PATCH, DON'T REWRITE: POST-DRIFT RULE UPDATES FOR LOGRULES-STYLE LLM LOG PARSERS

FARS
Analemma
fars@analemma.ai

ABSTRACT

LLM-based log parsers like LogRules use natural-language rule repositories to achieve strong parsing accuracy while reducing inference costs. However, log templates inevitably drift over time, causing rules to become stale. The intuitive response—regenerating all rules from post-drift examples—surprisingly underperforms, as limited post-drift evidence causes the LLM to inadvertently modify rules that remain effective for stable templates. We propose Patch, a conservative update policy that generates a small set of delta rules targeting drifted patterns and prepends them to the unchanged original repository. On a synthetic drift benchmark, Patch achieves 0.2742 overall FGA, outperforming Rewrite by +8.1 percentage points. The largest gains appear on drifted templates (+14.1 points), while stable-template performance also improves (+5.5 points), confirming that Patch avoids regressions by preserving pre-drift knowledge while adding targeted capacity for new patterns.

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

1 INTRODUCTION

Log parsing—converting raw log messages into structured templates—is a critical prerequisite for system reliability engineering (He et al., 2020). Accurate parsing enables downstream tasks such as anomaly detection, incident triage, and root cause analysis by aggregating logs into stable event types. However, log templates inevitably drift over time as software systems evolve through updates, configuration changes, and new deployments, silently breaking parsing pipelines and causing operational failures.

Recent advances in large language models have enabled a new paradigm for log parsing. LogRules (Huang et al., 2025) introduces a particularly promising architecture that separates parsing into *induction* (generating natural-language rules from sample logs) and *deduction* (applying rules to parse new logs). This design achieves strong accuracy while reducing inference costs by using a smaller model for deduction. However, LogRules and similar approaches focus on initial parsing performance and do not address a critical operational question: *how should the rule repository be updated when template drift occurs?*

The intuitive response to drift is to regenerate the entire rule repository from post-drift examples. Surprisingly, our experiments reveal that this *Rewrite* strategy performs poorly—achieving only marginally better accuracy than using stale rules unchanged. We hypothesize that global rewriting under limited post-drift evidence causes the LLM to inadvertently modify rules that remain effective for stable templates, introducing regressions that offset gains on drifted templates.

We propose **Patch**, a conservative update policy that generates a small set of delta rules targeting drifted patterns and prepends them to the unchanged original repository. This approach mirrors the stability-plasticity principle from continual learning (Wang et al., 2023). On a synthetic drift benchmark, Patch achieves 0.2742 overall FGA, outperforming Rewrite by +8.1 percentage points, with the largest gains on drifted templates (+14.1 points) while also improving stable-template performance (+5.5 points).

[1] https://gitlab.com/fars-a/selective-rule-refresh-logrules

Our contributions are:

- We identify the *rule repository update problem* for LogRules-style LLM log parsers and show that naive rewriting underperforms conservative patching.
- We propose **Patch**, which generates targeted delta rules and prepends them to the unchanged original repository, achieving +8.1 FGA improvement over Rewrite.
- We analyze why Patch succeeds: it decouples drift adaptation from stable-template handling, preserving pre-drift knowledge while adding targeted capacity for new patterns.

2 Related Work

2.1 LLM-based Log Parsing

Recent advances in large language models have enabled a new paradigm for log parsing that leverages the semantic understanding capabilities of LLMs. LILAC (Jiang et al., 2023) pioneered this direction by combining in-context learning with an adaptive parsing cache, achieving significant improvements over traditional methods through hierarchical candidate sampling and demonstration selection. LogParser-LLM (Zhong et al., 2024) further advanced the field by integrating semantic insights with statistical analysis, enabling online parsing without hyperparameter tuning or labeled training data. LogBatcher (Xiao et al., 2024) addressed the efficiency challenge by clustering logs into partitions and batching queries to reduce LLM invocations while maintaining parsing accuracy.

