STATEBUNDLE: CAUSE-CENTERED EVIDENCE SELECTION FOR AGENTIC DATA-CENTER OPERATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Autonomous data-center operations require agents to interpret heterogeneous telemetry and manage faults that propagate across physical facilities and the IT stack. In modern operational environments, redundant and fragmented telemetry complicates reasoning while increasing context and computational costs. We propose STATEBUNDLE, a hierarchical, cause-centered evidence-selection method that constructs compact observation bundles for operational agents. STATEBUNDLE selects nonredundant evidence anchors and retrieves corroborating observations through contrastive cross-modal alignment, hierarchical root-cause and propagation supervision, and task-oriented evidence-selection objectives. To enable controlled cross-layer evaluation, we further introduce DC-BENCH, a deterministic simulator-backed benchmark that couples physical facility with IT dynamics. The benchmark contains 72 task instances spanning 18 fault mechanisms and four operational objectives. Across ten evaluated agent systems, STATEBUNDLE reduces total model-token usage per episode by 37.4–80.9% relative to full-canonical observations, while improving task success for evaluated agents.

1 INTRODUCTION

Data centers underpin modern digital services and are becoming increasingly critical to large-scale AI workloads. The rapid growth of AI infrastructure has further increased their scale and hardware density; for example, production LLM training systems can coordinate more than 10,000 accelerators (Jiang et al., 2024b; Grattafiori et al., 2024). As these systems grow in scale and interdependence, faults can propagate across longer dependency chains and manifest through increasingly distributed telemetry, making it harder to identify their causes and assess their operational impact. At the same time, the larger blast radius of failures increases the cost of delayed or inappropriate interventions. Operating modern data centers reliably therefore becomes increasingly important and challenging. LLM-based agents offer a promising approach to augmenting human operators and automating parts of the observe–diagnose–act–verify loop (Chen et al., 2025b; Jha et al., 2025; Chen et al., 2025a).

However, the complexity of modern data-center environments also poses significant challenges for agentic operation. In particular, operational state is fragmented across logs, metrics, alerts, traces, configuration and deployment records, entities, and time windows (Lee et al., 2023; Chen et al., 2025b). Root-cause evidence is therefore distributed across observations and interspersed with noise rather than being co-located(Chen et al., 2024). This creates two bottlenecks. First, passing large volumes of telemetry to the agent consumes context and computation(Wang et al., 2024; Jiang et al., 2024a), while language models may overlook relevant evidence buried in long inputs (Liu et al., 2024). Second, agents must coordinate and reconcile observations related to the root cause, increasing the length of the reasoning process.

To address this representation bottleneck, we propose STATEBUNDLE, a hierarchical, cause-centered evidence-selection method for agentic data-center operations. It first uses contrastive cross-modal alignment to learn semantics and relationships across heterogeneous observations. Using root-cause and subsystem propagation-footprint supervision, STATEBUNDLE then learns incident- and subsystem-level representations. Task-oriented evidence-selection objectives further enable the model to distinguish root-cause-relevant observations from irrelevant or redundant telemetry. At inference time, it first selects a token-budgeted set of root-cause-relevant but nonredundant anchors

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

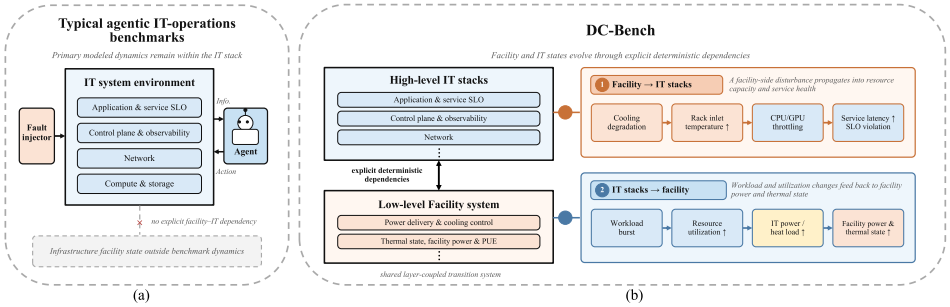

Figure 1: Comparison between typical agentic IT-operations benchmarks and DC-BENCH. (a) Existing benchmarks primarily model dynamics within the IT stack, leaving facility states outside the benchmark dynamics. (b) DC-BENCH couples facility and IT states through explicit deterministic dependencies; two concrete examples illustrate bidirectional cross-layer interactions from facility to IT and from IT workloads to facility power and thermal states.

and then augments each anchor with corroborating observations. The resulting bundle provides compact, structured multimodal evidence with multimodal root cause related evidence aggregated.

To evaluate the performance of STATEBUNDLE, a controlled environment that captures cross-layer data-center dynamics is essential. More broadly, realizing the potential of AI agents to automate critical data-center tasks requires reliable methods for measuring the effectiveness of proposed solutions (Jha et al., 2025). Recent benchmarks provide live environments for agents in cloud and IT operations (Chen et al., 2025b; Jha et al., 2025; Clark et al., 2026), however, their published settings largely center on cloud-native applications, Kubernetes, and operating systems. Although SREGym (Clark et al., 2026) includes hardware-level faults, it remains confined to the cloud-native computing stack. They overlook the modeling of facility infrastructure and the coupling of facility to high-level IT resources. As high-density AI infrastructure makes power and cooling first-order operational constraints (Shehabi et al., 2024; Go et al., 2025; Stojkovic et al., 2025), existing benchmarks provide only a partial view of the operational complexity that agents must handle in modern data centers.

To address these limitations, we introduce DC-BENCH, a cross-layer simulator-backed benchmark for agentic data-center operations. The benchmark explicitly models facility infrastructure together with high-level stacks, and represents facility and IT states within a shared dynamical system. Consequently, facility-side disturbances can affect IT resources and service performance, while workload and resource-utilization changes can feed back to facility power and thermal states through explicit deterministic dependencies. Figure 1 illustrates these bidirectional cross-layer interactions. The released DC-BENCH suite contains 72 task instances spanning 18 injected fault mechanisms and four objectives: detection, localization, root-cause analysis, and mitigation.

Our contributions are as follows:

- **Cause-centered evidence selection.** We propose STATEBUNDLE, which transforms noisy and redundant data-center telemetry into token-budgeted bundles of complementary evidence anchors and corroborating observations.
- **Cross-layer operations benchmark.** We introduce DC-BENCH, a layer-coupled data-center operations benchmark that captures interactions across facility, compute, network, control-plane, and application layers.
- **Empirical evaluation and failure analysis.** Experiments show that STATEBUNDLE substantially reduces token usage across all evaluated agents while improving observed aggregate success. Through a systematic analysis of agent outcomes and failure cases, we identify recurring challenges in agentic data-center operations and highlight directions for future research.

2 Related Work

2.1 Compact Representations for Operational Telemetry

Prior work reduces telemetry complexity at several levels. Log parsers such as Drain replace variable-rich messages with templates, reducing within-channel syntactic redundancy (He et al., 2017). Multimodal diagnostic methods then integrate signals across channels: DiagFusion embeds multimodal service events and reasons over dependency graphs (Zhang et al., 2023); Eadro jointly performs anomaly detection and root-cause localization from multi-source microservice data (Lee et al., 2023); and Nezha maps logs, metrics, and traces into homogeneous events and mines event graphs for fine-grained root-cause analysis (Yu et al., 2023). These systems learn model-internal representations for task-specific diagnosis, but do not construct an agent-facing, token-budgeted evidence state. In parallel, prompt-compression methods such as LLMLingua prune redundant prompt tokens (Jiang et al., 2023), while RECOMP extracts or summarizes retrieved documents to preserve downstream language-model utility under shorter contexts (Xu et al., 2024). Such methods operate on generic text rather than causal, typed operational observations. StateBundle combines these directions: it learns incident- and subsystem-level relevance from root-cause and propagation-footprint supervision, separates relevance from symptom aspect, and returns canonical observations as complementary anchors with corroborating support under a token budget. This preserves telemetry provenance and structure while optimizing the representation for closed-loop agent reasoning rather than standalone diagnosis or generic text compression.

2.2 Benchmarks for Agentic Data-Center and IT Operations

Representative data-center simulators such as CloudSim and GreenCloud model resource provisioning and energy-aware compute and network behavior, but are designed to study systems and resource-management policies rather than agent-facing incident handling (Calheiros et al., 2011; Kliazovich et al., 2012). Traditional AIOps evaluations likewise largely assess task-specific models on collected telemetry (Notaro et al., 2021). Recent work instead evaluates tool-using agents in executable environments. AIOpsLab orchestrates cloud deployments, workloads, fault injection, telemetry, and agent actions across the incident lifecycle (Chen et al., 2025b); ITBench broadens evaluation to SRE, security and compliance, and FinOps scenarios (Jha et al., 2025); and SRE-Gym introduces live cloud-native scenarios with cross-stack faults, ambient noise, metastability, and correlated failures (Clark et al., 2026). Cloud-OpsBench takes a complementary reproducibility-oriented approach by replaying immutable Kubernetes state snapshots through mocked operational tools for agentic root-cause analysis (Wang et al., 2026). These benchmarks establish interactive evaluation for cloud and IT operations, but remain primarily focused on the software and IT stack, without explicitly coupling facility-level power and thermal dynamics with higher-level system states in a shared incident transition model. This limitation becomes increasingly important for AI-oriented data centers: the growing scale and density of accelerator-based infrastructure impose substantially greater power-delivery and thermal-management demands (Meng et al., 2025; Cho & Moon, 2025; Colangelo et al., 2026). By contrast, DC-Bench extends agentic operations evaluation beyond the software stack by modeling cross-layer data-center dynamics and explicitly coupling facility-level thermal and power conditions with higher-level compute, storage, network, control-plane, observability, and application behavior.

3 StateBundle

StateBundle uses relevance scores between high-level root-cause representations and observation-level representations to select root-cause-relevant anchors, thereby retaining pivotal observations while filtering out noisy and irrelevant telemetry. It then retrieves corroborating evidence for each anchor according to feature similarity in the learned aspect space, strengthening the evidential support for root-cause reasoning and aggregating correlated observations. StateBundle adopts a three-stage training strategy. The three stages respectively pretrain the channel encoders and align multimodal features, learn global and subsystem-level representations, and optimize root-cause relevance scoring and evidence selection. The overall framework of StateBundle is illustrated in Fig. 2.

