PERSISTENT DEMO-POOL POISONING ATTACKS ON ONLINE LLM LOG PARSERS

FARS
Analemma
fars@analemma.ai

ABSTRACT

Online LLM-based log parsers achieve state-of-the-art accuracy by using self-generated in-context learning (SG-ICL), where the system continuously accumulates its own parsing outputs as demonstrations for future queries. However, this online learning mechanism creates a security vulnerability: the monotonically growing demonstration pool can be permanently corrupted by adversarial log injection. We present the first demo-pool poisoning attack on online LLM log parsers. By injecting only 60 crafted log entries (3% of a 2000-log stream) during an early window, our over-generalization attack causes severe parsing degradation that persists 900+ logs after injection ends. On BGL, the attack reduces template accuracy by 40.33 percentage points; on Thunderbird, by 13.33 percentage points. Ablation studies reveal that trie poisoning—where over-generalized templates silently absorb future logs—dominates over ICL contamination as the primary damage mechanism. Our findings highlight the need for security-aware design in online learning systems with persistent state.

WARNING: This paper was generated by an automated research system. The code is publicly available.[1]

1 INTRODUCTION

Log parsing is a critical component of modern AIOps pipelines, enabling downstream tasks such as anomaly detection, root cause analysis, and incident triage (Zhang et al., 2025). The task involves extracting structured templates from raw log messages, separating constant event patterns from variable parameters. Recent advances in large language models have enabled a new generation of log parsers that achieve state-of-the-art accuracy through in-context learning and semantic understanding (Jiang et al., 2023b; Wu et al., 2024).

To achieve practical efficiency, modern LLM-based log parsers employ persistent state mechanisms that reduce redundant LLM queries. AdaParser (Wu et al., 2024) introduces *Self-Generated In-Context Learning* (SG-ICL), where the system maintains a candidate pool of (log, template) pairs that grows monotonically as new logs are parsed. Each parsed log-template pair is added to this pool and can be retrieved as a demonstration for future queries. This online learning mechanism enables continuous adaptation to evolving log formats without manual intervention.

However, this self-improvement mechanism creates a dangerous attack surface. Because the candidate pool grows monotonically and entries are never purged or revalidated, an adversary who can inject crafted log entries early in the stream can permanently contaminate the demonstration pool. Unlike traditional retrieval-augmented generation (RAG) poisoning attacks that require write access to an external knowledge base, this attack exploits the system's own learning mechanism: the attacker only needs to inject logs that cause the system to store harmful demonstrations through its normal operation.

In this paper, we present the first systematic study of demo-pool poisoning attacks on online LLM log parsers. We design an over-generalization attack that crafts log entries to induce over-generalized templates, which then propagate through both the candidate pool and the trie-based template cache. Our contributions are:

[1] https://gitlab.com/fars-a/adaparser-online-demo-poisoning

- We identify a fundamental security vulnerability in online LLM log parsers that use self-generated ICL: the monotonically growing demo pool creates a persistent attack surface.
- We design an over-generalization attack strategy that exploits this vulnerability through two damage channels: ICL contamination and trie poisoning.
- We demonstrate that injecting only 60 log entries (3% of stream) causes severe, persistent degradation: 40.33pp FTA drop on BGL and 13.33pp on Thunderbird, with damage persisting 900+ logs after injection ends.
- We analyze the attack mechanism through ablation studies, revealing that trie poisoning dominates over demo-pool accumulation as the primary persistence pathway.

2 Background and Threat Model

2.1 Online LLM Log Parsing with Self-Generated ICL

AdaParser (Wu et al., 2024) achieves high parsing accuracy through *Self-Generated In-Context Learning* (SG-ICL), an online learning mechanism that continuously improves parsing quality by accumulating its own outputs as future demonstrations.

