WPI

ByteDance

ICDE 2026
MONTRÉAL, CANADA

KRONE: Hierarchical and Modular Log Anomaly Detection

Presenter: Prof. Elke Rundensteiner

Lei Ma∗, Jinyang Liu†, Tieying Zhang†, Peter M. VanNostrand∗, Dennis M. Hofmann∗, Lei Cao‡, Elke A. Rundensteiner∗, Jianjun Chen†

∗ Worcester Polytechnic Institute, Worcester, USA
† ByteDance Inc., San Jose, USA
‡ University of Arizona, Tucson, USA

SECTION 1

INTRODUCTION

Why Log Anomaly Detection Matters?

Logs are software's nervous system — unreadable at scale without automation.

Security & Reliability

Ground truth for intrusions and SLA breaches.

Scale & Complexity

TB-scale daily across hundreds of services.

Operational Cost

Outages cost thousands/minute; alerts drown real incidents.

The Problem of Flat Log AD Paradigm

Log sequence | L_1 | L_2 | L_3 | L_4 | L_5 | L_6 | ... |

billions per hour !

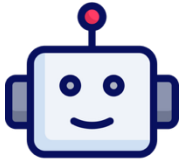
Deep Learning Models

LSTM[1], Transformer[2]...

Anomaly score = 0.6

Uninterpretable Detection

LLM detection [3,4]

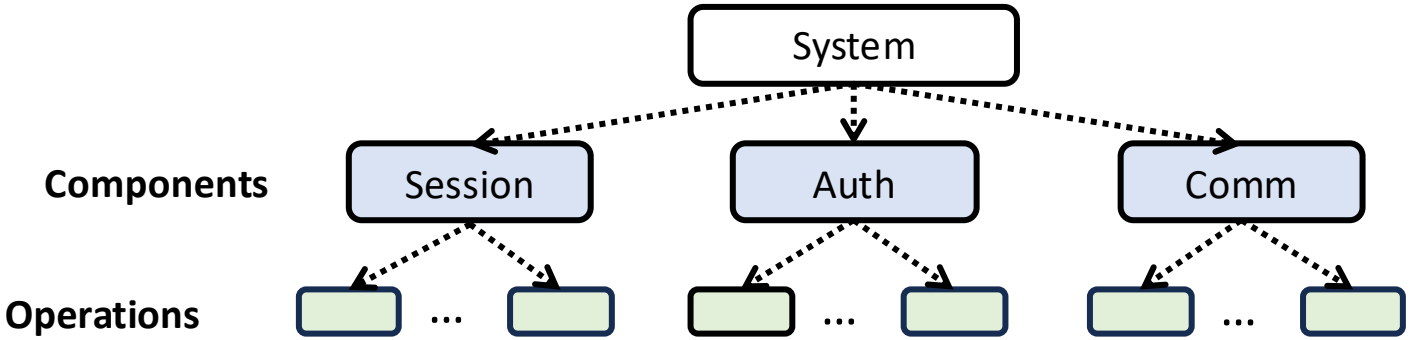


Context

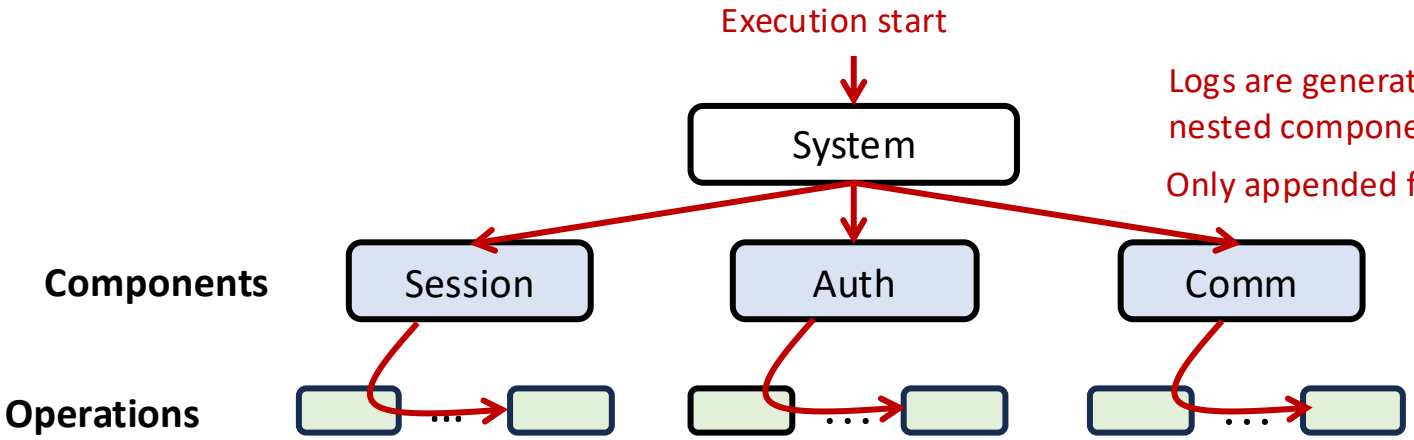
Context window limit & Huge Cost!