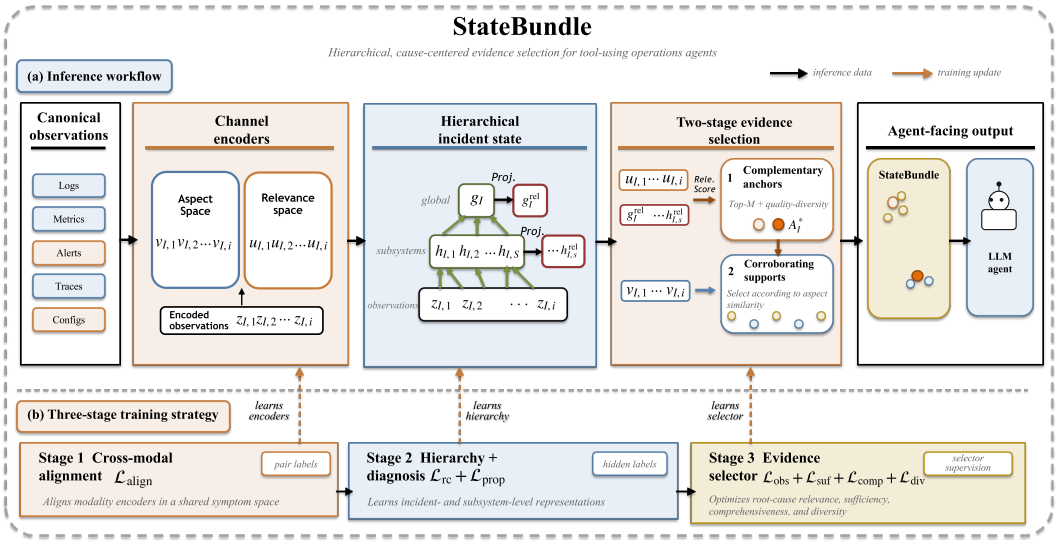

Figure 2: Framework of StateBundle

3.1 Root-cause Anchors and Corroborating Observations Selection

Modality encoding. Each canonical observation $o_{I,i}$, whose construction is detailed in Appendix A, is encoded independently and projected into the shared observation space:

$$\mathbf{z}_{I,i} = \mathrm{Norm}\big(P_{c_{I,i}} f_{c_{I,i}}(o_{I,i})\big) \in \mathbb{R}^d. \tag{1}$$

The released encoders combine a shared trainable hashed-token text embedding, a channel-specific fixed-width numeric projection, and a shared encoding of observable metadata. The metric encoder additionally applies a gated recurrent unit (GRU) to the robustly normalized value sequence and its observation mask. Appendix B.1 details the exact text fields, numeric feature construction, temporal preprocessing, and fusion operations used by the modality encoders.

To respectively obtain representations specialized for root-cause relevance and symptom evidence, we learn

$$\mathbf{u}_{I,i} = \mathrm{Norm}(P_{\text{rel}}\mathbf{z}_{I,i}), \quad \mathbf{v}_{I,i} = \mathrm{Norm}(P_{\text{asp}}\mathbf{z}_{I,i}). \tag{2}$$

The relevance embedding $\mathbf{u}_{I,i}$ measures consistency with the incident root cause; $\mathbf{v}_{I,i}$ captures the symptom or evidence aspect.

Hierarchy. Resource metadata deterministically assigns each observation to a subsystem $a(i) \in \mathcal{S}_I$. Attention pooling produces

$$\mathbf{h}_{I,s} = \mathrm{Pool}_{\text{sub}}\left(\{\mathbf{z}_{I,i} : a(i) = s\}\right), \qquad \mathbf{g}_I = \mathrm{Pool}_{\text{global}}\left(\{\mathbf{h}_{I,s} : s \in \mathcal{S}_I\}\right). \tag{3}$$

The subsystem representation summarizes a local failure manifestation, whereas the global representation summarizes incident-level evidence. A propagation head estimates whether subsystem s lies on the hidden propagation chain:

$$\widehat{m}_{I,s} = \sigma(g_{\text{prop}}\left[\mathbf{h}_{I,s} \| \mathbf{g}_I \| (\mathbf{h}_{I,s} \odot \mathbf{g}_I)\right]). \tag{4}$$

To compare observations with subsystem- and incident-level context in a common root-cause relevance space, we define

$$\mathbf{g}_I^{\text{rel}} = \mathrm{Norm}(P_g \mathbf{g}_I), \qquad \mathbf{h}_{I,s}^{\text{rel}} = \mathrm{Norm}(P_{\text{sub}}\mathbf{h}_{I,s}), \tag{5}$$

where P_g and P_{sub} are the global and subsystem relevance projections, respectively. Together with the observation relevance embedding $\mathbf{u}_{I,i} = \mathrm{Norm}(P_{\text{rel}}\mathbf{z}_{I,i})$, the retrieval score is

$$q_{I,i} = \lambda_g \,\mathrm{sim}(\mathbf{u}_{I,i}, \mathbf{g}_I^{\text{rel}}) + \lambda_s \,\mathrm{sim}(\mathbf{u}_{I,i}, \mathbf{h}_{I,a(i)}^{\text{rel}}) + \lambda_p \widehat{m}_{I,a(i)}. \tag{6}$$

The three terms capture incident-level root-cause relevance, subsystem-level relevance, and predicted propagation membership, respectively.

Complementary anchors and corroborating evidence. After duplicate suppression, we retain the top-M observations by $q_{I,i}$ as anchor candidates. Starting from an empty anchor set $\mathcal{A}$, we greedily select the feasible candidate with the largest marginal gain:

$$\Delta(i \mid \mathcal{A}) = q_{I,i} - s_{\text{div}} \sum_{j \in \mathcal{A}} [\text{sim}(\mathbf{v}_{I,i}, \mathbf{v}_{I,j}) - \delta]_+ . \tag{7}$$

Here, s_{div} controls the diversity penalty, δ is the tolerated aspect similarity, and $[x]_+ = \max(x, 0)$. The resulting anchor set $\mathcal{A}_I$ satisfies $\sum_{i \in \mathcal{A}_I} \ell_{I,i} \leq B_{\text{anchor}} \leq B$, where $\ell_{I,i}$ is the estimated serialized token cost of $o_{I,i}$. Selection stops when the maximum anchor count is reached or no budget-feasible candidate meets the minimum-gain threshold.

We then retrieve corroborating evidence for each anchor $a \in \mathcal{A}_I$ based on its distance to candidate observations in the aspect space. Candidates are ranked by the resulting score and scanned in descending order, retaining at most K supporting observations per anchor. Each candidate is admitted only if it has not already been selected and its cost fits the remaining global budget; otherwise, it is skipped. The budget is updated after every admission and shared across all anchors, ensuring

$$\sum_{i \in \mathcal{A}_I} \ell_{I,i} + \sum_{a \in \mathcal{A}_I} \sum_{j \in \mathcal{S}_{I,a}} \ell_{I,j} \leq B, \tag{8}$$

where $\mathcal{S}_{I,a}$ is the resulting support set for anchor a. Any budget not consumed by anchors remains available for supports, and each observation is included at most once. The agent receives

$$\mathcal{B}_I^B = \{(o_{I,a}, \{o_{I,j} : j \in \mathcal{S}_{I,a}\}) : a \in \mathcal{A}_I\} . \tag{9}$$

3.2 Multi-Stage Training

We train StateBundle in three stages to separate cross-modal representation learning, hierarchical incident supervision, and evidence selection.

Stage 1: cross-modal alignment. The modality encoders and aspect projection are trained with

$$\mathcal{L}^{(1)} = \lambda_{\text{align}} \mathcal{L}_{\text{align}}, \tag{10}$$

where $\mathcal{L}_{\text{align}}$ is a weighted multi-positive supervised contrastive loss. Positive pairs are mined across telemetry channels when they describe a compatible local manifestation, supported by either same-entity temporal overlap or an explicit observable correlation link. Hard negatives contain confidently incompatible local manifestations, whereas ambiguous relations are treated as neutral.

Stage 2: hierarchical supervision. Observation embeddings are pooled into subsystem representations $\mathbf{h}_{I,s}$ and a global incident representation $\mathbf{g}_I$. The global representation is projected by P_g into the root-cause relevance space and aligned with the correct root-cause prototype through $\mathcal{L}_{\text{rc}}$. In parallel, the propagation head predicts whether each subsystem belongs to the hidden propagation footprint through L_{prop}. We optimize

$$\mathcal{L}^{(2)} = \lambda_{\text{rc}} \mathcal{L}_{\text{rc}} + \lambda_{\text{prop}} \mathcal{L}_{\text{prop}}. \tag{11}$$

Stage 3: evidence selection. At the beginning of Stage 3, the observation and subsystem relevance projections P_{rel} and P_{sub} are initialized from the global relevance projection P_g learned in Stage 2; P_g is subsequently frozen. The resulting relevance scores $q_{I,i}$ define soft inclusion probabilities $\pi_{I,i}$ and normalized selection weights $\alpha_{I,i}$.

The observation-relevance objective $\mathcal{L}_{\text{obs}}$ encourages highly weighted observations to align with the global relevance representation of their own incident while discouraging alignment with incidents having different root causes. Using $\pi_{I,i}$, we reconstruct a selected incident representation $\mathbf{g}_I^{\text{sel}}$ and a complementary representation $\mathbf{g}_I^{\text{rem}}$ from the evidence left after selection. The Stage 3 objective is

$$\mathcal{L}^{(3)} = \lambda_{\text{obs}} \mathcal{L}_{\text{obs}} + \lambda_{\text{suf}} \mathcal{L}_{\text{suf}} + \lambda_{\text{comp}} \mathcal{L}_{\text{comp}} + \lambda_{\text{div}} \mathcal{L}_{\text{div}}. \tag{12}$$

Here, $\mathcal{L}_{\text{suf}}$ encourages the selected evidence to preserve the root-cause distribution inferred from the complete incident, $\mathcal{L}_{\text{comp}}$ requires removal of the selected evidence to reduce confidence in the correct root cause, and $\mathcal{L}_{\text{div}}$ discourages assigning high selection weight to observations expressing redundant symptom aspects. Complete loss definitions and pair-mining rules are given in Appendix C.

4 DC-BENCH

Cross-layer environment. DC-BENCH uses a deterministic discrete-time simulator for controlled closed-loop evaluation. Its state spans facility thermal and power conditions, compute and storage resources, network behavior, control-plane and observability state, and application workloads. At each tick,

$$\mathbf{x}_{t+1} = F_{\boldsymbol{\theta}}\left(\mathbf{x}_t, \mathbf{w}_t, \mathbf{f}_t, \mathbf{a}_t\right), \tag{13}$$

where $\mathbf{w}_t$ is workload demand, $\mathbf{f}_t$ contains active fault interventions, and $\mathbf{a}_t$ contains agent controls. Workload demand determines resource utilization; utilization determines server and rack power; IT power and cooling capacity determine thermal state; and power, thermal, compute, network, storage, control-plane, and application conditions jointly determine service capacity, queueing, latency, errors, and SLO status. Actions modify these transitions through cooling, migration, throttling, maintenance, policy, monitoring, sensor, and routing controls.

Executable incident tasks. Each episode resets a seeded environment, starts and stabilizes a workload, and then privately injects one evaluator-owned fault. The agent iteratively observes telemetry, advances time, applies task-permitted controls, and submits its result. The released suite contains 72 task instances obtained by crossing 18 fault mechanisms with detection, localization, root-cause analysis, and mitigation. Non-mitigation tasks expose only observation and time-advancement operations, whereas mitigation tasks additionally expose scenario-relevant write operations.

Causal telemetry and oracle isolation. The agent observes canonical logs, metrics, alerts, traces, and configuration records. For query time t_q and opaque cut token c_q, the interface exposes only observations whose event window ends by t_q, whose telemetry is available by t_q, and whose availability token equals c_q. Agent-visible telemetry is structurally separated from simulator-derived fault, target, and subsystem-effect annotations. The interface also applies rack-level host visibility and fail-closed validation against future records and forbidden oracle fields.

