How Do LLMs Reason About Distributed Systems Logs?

Marko Georgiev
Ss. Cyril and Methodius University
Skopje, North Macedonia

Mohamed Hassaan M Hydher
Graz University of Technology
Graz, Styria, Austria

Carlo Alberto Boano
Graz University of Technology
Graz, Styria, Austria

Olga Saukh
Graz University of Technology
Graz, Styria, Austria

Abstract

Large Language Models (LLMs) are increasingly used for system monitoring and log analysis, yet little is known about their ability to reconstruct execution semantics from long traces. We study this problem using distributed-system logs annotated with vector clocks, which support global-state inference tasks with exact ground truth: event concurrency, the latest consistent cut, and the recovery line. We benchmark a range of hosted and self-hosted LLMs on synthetic traces of up to 50'000 events. Models that load the full log as a file and invoke code execution solve these tasks reliably, often near-perfectly, even on the longest traces. In contrast, current local models fail on concurrency counting and achieve substantially lower accuracy on the remaining tasks. We additionally evaluate fault diagnosis on logs from the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL), where faults must be inferred from protocol behavior rather than computed directly. Our results suggest that accurate log analysis depends strongly on full-log access and structured computation, highlighting a key challenge for deploying local LLM-based monitoring.

CCS Concepts

• **Computer systems organization** → **Dependable and fault-tolerant systems and networks**; • **Software and its engineering** → *Software defect analysis*; • **Computing methodologies** → *Natural language processing*.

Keywords

Distributed systems, Large Language Models, Log analysis, Vector clocks, Consistent cut, Recovery line, IoT monitoring.

1 Introduction

Distributed IoT and edge deployments consist of many low-cost, failure-prone devices running unattended in harsh environments, so monitoring is essential and logs are often the only source of observability. Monitoring systems built for servers assume infrastructure that these deployments lack, and the logs themselves require semantic interpretation, which makes LLMs an attractive alternative. Each node records its local events and message exchanges, and global system behavior must be reconstructed from these partial views. Classical distributed-systems theory formalizes several aspects of this reconstruction, including causal ordering and concurrency between events [7], consistent global state [1], and the recovery line required for safe rollback [3]. These questions admit exact answers that can be derived from the log alone, making them a useful testbed for evaluating how well LLMs reason about system execution traces.

LLMs are increasingly applied to log analysis, primarily for open-ended tasks such as fault detection and diagnosis [5, 12], log parsing and explanation [8], root-cause analysis in cloud and microservice systems [2, 13], natural-language summarization of distributed executions [11], and benchmarking whether LLMs can locate the root cause of failures from such logs [15]. These tasks are typically evaluated against human-labeled fault categories, anomaly flags, or qualitative explanations. In contrast, much less is known about how well LLMs perform on formal log-analysis tasks with objectively verifiable answers, where correctness can be determined exactly from the trace itself, without human judgment or annotation.

That is the setting we study. We evaluate the three global-state inference tasks above on synthetic vector-clock traces: counting concurrent event pairs, identifying the latest consistent cut, and identifying the latest recovery line. Because the traces are generated by simulation, exact ground truth is available for every instance, enabling exact scoring. The traces range from 100 to 50'000 events. We benchmark five hosted LLMs and seven self-hosted open-weight models (4B–70B parameters), and we qualitatively analyze the reasoning traces they expose. As a complementary benchmark, we also study fault diagnosis in logs generated from the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [14], a routing protocol widely used in IoT deployments. Unlike the vector-clock tasks, where answers can be computed directly from formal definitions, RPL diagnosis requires inferring network faults from protocol behavior observed in the logs. The benchmark is again generated with known ground truth, allowing exact evaluation.

Three observations emerge from the results. First, models that remain near-perfect on the global-state inference

tasks combine access to the full log with code execution. Second, performance depends strongly on the task: concurrency counting, despite its simple definition, is not solved reliably without code execution, whereas consistent-cut and recovery-line inference improve with model capability. Third, the RPL benchmark presents a different challenge: the logs are short and require little computation, yet fault diagnosis remains difficult, suggesting that protocol understanding rather than formal inference is the limiting factor.