[1 -4] See Reference Page.

However, System Executions Are Hierarchical



However, System Executions Are Hierarchical



Logs are generated **hierarchically** by nested components and operations.

Only appended flatly.

Logs

1	Open session started	Session
2	Session Opened successfully	
3	Auth start	Auth
4	Auth succeeds	
5	Comm: GET_request	Comm
6	Comm: GET_response	

Logs for session operations

Logs for Auth operations

Logs for Comm operations

An Example of Hierarchical Anomaly Detection

Hierarchy = natural execution boundaries for structured anomaly detection

- **Low: Intra-Component**

1	Open session started
2	Session Opened successfully
3	Auth start
5	Comm: GET_request
6	Comm: GET_response

Session ✓

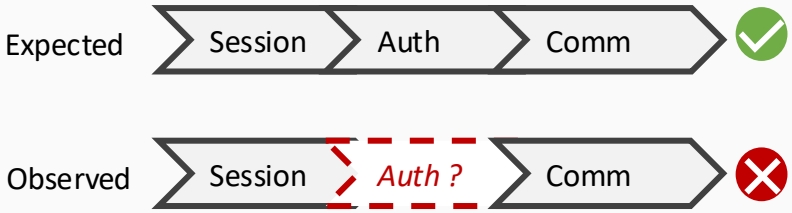
Auth ✗

Comm ✓

Intra-component Anomaly

Sessions look fine, but Auth is incomplete

- **High: Across-Component**



Inter-component Anomaly

Each component runs fine, but Auth never replies

Benefits of the Hierarchical Log AD Paradigm

1 Modular Detection

- Reusable atomic units + permutations
- Sequence detection → modular local tasks

2 Scalable LLM Log AD

- LLMs run on compact semantic-coherent units, not full traces
- Lower cost, latency, context

3 Localization & Explanation

- Anomalies bound to execution scope
- Grounded in semantic meaning
- Clearer root-cause analysis

KRONE Leads on Accuracy, Resource-Efficiency, and Interpretability

KRONE achieves the following on HDFS dataset:

Cheaper detectors, sharper results.

Pattern Matching

42% → 88%

F1-score with or without KRONE (using **only 1% training data**)

Smarter LLMs, lighter bills.

LLM Cost

$2.89 with GPT-5

On **1.1M log messages (10% HDFS)**

- ✓ Knowledge accumulation & reuse across sequences
- ✓ Automatic Anomaly Localization inside Log Sequences
- ✓ Knowledge-grounded LLM explanations

[5] See Reference Page.

Challenges of Hierarchical Log Anomaly Detection

C1 Abstracting Hierarchy from Black-Box Systems

No structure, no domain knowledge

- Systems expose no explicit structure
- Infer hierarchy from logs alone

C2 Lightweight Modeling 4 Massive Logs

Billions of logs, no hard-coded rules

- Heavy models infeasible at scale
- Manual rule maintenance breaks down
- Need efficient, adaptive abstraction

C3 Detection under Nested Semantics

Same high-level, different low-level

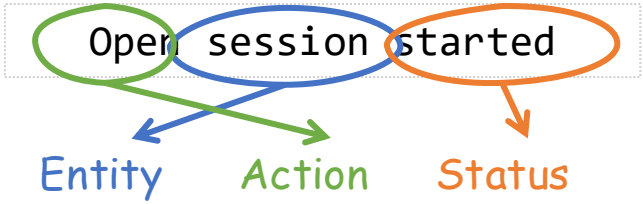
- High-level only: fast but coarse
- Incorporating Low-level detail: accurate but costly
- Balance fidelity with efficiency

