

Adaptive Schema Normalization for Deterministic Security Log Template Mining

Aditya Sourabh¹, Harsh P Kavatekar^{*2}

^{1,*2}Log Analysis and Security Operations Research.

^{*}Corresponding Author.

To Cite this Article: Aditya Sourabh¹, Harsh P Kavatekar^{*2}, "Adaptive Schema Normalization for Deterministic Security Log Template Mining" Indian Journal of Computer Science and Technology, Volume 05, Issue 03 (September-December 2026), PP: 176-181.



Copyright: ©2026 This is an open access journal, and articles are distributed under the terms of the Creative Commons Attribution License; Which Permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Abstract: Modern Security Operations Centers (SOCs) process heterogeneous telemetry generated by security appliances, cloud platforms, operating systems, applications, and network infrastructure. Rule-based parsers provide deterministic extraction but require repeated maintenance as formats evolve. Automated template-mining algorithms reduce this dependency through data-driven template discovery, yet commonly normalize field values while retaining vendor-specific field names. Equivalent events can therefore produce fragmented templates when schemas differ across products or change between software versions. This paper specifies an architecture that inserts adaptive schema normalization between value normalization and online template mining. The pipeline performs JSON envelope unwrapping, recursive flattening, deterministic value normalization, LLM-assisted canonical field mapping, schema-drift detection, Drain3 template mining, and versioned repository maintenance. Established canonical schemas such as OCSF can serve as the target schema, while the Adaptive Schema Normalization (ASN) mechanism automatically maps heterogeneous and previously unseen vendor fields into that target representation at ingestion time. Drain3 template mining and repository alias lookups remain deterministic; a field with no approved alias is instead grounded in its nearest existing canonical templates and passed to a large language model, which proposes a canonical mapping that an analyst must approve before the repository is updated. Drain3 consequently operates on canonical rather than vendor-specific schemas. The architecture is intended to reduce template fragmentation, improve cross-source interoperability, and preserve auditable, analyst-gated mapping decisions as telemetry schemas evolve.

Key Word: adaptive schema normalization, canonical schema, cybersecurity telemetry, Drain3, log parsing, schema drift, template mining.

I. INTRODUCTION

A. Motivation

Security telemetry serves as the foundation of modern cybersecurity operations. Every alert investigated, threat detected, forensic analysis performed, and compliance assessment conducted within a Security Operations Center (SOC) relies on accurately parsing and interpreting logs generated by diverse software systems. As enterprises increasingly adopt cloud-native platforms, containerized workloads, microservices, and distributed infrastructures, both the volume and heterogeneity of security telemetry continue to grow rapidly. Consequently, scalable and reliable log parsing has become a critical prerequisite for Security Information and Event Management (SIEM) platforms, Security Orchestration, Automation and Response (SOAR) systems, threat hunting platforms, and security analytics pipelines.

B. Evolution of Log Parsing

Log parsing techniques have evolved significantly over the past decade. Early production systems primarily relied on manually authored regular expressions, Grok patterns, and vendor-specific parsing rules to extract structured fields from raw log messages. While these approaches provide deterministic and accurate parsing for known log formats, they require continuous maintenance as vendors modify message structures, introduce new telemetry sources, or evolve existing schemas.

To reduce dependence on handcrafted parsing rules, automated template mining algorithms such as Spell, IPLoM, Drain, and Drain3 were introduced to discover structural patterns directly from streaming log data. Rather than relying on predefined rules, these approaches automatically cluster structurally similar events into reusable templates, significantly improving scalability while reducing parser maintenance. Among these algorithms, Drain3 has emerged as one of the most widely adopted production-ready parsers due to its online template discovery, incremental template evolution, and efficient fixed-depth parsing architecture.

C. Challenges in Production Log Parsing

Current deterministic log parsers primarily address structural variability by normalizing high-cardinality values such as timestamps, IP addresses, UUIDs, process identifiers, and numeric values. While value normalization effectively reduces template explosion, it does not resolve inconsistencies introduced by evolving field schemas. The unresolved systems problem lies between value normalization and template discovery. Existing miners can adapt to new token structures, but they do not determine whether heterogeneous field names denote the same canonical attribute. Consequently, schema variation is preserved as structural variation and accumulates in the template repository.