Evaluation. Detection, localization, and root-cause analysis are evaluated as binary outcomes: an episode is considered successful only if all task-specific criteria are satisfied. Mitigation requires evaluator-side recovery conditions that remain satisfied throughout a fixed stability window. Detailed task-specific evaluation criteria are provided in Appendix D.5.

5 EXPERIMENTAL DESIGN

5.1 EXPERIMENTAL SETUP

Data and evaluation suites. We train STATEBUNDLE on simulator-generated incidents whose observable canonical telemetry and privileged labels are stored separately. The final training, validation, and test splits contain 1,868/230/206 incidents and 7,472/920/824 active-fault snapshots, respectively. Splits are incident disjoint, and all seeds used by the 72-task agent benchmark are excluded from training. To evaluate out-of-distribution (OOD) generalization, we additionally exclude a predefined set of scenario-factor values from the training data. Dataset construction, and complete statistics are reported in Appendix E.1.

Agent baselines. We evaluate controlled tool-using agents backed by DeepSeek-V4-Flash-0731, GPT-5.6 Luna, Gemini 3.1 Flash-Lite, Claude Sonnet 5, GPT-5.6 Sol, Claude Opus 5, and GPT-6 Astra. We additionally evaluate the black-box coding agent Codex backed by GPT-5.6 Luna, GPT-5.6 Sol, and GPT-6 Astra. Black-box agents run in an isolated environment and interact with the benchmark only through the agent-visible task interface. All agents are limited to at most 20 interaction turns, and we conduct three independent trials for each benchmark task. Exact model configurations are reported in Appendix E.2.

5.2 END-TO-END AGENT PERFORMANCE

For each agent, we compare paired runs under two observation conditions: *Full-Canonical*, which exposes the complete causally available canonical telemetry snapshot, and STATEBUNDLE, which

Table 1: End-to-end results on DC-BENCH. W.S. denotes the corresponding agent augmented with STATEBUNDLE. Token usage reports the recorded average model-token consumption per episode. Best results in each column are shown in bold.

System	Detect ↑	Localize ↑	RCA ↑	Mitigate ↑	Overall ↑	Token usage ↓
Full-Canonical						
DeepSeek-V4-Flash-0731	61.1%	83.3%	61.1%	29.6%	58.8%	755,587
GPT-5.6 Luna	70.4%	88.9%	64.8%	59.3%	70.8%	203,239
Gemini 3.1 Flash-Lite	75.9%	87.0%	70.4%	57.4%	72.7%	139,114
Claude Sonnet 5	72.2%	94.4%	70.4%	37.0%	68.5%	136,477
GPT-5.6 Sol	83.3%	**98.1%**	77.8%	64.8%	81.0%	336,910
Claude Opus 5	68.5%	92.6%	70.4%	37.0%	67.1%	94,351
GPT-6 Astra	94.4%	96.3%	94.4%	37.0%	80.6%	121,022
Codex agents						
Codex (GPT-5.6 Luna)	92.6%	85.2%	68.5%	35.2%	70.4%	181,200
Codex (GPT-5.6 Sol)	83.3%	87.0%	79.6%	57.4%	76.9%	261,092
Codex (GPT-6 Astra)	90.7%	88.9%	83.3%	48.1%	77.8%	250,306
With STATEBUNDLE *(W.S.)*						
DeepSeek-V4-Flash-0731 (W.S.)	83.3%	74.1%	77.8%	27.8%	65.7%	206,742
GPT-5.6 Luna (W.S.)	92.6%	83.3%	75.9%	61.1%	78.2%	110,878
Gemini 3.1 Flash-Lite (W.S.)	83.3%	90.7%	79.6%	61.1%	78.7%	**26,552**
Claude Sonnet 5 (W.S.)	74.1%	96.3%	70.4%	53.7%	73.6%	54,246
GPT-5.6 Sol (W.S.)	**98.1%**	92.6%	90.7%	**68.5%**	**87.5%**	77,388
Claude Opus 5 (W.S.)	81.5%	88.9%	72.2%	50.0%	73.1%	51,530
GPT-6 Astra (W.S.)	96.3%	**98.1%**	**96.3%**	46.3%	84.3%	32,535
Codex agents						
Codex (GPT-5.6 Luna, W.S.)	90.7%	92.6%	72.2%	50.0%	76.4%	113,484
Codex (GPT-5.6 Sol, W.S.)	88.9%	90.7%	83.3%	64.8%	81.9%	124,938
Codex (GPT-6 Astra, W.S.)	92.6%	88.9%	85.2%	53.7%	80.1%	155,395

constructs a compact evidence bundle from the same snapshot. Paired runs share the same scenario manifest, topology, workload, fault target, seed, evaluator, and pre-fault initial state, and differ only in the agent-visible telemetry representation. We use a telemetry budget of $B = 4096$ estimated tokens for STATEBUNDLE. For each task objective, we report the success rate of each subtask across three trials, together with the overall success rate across all 72 benchmark tasks. We measure observation efficiency using token usage.

As shown in Table 1, STATEBUNDLE substantially reduces token usage for every evaluated agent, with reductions ranging from 37.4% to 80.9% relative to the corresponding Full-Canonical condition. Beyond reducing token usage, STATEBUNDLE also improves observed aggregate success to different extents.

Our results indicate that agent performance is not determined solely by the broad capability of the underlying language model. Claude Opus 5, for example, performs strongly on broad external capability evaluations (Artificial Analysis, 2026), yet its tool-using agent achieves only moderate overall success on our benchmark. Likewise, GPT-6 Astra does not consistently outperform GPT-5.6 Sol. Inspection of the episode trajectories of Claude Opus 5 agent indicates that failures also arise from over-inference from downstream symptoms, inadequate temporal reasoning over evolving system states, and premature termination before stable recovery is established. This discrepancy suggests that closed-loop data-center operation depends not only on backbone-model capability, but also on the agent's ability to acquire, organize, and act on operational evidence over multi-step trajectories. Detailed failure analyses are provided in Appendix E.4 and E.5.

5.3 TOKEN-BUDGET-MATCHED EVALUATION

To verify that the gains of STATEBUNDLE arise from its learned telemetry-selection strategy rather than merely from presenting agents with shorter inputs, we compare it against a token-matched deterministic baseline, channel-stratified truncation (TRUNC-B). Specifically, we divide the telemetry budget B equally among nonempty channels, rank observations within each channel by decreasing event end time, and iteratively redistribute unused budget until no additional observation can be retained. For a fair comparison, TRUNC-B also applies the same deterministic observation-retention policy used by STATEBUNDLE, thereby controlling for the effect of this policy and isolating the contribution of learned evidence selection. We evaluate both methods using GPT-5.6 Sol and Gemini

Table 2: Token-budget-matched comparison of STATEBUNDLE, LLMLingua-2, and TRUNC-B

System	Detect ↑	Localize ↑	RCA ↑	Mitigate ↑	Overall ↑	Token usage ↓
With STATEBUNDLE *(W.S.)*						
Gemini 3.1 Flash-Lite (W.S.)	83.3%	90.7%	79.6%	61.1%	78.7%	26,552
GPT-5.6 Sol (W.S.)	98.1%	92.6%	90.7%	68.5%	87.5%	77,388
LLMLingua-2						
Gemini 3.1 Flash-Lite	57.4%	61.1%	46.3%	29.6%	48.6%	41,274
GPT-5.6 Sol	55.6%	72.2%	44.4%	44.4%	54.2%	108,680
*Token-Budget-Matched (*TRUNC-B*)*						
Gemini 3.1 Flash-Lite	70.4%	74.1%	59.3%	35.2%	59.7%	32,430
GPT-5.6 Sol	72.2%	81.5%	63.0%	53.7%	67.6%	89,203

3.1 Flash-Lite as agent backends. As shown in Table 2, agents using TRUNC-B achieve substantially lower task success than their STATEBUNDLE counterparts, indicating the value of STATEBUNDLE's learned evidence-selection mechanism. Nevertheless, TRUNC-B remains capable of solving a subset of tasks because channel stratification, recency-based truncation, and the shared deterministic retention policy preserve some informative telemetry. In addition, although the two methods are subject to the same per-snapshot telemetry budget B, STATEBUNDLE also yields lower total model-token usage per episode. This reduction arises because agents equipped with STATEBUNDLE typically complete tasks in fewer interaction turns, thereby reducing the cumulative tokens consumed across the full agent trajectory.

5.4 COMPARISON WITH LEARNING-BASED COMPRESSION BASELINES

To compare STATEBUNDLE with an established learning-based compression method, we evaluate against LLMLingua-2 (Pan et al., 2024). We adapt LLMLingua-2 to select complete canonical observations rather than individual tokens. Concretely, for each observation, we use the released pretrained compressor to obtain token-level preservation probabilities, aggregate sub-token probabilities within words, and then average the resulting word-level scores to obtain an observation-level importance score. Observations are ranked by this score and greedily retained while satisfying the same telemetry budget. We additionally apply the same deterministic observation-retention policy used by STATEBUNDLE to ensure a controlled comparison.

We evaluate LLMLingua-2 and STATEBUNDLE using GPT-5.6 Sol and Gemini 3.1 Flash-Lite as agent backbones. As shown in Table 2, STATEBUNDLE achieves higher overall success while using fewer model tokens. One possible explanation is that LLMLingua-2 estimates importance primarily from the semantic content available within its bounded scoring context and does not explicitly model global cross-observation dependencies. In contrast, STATEBUNDLE constructs its compact representation using global relevance and relationships among observations, allowing evidence to be selected according to cross-observation context. These results suggest that modeling global semantic structure and inter-observation relationships is beneficial for compressing heterogeneous operational telemetry.

5.5 OUT-OF-DISTRIBUTION GENERALIZATION

To evaluate the generalizing capacity of STATEBUNDLE, we hold out selected factors along the workload, topology, and target axes during training. For OOD evaluation, we reconstruct the 72 benchmark problems by varying exactly one held-out factor per problem while keeping all remaining factors unchanged. This preserves maximal alignment with the original problem set and minimizes performance differences attributable to changes in evaluation instances rather than distribution shift itself. We use GPT-5.6 Sol and Gemini 3.1 Flash-Lite as agent backends and compare their in-distribution (ID) and OOD performance. Table 3 shows that STATEBUNDLE maintains comparable performance under held-out distribution shifts, suggesting that its compact evidence representation generalizes beyond the training configurations to changes in workload, topology, and fault target.

Table 3: Results of the out-of-distribution generalization evaluation. ID denotes evaluation on the in-distribution test set (i.e., the original benchmark problem set), whereas OOD denotes evaluation on the out-of-distribution test sets.

System	Detect ↑	Localize ↑	RCA ↑	Mitigate ↑	Overall ↑	Token usage ↓
In-distribution Test Set (ID)						
Gemini 3.1 Flash-Lite	83.3%	90.7%	79.6%	61.1%	78.7%	26,552
GPT-5.6 Sol	98.1%	92.6%	90.7%	68.5%	87.5%	77,388
Out-of-Distribution (OOD)						
Gemini 3.1 Flash-Lite	88.9%	88.9%	77.8%	63.0%	79.6%	27,384
GPT-5.6 Sol	92.6%	94.4%	90.7%	66.7%	86.1%	79,619

Table 4: Ablation study of StateBundle on DC-Bench using Codex (GPT-5.6 Luna).