SECTION 2

KRONE LOG ABSTRACTION MODEL

Our Observation — Hidden Hierarchy in Log Semantics

An example log message



A log message usually describes a **status** as the outcome of an **action** on an **entity.**

Log Templates

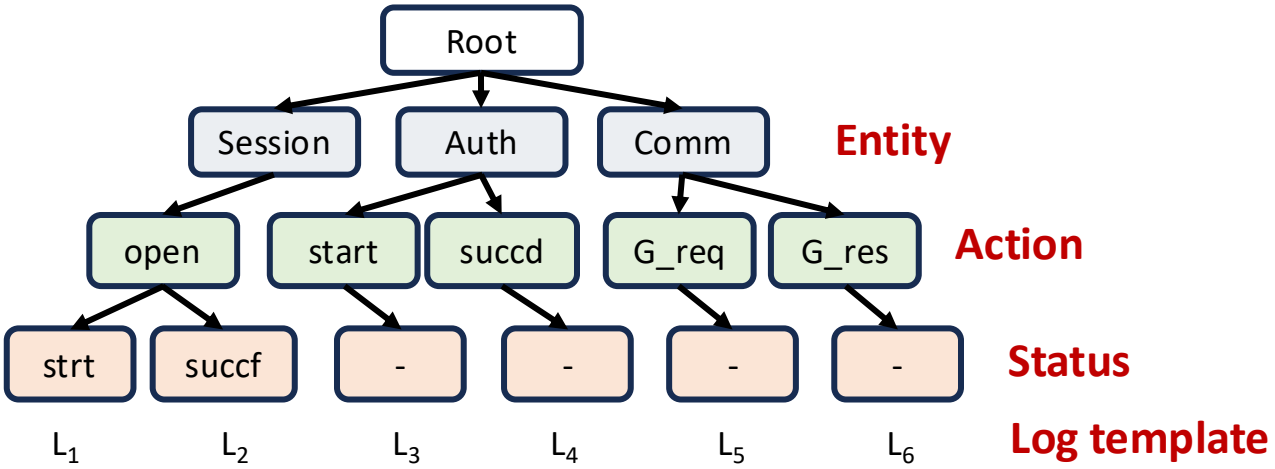
Index	Log Template
1	Open session started
2	Session Opened successfully
3	Auth start
4	Auth succeeds
5	Comm: GET_request
6	Comm: GET_response

Semantic Structure

Entity	Action	Status
Session	Open	started
		succfl
Auth	start	-
	succeed	-
Comm	GET_req	-
	GET_res	-

KRONE Log Abstraction Model

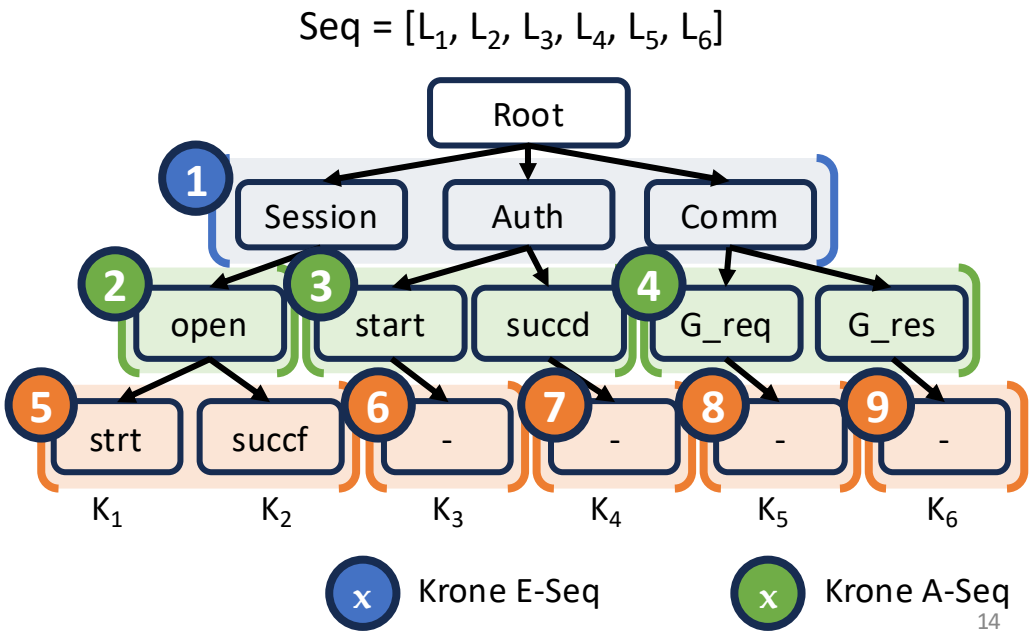
KRONE-Tree: Encodes the relationships between entity, action, status, and the log templates.