D. Contributions

This paper proposes an adaptive schema-aware log parsing architecture that extends conventional deterministic pre-processing with automated schema normalization prior to online template mining. The primary contributions are:

- Production-oriented architecture extending traditional preprocessing with adaptive schema normalization prior to online template mining
- LLM-assisted field mapping framework grounded in nearest canonical templates with analyst approval Schema drift detection mechanism for identifying previously unseen telemetry attribute.
- Demonstration of improved template consistency while maintaining deterministic parsing
- Canonical templates providing foundation for downstream security analytics

II. PROBLEM STATEMENT

A. Schema Fragmentation at Scale

Modern SOC ingest millions of security events daily from heterogeneous sources with different message formats, field naming conventions, and vendor-specific schemas. When equivalent security concepts are represented under different field names—such as source IP appearing as `srcip`, `sourceip`, `clientip`, `network.sourceip`, or `sourceAddress`—Drain3 and similar miners treat these as structurally distinct templates despite conveying identical information. This schema fragmentation has tangible consequences: templates proliferate unnecessarily, cross-source correlation becomes difficult, downstream analytics must accommodate field name variations, and operational queries must union over multiple field representations.

B. Limitations of Current Approaches

Template mining algorithms reduce manual parsing overhead by discovering structural patterns from log streams. Yet they inherit vendor schemas as-is, perpetuating the naming conventions and attribute structures present in the original telemetry. When a vendor introduces a new field or renames an existing one between software versions, the miner observes this as schema drift and creates new templates rather than recognizing semantic equivalence.

C. Cross-Vendor Schema Variation Examples

Consider authentication events across three security products:

- Product A: `user.name`, `session.id`, `auth.result`
- Product B: `username`, `sessiontoken`, `loginstatus`
- Product C: `principal.account`, `session.identifier`, `outcome`

Each system captures the same security event: a user authenticating with a session. Yet naive template mining produces three distinct templates because field names differ. Canonical schema normalization transforms this into a single unified template representation, enabling consistent downstream detection, alerting, and forensic analysis.

Aspect	Regex	Spell	IPLoM	Drain3	ASN
Value normalization	Yes	Yes	Yes	Yes	Yes
Template discovery	No	Yes	Yes	Yes	Yes
Schema normalization	No	No	No	No	Yes
Online processing	Yes	Yes	No	Yes	Yes
Production-ready	High	Mod	Low	Very High	Very High

Table 1: Scope of Log Parsing Techniques

III. BACKGROUND AND RELATED WORK

Log parsing research has progressed through three distinct eras. The first generation relied entirely on manually authored patterns: regular expressions, Grok rules, and rule engines. These deterministic approaches provide high accuracy for known formats but scale poorly and require constant maintenance as vendors update their telemetry schemas.

The second generation introduced automated template mining: Spell (2016), IPLoM (2012), Drain (2017), and Drain3 (2020+). These algorithms discover structural patterns directly from log streams by identifying tokens that vary across messages (wildcards) versus those that remain constant (invariants). Drain3 employs a fixed-depth prefix tree to route log messages, computing token-level similarity against candidate templates and either merging with existing clusters or creating new templates as needed.

The third generation—adaptive schema-aware parsing—must address field-level normalization, not just value-level normalization. Existing research has primarily focused on improving rule-based parsing or automated template discovery. Current deterministic parsers normalize field values but rarely normalize field schemas. This work addresses this limitation by introducing an adaptive schema normalization layer that operates prior to template mining.

IV. PROPOSED ADAPTIVE SCHEMA NORMALIZATION ARCHITECTURE

The proposed architecture extends deterministic log parsing with adaptive schema normalization prior to online template mining. It enables structurally identical events from different vendors to produce consistent templates despite field naming differences. The architectural problem is to prevent transport variation, value variation, and vendor schema variation from reaching the template miner as unrelated structures.

A. Pipeline Design

The architecture consists of six sequential processing stages:

Stage 1: JSON Unwrapping – Security telemetry is frequently encapsulated within multiple transport-specific envelopes or embedded JSON payloads. The first stage recursively extracts the original security event by decoding escaped JSON strings, removing transport envelopes, and recovering the underlying payload.

Stage 2: Recursive JSON Flattening – Security events often contain nested structures: objects with sub-objects, arrays of events, hierarchical attribute relationships. Stage 2 flattens arbitrarily deep JSON hierarchies into dot-notation key-value pairs while preserving hierarchical content.

Stage 3: Deterministic Value Normalization – High-cardinality values (timestamps, IP addresses, UUIDs, process IDs, session tokens) cause template explosion when left unnormalized. Stage 3 applies deterministic pattern recognition to replace these values with semantic placeholders: IPs → <IP>, timestamps → <TIMESTAMP>, UUIDs → <UUID>, numeric IDs → <NUM>, hashes → <HEX>.

