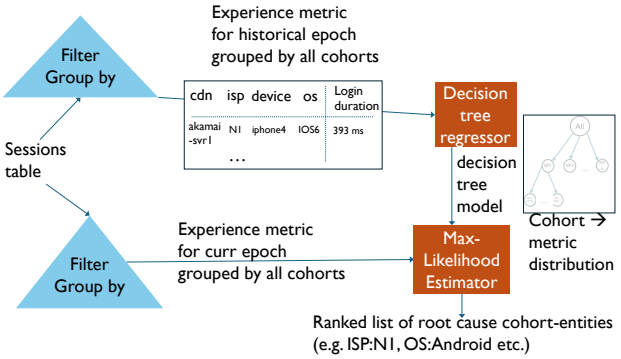
(b) Experts 3-8: Differential diagnosis on URL statistics



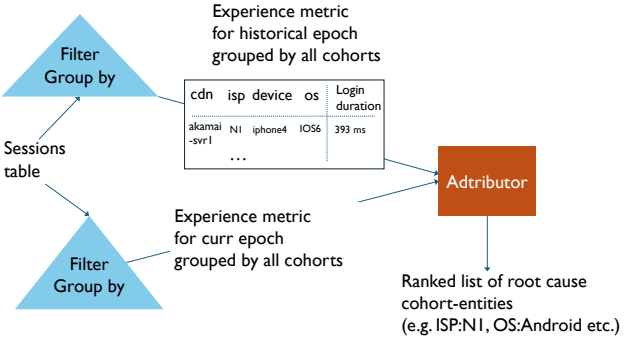
(c) Experts 9,10: MLE on (URL + cohort) summary statistics



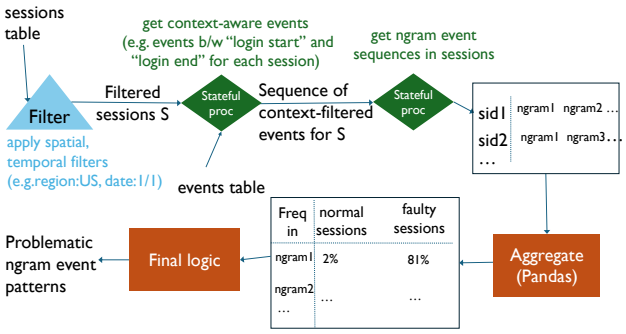
(d) Expert 11: Differential diagnosis on cohort statistics



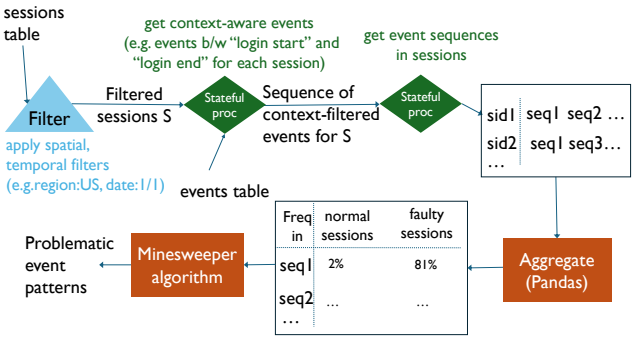
(e) Expert 12: MLE on cohort statistics



(f) Expert 13: Adtributor



(g) Expert 14: Ngram event pattern



(h) Expert 15: Minesweeper event pattern

Figure 14: Experts used in our mixture model written as DAG of operators.