KRONE Log Abstraction Model

Log Sequence Decomposition: Log sequence → structured set of KRONE-Seqs

Log Sequence Detection → Modular detections on KRONE-Seqs



KRONE-Seqs

1. [Session, Auth, Comm]
2. Session→[open]
3. Auth → [start, succeeds]
4. Comm → [GET_request, GET_response]
5. Session → open →[started, successful]
6. Auth → start → [-]
7. Auth → succeeds → [-]
8. Comm → GET_request →[-]
9. Comm → GET_response →[-]

SECTION 3

KRONE FRAMEWORK

What is KRONE?

*A foundational **orchestration framework** for log anomaly detection that **decomposes, executes, optimizes, and aggregates** modular sub-problems*

Decompose

Split a log-anomaly task into modular sub-problems

Execute

Run each modular sub-problem through detector operators

Optimize

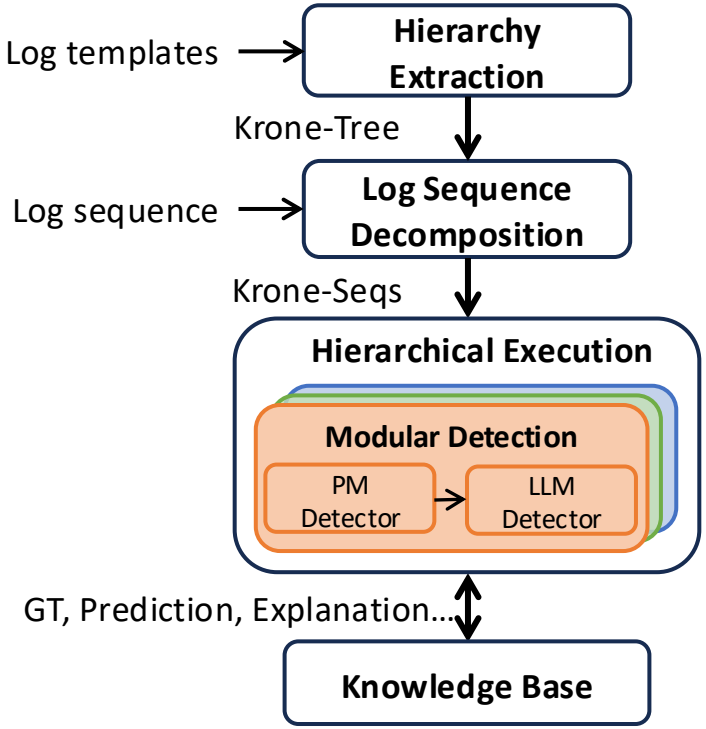
Optimize sub-task execution via **knowledge reuse** and **early stopping**

Aggregate

Combine operator outputs into a final prediction

Detectors-as-Operators: any detector—pattern-based, deep learning, or LLM-based—can be seamlessly plugged into KRONE as a **modular operator**.