System	Detect ↑	Localize ↑	RCA ↑	Mitigate ↑	Overall ↑	Token usage ↓
Codex (GPT-5.6 Luna, W.S.)	90.7%	92.6%	72.2%	50.0%	76.4%	113,484
Codex (GPT-5.6 Luna, W.S., w/o HR)	72.2%	77.8%	57.4%	40.7%	62.0%	174,084
Codex (GPT-5.6 Luna, W.S., w/o CE)	88.9%	85.2%	64.8%	50.0%	72.2%	135,556
Codex (GPT-5.6 Luna, W.S., w/o $\mathcal{L}_{\text{obs}}$)	87.0%	77.8%	74.1%	44.4%	70.8%	128,523
Codex (GPT-5.6 Luna, W.S., w/o $\mathcal{L}_{\text{suf}}$)	81.5%	85.2%	64.8%	51.9%	70.8%	132,585
Codex (GPT-5.6 Luna, W.S., w/o $\mathcal{L}_{\text{comp}}$)	94.4%	79.6%	55.6%	48.1%	69.4%	134,512
Codex (GPT-5.6 Luna, W.S., w/o $\mathcal{L}_{\text{div}}$)	79.6%	94.4%	68.5%	38.9%	70.4%	129,414

5.6 Ablation studies

To assess the contribution of the key design choices in StateBundle, we conduct a series of ablation studies. First, to evaluate the hierarchical representation, we remove the subsystem layer, the propagation loss $\mathcal{L}_{\text{prop}}$, and the corresponding propagation-dependent terms in $\mathcal{L}^{(3)}$. Second, to assess the contribution of corroborating evidence, we disable support-evidence retrieval and retain only cause-relevant anchors under the same total token budget B. Finally, to evaluate the Stage 3 evidence-selection objectives, we independently remove $\mathcal{L}_{\text{obs}}$, $\mathcal{L}_{\text{suf}}$, $\mathcal{L}_{\text{comp}}$, and $\mathcal{L}_{\text{div}}$. We use Codex backed by GPT-5.6 Luna for all ablation experiments.

As shown in Table 4, all ablations degrade the end-to-end performance of StateBundle while increasing token consumption, demonstrating that each component contributes to the overall effectiveness of the method. In particular, removing the hierarchical representation produces the largest degradation, reducing overall success from 76.4% to 62.0% and increasing token usage by 53.4%. This result highlights the importance of explicitly organizing evidence across subsystem and propagation levels. Removing corroborating-evidence retrieval also reduces overall success to 72.2%, suggesting that cause-relevant anchors alone are insufficient and that supporting observations provide useful complementary evidence. Similarly, removing $\mathcal{L}_{\text{obs}}$, $\mathcal{L}_{\text{suf}}$, $\mathcal{L}_{\text{comp}}$, or $\mathcal{L}_{\text{div}}$ consistently reduces overall success while increasing token usage to varying degrees, supporting the contribution of each Stage 3 objective.

6 Limitations and Conclusion

We introduced StateBundle, a cause-centered method for constructing compact telemetry bundles, and DC-Bench, a cross-layer, simulator-backed benchmark for data-center operations. Experimental results show that StateBundle substantially reduces token usage across all evaluated agents while improving task performance to varying degrees. Through DC-Bench, we further identify several factors beyond the underlying model capability that influence agentic data-center operation performance.

This work has several limitations. Its generalization to real-world deployments and incidents with simultaneous root causes remains to be established. In addition, training relies on simulator-derived root-cause and propagation labels, while a finite evidence budget may omit operationally relevant observations.

AI USE STATEMENT

Generative AI tools were used to assist with editing the manuscript for clarity and readability, preparing draft scientific illustrations, supporting qualitative data analysis, and implementing and editing portions of the source code. We have not used generative AI tools for the rest of the required disclosure tasks. All AI-assisted text and figures were manually reviewed and revised by the authors before inclusion in the final manuscript. AI-assisted code was reviewed by two authors and tested to verify its correctness and intended functionality. The authors take full responsibility for the final manuscript, scientific claims, figures, and software artifacts produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

Section 4 describes the simulator and episode protocol, while Appendix D.5 provides the task-specific evaluation criteria. Section 3 and Appendices A, B.1, and C describe canonicalization, modality encoding, and training objectives. Section 5.1, together with Appendices E.1 and E.2, document dataset construction and evaluation settings. Finally, we release the source code to support reproducibility.

REFERENCES

Artificial Analysis. Artificial analysis intelligence index v4.3. https://artificialanalysis.ai/evaluations/artificial-analysis-intelligence-index, 2026. Accessed September 2026.

Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011. doi: 10.1002/spe.995.

Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems*, EuroSys '24, pp. 674–688. ACM, 2024. doi: 10.1145/3627703.3629553.

Yinfang Chen, Jiaqi Pan, Jackson Clark, Yiming Su, Noah Zheutlin, Bhavya Bhavya, Rohan Arora, Yu Deng, Saurabh Jha, and Tianyin Xu. STRATUS: A multi-agent system for autonomous reliability engineering of modern clouds. In *Advances in Neural Information Processing Systems*, volume 38, 2025a. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/47a6e9e2c3019f13ad94a0f259fe4970-Abstract-Conference.html.

Yinfang Chen, Manish Shetty, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Jonathan Mace, Chetan Bansal, Rujia Wang, and Saravan Rajmohan. AIOpsLab: A holistic framework to evaluate AI agents for enabling autonomous clouds. *Proceedings of Machine Learning and Systems*, 7, 2025b. URL https://proceedings.mlsys.org/paper_files/paper/2025/hash/d1f9e4a9f109b6e8b75ed362736f22ec-Abstract-Conference.html.

Jinkyun Cho and Joo Hyun Moon. Numerical coupling of energy efficiency and thermal performance for cold plate cooling optimization in high-density compute ai data centers. *Energy and Buildings*, 348:116441, 2025. doi: 10.1016/j.enbuild.2025.116441.

Jackson Clark, Yiming Su, Saad Mohammad Rafid Pial, Yifang Tian, Lily Gniedziejko, Hans-Arno Jacobsen, Yinfang Chen, and Tianyin Xu. SREGym: A live benchmark for AI SRE agents with high-fidelity failure scenarios, 2026. URL https://arxiv.org/abs/2605.07161.

Philip Colangelo, Ayse K. Coskun, Jack Megrue, et al. Ai data centres as grid-interactive assets. *Nature Energy*, 11:254–261, 2026. doi: 10.1038/s41560-025-01927-1.

Seokjin Go, Joongun Park, Spandan More, Hanjiang Wu, Irene Wang, Aaron Jezghani, Tushar Krishna, and Divya Mahajan. Characterizing the efficiency of distributed training: A power, performance, and thermal perspective. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, MICRO '25, pp. 626–642. ACM, 2025. doi: 10.1145/3725843.3756111.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. The Llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783v3. Version 3.

Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*, pp. 33–40. IEEE, 2017. doi: 10.1109/ICWS.2017.13.

Saurabh Jha, Rohan R. Arora, Yuji Watanabe, Takumi Yanagawa, Yinfang Chen, Jackson Clark, Bhavya Bhavya, Mudit Verma, Harshit Kumar, Hirokuni Kitahara, Noah Zheutlin, Saki Takano, Divya Pathak, Felix George, Xinbo Wu, Bekir O Turkkan, Gerard Vanloo, Michael Nidd, Ting Dai, Oishik Chatterjee, Pranjal Gupta, Suranjana Samanta, Pooja Aggarwal, Rong Lee, Jae-Wook Ahn, Debanjana Kar, Amit Paradkar, Yu Deng, Pratibha Moogi, Prateeti Mohapatra, Naoki Abe, Chandrasekhar Narayanaswami, Tianyin Xu, Lav R. Varshney, Ruchi Mahindru, Anca Sailer, Laura Shwartz, Daby Sow, Nicholas C. M. Fuller, and Ruchir Puri. ITBench: Evaluating AI agents across diverse real-world IT automation tasks. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 27134–27197. PMLR, 2025. URL https://proceedings.mlr.press/v267/jha25a.html.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13358–13376. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.825.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1658–1677. Association for Computational Linguistics, 2024a. doi: 10.18653/v1/2024.acl-long.91. URL https://aclanthology.org/2024.acl-long.91/.

Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, Yulu Jia, Sun He, Hongmin Chen, Zhihao Bai, Qi Hou, Shipeng Yan, Ding Zhou, Yiyao Sheng, Zhuo Jiang, Haohan Xu, Haoran Wei, Zhang Zhang, Pengfei Nie, Leqi Zou, Sida Zhao, Liang Xiang, Zherui Liu, Zhe Li, Xiaoying Jia, Jianxi Ye, Xin Jin, and Xin Liu. MegaScale: Scaling large language model training to more than 10,000 GPUs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pp. 745–760. USENIX Association, 2024b. URL https://www.usenix.org/conference/nsdi24/presentation/jiang-ziheng.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 18661–18673, 2020.

Dzmitry Kliazovich, Pascal Bouvry, and Samee Ullah Khan. Greencloud: A packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, 62(3):1263–1283, 2012. doi: 10.1007/s11227-010-0504-1.

Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R. Lyu. Eadro: An end-to-end troubleshooting framework for microservices on multi-source data. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 1750–1762. IEEE, 2023. doi: 10.1109/ICSE48619.2023.00150.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638.

Qingkai Meng, Hao Zheng, Zhenhui Zhang, ChonLam Lao, Chengyuan Huang, Baojia Li, Ziyuan Zhu, Hao Lu, et al. Astral: A datacenter infrastructure for large language model training at scale. In *Proceedings of the ACM SIGCOMM 2025 Conference*, pp. 609–625, 2025. doi: 10.1145/3718958.3750521.

Paolo Notaro, Jorge Cardoso, and Michael Gerndt. A survey of AIOps methods for failure management. *ACM Transactions on Intelligent Systems and Technology*, 12(6):81:1–81:45, 2021. doi: 10.1145/3483424.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, et al. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 963–981, 2024.

Arman Shehabi, Alex Newkirk, Sarah J Smith, Alex Hubbard, Nuoa Lei, Md Abu Bakar Siddik, Billie Holecek, Jonathan Koomey, Eric Masanet, and Dale Sartor. 2024 united states data center energy usage report. Technical report, Lawrence Berkeley National Laboratory, 2024.

Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Esha Choukse, Haoran Qiu, Rodrigo Fonseca, Josep Torrellas, and Ricardo Bianchini. TAPAS: Thermal- and power-aware scheduling for LLM inference in cloud platforms. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '25, pp. 1266–1281. ACM, 2025. doi: 10.1145/3676641.3716025.

Yilun Wang, Guangba Yu, Haiyu Huang, Zirui Wang, Yujie Huang, Pengfei Chen, and Michael R. Lyu. Cloud-opsbench: A reproducible benchmark for agentic root cause analysis in cloud systems, 2026.

Zefan Wang, Zichuan Liu, Yingying Zhang, Aoxiao Zhong, Jihong Wang, Fengbin Yin, Lunting Fan, Lingfei Wu, and Qingsong Wen. RCAgent: Cloud root cause analysis by autonomous agents with tool-augmented large language models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24. ACM, 2024. doi: 10.1145/3627673.3680016.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. RECOMP: Improving retrieval-augmented LMs with context compression and selective augmentation. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=mlJLVigNHp.

Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. Nezha: Interpretable fine-grained root causes analysis for microservices on multi-modal observability data. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 553–565. ACM, 2023. doi: 10.1145/3611643.3616249.

Shenglin Zhang, Pengxiang Jin, Zihan Lin, Yongqian Sun, Bicheng Zhang, Sibo Xia, Zhengdan Li, Zhenyu Zhong, Minghua Ma, Wa Jin, Dai Zhang, Zhenyu Zhu, and Dan Pei. Robust failure diagnosis of microservice system through multimodal data. *IEEE Transactions on Services Computing*, 16(6):3851–3864, 2023. doi: 10.1109/TSC.2023.3290018.

Table 5: Source-code segmentation and consolidation policies for canonical observations. "Window" denotes the query lookback interval ending at t_q.

Channel	Adapter input	Observation boundary	Canonical payload
Log	Sanitized observable event records	Events assigned to the same subsystem and sharing the same normalized message template are consolidated within the query window.	A template aggregate containing the normalized template, event type, count, maximum severity, severity histogram, causal rarity, burst rate, summaries of structured event details, and first/last offsets from the window start.
Metric	Metric samples from causally available state captures	Samples with the same metric name and entity are consolidated into one time-series segment over the query window.	A series segment containing the name, unit, scale, timestamps, values, missingness mask, descriptive statistics, causal normalization reference, and resource identifier.
Alert	Safe active-alert sets from successive captures	Successive states with the same observable alert type and target are consolidated into one firing episode, ending at resolution or at the query time if still active.	A deduplicated episode containing the fingerprint, type, message, target, firing or resolved status, severity, threshold fields, duration, and safe details.
Trace	Simulator-derived per-capture request-flow records	Records sharing the same operation, source, and destination are consolidated over the query window.	A request-flow group containing the operation, endpoints, request and status counts, retry count, latency summary, and the current critical-path flag.
Config.	Nested safe configuration snapshots	Each flattened configuration path is represented by its visible state or by a `set`, `update`, or `remove` transition.	A scoped path–value record containing the path, typed current and previous values, scope, `set`/`update`/`remove`/`state` operation, and change time.

A TELEMETRY CANONICALIZATION

In DC-BENCH, causally available telemetry is first converted into canonical observations to normalize heterogeneous telemetry across channels. Canonical observations serve as the common input representation for all experimental configurations evaluated in this paper.

Let $\mathcal{C} = \{\log, \text{metric}, \text{alert}, \text{trace}, \text{config}\}$. At query time t_q, a deterministic channel adapter A_c converts the causally available observable history $R_{I,c}$ of channel c into canonical observations:

$$\mathcal{O}_I(t_q) = \bigcup_{c \in \mathcal{C}} A_c(R_{I,c}, t_q), \qquad o_{I,i} = (\text{id}_{I,i}, c_{I,i}, w_{I,i}, p_{I,i}, \rho_{I,i})\,. \tag{14}$$

Here, $o_{I,i} \in \mathcal{O}_I(t_q)$ denotes an individual canonical observation in the collection $\mathcal{O}_I(t_q)$; $\text{id}_{I,i}$ is its stable identifier; $w_{I,i}$ is the closed event interval; $p_{I,i}$ is the channel-specific payload; and $\rho_{I,i}$ contains structural metadata, including subsystem assignment, entity roles, and correlation identifiers linking related observations. They segment logs by subsystem and normalized template, metrics by metric name and entity, alerts by type–target firing episodes, request-flow traces by operation–source–destination, and configurations by flattened path at each visible state or change time.

A.1 ADAPTER AND BOUNDARY FOR EACH MODALITY

Table 5 summarizes the modality-specific boundaries used to construct canonical observations and the resulting payload for each telemetry channel. The following parts detail the channel-specific normalization and consolidation performed by each adapter.

Log adapter. Before grouping, the adapter replaces numeric substrings with `<number>` and recognized server, rack, or cooling-unit identifiers with `<resource>`. The template identifier is a deterministic hash of the event type and normalized message. The subsystem is inferred from the observable event type. Historical rarity is the reciprocal of the causally visible occurrence count of the template, and burst rate is computed from the number of retained events and their observed span. Variable summaries are derived only from structured `details`: numeric fields retain count, minimum, maximum, and mean, whereas nonnumeric fields retain counts of their most frequent rendered values. The observation interval is the first-to-last occurrence span.

Metric adapter. For each metric–entity pair, missing cadence points are represented by `None` values and a same-length Boolean missingness mask. The payload statistics are count, minimum,

maximum, mean, median, p95, population standard deviation, endpoint slope, and last observed value. The normalization reference is computed from causally available samples preceding the lookback window when they exist; otherwise, the observed values in the current segment are used as a fallback. It records the reference median, interquartile range, last value, sample count, and optional reference end time.

Alert adapter. The fingerprint contains only the observable alert type and target. Repeated active snapshots therefore remain one episode rather than becoming multiple observations. Threshold information is copied from safe detail fields whose names contain `threshold` or `limit`; the episode duration is the firing-to-resolution interval, or the firing-to-query interval for an active alert.

Trace adapter. The current trace source is a request-flow abstraction rather than a full OpenTelemetry span tree. For each running workload or tenant and simulator capture, it constructs an operation from the workload class, a source `tenant:<id>`, a destination `service:<workload-class>`, request count, error count, retry count, and mean/p95 latency. Canonicalization sums the counts and retries within each operation–source–destination group, derives successful requests as total minus errors, averages the per-capture mean latencies, and takes the maximum available p95. The released adapter marks each emitted group as being on the critical path.

Configuration adapter. Flattening and differencing, rather than neural encoding, determine the configuration observation boundary. A new path is `set`, a changed existing path is `update`, and a disappeared path is `remove`. Each group of same-capture changes receives an opaque revision identifier. The adapter retains both the complete set of changes inside the query window and the latest visible state of every path so that an unchanged but operationally relevant setting is not lost when its last update predates the window.

B StateBundle Details

B.1 Modality-Specific Encoders

Shared encoding framework. All modality encoders follow a common three-branch design that combines textual content, numeric attributes, and observable structural metadata. The text branch lowercases and tokenizes the selected textual fields, maps each token deterministically with BLAKE2b to a trainable vocabulary entry, and mean-pools the resulting token embeddings. The numeric branch converts channel-specific numerical attributes into a fixed-width feature vector and projects it to the encoder dimension. For structured payloads, numerical fields are summarized using simple distributional statistics such as count, mean, dispersion, range, magnitude, and the fractions of zero and negative values. The metadata branch represents information shared across modalities, including the primary subsystem, entity roles, timing and availability information, missingness, parsing confidence, and other data-quality indicators. The text, numeric, and metadata representations are projected to the same dimension, summed, and layer-normalized. The metric encoder additionally includes a temporal branch for the canonical value sequence. The resulting channel-specific representation is subsequently projected and normalized to obtain the shared observation embedding $\mathbf{z}_{I,i}$ defined in the main paper.

Log encoder. For logs, the text branch represents the normalized message template together with the observable event type and severity. The numeric branch captures the frequency and temporal behavior of the template, including its occurrence count, causal rarity, burst rate, severity distribution, numerical summaries of structured event details, and the first and last occurrence offsets within the observation window. Thus, the encoder represents both the semantic event pattern and how that pattern manifests during the incident window.

Metric encoder. For metrics, the text branch represents the metric name, unit, scale, and resource description. The numeric branch summarizes the canonical segment, including its descriptive statistics, normalization reference, sampling interval, missingness, and sequence length.

The metric values are additionally processed as a temporal sequence. Let x_t denote the canonical metric value at position t, and let $m_t \in \{0, 1\}$ indicate whether that value is observed. Let μ_{ref} and r_{ref} denote, respectively, the median and interquartile range of the causal normalization reference. Each observed value is robustly normalized as

$$\widetilde{x}_t = \frac{x_t - \mu_{\text{ref}}}{|r_{\text{ref}}| + \epsilon}, \tag{15}$$

where $\epsilon > 0$ prevents division by zero; missing positions are assigned zero while remaining distinguishable through m_t. The normalized sequence and observation mask are resampled to a fixed length and jointly processed by a gated recurrent unit (GRU). Its final representation captures the temporal evolution of the metric and is fused with the text, numeric, and metadata branches.

Alert encoder. For alerts, the text branch represents the alert message, type, severity, status, and target. The numeric branch summarizes available numerical attributes of the canonical alert episode, including its duration, threshold-related values, and other numeric details. The resulting representation therefore captures both the semantic alert description and the quantitative characteristics of the episode.

Trace encoder. For traces, the text branch represents the operation, source, destination, and request-status information of the canonical request-flow group. The numeric branch summarizes request and status counts, retries, latency measurements, and the critical-path indicator. The encoder therefore represents the behavior of an interaction between two endpoints, while the construction of that interaction group itself remains a canonicalization responsibility.

Configuration encoder. For configurations, the text branch represents the configuration path, current and previous values, value type, scope, and operation. The numeric branch summarizes numerical current or previous values and timing information when present. Consequently, the encoder captures the semantic meaning and magnitude of a configuration state or change.

B.2 Subsystem and Global Pooling

Let $\mathcal{O}_{I,s} = \{i : a(i) = s\}$. We use masked attention pooling:

$$\alpha^s_{I,i} = \frac{\exp\left(\mathbf{w}_s^\top \tanh(W_s \mathbf{z}_{I,i})\right)}{\sum_{j \in \mathcal{O}_{I,s}} \exp\left(\mathbf{w}_s^\top \tanh(W_s \mathbf{z}_{I,j})\right)}, \tag{16}$$

$$\mathbf{h}_{I,s} = \sum_{i \in \mathcal{O}_{I,s}} \alpha^s_{I,i} \mathbf{z}_{I,i}. \tag{17}$$

Global pooling is analogous:

$$\mathbf{g}_I = \sum_{s \in \mathcal{S}_I} \beta_{I,s} \mathbf{h}_{I,s}. \tag{18}$$

Observation dropout is applied during training to reduce self-retrieval bias.

C Complete Training Objectives

C.1 Cross-Modal Alignment

For each symptom vector $\mathbf{v}_{I,i}$, we compute a normalized representation

$$\overline{\mathbf{v}}_{I,i} = \frac{\mathbf{v}_{I,i}}{\|\mathbf{v}_{I,i}\|_2}. \tag{19}$$

Strong positive pairs include observations from different telemetry channels that describe a compatible local symptom for the same entity and overlapping time interval. Observations linked by the same trace, alert episode, deployment, configuration change, or other observable correlation identifier are also treated as strong positives when they describe the same local manifestation. Observations from adjacent windows are positive only when they belong to the same continuous anomaly or alert episode.

Topology adjacency or common incident membership alone is insufficient to define a positive pair. Observations from different stages of the same symptom-propagation chain may describe complementary evidence and are therefore treated as neutral unless they can be verified to represent the same local manifestation. Uncertain pairs are also masked as neutral rather than being forced into either the positive or negative set.