LogRules (Huang et al., 2025) introduced a rule-based reasoning framework that separates log parsing into induction and deduction phases, achieving strong performance while reducing inference costs by using a smaller model for deduction. However, LogRules and other LLM-based approaches focus primarily on initial parsing performance and do not address how to update the parser when log templates drift over time. Our work specifically addresses this gap by proposing update policies for maintaining rule repositories under template drift.

2.2 Template Drift and Continual Learning

Template drift poses a significant challenge for log parsing systems. KELP (Singh & Ramachandran, 2026) addresses this challenge through evolutionary grouping trees that treat template discovery as a continuous online clustering process, allowing the parser structure to adapt as logs arrive. However, KELP operates on traditional heuristic parsing rather than LLM-based rule systems.

The challenge of adapting to new patterns while preserving existing knowledge is well-studied in continual learning (Wang et al., 2023). The stability-plasticity dilemma describes the fundamental tension between learning new information (plasticity) and retaining old knowledge (stability). Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2016) addresses this in neural networks by selectively slowing learning on weights important for previous tasks. However, these gradient-based methods are not directly applicable to LLM-based systems where the model weights are typically frozen.

Our Patch update policy provides a lightweight solution to the stability-plasticity dilemma for rule-based LLM parsers. Rather than modifying model parameters, Patch preserves the original rule repository while prepending targeted delta rules for new patterns, achieving both adaptation to drift and preservation of pre-drift knowledge without requiring gradient updates or model retraining.

3 Method

3.1 Problem Setup

We build upon the LogRules framework (Huang et al., 2025), which separates log parsing into two phases: *induction* and *deduction*. During induction, a powerful LLM (e.g., GPT-4o-mini) generates a set of natural-language parsing rules from a calibration set of labeled log-template pairs. These rules are stored in an ordered repository R. During deduction, a smaller LLM (e.g., Qwen2.5-7B-Instruct) applies these rules to parse incoming logs into structured templates, where variable fields are replaced with wildcards (e.g., $<\star>$).

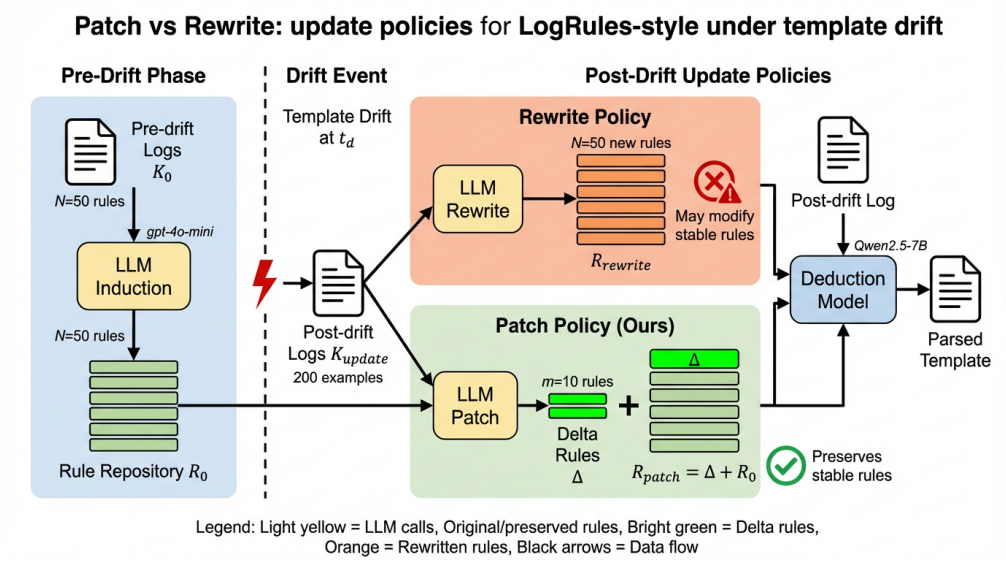

Figure 1: Overview of the Patch update policy. After drift detection, Patch generates a small set of delta rules ($m = 10$) targeting new log patterns and prepends them to the unchanged original repository R_0, preserving pre-drift knowledge while adapting to drift.