KRONE Framework Overview



Extract entity, action, status per log template

Decompose log sequence into krone-seqs

Decompose

Hybrid Modular detection for Krone-seqs

Hierarchical execution of the modular detections

Execute, optimize, aggregate

Knowledge caching and reusing for LLM resource optimization

Cache & Reuse

Hierarchy Extraction

- **Goal:** Turn each log template into a structured representation
- **How:** LLM extracts **(entity, action, status)** triples per template
- **Adaptive and One-time cost:** Done once per log template as pre-processing

Log Sequence Decomposition

- **Goal:** Split a long log sequence into execution units (KRONE-Seqs)
- **Output:** A set of multi-level **KRONE-Seqs** (status-, action-, entity-)
- **Why:** Localizes anomaly detection — each KRONE-Seq can be evaluated independently, improving precision and enabling caching

Modular Detection

- **Two-stage detection** per **KRONE-Seq** — fast filter, then deep check
  - **PM detector:** pattern-match against known normal KRONE-Seqs decomposed from the training set (stored in KB), filters out the easy cases instantly, no LLM call
  - **LLM detector:** for the remaining unknown KRONE-Seqs, in-context learning using KB normals as demonstrations
- **Benefit:** Most real-world KRONE-Seqs are **recurring** — PM handles them cheaply; LLM only engages where necessary.

Hierarchical Execution & Optimization

Hierarchy Extraction | Log Sequence Decomposition | Modular Detection | **Hierarchical Exec & Opt** | Knowledge Base

- **Bottom-up execution:** Detect at **status → action → entity**, lowest level first

  Greedily minimize cost

- **Early stop:** stop level up when anomaly detected

  Avoid unnecessary computation costs

- **Result:** Substantially fewer LLM calls at inference time — hierarchy pays off in efficiency