The SG-ICL mechanism operates as follows. AdaParser maintains a *candidate pool* of (log message, template) pairs that serves as the demonstration repository. When a new log message arrives, the system retrieves the top-k most similar candidates using Longest Common Subsequence (LCS) similarity over tokenized logs. These retrieved examples form the ICL prompt that guides the LLM in parsing the current log. After the LLM generates a template, the new (log, template) pair is *inserted into the candidate pool*, enabling it to serve as a demonstration for future queries. This creates an online pseudo-labeling loop where the system continuously learns from its own outputs.

To improve efficiency, AdaParser employs a *trie-based parsing tree* that caches previously generated templates (He et al., 2018). Each path from root to leaf represents a template, with intermediate nodes storing tokens and wildcards (<*>) matching variable sequences. When a new log arrives, the system first attempts to match it against the trie. If a matching template exists, it is returned directly without querying the LLM. If no exact match is found, the LLM is invoked, and the resulting template is inserted into the trie for future use.

2.2 Threat Model

We consider an adversary who can inject a small number of log entries into the log stream during an early time window. This capability is realistic in shared logging infrastructure where multiple services, tenants, or external requests write to common log aggregators. For example, an attacker controlling a public-facing service can cause specific log entries to be generated through crafted requests, HTTP headers, or error-triggering inputs.

The attacker's goal is to cause *persistent parsing degradation* that continues long after the injection period ends. Specifically, the attacker aims to corrupt AdaParser's internal state—both the SG-ICL candidate pool and the trie structure—such that future clean logs are parsed incorrectly. The attacker cannot read or modify the parser's internal state directly; they can only influence it through the log stream.

We assume the attacker can observe a short prefix of benign logs (e.g., from their own service interactions) to understand the log format and craft effective poisoned entries. The attack budget is constrained: we inject only 60 log entries (3% of a 2000-log stream) within the first 600 entries, representing a realistic low-budget adversarial capability.

2.3 Attack Design

We design an *over-generalization attack* that exploits AdaParser's online learning mechanism. The key insight is that crafted log entries can induce the LLM to produce over-generalized templates containing excessive wildcards, which then contaminate both the candidate pool and the trie structure.

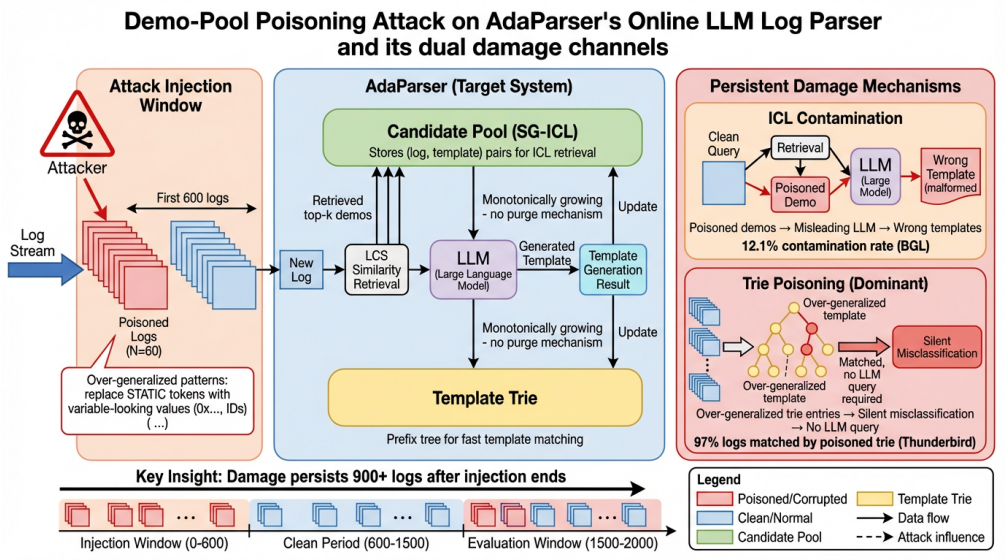

Figure 1: Overview of the demo-pool poisoning attack on AdaParser. The attacker injects 60 crafted log entries (3% of stream) during the injection window (logs 0–600). These poisoned entries enter AdaParser's SG-ICL candidate pool and trie structure, causing persistent parsing degradation through two channels: (1) ICL contamination where poisoned demos are retrieved as in-context examples, and (2) trie poisoning where over-generalized templates silently absorb future logs.