→ <UUID>, numeric IDs → <NUM>, hashes → <HEX>.

Stage 4: Adaptive Schema Normalization – This is the novel contribution. Stage 4 maps each flattened, nor-malized field to its canonical equivalent using three mechanisms: (i) Repository Lookup for approved aliases, (ii) LLM-Assisted Grounding retrieving the K nearest canonical templates and passing them to an LLM with the unre-solved field, and (iii) Analyst Approval Gate requiring human validation before any mapping enters the repository.

Stage 5: Drain3-Based Template Mining – With all fields now in canonical form and values normalized, Drain3 operates on a canonicalized event stream. The algorithm discovers templates by routing incoming token sequences through a fixed-depth prefix tree, computing similarity against candidate templates, and either merging with existing clusters or creating new ones.

Stage 6: Canonical Repository & Schema Evolution – Maintains versioned repository with approved mappings and schema drift detection.

B. Field Mapping Decision Flow

For an unresolved incoming field f – one with no approved alias in the repository – the engine retrieves the K nearest existing canonical templates. The drifted template, its unmapped field(s), and the canonical field names are passed to a large language model, prompted to propose which canonical field most plausibly denotes. The proposal is grounded entirely in retrieved template context and is never applied to an event until an analyst approves it.

C. Lifecycle of Canonical Field Mapping

The lifecycle transitions through states: Unresolved → Under Review → Active Alias. Upon drift detection, the mapping transitions to Reopened for re-evaluation. Rejected proposals leave fields unresolved.

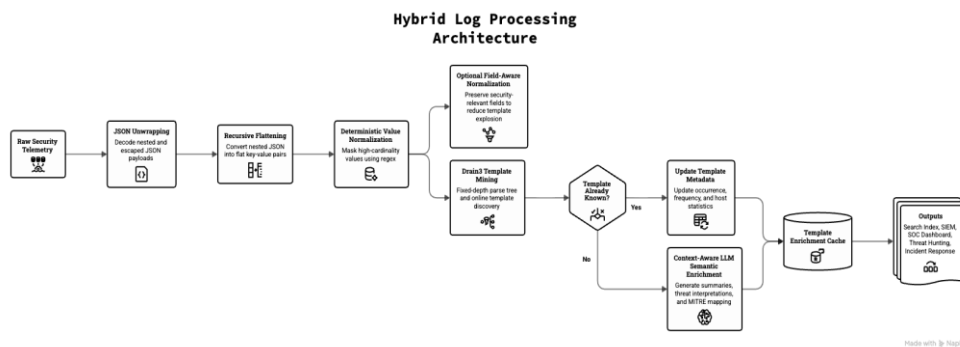


Figure 1: Figure 2: Field mapping decision flow showing unresolved fields grounded in nearest canonical templates, LLM proposal, and analyst approval gate.

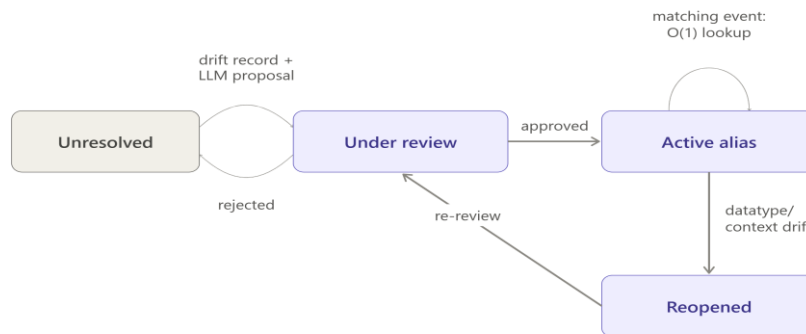


Figure 2: Figure 3: Lifecycle of canonical field mapping transitioning from unresolved through review to active state, with drift detection enabling reopening.

V.DRAIN3 OPERATION ON CANONICAL EVENTS

The remaining structural variation must be clustered without comparing each event with the entire template repository. Each canonical event is serialized with deterministic key ordering and tokenized before Drain3 processing. Routing first uses token count and then stable prefix tokens to reach a bounded parse-tree leaf. Similarity is evaluated only against clusters in that leaf. For an incoming token sequence L and candidate template T , the similarity measure is:

$$\text{sim}(L, T) = \frac{|\{i : T_i \neq \langle * \rangle \wedge T_i = L_i\}|}{|L|}. \quad (1)$$

If the configured threshold is satisfied, the cluster is reused. Tokens that vary within the cluster are generalized according to the Drain3 update rule. If no candidate satisfies the threshold, a new cluster is created. Canonicalization occurs before this comparison, so vendor aliases do not lower structural similarity.