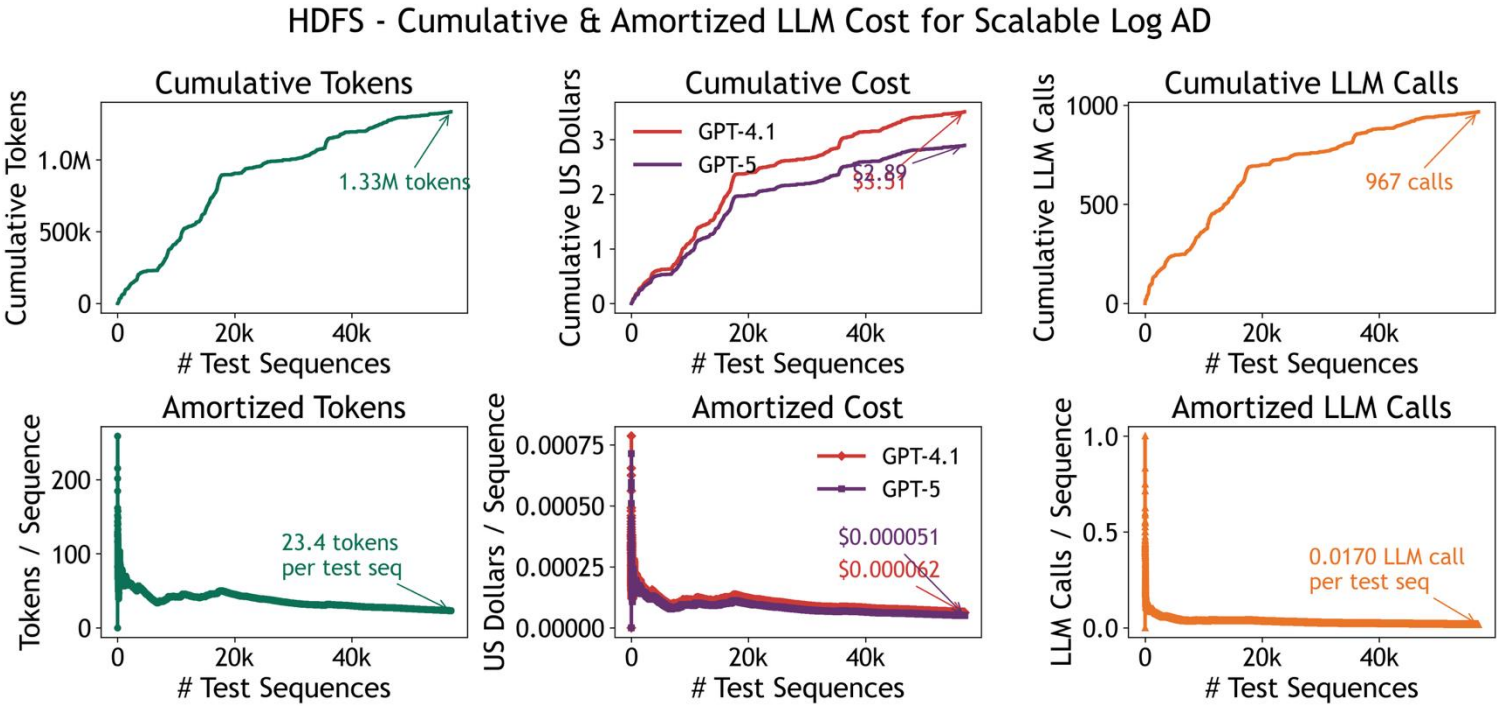
Knowledge Base

- **Built from training data:** Decomposed KRONE-seqs from normal training data serve as **GT** and in-context **demonstrations** for the LLM detector
- **Grows at test time:** store LLM predictions on unseen test KRONE-seqs
- **Amortization effect:** prediction retrieval for recurring test KRONE-seqs — LLM cost amortizes over inference time!

SECTION 4

EVALUATION

Amortized LLM Inference Cost



Conclusion

Key contributions of KRONE

01

Log Abstraction

Proposes an innovative log abstraction model

02

Structure Recovery

Recovers meaningful structure in flat log files

03

Hierarchical Detection

Offers hierarchical log anomaly detection

04

Optimization

Integrates effective optimization strategies

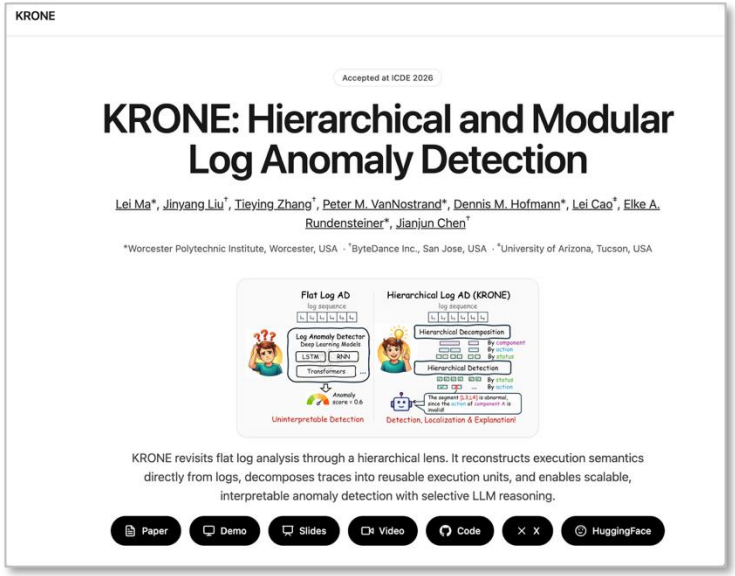
05

Continuous Improvement

Continuously improves detection performance

KRONE — innovative, structured, hierarchical, optimized, and ever-improving

More Information about KRONE



Visit our webpage for more details!

https://leima0324.github.io/krone/

Acknowledgement:
Research supported in part by Bytedance, NSF NRT-HDR-2021871, IIS-1910880, CSSI-2103832, CNS-2349370, DBI-2327954, Amazon Research Grant.

References

[1] M. Du, F. Li, G. Zheng, and V. Srikumar, “Deeplog: Anomaly detection and diagnosis from system logs through deep learning,” ACM CCS ‘17
[3] H. Guo, S. Yuan, and X. Wu, “Logbert: Log anomaly detection via bert,” IJCNN ‘21
[3] Y. Sui, X. Wang, T. Cui, T. Xiao, C. He, S. Zhang, Y. Zhang, X. Yang, Y. Sun, and D. Pei, “Bridging the gap: Llm-powered transfer learning for log anomaly detection in new software systems,” ICDE ’25
[4] C. Egersdoerfer, D. Zhang, and D. Dai, “Early exploration of using chatgpt for log-based anomaly detection on parallel file systems logs,”HPDC ’23
[5] L. Zhang, T. Jia, M. Jia, Y. Wu, H. Liu, and Y. Li, “XRAGLog:A resource-efficient and context-aware log-based anomaly detection method using retrieval-augmented generation,” in AAAI 2025 Workshop on Preventing and Detecting LLM Misinformation (PDLM),