This paper contributes: (i) two log-analysis benchmarks with automatically generated ground truth: three global-state inference tasks on vector-clock traces and an RPL fault-diagnosis benchmark for IoT routing logs; (ii) an exact-scored evaluation of five hosted and seven self-hosted open-weight LLMs on traces ranging from 100 to 50'000 events, complemented by a qualitative analysis of the solution strategies exposed by the models; and (iii) an empirical characterization of the capabilities required for accurate log analysis across both benchmarks. These findings highlight the challenges of deploying local LLM-based log analysis, where full-log access and code execution are often unavailable.

2 Tasks and Benchmarks

2.1 Global-State Inference from Logs

The traces are generated from a distributed system with four communicating processes (Alice, Bob, Carol, and Dave). Each event is annotated with a vector clock, a four-component logical timestamp with one component per process. Vector clocks are updated in the standard way: a process increments its own component on each event, while a receive event additionally merges the sender's clock by componentwise maximum [4, 9]. This representation makes causality directly testable: event A causally precedes event B iff $VC(A) < VC(B)$ componentwise (i.e., $VC(A) \leq VC(B)$ and at least one component is strictly smaller). If neither vector clock dominates the other, the events are concurrent. Each log entry records the process, event type, and vector clock:

```
Alice  SEND_EVENT      VC=[2,0,0,0]
Bob    RECEIVE_EVENT   VC=[2,2,0,0]
Dave   CHECKPOINT      VC=[0,0,0,3]
Dave   RECEIVE_FAILED  VC=[0,0,0,5]
```

The event types are process start, send, receive, failed receive (a lost message: the receiver ticks its clock but merges nothing), checkpoint, process failure, and three kinds of local work. Over such a trace, we pose three questions.

Concurrency counting. How many unordered pairs of distinct events are concurrent? The answer is a single integer.

Latest consistent cut. A cut selects one event from each process. It is consistent if it is causally closed, meaning that no receive event is included without its corresponding send. The task is to identify the latest such cut and report it as one vector clock per process. In our traces, the complete execution forms a consistent cut, so the latest consistent cut is simply the final event of each process. The task therefore reduces to locating and extracting these states from the log.

Latest recovery line. This task seeks the latest consistent cut restricted to checkpoint events [3]. Unlike the previous task, the most recent checkpoint on each process does not necessarily form a consistent global state. The task is therefore to identify the most recent causally consistent set of checkpoints, representing the point from which a coordinated rollback can safely resume execution.

Each task has a uniquely determined answer derived from the trace and the task definition: an integer for concurrency counting and sixteen vector-clock values (four processes by four components) for each cut. All three tasks rely on the same core operations: comparing vector clocks, reasoning about causality, and enforcing consistency across processes.

We generate the traces using an event-based simulator of the above system. Because the simulator has access to the complete execution, it produces each trace together with the exact answer to all three tasks, enabling exact evaluation without manual annotation. We release the generator, traces, and ground-truth labels as part of the benchmark[1].

We evaluate 26 traces spanning ten lengths from 100 to 50'000 events (five traces at 100; three each at 200, 500, 1'000, 1'500, and 2'000; two each at 5'000 and 10'000; and one each at 20'000 and 50'000). These sizes are chosen to stress-test the models and identify where performance begins to degrade. We stop at 50'000 events for two reasons. First, exact ground-truth generation becomes increasingly expensive, with the concurrency-counting task requiring tens of millions of event-pair comparisons. Second, models that successfully process a 50'000-event trace typically do so by loading the log as a file and delegating the computation to code, making larger trace processing qualitatively similar from the model's perspective.