A. Worked Example

Consider two normalized vendor events:

source_ip=<IP> destination_ip=<IP> destination_port=<NUM> client_ip=<IP> dst_ip=<IP> dst_port=<NUM>

After repository lookup and LLM-assisted mapping where no alias was previously approved, both canonicalize to:

src_ip=<IP> dst_ip=<IP> dst_port=<NUM>

The common representation follows one parse-tree path and reuses one template. Without schema normalization, the differing key tokens would reduce similarity or route the events to separate clusters.

B. Template Splitting and Tuning

Some canonical events must remain separate. A different token count, a stable protocol discriminator, or an additional field group may represent a genuinely different structure. The similarity threshold, tree depth, branch limit, and cluster limit therefore control the balance between unsafe merging and excessive fragmentation. Configuration values are versioned and evaluated against holdout time windows; they are not changed implicitly by runtime events.

C. Array Handling

Arrays must be stabilized before template mining because element order and cardinality can otherwise alter token positions. Arrays with meaningful order are flattened using indexed paths such as `dns.answers.0.type`. Arrays that represent unordered sets are sorted by a configured stable key before flattening. For highly variable collections, bounded cardinality summaries and explicit overflow markers are used. The selected policy is defined per canonical field so that identical inputs always produce identical token sequences.

VI.BENEFITS AND OPERATIONAL IMPACT

A. Reduced Template Fragmentation

Template fragmentation—the proliferation of templates caused by schema variation—imposes substantial operational costs. By normalizing schemas before template mining, ASN prevents equivalent events from producing multiple templates, directly improving memory efficiency, query performance, and ML model accuracy for anomaly detection.

B. Cross-Source Correlation and Threat Hunting

Modern attacks span multiple security tools. With vendor-specific schemas, threat hunting queries must union across field name variations. With canonical schemas, queries simplify dramatically, enabling faster query development, fewer missed correlations, and more maintainable detection rules.

C. Deterministic Replaying and Forensics

Security investigations often require replaying events through analysis pipelines. ASN enables deterministic replay: the same sequence of field mappings applied to historical events produces identical results today. Repository version-ing preserves the exact mappings in effect when events were processed, enabling bit-for-bit reproducible analysis.

D. Schema Evolution Without Re-Parsing

When a vendor upgrades from v1.0 to v2.0 with schema changes, traditional parsers require rule updates. With ASN, schema evolution is managed at the mapping layer: new vendor fields are recognized as schema drift, grounded in existing templates, proposed by LLM, and approved by analysts.

E. Multi-Vendor Environments

Organizations typically run security stacks from multiple vendors. ASN allows these heterogeneous sources to be normalized into a unified canonical representation, enabling a single SIEM data model, unified threat detection, and consistent compliance reporting.

F.LLM-Assisted Field Mapping Reduces Manual Work

Field mapping is historically a manual, error-prone task. By grounding LLM proposals in retrieved canonical templates and requiring analyst approval, ASN reduces mapping effort while maintaining auditability.

VII. EXPERIMENTAL EVALUATION

A. Evaluation Metrics

The evaluation uses the following measures:

- template count and singleton-template ratio before and after schema normalization;
- template reuse across vendors and software versions;
- LLM-proposal precision, recall, and abstention rate against analyst-validated field pairs;
- LLM-suggestion acceptance, edit, and rejection rate by analysts;
- drift-detection delay and false-positive rate;
- analyst decisions required per newly observed field;
- event throughput, end-to-end latency, and repository lookup latency;
- Deterministic replay agreement for fixed parser and repository versions.

Let K_b and K_c denote the numbers of templates before and after canonical mapping. Template-fragmentation reduction is reported as:

$$R_f = 1 - \frac{K_c}{K_b}. \quad (2)$$

This measure is interpreted together with mapping precision because a lower template count is not beneficial when semantically different fields have been merged.

B. Cross-System Evaluation

Claims concerning canonical mapping require a separate benchmark containing equivalent events expressed under heterogeneous schemas, controlled field-renaming experiments, or analyst-labelled mapping pairs.

