# Long-Horizon Semantic Complex Event Processing over Data and Event Streams with LLMs

Junhan Liu
Brown University
junhan_liu@brown.edu

## ABSTRACT

Monitoring continuous data streams for meaningful signals increasingly requires long horizon, stateful reasoning over unstructured content. Existing large language model frameworks are primarily designed for stateless, one-shot interactions, while traditional Complex Event Processing systems support temporal pattern detection over structured and typed event streams. This leaves a critical gap for applications that must detect evolving semantic phenomena in free text streams.

To bridge this gap, this paper introduces *sem_pattern*, a semantic event pattern operator that extends CEP capabilities to unstructured text. Implemented within the VectraFlow streaming dataflow system, *sem_pattern* avoids requiring predefined structured input schemas by using language models to materialize atomic semantic events from raw documents. It then evaluates these extracted signals against user-specified temporal patterns using a deterministic, automata-based runtime.

By decoupling semantic extraction from temporal state progression, the system efficiently compiles natural language monitoring goals into executable operator graphs and maintains intermediate temporal states across continuous inputs. Ultimately, *sem_pattern* provides a principled, interpretable, and scalable execution model for detecting complex event sequences over long-running unstructured streams.

## 1 INTRODUCTION

Large language models have emerged as a powerful tool for interpreting unstructured data, including few shot reasoning and structured information extraction from text [7, 10]. Nevertheless, most existing pipelines remain stateless and episodic by evaluating isolated prompts over static corpora without supporting persistence or reasoning that unfolds over time. Many real world applications require the detection of signals emerging across sequences of events rather than single documents. For instance, identifying a deteriorating patient, flagging escalating compliance violations, or detecting multistep fraud patterns requires long horizon stateful reasoning over unstructured streams. This capability remains largely inaccessible to both single prompt evaluations and modern agentic models.

Existing systems address only part of this challenge. Frameworks such as Palimpzest, LOTUS, and DocETL introduce semantic operators for unstructured data but primarily operate in batch settings with limited support for cross record state [17, 21, 23]. In contrast, complex event processing systems enable long horizon temporal reasoning over streams through event pattern matching, automata based execution, and formal event semantics [2, 8, 12, 24]. Production systems such as FlinkCEP and Esper further demonstrate the practical importance of event pattern operators in streaming engines [6, 11]. However, these systems assume structured event streams with predefined schemas and event types. Neither approach suffices when events must first be inferred from unstructured text before being reasoned over temporally.

To address the disconnect between unstructured text comprehension and rigorous temporal state tracking, this paper presents the semantic pattern operator. This architecture unifies semantic interpretation with complex event detection in a single framework, building on recent efforts to integrate semantic and vector-based processing into streaming dataflows [9, 18]. Specifically, VectraFlow extends traditional event recognition to unstructured text by executing across three coupled layers. First, a semantic extraction layer maps the input stream of documents to explicit atomic events, resolving structured metadata directly while using models to populate unstructured attributes. Second, a rule specification and compilation layer allows users to define patterns via a domain-specific language or a natural language interface. This layer translates intent into an executable nondeterministic finite automaton governed by explicit transition semantics. Third, an event time execution layer manages concurrent partial matches over partitioned event streams, using watermark buffers to reconcile discrepancies between ingestion time and event occurrence [3].

This design fundamentally separates semantic uncertainty from temporal rule execution. Unlike monolithic classifiers that attempt to predict complex satisfaction in a single pass, *sem_pattern* materializes intermediate atomic events and delegates reasoning to a deterministic automaton. This approach makes detection compositional and auditable while supporting standard streaming mechanisms such as keyed state isolation and temporal pruning.

## 2 RELATED WORK

**Complex Event Processing.** Complex event processing engines are designed to detect high-level situations from low-level event streams based on precise temporal and logical constraints [5, 8, 12]. While foundational systems excel at stateful pattern matching [6, 24], they rigidly assume that input streams consist of typed, machine readable tuples. This structural assumption is limiting in domains such as clinical analytics where evidence for an event is often distributed across heterogeneous free text. *sem_pattern* builds upon the formal execution models of traditional engines but extends them to operate over unstructured documents via upstream semantic materialization.

**Information Extraction with LLMs.** The extraction layer of *sem_pattern* treats large language models as schema guided extractors to transform unstructured text into structured records. This strategy aligns with recent advancements that leverage the few-shot reasoning capabilities of models to populate complex schemas from domain-specific literature and records [7, 10]. By isolating the

**Table 1: Core operators in the declarative CEP DSL.**

| Operator | Semantics |
|---|---|
| Seq ($A \to B \to C$) | Ordered, discontiguous occurrence of subpatterns. |
| Or ($A \vee B$) | Match of either subpattern. |
| And ($A \wedge B$) | Match of both subpatterns in any order. |
| Not ($\neg A$) | Absence of a subpattern within a bounded interval. |
| Within (`within` $\Delta t$) | Event-time bound on a subpattern. |
| Times (`times(`$n$`)`) | Exact repetition of a subpattern. |

extraction phase, our system bounds the workload of the model to per-document interpretation rather than forcing it to manage multi-document temporal logic, which often leads to context window exhaustion.