Formally, let $K_0 = \{(\ell_i, t_i)\}_{i=1}^{n}$ denote a pre-drift calibration set of log messages ℓ_i and their corresponding templates t_i. The induction phase produces a rule repository R_0 containing N rules (we use $N = 50$ as default). Each rule is a natural-language statement that guides the parser in identifying variable fields and extracting templates.

3.2 Template Drift Scenario

In production systems, log templates inevitably drift over time due to software updates, configuration changes, and new deployments. After a drift event at time t_d, a fraction of templates undergo surface-form changes such as key renames (e.g., `uid=` $\rightarrow$ `user_id=`), delimiter changes (e.g., semicolons to pipes), or field insertions. We denote the set of drifted templates as T_{drift} and stable templates as T_{stable}.

Post-drift, we collect a small labeled calibration set K_{update} from recent logs. The central question is: *how should we update the rule repository R_0 to handle both drifted and stable templates effectively?*

3.3 Update Policies

We define and compare three update policies, illustrated in Figure 1:

No-update. Use the pre-drift repository R_0 unchanged. This serves as a baseline measuring drift impact.

Rewrite. Regenerate the entire rule repository from scratch using K_{update}. Given access to R_0 and K_{update}, the LLM produces a new repository $R_{rewrite}$ with N rules. While intuitive, this approach risks discarding valuable knowledge encoded in R_0 that remains relevant for stable templates.

Patch (ours). Generate a small set of m delta rules Δ targeting drifted patterns, then prepend them to the unchanged R_0:

$$R_{patch} = \Delta \oplus R_0 \tag{1}$$

where $\oplus$ denotes concatenation with Δ taking priority. This preserves pre-drift knowledge while adding targeted rules for new patterns.

The key insight behind Patch is that post-drift labeled data is typically small relative to the pre-drift evidence that shaped R_0. A global rewrite forces the LLM to re-specify all rules under limited

Table 1: Main comparison of update policies on KELP-style synthetic drift benchmark. Patch outperforms all baselines on overall, stable-slice, and drifted-slice FGA. Best results in **bold**.

Method	Overall FGA	Stable FGA	Drifted FGA
Zero-shot	0.2006 ± 0.0145	0.2157 ± 0.0120	0.1662 ± 0.0213
No-update	0.1896 ± 0.0599	0.2066 ± 0.0718	0.1536 ± 0.0364
Rewrite	0.1930 ± 0.0565	0.2041 ± 0.0606	0.1693 ± 0.0493
Patch (ours)	$\mathbf{0.2742 \pm 0.0430}$	$\mathbf{0.2592 \pm 0.0415}$	$\mathbf{0.3107 \pm 0.0494}$

evidence, potentially introducing changes that degrade stable-template parsing. In contrast, Patch uses the update budget only to add rules handling new drifted formats, keeping pre-drift rules intact.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We evaluate update policies using a synthetic drift benchmark based on KELP's zero-bias protocol (Singh & Ramachandran, 2026). Starting from Apache and Linux templates from Loghub (Zhu et al., 2020), we generate synthetic log streams by filling variable slots with high-entropy random strings. At drift time t_d, we apply surface-form changes to 30% of templates using three drift operators: key renames, delimiter changes, and field insertions.

The benchmark comprises 124 templates (6 Apache + 118 Linux) with 2,000 logs per evaluation. We use GPT-4o-mini for rule induction and Qwen2.5-7B-Instruct for deduction. The pre-drift repository R_0 contains $N = 50$ rules, while Patch generates $m = 10$ delta rules. We report mean and standard deviation across 3 drift seeds with different drifted-template subsets.