For anchor i, let $\mathcal{P}_i$ denote its positive set, w_{ip} the confidence of positive pair (i, p), and γ_{ia} the contrast weight of candidate a. We use a weighted multi-positive supervised contrastive objective (Khosla et al., 2020):

$$\mathcal{L}_{\text{align}} = -\frac{1}{N}\sum_{i}\frac{1}{\sum_{p\in\mathcal{P}_i} w_{ip}}\sum_{p\in\mathcal{P}_i} w_{ip}\log\frac{\exp\left(\text{sim}\left(\overline{\mathbf{v}}_i, \overline{\mathbf{v}}_p\right)/\tau\right)}{\sum_{a\neq i}\gamma_{ia}\exp\left(\text{sim}\left(\overline{\mathbf{v}}_i, \overline{\mathbf{v}}_a\right)/\tau\right)}. \tag{20}$$

Regular candidates receive $\gamma_{ia} = 1$, verified hard negatives receive $\gamma_{ia} > 1$, and possible false negatives or otherwise ambiguous pairs receive $\gamma_{ia} = 0$. The rules for building positive, neutral, and negative pairs are:

Positive pairs include:

- observations from different telemetry channels that describe a compatible local symptom for the same entity and overlapping time interval;
- observations linked by the same trace, alert episode, deployment, configuration change, or other observable correlation identifier, when they describe the same local manifestation;
- observations from adjacent windows when they belong to the same continuous anomaly or alert episode.

Topology adjacency or common incident membership alone is insufficient to define a positive pair.

Neutral pairs include:

- observations from different stages of the same symptom-propagation chain that provide complementary evidence but cannot be verified to represent the same local manifestation;
- pairs whose relationship cannot be determined with sufficient confidence;
- possible false negatives that should not be forced into either the positive or negative set.

Hard negative pairs include only observations whose local manifestations are confidently incompatible:

- observations describing clearly different symptom families, such as packet loss versus thermal overload;
- the metric or alert with same signal name but in incompatible states, such as anomaly onset versus recovery or firing versus resolved;
- the same log template with variable distributions indicating different operational states;
- the same metric name with incompatible temporal behavior, such as a sudden anomaly versus normal seasonal variation;
- observations from the same entity or time interval that are verified to describe different local manifestations.

C.2 Root-Cause Alignment

Let d_y^{rc} be a textual or structured description of root-cause class y and

$$\mathbf{e}_y^{\text{rc}} = \text{Norm}\left(E_{\text{rc}}(d_y^{\text{rc}})\right). \tag{21}$$

where E_{rc} denote the root-cause description encoder that maps d_y^{rc} into the same root-cause relevance space as $\mathbf{g}_I^{\text{rel}}$. The global loss is

$$\mathcal{L}_{\text{rc}} = -\sum_{I}\log\frac{\exp\left(\text{sim}(\mathbf{g}_I^{\text{rel}}, \mathbf{e}_{y_I^{\text{rc}}}^{\text{rc}})/\tau_{\text{rc}}\right)}{\sum_{y\in\mathcal{Y}_{\text{rc}}}\exp\left(\text{sim}(\mathbf{g}_I^{\text{rel}}, \mathbf{e}_y^{\text{rc}})/\tau_{\text{rc}}\right)}. \tag{22}$$

where $\mathcal{Y}_{\mathrm{rc}}$ denotes the set of root-cause classes considered during training, $y_I^{\mathrm{rc}} \in \mathcal{Y}_{\mathrm{rc}}$ is the ground-truth root-cause class for incident I.

C.3 Propagation Supervision

The propagation head is

$$\widehat{m}_{I,s} = \sigma\left(g_{\mathrm{prop}}\left[\mathbf{h}_{I,s} \| \mathbf{g}_I \| (\mathbf{h}_{I,s} \odot \mathbf{g}_I)\right]\right). \tag{23}$$

With class weights w_+ and w_-,

$$\begin{aligned}\mathcal{L}_{\mathrm{prop}} = -\sum_{I,s} \big[& w_+ m_{I,s} \log \widehat{m}_{I,s} \\ & + w_-(1 - m_{I,s}) \log(1 - \widehat{m}_{I,s})\big].\end{aligned} \tag{24}$$

where $m_{I,s} \in \{0, 1\}$ denote the ground-truth propagation-membership label

C.4 Latent Observation Relevance

We project each subsystem representation into the relevance space as

$$\mathbf{h}_{I,s}^{\mathrm{rel}} = \mathrm{Norm}(P_{\mathrm{sub}} \mathbf{h}_{I,s}), \tag{25}$$

and define the observation relevance score as

$$q_{I,i} = \lambda_g \,\mathrm{sim}(\mathbf{u}_{I,i}, \mathbf{g}_I^{\mathrm{rel}}) + \lambda_s \,\mathrm{sim}(\mathbf{u}_{I,i}, \mathbf{h}_{I,a(i)}^{\mathrm{rel}}) + \lambda_p \widehat{m}_{I,a(i)}. \tag{26}$$

The three terms respectively capture incident-level relevance, subsystem-level relevance, and the predicted propagation membership of the subsystem containing observation i.

At the beginning of this stage, the observation and subsystem relevance projections P_{rel} and P_{sub} have not yet received direct relevance supervision. We therefore initialize both from the global relevance projection learned in Stage 2,

$$P_{\mathrm{rel}} \leftarrow P_g, \qquad P_{\mathrm{sub}} \leftarrow P_g. \tag{27}$$

Accordingly, before Stage 3 optimization, the relevance score is initialized as

$$\begin{aligned}q_{I,i}^{(0)} = {} & \lambda_g \,\mathrm{sim}\left(\mathrm{Norm}(P_g \mathbf{z}_{I,i}), \mathrm{Norm}(P_g \mathbf{g}_I)\right) \\ & + \lambda_s \,\mathrm{sim}\left(\mathrm{Norm}(P_g \mathbf{z}_{I,i}), \mathrm{Norm}(P_g \mathbf{h}_{I,a(i)})\right) + \lambda_p \widehat{m}_{I,a(i)}.\end{aligned} \tag{28}$$

This initialization places observation-, subsystem-, and incident-level representations in the root-cause relevance space learned during Stage 2, avoiding an arbitrary initial selector. Throughout Stage 3, P_g is frozen, while P_{rel} and P_{sub} remain trainable.

Then we use a soft inclusion gate and a normalized pooling weight to select the root-cause-related observation embeddings

$$\pi_{I,i} = \sigma\left(\frac{q_{I,i} - \tau_{\mathrm{rel}}}{T_{\mathrm{gate}}}\right), \tag{29}$$

$$\alpha_{I,i} = \frac{\pi_{I,i} \exp(q_{I,i}/T_{\mathrm{rel}})}{\sum_{j=1}^{N_I} \pi_{I,j} \exp(q_{I,j}/T_{\mathrm{rel}}) + \epsilon}. \tag{30}$$

The positive observation-alignment loss is

$$\mathcal{L}_{\mathrm{obs}}^{+} = \sum_{I,i} \mathrm{sg}(\alpha_{I,i}) \left[1 - \mathrm{sim}\left(\mathbf{u}_{I,i}, \mathrm{sg}(\mathbf{g}_I^{\mathrm{rel}})\right)\right]. \tag{31}$$

For negative incidents $\mathcal{N}_I = \{J : y_J^{\mathrm{rc}} \neq y_I^{\mathrm{rc}}\}$,

$$\mathcal{L}_{\mathrm{obs}}^{-} = \sum_{I,i} \mathrm{sg}(\alpha_{I,i}) \frac{1}{|\mathcal{N}_I|} \sum_{J \in \mathcal{N}_I} \left[\mathrm{sim}(\mathbf{u}_{I,i}, \mathbf{g}_J^{\mathrm{rel}}) - \mu_{\mathrm{neg}}\right]_+. \tag{32}$$

Then

$$\mathcal{L}_{\mathrm{obs}} = \mathcal{L}_{\mathrm{obs}}^{+} + \lambda_{\mathrm{neg}} \mathcal{L}_{\mathrm{obs}}^{-}. \tag{33}$$

C.5 Sufficiency and Comprehensiveness

Using the same hierarchy as the full model,

$$\mathbf{g}_I^{\mathrm{sel}} = \mathrm{HierPool}\left(\{\pi_{I,i}\mathbf{z}_{I,i}\}_{i=1}^{N_I}\right), \tag{34}$$

$$\mathbf{g}_I^{\mathrm{rem}} = \mathrm{HierPool}\left(\{(1-\pi_{I,i})\mathbf{z}_{I,i}\}_{i=1}^{N_I}\right). \tag{35}$$

Here, $\mathbf{g}_I^{\mathrm{sel}}$ is the incident representation constructed by emphasizing observations according to their soft inclusion gates $\pi_{I,i}$, whereas $\mathbf{g}_I^{\mathrm{rem}}$ is the complementary representation obtained by weighting each observation with $1-\pi_{I,i}$.

To evaluate whether these representations preserve the root-cause information, we use the same root-cause prototype space introduced in the root-cause alignment objective. For $x \in \mathrm{full}, \mathrm{sel}, \mathrm{rem}$, we first project the corresponding global representation into the root-cause relevance space:

$$\mathbf{g}_I^{x,\mathrm{rel}} = \mathrm{Norm}\left(P_{\mathrm{g}}\mathbf{g}_I^{x}\right). \tag{36}$$

The root-cause score for class $y \in \mathcal{Y}_{\mathrm{rc}}$ is

$$s_I^x(y) = \frac{\mathrm{sim}\left(\mathbf{g}_I^{x,\mathrm{rel}}, \mathbf{e}_y^{\mathrm{rc}}\right)}{\tau_{\mathrm{rc}}}, \tag{37}$$

where $\mathbf{e}_y^{\mathrm{rc}}$ is the prototype of root-cause class y and $\tau_{\mathrm{rc}} > 0$ is the temperature parameter. The corresponding root-cause probability distribution is

$$p_I^x(y) = \frac{\exp\left(s_I^x(y)\right)}{\sum_{y' \in \mathcal{Y}_{\mathrm{rc}}} \exp\left(s_I^x(y')\right)}. \tag{38}$$

We denote the complete distributions by $\mathbf{p}_I^{\mathrm{full}}$, $\mathbf{p}_I^{\mathrm{sel}}$, and $\mathbf{p}_I^{\mathrm{rem}}$.

Sufficiency. The selected observations should retain the root-cause information available in the complete telemetry. We therefore minimize the divergence between the root-cause distribution predicted from the full incident and that predicted from the selected observations:

$$\mathcal{L}_{\mathrm{suf}} = \sum_I D_{\mathrm{KL}}\left(\mathrm{sg}\left(\mathbf{p}_I^{\mathrm{full}}\right) \| \mathbf{p}_I^{\mathrm{sel}}\right), \tag{39}$$

Comprehensiveness. Sufficiency alone is not enough because a selected subset could reproduce the full prediction while containing observations that are not essential. We therefore additionally require the correct-root-cause confidence to decrease when the selected evidence is suppressed by the complementary weights $1-\pi_{I,i}$. Let y_I^{rc} denote the ground-truth root-cause class of incident I. We define

$$\mathcal{L}_{\mathrm{comp}} = \sum_I \left[m_{\mathrm{comp}} - \log p_I^{\mathrm{full}}\left(y_I^{\mathrm{rc}}\right) + \log p_I^{\mathrm{rem}}\left(y_I^{\mathrm{rc}}\right)\right]+, \tag{40}$$

where $[x]+ = \max(0, x)$ and $m_{\mathrm{comp}} > 0$ is a margin hyperparameter specifying the minimum desired decrease in correct-root-cause log-confidence after the selected evidence is removed.