2.2 RPL Fault Diagnosis

The vector-clock tasks have formally defined answers. To assess whether the same performance patterns hold on a more operational problem, we constructed a companion benchmark on logs from RPL—the IETF standard routing protocol for low-power IoT networks [14]—in which the fault behind a log must be inferred from observed symptoms rather than computed from a definition. The protocol organizes the nodes into a destination-oriented directed acyclic graph rooted at a border router: each node selects a preferred parent toward the root by an objective function, advertises the graph with DIO messages paced by the Trickle timer, and registers downward routes with DAO messages [14]. What

[1] https://github.com/hassaanhydher/llm-distributed-logs

Table 1: The seven injected RPL fault types. Each is injected once into an otherwise healthy run; because the log contains no explicit fault marker, the diagnosis must be inferred from the protocol's reactions.

Fault	What happens
Blackhole	A node stops forwarding traffic, causing downstream nodes to lose connectivity and eventually time it out.
Asymmetric link	A link becomes unreliable in one direction, causing only one endpoint to perceive the degradation.
Link degradation	A link degrades in both directions, causing nodes to abandon it and select new parents.
Parent churn	A node oscillates between two parents, re-selecting and re-announcing repeatedly.
Silent cycle	Two nodes pick each other as parent from stale ranks, forming an undetected loop.
DAO failure	A node's unicast is blocked, so its downward-route (DAO) registration never reaches the root.
Trickle storm	Every node resets its Trickle timer at once, flooding the network with control traffic.

makes diagnosis hard is that faults seldom appear as explicit errors: a degraded link or a routing loop appears only through the protocol's reactions, such as a parent switch, a liveness timeout, a rank change, or a burst of control traffic [6]; moreover, different faults often produce overlapping symptoms.

A custom simulator runs a small RPL network consisting of five or six nodes (one acting as root, the rest as source/-forwarding devices) for three minutes, and generates logs modeled after Contiki-NG [10] RPL output. Into an otherwise healthy run, exactly one fault is injected, making its type, location, and onset time known precisely. Several of the faults are internal protocol misbehaviors that a Contiki-NG/Cooja setup does not expose as controls; reproducing them there would mean altering the routing implementation for each scenario. The benchmark covers seven fault types (Table 1) plus a healthy control, with two seeds per scenario, yielding sixteen logs in total. Like the vector-clock tasks, it is scored exactly and automatically, but the challenge shifts from formal inference over long traces to interpreting protocol behavior. We keep the networks small so that a fault's signature stays visible rather than buried in traffic from many nodes.

3 Study Design

This section describes the evaluated models, prompting and scoring, and our analysis of the models' visible reasoning.

3.1 Models

We evaluate twelve models in two groups: five hosted assistants accessed through their chat interfaces and seven self-hosted open-weight models. The hosted group comprises ChatGPT 5.5-Thinking, ChatGPT o3, Claude Opus 4.7, Gemini 3.5 Flash-Extended, and DeepSeek Expert mode[2]. These interfaces differ in their support for log analysis, including file upload, code execution, and accepted file size. The self-hosted group comprises Gemma 4 at three sizes, Llama 3.1 at 8B and 70B, Qwen3-30B, and a distilled DeepSeek-R1 8B, all served from a local GPU server through an API. We report the self-hosted results separately in Section 4.4. The RPL benchmark uses the seven self-hosted models and all five hosted ones.

3.2 Prompts and Scoring

Each global-state inference task uses a fixed zero-shot prompt applied verbatim across all models and traces. The prompt provides the relevant definition, specifies the task and output format, and appends the trace, with no worked examples. For the consistent-cut and recovery-line tasks, models are also asked to provide a brief justification, which is not scored. For RPL, the prompt lists the fault types in Table 1 and requests a JSON output with the fault type, affected nodes, and onset time. All prompts are released with the benchmark[1].

Concurrency counting is scored by exact match against the ground-truth count. For the consistent-cut and recovery-line tasks, we assign partial credit based on the number of correctly predicted vector-clock values (out of sixteen). This allows us to quantify how answer quality degrades as traces grow. One RPL diagnosis is counted as correct only when the fault type, affected nodes, and onset time (within 15 seconds of the ground truth) all match, and reporting any fault on a healthy log counts as a failure. Empty, malformed, or unreadable responses are scored as failures.