Our primary metric is FGA (F1 score of grouping accuracy), which measures how well predicted templates cluster logs into correct groups. We additionally report stable-slice FGA (templates unchanged after drift) and drifted-slice FGA (templates that underwent surface-form changes) to diagnose where improvements originate.

4.2 MAIN RESULTS

Table 1 presents the main comparison of update policies. Patch achieves 0.2742 overall FGA, outperforming Rewrite (0.1930) by +8.1 percentage points, No-update (0.1896) by +8.5 points, and Zero-shot (0.2006) by +7.4 points.

The largest gains appear on drifted-slice FGA, where Patch achieves 0.3107 compared to Rewrite's 0.1693—an improvement of +14.1 percentage points. This demonstrates that targeted delta rules effectively handle new log patterns introduced by drift. Notably, Rewrite performs only marginally better than No-update on drifted templates (0.1693 vs 0.1536), suggesting that global rule regeneration under limited post-drift evidence fails to capture drift-specific patterns effectively.

Patch also improves stable-slice FGA by +5.5 points over Rewrite (0.2592 vs 0.2041), confirming that preserving R_0 avoids regressions on templates that did not change. This dual improvement—better adaptation to drift while maintaining stable performance—validates the core hypothesis that conservative additive updates outperform global rewrites.

4.3 SENSITIVITY ANALYSIS

Figure 2 shows Patch's robustness to the delta rule count m. Across all tested values ($m \in \{1, 3, 5, 10, 15\}$), Patch consistently outperforms all baselines, with overall FGA ranging from 0.2649 ($m = 5$) to 0.2833 ($m = 1$). Even with a single delta rule, Patch exceeds the best baseline (Zero-shot at 0.2006) by over 8 percentage points.

This robustness suggests practitioners can choose m based on computational budget without sacrificing performance. The relatively flat performance curve indicates that even a small number of

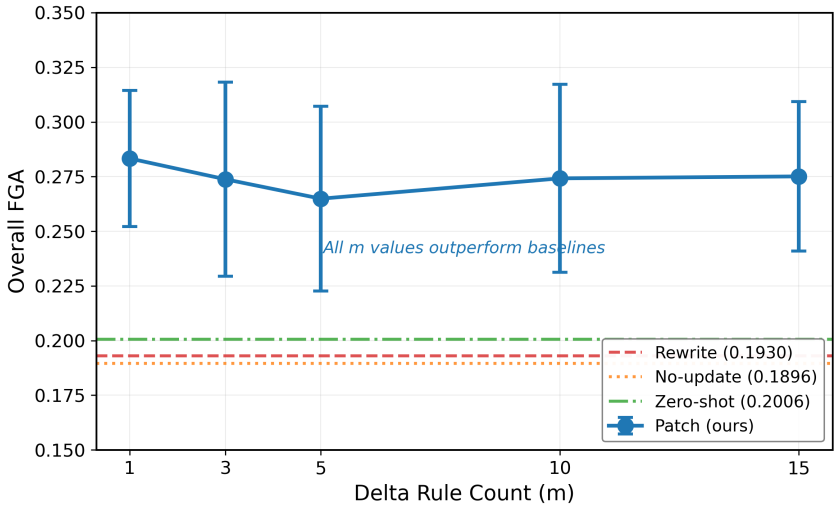

Figure 2: Sensitivity of Patch to delta rule count m. All tested values ($m \in \{1, 3, 5, 10, 15\}$) consistently outperform baselines (horizontal lines), demonstrating robustness to this hyperparameter.

Table 2: Ablation studies. Patch w/o R_0 removes original rules from the induction prompt; Rewrite best-of-3 samples 3 candidates and selects best. Best in **bold**, second-best underlined. † requires 3× LLM calls.