**Semantic Parsing and Synthesis.** The ability to compile natural language into executable pattern representations shares lineage with the broader semantic parsing literature, which traditionally translates natural language intent into executable queries such as SQL [25]. However, rather than targeting relational algebra, our compilation targets stateful automata. Furthermore, the use of critique and refinement loops to detect structural errors and missing temporal bounds reflects modern test time refinement strategies for model outputs [19], ensuring the synthesized patterns are logically sound.

**Stream Processing Semantics.** Handling out-of-order data is a fundamental challenge in distributed stream processing. Modern engines manage this via watermark buffers, which provide a mechanism to track progress in event time despite ingestion delays [3]. The *sem_pattern* integrates this temporal processing model into its runtime. This integration ensures that the delayed ingestion of unstructured text does not compromise the correctness of sequence matching or negation timeouts, even when a note processed today describes an event that occurred days earlier.

# 3 METHODOLOGY

## 3.1 Declarative CEP DSL and NFA-Based Pattern Execution

The `sem_pattern` operator exposes semantic event rules through a declarative domain-specific language. The goal of the language is to specify the temporal condition to be detected, while leaving physical execution to the compiler and runtime. The language operates over a stream of atomic events extracted via the semantic extraction layer (Layer 2 of Figure 1). Each event carries an entity key, an event timestamp, an event type, and a set of attributes. An atomic predicate $E(\phi)$ matches an input event when the event satisfies the type constraint and the attribute predicate $\phi$.

Table 1 summarizes the core operators. These operators express complex event rules as logical patterns rather than procedural state-machine code. For example, a rule requiring event $A$ followed by the absence of event $B$ within thirty days is represented as a structured pattern expression (depicted as the Compiled Pattern AST in Layer 1 of Figure 1) and compiled into an executable automaton.

Formally, a pattern $P$ is defined inductively as follows:

$$\begin{aligned} P ::= {} & E(\phi) \mid \textsc{Seq}(P_1, \ldots, P_n) \mid \textsc{Or}(P_1, P_2) \\ & \mid \textsc{And}(P_1, P_2) \mid \textsc{Not}(P) \mid \textsc{Within}(P, \Delta) \\ & \mid \textsc{Times}(P, m) \end{aligned}$$

where $\Delta$ denotes an event-time interval and $m$ denotes an exact repetition count. Sequence semantics are discontiguous: unrelated events may occur between two matched subpatterns as long as ordering and time constraints are preserved. This choice is important for long-horizon monitoring, where relevant events are often separated by many irrelevant stream records.

The compiler lowers each pattern into a nondeterministic finite automaton (illustrated in Layer 3 of Figure 1), following standard automata-based execution models for complex event recognition [8]. The compiled automaton is defined as $\mathcal{N} = (Q, q_0, F, T)$, where $Q$ is the set of states, $q_0$ is the initial state, $F \subseteq Q$ is the set of accepting states, and $T$ is the transition relation. The `sem_pattern` operator uses three transition types. A Take transition consumes the current event when its guard predicate is satisfied. A Proceed transition is an $\varepsilon$-transition that advances the automaton without consuming an event. An Ignore transition preserves a partial match while skipping an event that is irrelevant to the current subpattern; this is the mechanism that implements discontiguous matching for sequence patterns.

Negation requires bounded event-time semantics because the absence of an event cannot be established from a single input record. For a pattern containing Not, the compiler creates a monitoring state with an Ignore loop guarded by the complement of the forbidden predicate. If the forbidden predicate is satisfied before the temporal bound closes, the corresponding partial match is discarded. If the negated subpattern appears at the end of a rule, the runtime defers emission until the event-time boundary has elapsed. Thus, negative matches are produced only when the watermark indicates that no earlier event within the configured lateness bound can still arrive.

At runtime, the automaton engine maintains a set of active partial matches. For each incoming event, the engine first creates any new runs enabled from the initial state and then advances existing runs through all enabled transitions. Each partial match records its current state, consumed events, entity key, and start timestamp. Unless a rule specifies a more restrictive selection policy, the `sem_pattern` operator applies *all-matches* semantics, meaning that multiple valid partial matches may coexist and emit independently. As illustrated on the right side of Figure 1, keyed streams are routed to per-key NFA rule detectors that emit complex event detection results in real time.

For partitioned streams, the `sem_pattern` operator evaluates each entity key independently while sharing the same compiled automaton topology. This ensures that events from different entities, for example Patient 1 in Layer 3 of Figure 1, never contribute to the same complex event instance. Event-time execution is handled through a watermark buffer, following standard stream-processing semantics [3]. The runtime processes events in timestamp order up to a configured lateness bound and prunes partial matches whose temporal windows can no longer produce valid completions.