3.3 Capturing Reasoning

We examine the reasoning traces exposed through the hosted models' interfaces in addition to task accuracy for the vector-clock tasks. We identify recurring solution strategies, such as delegating the computation to a script when code execution is available and relying on textual analysis when it is not, and discuss these qualitatively in Section 4.1. Because we do not apply a formal faithfulness measure, the exposed reasoning traces are treated as descriptive artifacts rather than evidence of the models' internal reasoning processes.

4 Results and Discussion

We first examine how the models approach the vector-clock tasks, then evaluate their accuracy and characterize the conditions under which performance degrades. We next consider the RPL benchmark, where success depends on interpreting protocol behavior rather than performing formal

[2]Shortened from here on to GPT-5.5, o3, Opus 4.7, Gemini, and DeepSeek.

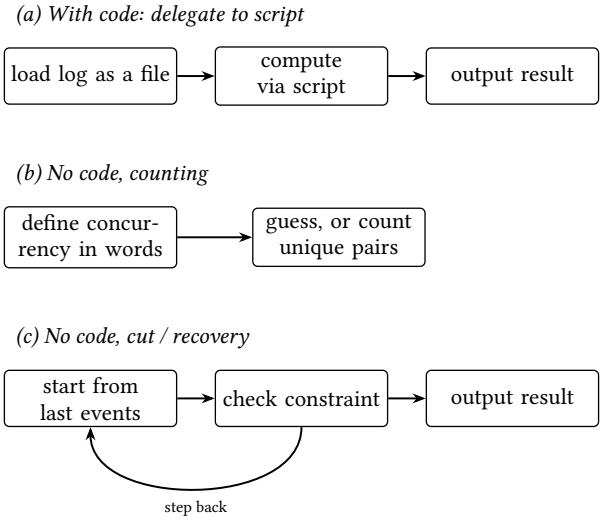

Figure 1: Solution strategies inferred from reasoning traces. With code execution, the strongest models delegate computation to a script (a). Without it, concurrency counting often reduces to guessing (b), whereas the consistent-cut and recovery-line tasks elicit stepwise search procedures (c).

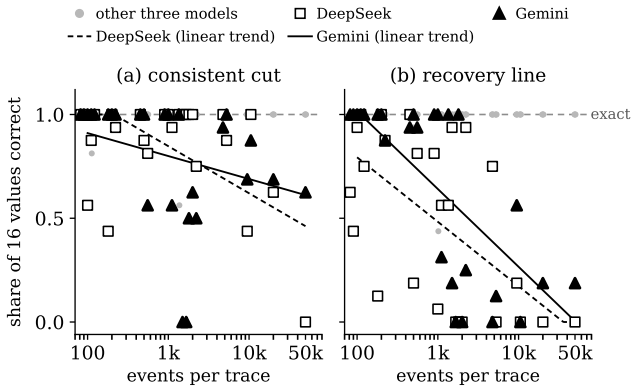

Figure 2: Partial credit on the consistent-cut (a) and recovery-line (b) tasks, measured as the fraction of the sixteen vector-clock values matching the ground truth. Each marker represents one trace (1–5 traces per length); markers on the gray dashed line at 1.0 are exact answers. Black trend lines show linear fits for Gemini (solid) and DeepSeek (dashed). Small gray markers denote GPT-5.5, o3, and Opus 4.7. DeepSeek returned no answer at 50'000 events.

inference. Finally, we evaluate the self-hosted open-weight models across both benchmarks.

4.1 Solution Strategies

Examining the reasoning traces exposed by the hosted interfaces, we find one strategy that recurs across successful runs and several others that recur on failures (Figure 1). We treat these patterns as observations rather than a verified account of the models' internal processes. The most striking distinction is the use of code execution: successful runs consistently delegate the computation to a script, whereas failures rely on purely textual approaches.