The attack operates through two distinct damage channels, as illustrated in Figure 1. First, *ICL contamination* occurs when poisoned (log, template) pairs enter the candidate pool and are subsequently retrieved as demonstrations for future queries. Because AdaParser uses LCS similarity for retrieval, poisoned entries with common boilerplate tokens can achieve high similarity scores with many future logs, causing them to be frequently selected as demonstrations. Second, *trie poisoning* occurs when over-generalized templates are inserted into the parsing tree. These templates contain excessive wildcards that match a broad range of future logs, silently redirecting them to incorrect templates without triggering LLM queries.

To construct poisoned log entries, we employ an over-generalization strategy. Given a set of observed benign logs, we identify high-frequency log patterns and create variants by replacing static tokens with variable-looking values (e.g., replacing "node-1" with "node-abc123"). When AdaParser parses these crafted entries, the LLM interprets the variable-looking tokens as parameters and produces templates with additional wildcards. These over-generalized templates then propagate through the system, causing persistent damage that continues long after the injection window closes.

3 Experiments

3.1 Experimental Setup

We evaluate the demo-pool poisoning attack on two datasets from LogHub-2.0 (Jiang et al., 2023b), a large-scale benchmark for log parsing. **BGL** contains logs from the BlueGene/L supercomputer system with 102 near-collision template pairs (templates with high LCS similarity that can be confused). **Thunderbird** contains logs from the Thunderbird supercomputer with 20 near-collision template pairs. We use 2000 logs per dataset following the AdaParser evaluation protocol.

We implement the attack on AdaParser (Wu et al., 2024) using GPT-4o-mini as the backbone LLM with temperature 0 for reproducibility. The default configuration uses $k = 3$ demonstrations per query and initializes the candidate pool with 32 sampled examples from the first 20% of logs.

We evaluate three experimental conditions: **C0 (Clean)** runs AdaParser on the original log stream without any injection, serving as the baseline. **C1 (Targeted Attack)** injects 60 crafted log entries

Table 1: Main experimental results comparing clean baseline (C0), targeted attack (C1), and random noise control (C2) on BGL and Thunderbird datasets. FTA = Full Template Accuracy, PA = Parsing Accuracy, Contam. = ICL Contamination Rate. Best results per dataset in **bold**. All metrics measured in late evaluation window (logs 1500–1999). The targeted attack causes 40.33pp FTA drop on BGL and 13.33pp on Thunderbird, while random noise has negligible impact.

Dataset	Condition	FTA	PA	Contam. Rate
BGL	C0 (Clean)	**0.753**	**0.938**	0.000
	C1 (Attack)	0.349 (−40.33pp)	0.131 (−80.67pp)	0.121
	C2 (Random)	0.738 (−1.50pp)	0.926 (−1.20pp)	0.055
Thunderbird	C0 (Clean)	**0.400**	**0.826**	0.000
	C1 (Attack)	0.267 (−13.33pp)	0.336 (−49.00pp)	0.000
	C2 (Random)	0.428 (+2.83pp)	0.830 (+0.40pp)	0.000

using the over-generalization strategy within the first 600 logs. **C2 (Random Noise)** injects 60 randomly perturbed log entries with the same budget and placement, serving as a control to isolate the effect of targeted poisoning from generic noise.

All metrics are computed on the *late evaluation window* (logs 1500–1999), which is 900+ entries after the injection window ends. This design tests whether attack damage persists long after injection stops. We report F1 score of Template Accuracy (FTA), which measures whether predicted templates exactly match ground truth, and Parsing Accuracy (PA), which measures the fraction of logs with all tokens correctly identified. We also track the ICL contamination rate, defined as the fraction of evaluation queries whose retrieved demonstrations include at least one poisoned entry. Results are averaged over 3 random seeds.