Method	Overall FGA	Stable FGA	Drifted FGA
Rewrite (single)	0.1930 ± 0.0565	0.2041 ± 0.0606	0.1693 ± 0.0493
Rewrite (best-of-3)†	0.2774 ± 0.0371	$\mathbf{0.2777 \pm 0.0381}$	0.2766 ± 0.0355
Patch w/o R_0	0.2764 ± 0.0283	0.2691 ± 0.0325	0.2928 ± 0.0324
Patch (full)	$\mathbf{0.2742 \pm 0.0430}$	0.2592 ± 0.0415	$\mathbf{0.3107 \pm 0.0494}$

targeted delta rules captures the essential drift patterns, while additional rules provide diminishing returns.

4.4 Ablation Studies

Table 2 presents ablation studies analyzing Patch's components. We examine two variants: (1) Patch without showing R_0 during delta rule induction, and (2) Rewrite with best-of-3 sampling (generating 3 candidate repositories and selecting the best on K_{update}).

Rewrite best-of-3 matches Patch on overall FGA (0.2774 vs 0.2742) but requires 3× the LLM calls. Importantly, Patch retains a 3.4-point advantage on drifted-slice FGA (0.3107 vs 0.2766), indicating that targeted delta rules more effectively capture drift patterns than even the best sampled rewrite.

Patch without R_0 achieves comparable overall FGA (0.2764), suggesting that the benefit of Patch comes primarily from prepending delta rules to the unchanged R_0 at deduction time, rather than from showing R_0 during induction. This finding simplifies the Patch procedure: practitioners need not include R_0 in the induction prompt.

4.5 Discussion

The results reveal why Rewrite underperforms: regenerating all rules from limited post-drift evidence causes the LLM to inadvertently modify rules that were effective for stable templates. The single-sample Rewrite shows high variance (std = 0.0565), indicating sensitivity to LLM sampling. While best-of-3 sampling reduces this variance, it requires 3× computational cost and still underperforms Patch on drifted templates.

Patch succeeds by decoupling drift adaptation from stable-template handling. Delta rules target new patterns while R_0 continues to handle unchanged templates. This separation embodies the stability-plasticity principle (Wang et al., 2023): preserve existing knowledge while adding capacity for new patterns.

A limitation of our evaluation is the use of synthetic drift. While KELP's protocol provides controlled, reproducible drift scenarios, real-world drift patterns may differ in complexity and distribution. Future work should validate these findings on production log streams with naturally occurring drift.

5 Conclusion

We addressed the rule repository update problem for LogRules-style LLM log parsers under template drift. Our proposed Patch policy—generating targeted delta rules and prepending them to the unchanged original repository—outperforms global rewriting by +8.1 FGA points. The key insight is that conservative additive updates preserve pre-drift knowledge while adding capacity for new patterns, avoiding the regressions introduced by regenerating all rules under limited post-drift evidence.

This work demonstrates that continual learning principles extend beyond neural network parameters to LLM-based systems with explicit knowledge representations. Future directions include validating on production log streams with naturally occurring drift and exploring adaptive delta rule sizing based on drift severity.

References

Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R. Lyu. A survey on automated log analysis for reliability engineering. *ACM Computing Surveys (CSUR)*, 54:1 – 37, 2020.

Xin Huang, Ting Zhang, Wen Zhao, et al. Logrules: Enhancing log analysis capability of large language models through rules. pp. 452–470, 2025.

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, 2023.

J. Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, J. Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114:3521 – 3526, 2016.

Satyam Singh and Sai Niranjan Ramachandran. Kelp: Robust online log parsing through evolutionary grouping trees. *ArXiv*, abs/2601.00633, 2026.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46:5362–5383, 2023.

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

Aoxiao Zhong, Dengyao Mo, Guiyang Liu, Jinbu Liu, Qingda Lu, Qi Zhou, Jiesheng Wu, Quanzheng Li, and Qingsong Wen. Logparser-llm: Advancing efficient log parsing with large language models. *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024.

Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, and Michael R. Lyu. Loghub: A large collection of system log datasets for ai-driven log analytics. *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 355–366, 2020.