When GPT-5.5, o3, Gemini, and Opus 4.7 answer correctly, they consistently follow the same delegate-to-script workflow: the log is loaded as a file, a short script is used to compute the answer, and the result is returned with little accompanying explanation. This pattern appears across all three tasks. Gemini alone adds an explicit verification step, re-running the computation on a subset of the log before reporting the final answer.

The observed failures occur only in runs without code execution, and their form depends on the task. For concurrency counting, models typically state the definition correctly but, unable to carry out the required computation, either guess the answer or report the total number of event pairs rather than the concurrent ones. For the consistent-cut and recovery-line tasks, they begin from the latest events, check the consistency constraints, and step backwards iteratively, performing in text the search that a script can automate. Gemini follows these patterns whenever it does not generate code and often concludes with a confident but incorrect answer. DeepSeek, which lacks code execution entirely, relies on them throughout and frequently acknowledges that its answer is likely incorrect before giving it; its exact solutions for the consistent-cut task are limited to the trivial cases where the answer is simply the final events. Across the hosted models, successful runs consistently involve code execution, whereas failed runs rely on textual analysis alone.

Table 2: Exact answers on vector-clock tasks over the 26 traces (100 to 50'000 events).

Task	GPT-5.5	o3	Opus 4.7	Gemini	DeepSeek
Counting	25	25	25	22	0
Cut	26	26	24	14	14
Recovery	25	25	25	12	3

Table 3: RPL fault diagnosis. A full diagnosis requires the fault type, affected nodes, and onset time (within 15 seconds) to all be correct, over all sixteen logs; the fault-type row counts the fourteen fault-injected logs.

	GPT-5.5	o3	Opus 4.7	Gemini	DeepSeek
Full diagnosis	5/16	8/16	8/16	6/16	4/16
Fault type	8/14	11/14	11/14	9/14	10/14

4.2 Exact-Match Correctness

Table 2 reports exact-answer accuracy across the 26 traces. The three strongest models (GPT-5.5, o3, and Opus 4.7) remain near-perfect on all three tasks as trace length increases to 50'000 events. Together, they miss only eight of 234 answers, all on traces of at most 2'000 events. From 5'000 events onward, all three models answer every task correctly. The errors therefore appear as isolated failures rather than a systematic effect of trace length.

Gemini's performance differs sharply across tasks. Its concurrency counts are exact at every size from 1'000 to 20'000 events and break only at 50'000, where the answer is off by four orders of magnitude. The three remaining misses occur on small traces of 200 to 500 events. In contrast, exact answers for the consistent-cut and recovery-line tasks become increasingly rare beyond a few hundred events and disappear entirely above 10'000. Figure 2 shows how the errors evolve: even at 50'000 events, Gemini recovers most of the sixteen values in the consistent cut, whereas recovery-line accuracy drops sharply and retains only a small fraction of the correct values beyond 5'000 events. Trace length alone therefore does not explain Gemini's behavior; performance also depends strongly on the task.

DeepSeek, the only large model evaluated without code execution, shows an uneven profile. None of its 26 counts are exact and only three recovery lines are correct, yet on the consistent cut it matches Gemini at 14 of 26. These are the cases where the cut reduces to each process's final events, which can be read off rather than computed, and partial credit holds up through the mid-sized traces (Figure 2a). On the 50'000-event trace, it returned no answer on either task.

No single limitation explains the failures. DeepSeek performs worst on concurrency counting, the task with the simplest rules but the greatest computational burden, requiring comparisons between all pairs of events. It fails even on 100-event traces, which are far smaller than any relevant context limit, suggesting that the difficulty lies in the computation rather than in processing the trace. Gemini exhibits a different pattern: its concurrency counts remain exact up to 20'000 events, whereas exact answers for the consistent-cut and recovery-line tasks begin to disappear at much smaller sizes. Only at 50'000 events does performance collapse for both models. The reasoning traces provide a possible explanation. Successful runs consistently delegate the computation to code, whereas unsuccessful runs rely on textual analysis alone (Section 4.1). This distinction is particularly visible on the counting task, where code execution naturally offloads the pairwise comparisons required by the definition. We therefore interpret the results as consistent with the view that accurate performance on these tasks depends on a combination of full-log access and code execution.