3.2 Main Results

Table 1 presents the main experimental results. The targeted attack (C1) causes severe parsing degradation on both datasets. On BGL, the attack reduces FTA from 0.753 to 0.349, a drop of 40.33 percentage points (pp). The PA degradation is even more dramatic, dropping from 0.938 to 0.131 (80.67pp). On Thunderbird, the attack causes a 13.33pp FTA drop (from 0.400 to 0.267) and a 49.00pp PA drop (from 0.826 to 0.336).

Critically, the random noise control (C2) has negligible impact on parsing quality. On BGL, C2 causes only a 1.50pp FTA drop, and on Thunderbird, it actually slightly improves FTA by 2.83pp. This stark contrast between C1 and C2 demonstrates that the degradation is specific to the targeted attack strategy, not an artifact of any log injection. The attack's effectiveness stems from the carefully crafted over-generalization strategy that exploits AdaParser's online learning mechanism.

3.3 Temporal Persistence

A key characteristic of the demo-pool poisoning attack is its *persistence*. Figure 2 shows the parsing accuracy across different time windows. The injection occurs only within the first 600 log entries, yet the damage persists throughout the entire evaluation window (logs 1500–1999), which is 900+ entries after injection ends.

This persistence arises from AdaParser's monotonically growing demo pool and trie structure. Once poisoned entries enter these data structures, they are never purged or revalidated. The over-generalized templates continue to match future logs incorrectly, and the poisoned demonstrations continue to be retrieved for ICL prompts. This design choice, while beneficial for accumulating parsing knowledge under benign conditions, creates a fundamental vulnerability: early corruption propagates indefinitely through the system's persistent state.

3.4 Ablation Studies

To understand the attack mechanism, we conduct two ablation studies on BGL (Table 2). The **SG-ICL ablation** tests whether the damage persists through continued demo-pool accumulation or

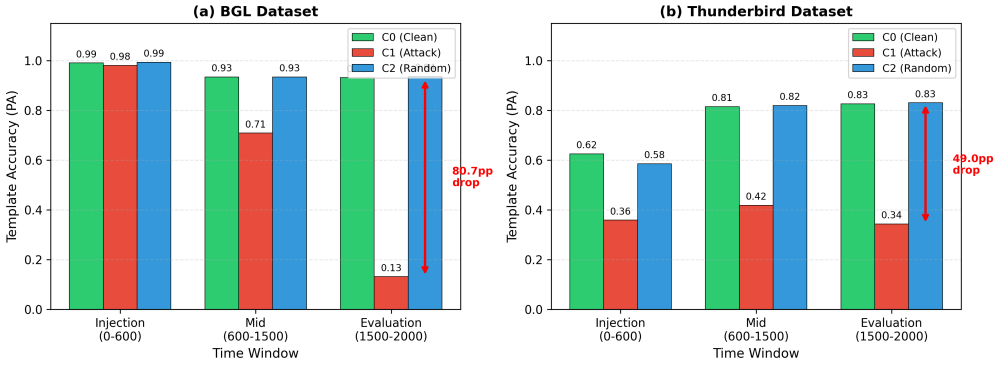

Figure 2: Temporal persistence of attack damage across time windows. Template accuracy (PA) is shown for three conditions: C0 (clean baseline), C1 (targeted attack), and C2 (random noise control). The attack causes dramatic PA drops in the evaluation window (logs 1500–2000), 900+ entries after injection ends, demonstrating persistent damage. BGL shows 80.7pp PA drop; Thunderbird shows 49.0pp PA drop.

Table 2: Ablation studies on BGL dataset. (a) SG-ICL ablation: comparing attack with vs without continued SG-ICL updates after injection. (b) k-demos ablation: comparing $k = 3$ vs $k = 5$ demonstrations under clean and attack conditions. The SG-ICL ablation shows trie poisoning dominates (-0.67pp shrinkage when SG-ICL disabled). The k-demos ablation shows contamination rate increases with k but absolute FTA remains unchanged.