C.6 Diversity

The soft anchor-diversity loss is

$$\mathcal{L}_{\mathrm{div}} = \sum_I \frac{\sum_{i<j} \alpha_{I,i}\alpha_{I,j}\left[\mathrm{sim}\left(\mathrm{sg}(\mathbf{v}_{I,i}), \mathrm{sg}(\mathbf{v}_{I,j})\right) - \delta_{\mathrm{div}}\right]_+}{\sum_{i<j} \alpha_{I,i}\alpha_{I,j} + \epsilon}. \tag{41}$$

In this stage, the aspect projection P_{asp} is frozen, and the gradients flow only through the selection weights $\alpha_{I,i}$, encouraging the relevance selector not to allocate large weights to multiple redundant observations.

Table 6: Fault mechanisms and principal implemented propagation paths in the DC-BENCH. Arrows indicate modeled dependencies; semicolons separate additional or parallel effects.

No.	Problem name	Implemented effects and propagation
1	Cooling degradation	Reduced cooling-unit capacity and an additional global thermal offset → rack and server temperatures → physical-temperature-driven thermal throttling and CPU-frequency reduction → compute service capacity → workload queue and latency; temperatures also affect thermal status and facility SLA.
2	Rack hotspot	Rack-specific thermal offset → rack and server temperatures → physical-temperature-driven thermal throttling and CPU-frequency reduction → compute service capacity → workload queue and latency; temperatures also affect thermal status and facility SLA.
3	Power overload	Increased healthy-server CPU utilization and additional power draw → rack power and temperatures → physical thermal derating → compute service capacity and workload queue/latency; power/thermal status and facility SLA are also affected; facility power and PUE are recomputed; a separate direct workload latency penalty is added.
4	Server failure	Server unavailability → effective workload allocation and compute capacity → workload queue and latency; failed servers consume no power, affecting rack thermal state; failure-ratio SLA checks and a direct workload latency penalty also apply.
5	Network partition	Rack-scoped reachability loss → attenuation or removal of effective server-allocation weights → compute service capacity → workload queue and latency.
6	Storage I/O saturation	Reduced effective storage IOPS capacity → storage utilization → storage latency penalty → application response latency.
7	Control-plane degradation	Reduced service-capacity multiplier → workload compute capacity → queue and response latency; scheduler/API latency and pending-operation metrics are also affected, and a separate direct workload latency penalty is added.
8	Application error	Application/tenant-scoped error injection → request-error rate → dropped requests and facility SLA; a separate direct application latency penalty is added.
9	Thermal sensor miscalibration	Rack/server sensor bias → reported temperatures and sensor disagreement; reported rack inlet temperature also drives thermal status and facility SLA.
10	Power budget violation	Rack operating power budget derived from power draw at fault injection and held fixed → power-budget utilization and violation status → power-budget alerts and facility SLA.
11	Intermittent server failure	Periodic server unavailability and health flapping → effective workload allocation and compute capacity → workload queue and latency; server power, rack thermal state, and facility SLA checks are also affected.
12	Thermal throttling	Direct reduction of server thermal-throttle factor and CPU-frequency scale; a parallel rack thermal offset raises temperatures and can additionally derate healthy servers in that rack. The stricter direct or physical thermal limit → compute service capacity → workload queue and latency; temperatures also affect thermal status and facility SLA.
13	Network congestion burst	During an active workload burst, reduced effective network-path capacity → network congestion ratio → network latency penalty → application response latency.
14	ToR packet loss	Rack-path degradation produces loss, retransmission, and network-error metrics; affected workload share scales the corresponding workload metrics and latency penalty. The penalty contributes to application latency; workload network errors affect facility SLA.
15	Autoscaler misconfiguration	Misconfigured autoscaler policy → effective server-count limit, bounded by healthy-server count → workload allocation and compute capacity → workload queue and latency.

Continued on next page

Table 6 – continued from previous page

No.	Problem name	Implemented effects and propagation
16	Monitoring pipeline failure	Monitoring-pipeline degradation → telemetry lag, stale update timestamps, missing-data ratios, and pipeline status → monitoring-health alerts.
17	Placement policy misconfiguration	Rack-concentrating placement policy and reduced server-count limit → workload allocation and compute capacity → workload queue and latency; allocation also affects server utilization → rack power and temperatures → physical thermal derating and further compute-capacity reduction when the thermal threshold is exceeded.
18	Load-balancer misconfiguration	Skewed backend weights and unhealthy-backend routing → effective allocation, compute capacity, and request-error rate; capacity affects queue and latency, while request errors produce dropped requests and affect facility SLA.

D Coupled Benchmark Specification

D.1 Details of the fault mechanisms

The benchmark contains 18 fault mechanisms spanning facility, compute, storage, network, control-plane, observability, and application layers. Each mechanism is implemented with explicit cross-layer effects, allowing faults to propagate from their source subsystem to downstream operational states and service performance. Table 6 summarizes these mechanisms and their principal affected-layer chains in the simulator.

D.2 Scenario Configuration and Episode Lifecycle

Each task is instantiated from a scenario manifest specifying the random seed, configuration overrides, workload, fault mechanism and target, fault severity and duration, permitted actions, and evaluation requirements. Table 7 summarizes the base simulator configuration and shared settings of the released 72-task manifest. Configuration overrides take precedence over the base parameters. The released scenarios use web-service and network-heavy workloads with workload noise disabled; demand profiles and placement settings are scenario-specific.

Table 7: Base simulator parameters and shared episode settings in the released scenario manifest.

Parameter	Value
Base topology	1 room, 2 rows, 8 racks, 80 servers
Cooling units	2 units, each with 50 kW baseline capacity
Ambient temperature	22 °C
Rack power limit / operating budget	8 kW / 4 kW
Rack warning / critical temperature	27 °C / 32 °C
Simulator tick / automatic advancement	1 s / disabled
Pre-injection stabilization	5 ticks
Injected-fault duration	1,800 simulated seconds

An episode resets the environment with the scenario seed and configuration, starts the prescribed workload, and advances through the stabilization period. The evaluator then privately injects one fault and applies any scenario-defined post-injection settling ticks before exposing the resulting observations. Fault type, target, and propagation annotations remain separate from agent-visible telemetry. Fault expiry is governed by simulator time rather than the wall-clock time consumed by model inference.

D.3 Coupled State Transitions

Allocation, power, and temperature. Workload placement determines the participating servers and their utilization, subject to server health, placement restrictions, autoscaler limits, routing, and

network reachability. Healthy-server power combines idle power with CPU- and GPU-utilization-dependent contributions; failed servers consume no power, whereas servers in maintenance consume half the idle power. Rack power is the sum of its server contributions. Physical rack inlet temperature combines ambient temperature, a linear rack-power heat term, a cooling term proportional to the positive surplus of available cooling over total IT power, and any global or rack-specific fault offsets. Changes in workload allocation therefore affect both electrical and thermal state.

Thermal feedback to compute. For physical rack inlet temperature T, the simulator applies the compute-capacity factor

$$\eta(T) = \max\left\{0.2,\ 1 - 0.5\max\left(0, \frac{T - T_{\text{warn}}}{T_{\text{crit}} - T_{\text{warn}}}\right)\right\}, \tag{42}$$

where $\eta(T)$ is the physical thermal-derating factor and T_{warn} and T_{crit} are the configured warning and critical temperatures. The factor is 1 at or below the warning threshold, 0.5 at the critical threshold, and bounded below by 0.2. For a healthy server, this physical limit is combined with any directly injected thermal-throttling limit by taking the smaller factor; CPU-frequency scale follows the resulting factor. Placement, power, temperature, and thermal factors are resolved through a bounded iterative recalculation without advancing simulator time or repeatedly accumulating queue demand. Sensor bias changes reported temperatures and thermal status, but not the physical temperature used for compute derating. This thermal relation is a deterministic simulator approximation rather than a calibrated hardware model.

Capacity, queueing, and service performance. Effective service capacity depends on eligible compute resources, their thermal factors, workload resource requirements, and control-plane and routing modifiers. For an active single workload with eligible servers, let Q_t be the existing queue length, D_t and C_t the demand and service-capacity rates used for an update, and Δt the elapsed simulator time. Backlog evolves as

$$Q_{t+\Delta t} = \max\{0, \operatorname{round}(Q_t + (D_t - C_t)\Delta t)\}, \tag{43}$$

where round converts the result to an integer request count. A capacity reduction produces backlog only when demand exceeds the remaining capacity; existing backlog drains when capacity exceeds demand. Network congestion and storage utilization are demand-to-effective-capacity ratios that contribute separate latency penalties. In particular, storage utilization denotes I/O-capacity pressure, not occupied disk space, and can increase under unchanged I/O demand when storage throughput capacity falls. Queueing, service time, network and storage penalties, and fault-specific penalties jointly determine application latency.

D.4 Agent Interface and Operational Health

Permitted interactions. Detection, localization, and RCA permit observation and explicit time advancement, whereas mitigation additionally exposes a scenario-specific subset of controls. Control families include cooling adjustment, workload migration and throttling, server maintenance, sensor calibration, autoscaler and placement-policy updates, monitoring repair, and load-balancer reconfiguration. Requests are checked against the action interface and the task's permitted operations. Observation alone does not advance time; time-advancement operations execute the requested ticks, and a control may be followed by a specified number of ticks before updated telemetry is returned. The agent receives observable telemetry rather than evaluator-owned fault annotations or scoring state.

Control semantics. Partial placement-policy updates modify supplied fields while preserving omitted fields. Thus, changing a placement strategy does not automatically remove an existing server-count cap. Workload migration moves a fraction of the source rack's current configured allocation, not a fraction of its original allocation; the source must contain allocated workload and the destination must have healthy servers. Consequently, repeated fractional migrations need not fully evacuate a rack, and a successful control invocation does not itself establish recovery.

Aggregate health. The facility SLA is a simulator-wide binary status, marked violated if any rack has critical reported thermal status, exceeds its power limit or operating power budget, or if

the failed-server fraction, workload queue, network-error rate, or application-error rate violates its corresponding condition. The default thresholds are a failed-server fraction above 5%, a queue above 1,000 requests, and any positive network- or application-error rate. Latency, network congestion, and thermal-throttle factors are not direct predicates of this aggregate status.

D.5 Task-Specific Evaluation Criteria

Detection. Rather than treating detection as a binary decision of whether the system is anomalous, we use a stricter success criterion that requires the agent to identify the incident and substantiate its diagnosis. A detection episode is considered successful only if:

- the agent affirmatively identifies an incident;
- the agent correctly specifies the underlying fault mechanism; and
- the agent provides at least one supporting evidence term that appears in its observation history.

Localization. A localization task is successful if the agent identifies the correct faulty component or operational scope. The submitted target must match an identifier or accepted alias defined for the scenario.