Table 4: Open-weight models hosted and run in context, without tools, on the 22 traces up to 5'000 events that fit their context windows. No model produced a single correct concurrency count, so only the consistent-cut and recovery-line results appear; smaller (≤8B) models on the left, larger on the right.

Model	Cut	Recovery	Model	Cut	Recovery
Gemma-4-E4B	6	3	Gemma-4-26B-A4B	14	6
Llama-3.1-8B	2	2	Qwen3-30B	14	2
DeepSeek-R1-8B	0	0	Gemma-4-31B	4	9
			Llama-3.1-70B	1	2

4.3 RPL Observations

The contrast with the vector-clock tasks is instructive (Table 3). The hosted models achieve a complete diagnosis on only four to eight of the sixteen logs and identify the correct fault type on eight to eleven of the fourteen fault-injected logs. The errors are not random. All models fail both silent-cycle instances; GPT-5.5, o3, Opus 4.7, and DeepSeek correctly identify the fault and affected nodes but place the onset outside the 15-second tolerance window, consistent with a fault whose primary symptom is the absence of expected protocol activity. Across all models, onset time is the most error-prone component of the diagnosis. Unlike the vector-clock tasks, these logs are short and place little demand on context length or computation. The challenge instead lies in interpreting protocol behavior, where access to the full log and code execution offers little apparent advantage.

4.4 Local and Deployable Models

Table 4 turns to the open-weight models that we host and run in context, without tools, as the deployable counterpart to the hosted assistants above. The limiting factor is context: none of these models ingests a log beyond 5'000 events, so the table covers the 22 traces up to that length. The longer logs do not fit, which is itself a deployment limitation.

Concurrency counting stands apart. Not a single self-hosted model, spanning 4B to 70B parameters, produces a correct count on any trace[3]. Increasing model size does not overcome the need to perform the underlying computation: all of these models can read the trace, but none can reliably carry out the pairwise comparisons required by the task. By contrast, models with access to code execution solve the same task almost perfectly.

[3]In preliminary tests, smaller models (Llama 3.2 1B and 3B, Qwen 2.5 3B, and a distilled DeepSeek-R1 1.5B) also failed all three tasks at every size, and produced at most one correct RPL diagnosis out of sixteen.

The consistent-cut and recovery-line tasks show a different pattern, and performance on one does not imply performance on the other. Qwen3-30B answers 14 of 22 consistent-cut instances correctly but only 2 recovery-line instances, whereas Gemma-4-31B shows the opposite tendency, with 4 correct cuts and 9 correct recovery lines. The stronger recent models achieve non-trivial success on at least one of the two tasks—Gemma-4-26B-A4B and Qwen3-30B both reach 14 of 22 on the consistent cut—and even many incorrect answers remain close to the ground truth, with three models averaging over 0.8 partial credit on that task. Performance therefore tracks model capability more closely than parameter count: the 70B model is not the strongest in the group, whereas a 26B-class model is. The weakness is not uniform across tasks but concentrated on concurrency counting, the only task that requires large-scale computation rather than extracting or searching for structure in the trace. This mirrors the pattern observed among the hosted models.

The same models also underperform on the RPL benchmark, where neither long traces nor substantial computation are required: their full-diagnosis rates range from 0 to 3 of 16, compared with 4 to 8 of 16 for the hosted models. Across both benchmarks, the self-hosted models remain the furthest from reliable log analysis. Whether code execution can narrow this gap is an open question.

5 Conclusion and Future Work

We evaluated LLMs on three formally defined global-state inference tasks over vector-clock logs—counting concurrent event pairs, identifying the latest consistent cut, and identifying the latest recovery line—using traces generated together with exact ground truth. Across five hosted models and seven self-hosted open-weight models, the strongest performance came from systems that combined access to the full log with code execution. Models without code execution, regardless of size, consistently struggled on concurrency counting, the task with the greatest computational demands. On an RPL fault-diagnosis benchmark, where the logs are short and computation is not the primary challenge, performance was substantially lower and the difficulty shifted to interpreting protocol behavior. Together, these results suggest that accurate log analysis depends on a combination of full-log access, external computation, and domain understanding.