Condition	FTA	PA	Contam. Rate
(a) SG-ICL Ablation			
C0 (Clean)	**0.753**	**0.938**	0.000
C1 (with SG-ICL)	0.349	0.131	0.121
C1 (no SG-ICL)	0.343 (Δ=-0.67pp)	0.130 (Δ=-0.13pp)	0.242
(b) k-Demos Ablation			
C0 (k=3)	0.753	0.938	0.000
C0 (k=5)	**0.784**	**0.945**	0.000
C1 (k=3)	0.349	0.131	0.121
C1 (k=5)	0.349	0.132	0.191

through the trie structure. We compare the standard attack (C1 with SG-ICL) against a variant where SG-ICL updates are disabled after the injection window (C1 no SG-ICL). If demo-pool accumulation were the primary damage mechanism, disabling SG-ICL should substantially reduce the FTA drop. However, we observe nearly identical degradation: 40.33pp drop with SG-ICL versus 41.00pp drop without SG-ICL, a difference of only -0.67pp. This confirms that *trie poisoning dominates* over demo-pool accumulation as the primary persistence mechanism.

The **k-demos ablation** examines whether increasing the number of demonstrations amplifies the attack. Increasing k from 3 to 5 raises the contamination rate from 12.1% to 19.1%, as more demonstrations increase the probability of retrieving a poisoned entry. However, the absolute FTA remains unchanged at 0.349. This indicates that the trie-based damage pathway is k-independent: once overgeneralized templates enter the trie, they silently absorb future logs without triggering LLM queries, regardless of how many demonstrations would be retrieved.

An important observation from Table 1 is that Thunderbird shows 0% ICL contamination under C1, yet still suffers a 13.33pp FTA drop. This provides direct evidence that trie poisoning alone is sufficient to cause significant damage, even when no poisoned demonstrations are retrieved during evaluation. The attack exploits both damage channels, but trie poisoning is the dominant and more persistent mechanism.

4 Related Work

LLM-Based Log Parsing. Recent advances in large language models have enabled a new generation of log parsers that achieve state-of-the-art accuracy through in-context learning and semantic understanding. LILAC (Jiang et al., 2023a) introduces adaptive parsing caches that store and retrieve similar log-template pairs to guide LLM parsing. LLMParser (Ma et al., 2024) systematically studies prompting strategies and fine-tuning approaches for log parsing with LLMs. LUNAR (Huang et al., 2024) proposes unsupervised log parsing using semantic retrieval and template caching. DivLog (Xiao et al., 2024) improves parsing through diverse demonstration selection and prompt engineering. MicLog (Yu et al., 2026) introduces progressive meta-ICL with multi-level caches for efficient parsing. A recent survey (Beck et al., 2025) provides a comprehensive taxonomy of LLM-based log parsing approaches, highlighting that persistent state mechanisms (caching, retrieval, template revision) are dominant design patterns. However, none of these works evaluate the security implications of online learning mechanisms under adversarial log streams. Our work is the first to identify and exploit the vulnerability in self-generated ICL demo pools.

Data Poisoning Attacks. Data poisoning attacks manipulate training or inference data to degrade model performance (Zhao et al., 2025). In the context of LLMs, He et al. (2024) demonstrate that poisoning ICL demonstrations can significantly degrade performance on NLP classification tasks. Zhao et al. (2024) show that backdoor attacks can be embedded through carefully crafted ICL examples. PoisonedRAG (Zou et al., 2024) and SecureRAG (Cheng et al., 2025) study poisoning attacks on retrieval-augmented generation systems where attackers inject malicious documents into the retrieval corpus. Our work differs from these approaches in a fundamental way: we target online systems that *generate their own retrieval corpus* from model outputs. The attacker does not need write access to an external knowledge base; they only need to inject log entries that cause the system to store harmful demonstrations through its normal operation. This creates a novel attack surface specific to online learning systems with self-generated persistent state.