Root-cause analysis (RCA). To evaluate RCA, we require the agent to identify not only the underlying fault mechanism, but also where the fault occurs, the operational domain to which it belongs, and the evidence supporting the diagnosis. Concretely, an RCA episode is successful only if:

- the agent correctly identifies the underlying fault mechanism;
- the agent correctly identifies the faulty component;
- the agent correctly identifies the operational domain; and
- the agent provides at least one supporting evidence term that appears in its observation history.

Mitigation. A mitigation task is successful if the agent performs all required control actions before the injected fault expires and restores the system to a healthy state. After the mitigation actions, the evaluator checks the system for ten consecutive simulator ticks. During this period, all scenario-specific health requirements must remain satisfied. The evaluator uses privileged simulator state for this verification.

Because deterministic term matching cannot enumerate all semantically valid ways in which an agent may express a correct diagnosis, we use an LLM-based semantic judge for detection, localization, and RCA tasks. Specifically, in our experiments, GPT-6 Astra with medium reasoning effort evaluates each required component of the agent submission against the scenario ground truth, accounting for semantically equivalent descriptions rather than requiring an exact lexical match. This procedure reduces false negatives caused by differences in wording or output structure while preserving the task-specific correctness criteria defined above.

D.6 Evaluation of the LLM-based semantic judge

To assess the reliability of the LLM-based semantic judge, we audit 54 agent submissions covering all 18 fault mechanisms and the three diagnostic objectives (detection, localization, and RCA). The submissions are randomly sampled from the complete set of agent outputs, with one response selected for each fault-mechanism–objective combination. Two reviewers independently evaluate each submission against the applicable task-specific criteria using the submitted answer, scenario ground truth, and agent-visible observation history. Disagreements are resolved through adjudication to establish the human reference.

We compare the deployed LLM semantic judge with a deterministic matcher based on a human-reviewed, field-specific alias library constructed independently of the audited responses. Both evaluators assess the same submissions under the same task-specific criteria, with a response considered successful only when all applicable requirements are satisfied. We report agreement with the human

Table 8: Human-reference audit of the diagnostic evaluators. FPR and FNR denote false-acceptance and false-rejection rates, respectively.

Evaluator	Correct Rate↑	FPR ↓	FNR ↓
Lexicon-based matcher	87.04%	0	12.96%
LLM semantic judge	100%	0	0

reference together with false-acceptance and false-rejection rates. To assess stability, the LLM judge is run independently three times on each submission.

As shown in Table 8, the LLM semantic judge achieves 100% agreement with the adjudicated human reference across the audited submissions. In comparison, the lexicon-based matcher achieves 87.04% agreement and exhibits a 12.96% false-rejection rate, indicating that exact or alias-based matching can reject semantically correct responses that are expressed differently from the predefined lexical forms.

E Extended Experimental Design

E.1 Dataset Generation

Episode generation. We generate the StateBundle training dataset using the cross-layer simulator with a deterministic sampler over fault mechanisms and targets, topologies, workloads, fault severities, and injection timing. Each episode comprises baseline, active-fault, and recovery phases. Canonical observations are stored separately from privileged simulator-derived labels. Stage 1 constructs training-only positive, hard-negative, and neutral cross-modal pairs, while pair-aware batching ensures that both endpoints of each selected positive pair appear in the same batch. When no reliable cross-modal positive exists, two semantics-preserving views of the same observation provide the positive pair. For Stages 2–3, we sample K_{snap} temporally separated snapshots from the active-fault phase of each episode; we set $K_{\text{snap}} = 4$ in all experiments. Limiting the number of snapshots per episode prevents longer incidents from contributing disproportionately more training examples. Table 9 summarizes the final dataset after quality filtering and stage-specific preparation.

Episodes are assigned to incident-disjoint splits to construct training validation and test set before Stage-1 pair mining and Stage-2/3 snapshot sampling. The generator excludes every seed reserved by the agent benchmark.

Table 9: StateBundle dataset statistics after quality filtering and stage-specific preparation. Pair counts are used only by Stage 1.

Split	Incidents	Snapshots	Observations	Positive	Hard neg.	Neutral
Train	1868	7472	924946	100000	100000	100000
Validation	230	920	115364	10000	10000	10000
Test	206	824	102945	10000	10000	10000

E.2 Agent Configurations

Controlled agents. The controlled agent uses one fixed system prompt, response schema, tool wrapper, and state machine for every LLM backend. Only provider-specific API parameters differ. The detailed model configurations are reported in Table 10. To reduce sampling variability, we set the temperature to 0 for models that support configurable temperature values. For Claude Sonnet 5, Claude Opus 5, and GPT-6 Astra, which do not support custom temperature settings, we use the provider defaults.

Black-box systems. Codex agents are evaluated as complete agent systems through an isolated workspace containing only the benchmark tool bridge and public task instructions.

Table 10: Model and agent-system configuration.

Display name	Provider/system	Reasoning setting	Temperature	Max output	Context limit	Token accounting
DeepSeek-V4-Flash-0731	DeepSeek	Thinking disabled	0	1,024	1,000,000	API I/O
GPT-5.6 Luna	OpenAI	`none`	0	1,024	1,050,000	API I/O
Gemini 3.1 Flash-Lite	Google	`minimal`	0	1,024	1,048,576	API I/O
Claude Sonnet 5	Anthropic	`medium`	Default	1,024	1,000,000	API I/O
GPT-5.6 Sol	OpenAI	`medium`	0	1,024	1,050,000	API I/O
Claude Opus 5	Anthropic	`medium`	Default	1,024	1,000,000	API I/O
GPT-6 Astra	OpenAI	`medium`	Default	1,024	1,050,000	API I/O
Codex	OpenAI	`none / medium`[a]	Unset	Unset	1,050,000	Codex trace I/O

[a] Codex runs backed by GPT-5.6 Luna use reasoning `none`; runs backed by GPT-5.6 Sol and GPT-6 Astra use reasoning `medium`.

Table 11: Sensitivity analysis of the StateBundle telemetry token budget B.

System	Detect ↑	Localize ↑	RCA ↑	Mitigate ↑	Overall ↑	Token usage ↓
Codex (GPT-5.6 Luna, W.S., $B = 2048$)	87.0%	94.4%	72.2%	33.3%	71.8%	82,717
Codex (GPT-5.6 Luna, W.S., $B = 4096$)	90.7%	92.6%	72.2%	50.0%	76.4%	113,484
Codex (GPT-5.6 Luna, W.S., $B = 8192$)	90.7%	87.0%	74.1%	44.4%	74.1%	150,598
Codex (GPT-5.6 Luna, W.S., $B = 16384$)	83.3%	90.7%	77.8%	50.0%	75.5%	174,291

E.3 Sensitivity to the Telemetry Token Budget

To assess the sensitivity of StateBundle to the telemetry token budget B, we evaluate $B \in \{2048, 4096, 8192, 16384\}$ while keeping all other experimental settings fixed. We use Codex backed by GPT-5.6 Luna as the agent for this analysis. As shown in Table 11, increasing the budget from 2048 to 4096 improves overall success from 71.8% to 76.4%. Further increasing the budget, however, does not lead to monotonic improvements, while token usage increases steadily as the budget grows. These results indicate that providing additional telemetry beyond a moderate budget does not necessarily improve downstream agent performance.

E.4 Analysis of High-Failure Tasks

Placement-policy mitigation. For placement-policy misconfiguration mitigation, failures are dominated by incomplete restoration of the effective scheduling state. Agents often repair the placement strategy while leaving an independent server-allocation cap unchanged.

Intermittent-server mitigation. For intermittent-server mitigation, failures arise primarily from incomplete containment and insufficient temporal verification. Agents commonly identify the unstable host but reverse isolation prematurely.

Based on these dominant failure modes, the two tasks suggest a common intrinsic limitation in *goal-directed planning*. Repairing the placement strategy and isolating an unstable host are both locally appropriate interventions, but agents often treat such local success as a stopping criterion without verifying whether the resulting system state satisfies the complete recovery objective. This pattern motivates closed-loop decision-making with feedback-driven replanning, in which agents continuously evaluate recovery progress and revise their actions until the operational objective is satisfied.

Table 12: Selected high-failure tasks and the domains involved.

Fault mechanism	Task	Recorded success	Domains involved
Placement-policy misconfiguration	Mitigation	6.7%	Control plane, compute, service performance
Intermittent server failure	Mitigation	13.3%	Host availability, placement, service performance

E.5 Analysis of Common Failure Mechanisms of Claude Opus 5

Although Claude Opus 5 performs strongly on broad external capability evaluations (Artificial Analysis, 2026), it achieves only moderate performance on DC-Bench. To characterize the operational

Table 13: Failure patterns for Claude Opus 5.

Failure pattern	Episodes	Total	Percentage
Over-inference from downstream symptoms	58	129	44.96%
Inadequate temporal reasoning over evolving system states	18	129	13.95%
Premature termination before stable recovery is established	22	129	17.05%

behaviors associated with these outcomes, we conduct a focused case study of recurring failure patterns in agents backed by Claude Opus 5.

Table 13 summarizes these patterns across all three trials under both *Full-Canonical* and STATE-BUNDLE. Each percentage denotes the proportion of all recorded failed episodes in which the corresponding pattern is observed, regardless of whether it is the primary cause of failure. The categories are therefore not mutually exclusive, and one failed episode may contribute to multiple categories.

Over-inference from downstream symptoms. This is the most frequently observed pattern, occurring in nearly half of the failed episodes. The agent recognizes evidence generated downstream of the actual fault but interprets these symptoms as an additional or alternative root cause without sufficient supporting evidence.

For example, the agent observes an explicit load-balancer imbalance alert, 84.46% unhealthy routing, and a reduction in service capacity from 24,000 to 1,184 requests/s, yet additionally concludes that there is a "simultaneous compute resource failure." The capacity reduction is consistent with the observed routing impairment and does not, by itself, establish a separate compute failure. This case illustrates the importance of distinguishing downstream consequences from independent causes and testing additional hypotheses against the available evidence.

Inadequate temporal reasoning over evolving system states. This pattern arises when the agent attributes an incident to a condition that already existed during healthy operation, without identifying the change that explains the onset of the anomaly.

For example, the agent attributes a power-budget incident to concentrated workload placement. However, the same placement was already present at $t = 0$, while rack power remained approximately 1.35 kW throughout the healthy baseline. Concentrated placement may still be a contributing condition, but its presence alone does not explain why the violation emerged later. The diagnosis therefore requires an explicit comparison between the baseline and incident states to distinguish persistent conditions from fault-induced changes.

Premature termination before stable recovery is established. This pattern occurs when an agent improves part of the system but terminates before verifying that the remaining bottlenecks and operational violations have been resolved. It also reflects a tendency to equate locally appropriate interventions with successful end-to-end recovery, a behavior that also recurs in the high-failure tasks shared across agents discussed in Appendix E.4.

For example, in a network-congestion mitigation episode, the agent improves compute allocation and drains the request queue, but submits while mean latency and network congestion remain high:

$$\text{mean latency} = 4511.38\ \text{ms}, \qquad \text{network congestion} = 30.3611. \tag{44}$$

Clearing the queue demonstrates a local improvement, not complete end-to-end recovery. The remaining network impairment requires further investigation and intervention, followed by verification that recovery persists over time.

These findings suggest that model capability alone may not translate into effective data-center operation. Improving the overall agent architecture and operational workflow is therefore a promising complement to advances in the underlying models.