Our study has several limitations. The evaluation covers 26 vector-clock traces and 16 RPL logs, so the reported accuracies should be viewed as indicative rather than definitive. It also does not separate model capability from tool support: no model was evaluated both with and without code execution. Our analysis of model reasoning relies solely on interface-exposed traces and lacks a formal faithfulness measure. Thus, the solution strategies in Section 4.1 should be viewed as observations of model behavior rather than verified accounts of the mechanisms underlying its answers.

Several directions remain open. The most immediate is to equip self-hosted open-weight models with code execution and evaluate whether this closes the gap on computation-heavy tasks such as concurrency counting. More broadly, the benchmarks can be expanded with more, larger, and noisier traces, extra processes and failure modes, richer RPL scenarios, and logs from real deployments. Further directions include simpler global-state tasks that better separate trace interpretation from computation, and quantitative methods to evaluate the faithfulness of exposed reasoning traces.

Acknowledgments. This research was funded in whole, or in part, by the Austrian Science Fund (FWF) [10.55776/DFH5]. For the purpose of open access, the author has applied a CC BY public copyright license to any Author Accepted Manuscript version arising from this submission. The computational results have been achieved using the Austrian Scientific Computing (ASC) infrastructure.

References

[1] K.M. Chandy and L. Lamport. 1985. Distributed Snapshots: Determining Global States of Distributed Systems. *ACM Trans. Comput. Syst.* 3, 1 (1985), 63–75.

[2] Y. Chen et al. 2024. Automatic Root Cause Analysis via Large Language Models for Cloud Incidents. In *Proc. of the 19th EuroSys Conference*. 674–688.

[3] E.N. Elnozahy et al. 2002. A Survey of Rollback-Recovery Protocols in Message-Passing Systems. *ACM Comput. Surv.* 34, 3 (2002), 375–408.

[4] C.J. Fidge. 1988. Timestamps in Message-Passing Systems That Preserve the Partial Ordering. In *Proceedings of the 11th Australian Computer Science Conference*. 56–66.

[5] Y. Ji et al. 2025. Adapting Large Language Models to Log Analysis with Interpretable Domain Knowledge. In *Proc. of the 34th CIKM Conference*.

[6] H.-S. Kim et al. 2017. Challenging the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL): A Survey. *IEEE Commun. Surveys Tuts.* 19, 4 (2017), 2502–2525.

[7] L. Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM* 21, 7 (1978), 558–565.

[8] Y. Liu et al. 2024. Interpretable Online Log Analysis Using Large Language Models with Prompt Strategies. In *Proc. of the 32nd ICPC Conference*. 35–46.

[9] F. Mattern. 1989. Virtual Time and Global States of Distributed Systems. In *Parallel and Distributed Algorithms*. North-Holland, 215–226.

[10] G. Oikonomou et al. 2022. The Contiki-NG open source operating system for next generation IoT devices. *SoftwareX* 18 (2022), 101089.

[11] D. Schiese and A. Both. 2025. Post-hoc LLM-Supported Debugging of Distributed Processes. In *Proc. of the 25th ICWE Conference*. arXiv:2508.14540.

[12] Y. Sun et al. 2025. Accurate and Interpretable Log-Based Fault Diagnosis Using Large Language Models. *IEEE Trans. Serv. Comput.* 18, 5 (2025), 2602–2615.

[13] P. Tang et al. 2025. MicroRCA-Agent: Microservice Root Cause Analysis Method Based on Large Language Model Agents. arXiv:2509.15635.

[14] T. Winter et al. 2012. *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*. RFC 6550. IETF.

[15] J. Xu et al. 2025. OpenRCA: Can Large Language Models Locate the Root Cause of Software Failures?. In *Proc. of the 13th ICLR Conference*.