VIII. THREATS TO VALIDITY AND LIMITATIONS

A. Mapping Ground Truth

Canonical equivalence is domain dependent. Analyst-labelled mappings may contain disagreement, and a field name can have different meanings across products. Vendor- and version-scoped aliases are therefore required.

B. Normalization Sensitivity

Over-normalization may merge distinct concepts, whereas under-normalization preserves fragmentation. Mapping precision and abstention must be reported with template-count reduction.

C. Dataset Scope

The current event sets demonstrate deterministic template mining but do not cover all vendor schema changes. Evaluation across multiple products and controlled schema versions is required before generalizing drift results.

D. Distribution Change

Representative value profiles may change because of operational behaviour rather than an actual schema change. Because resolution is analyst-gated rather than score-driven, a sustained shift is simply re-surfaced as drift for review rather than resolved automatically.

E. Array and Ordering Effects

Indexed arrays can increase template count when order changes, while sorting can remove meaningful order. Array policies must consequently be selected and versioned by field semantics.

F. Mapping Integrity and Adversarial Risk

Because unresolved-field mapping is LLM-proposed, the primary integrity risk is direct manipulation of the model's input. A compromised or malicious log source could craft field names, values, or structure to steer proposals toward unrelated canonical fields. The mandatory analyst-approval gate is therefore not a convenience but the primary safe-guard: no LLM proposal alters parser state until approved, every proposal is logged with retrieved template context, and repository updates remain scoped per vendor so one source cannot influence mappings for another.

IX. DEPLOYMENT AND REPRODUCIBILITY

A production rollout proceeds through four controlled phases:

- validate envelope decoding, flattening, key ordering, and malformed-event fallbacks;
- establish a versioned canonical repository from known vendor schemas;
- measure LLM-suggestion acceptance rate and analyst agreement on a shadow stream before enabling the re-view queue in production;
- activate canonical Drain3 mining while monitoring drift, fragmentation, latency, and replay agreement.

Reproducibility requires dataset hashes, parser parameters, normalization patterns, canonical repository snapshots, nearest-

template retrieval settings (K), the LLM model version and prompt template, and analyst decisions to be retained. Each output event records its parser version, schema version, and template identifier. Historical runs can then distinguish genuine telemetry change from configuration, model, or repository change.

X. FUTURE WORK

Future evaluation should construct labelled cross-vendor schema benchmarks, measure LLM-suggestion accuracy and acceptance rate across telemetry domains and model choices, and investigate privacy-preserving sharing of canonical aliases. Downstream reasoning systems may consume canonical templates for detection or investigation, but such reasoning remains separate from the deterministic parser and does not modify mappings without validation.

XI. CONCLUSION

Deterministic value normalization alone does not prevent template fragmentation caused by heterogeneous or evolving field schemas. The revised architecture places LLM-assisted schema normalization before Drain3 so online discovery operates on canonical field identities. Approved aliases are reused automatically, unresolved fields are grounded in nearest canonical templates and proposed by an LLM for analyst validation, and versioned repositories preserve auditability and replay.

This progression from rule-based extraction to automated mining and adaptive schema normalization provides stable canonical templates for interoperable security analytics. The architecture is practical: it preserves determinism, maintains auditability, integrates with existing tools (Drain3), and scales to production telemetry volumes. Organizations adopting this approach can expect reduced template counts, improved cross-source correlation, and lower maintenance burden as they scale their security operations.

The integration of large language models into a deterministic, auditable, analyst-gated pipeline demonstrates how modern AI can enhance rather than replace human judgment in security-critical systems. The mapped repository becomes a shared knowledge asset, encoding the organization's understanding of vendor schemas and field semantics, beneficial for training new analysts, onboarding new tools, and defending against schema drift.

REFERENCES

1. P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *IEEE International Conference on Web Services (ICWS)*, 2017.
2. LogPAI, "Drain3," GitHub repository, <https://github.com/logpai/Drain3>.
3. P. He et al., "An evaluation study on log parsing and its use in log mining," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020.
4. OCSF Project, "Open cybersecurity schema framework (ocsf)," <https://schema.ocsf.io/>.
5. Elastic, "Elastic common schema (ecs) reference," <https://www.elastic.co/guide/en/ecs/current/index.html>.
6. Splunk Inc., "Splunk common information model (cim) add-on," <https://docs.splunk.com/Documentation/CIM>.
7. Open Information Security Foundation, "Suricata ids/ips/nsm engine," <https://suricata.io/>