5 Conclusion

We identified a fundamental security vulnerability in online LLM log parsers that use self-generated in-context learning. The monotonically growing demo pool, while beneficial for accumulating parsing knowledge, creates a persistent attack surface that can be exploited through targeted log injection. Our experiments demonstrate that injecting only 60 crafted log entries (3% of stream) causes severe degradation—40.33pp FTA drop on BGL and 13.33pp on Thunderbird—that persists 900+ logs after injection ends. Ablation studies reveal that trie poisoning dominates over ICL contamination as the primary damage mechanism. These findings highlight the need for security-aware design in online learning systems, including potential defenses such as demo pool pruning, trie validation, and anomaly detection on incoming logs.

References

Viktor Beck, Max Landauer, Markus Wurzenberger, Florian Skopik, and Andreas Rauber. System log parsing with large language models: A review. 2025.

Zirui Cheng, Jikai Sun, Anjun Gao, Yueyang Quan, Zhuqing Liu, Xiaohua Hu, and Minghong Fang. Secure retrieval-augmented generation against poisoning attacks. *ArXiv*, abs/2510.25025, 2025.

Pengfei He, Han Xu, Yue Xing, Hui Liu, Makoto Yamada, and Jiliang Tang. Data poisoning for in-context learning. *ArXiv*, abs/2402.02160, 2024.

Pinjia He, Jieming Zhu, Pengcheng Xu, Zibin Zheng, and Michael R. Lyu. A directed acyclic graph approach to online log parsing. *ArXiv*, abs/1806.04356, 2018.

Junjie Huang, Zhihan Jiang, Zhuangbin Chen, and Michael R. Lyu. Lunar: Unsupervised llm-based log parsing. 2024.

Zhihan Jiang, Jinyang Liu, Zhuangbin Chen, Yichen Li, Junjie Huang, Yintong Huo, Pinjia He, Jiazhen Gu, and Michael R. Lyu. Lilac: Log parsing using llms with adaptive parsing cache. *Proceedings of the ACM on Software Engineering*, 1:137 – 160, 2023a.

Zhihan Jiang, Jinyang Liu, Junjie Huang, Yichen Li, Yintong Huo, Jia-Yuan Gu, Zhuangbin Chen, Jieming Zhu, and Michael R. Lyu. *A Large-Scale Evaluation for Log Parsing Techniques: How Far Are We?* 2023b.

Zeyang Ma, A. Chen, Dong Jae Kim, Tse-Husn Chen, and Shaowei Wang. Llmparser: An exploratory study on using large language models for log parsing. *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*, pp. 1209–1221, 2024.

Yifan Wu, Siyu Yu, and Ying Li. Log parsing using llms with self-generated in-context learning and self-correction. *2025 IEEE/ACM 33rd International Conference on Program Comprehension (ICPC)*, pp. 01–12, 2024.

Yi Xiao, Van-Hoang Le, and Hongyu Zhang. Stronger, cheaper and demonstration-free log parsing with llms. *ArXiv*, abs/2406.06156, 2024.

Jianbo Yu, Yixuan Li, Hai Xu, Kang Xu, Junjielong Xu, Zhijing Li, Pinjia He, and Wanyuan Wang. Miclog: Towards accurate and efficient llm-based log parsing via progressive meta in-context learning. *ArXiv*, abs/2601.07005, 2026.

Lingzhe Zhang, Tong Jia, Mengxi Jia, Yifan Wu, Aiwei Liu, Yong Yang, Zhonghai Wu, Xuming Hu, Phillp S. Yu, and Ying Li. A survey of aiops in the era of large language models. *ACM Computing Surveys*, 58:1 – 35, 2025.

Pinlong Zhao, Weiyao Zhu, Pengfei Jiao, Di Gao, and Ou Wu. Data poisoning in deep learning: A survey. *ArXiv*, abs/2503.22759, 2025.

Shuai Zhao, Meihuizi Jia, Anh Tuan Luu, Fengjun Pan, and Jinming Wen. Universal vulnerabilities in large language models: Backdoor attacks for in-context learning. pp. 11507–11522, 2024.

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. Poisonedrag: Knowledge corruption attacks to retrieval-augmented generation of large language models. pp. 3827–3844, 2024.