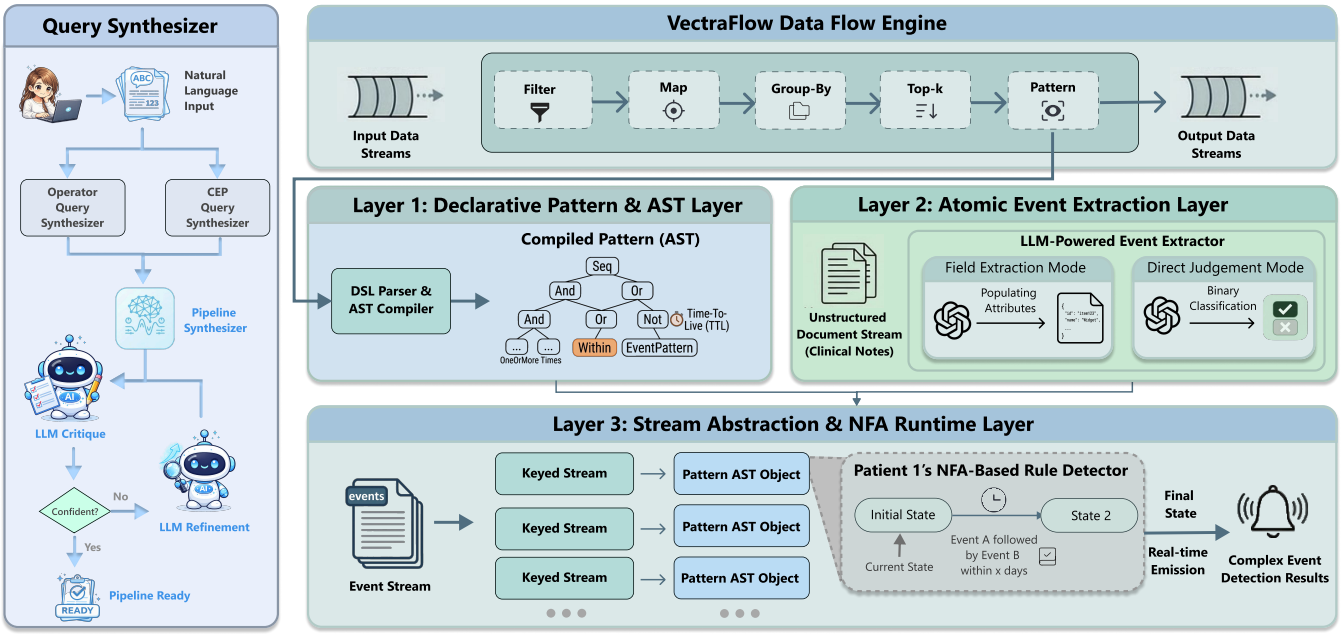


**Figure 1: Event Processing Architecture.**

## 3.2 Natural Language to DSL Compilation with Self Refinement

The `sem_pattern` operator allows users to describe monitoring goals in natural language. Executing these descriptions directly with a language model would make the detection logic opaque and difficult to verify. To preserve transparency, the operator translates each natural language intent into the structured DSL used by the execution engine. This follows the semantic parsing paradigm, where a natural language utterance is mapped to an executable logical form [25]. In our setting, the logical form is a typed CEP pattern AST built from the operators supported by `sem_pattern`.

The compiler accepts free-form rule text describing events, ordering, absence, repetition, and temporal bounds. Instead of allowing the language model to emit arbitrary code, the generation target is constrained to a typed intermediate representation whose constructors correspond directly to the supported DSL operators, including Seq, And, Or, Not, Within, and Times. The generated pattern is therefore inspectable before execution and can be lowered by the same compiler used for manually authored `sem_pattern` rules.

The operator adopts a pattern-first compilation strategy. The initial model call constructs the pattern AST and assigns descriptive names to the atomic event types referenced by the rule. The compiler then extracts these referenced event types and issues a second model call to define only the corresponding event catalog, including event descriptions, attributes, and extraction prompts. This ordering keeps the event catalog tightly coupled to the executable pattern, avoids unused event definitions, and provides the semantic extraction layer with schemas aligned to the downstream runtime query.

After initial generation, the compiler performs a critique phase before accepting the DSL representation. Deterministic structural checks verify that each operator is well formed and that every event referenced by the pattern has a corresponding catalog entry. Static analysis flags unsafe or underspecified constructs, especially unbounded negation and missing temporal constraints. An optional model-driven alignment check compares the generated pattern and event catalog against the original rule, identifying missing conditions, extra assumptions, ambiguous intent, and low-confidence translations.

When the critique reports errors or insufficient confidence, the `sem_pattern` compiler invokes a bounded self-refinement loop. The refiner receives the original rule, the current AST, the event catalog, deterministic error messages, alignment feedback, and any user clarifications collected through a callback mechanism. It then produces a repaired pattern representation under the same typed schema constraints. Clarifications are accumulated across rounds so that the system does not ask redundant questions. This procedure is similar in spirit to iterative language model refinement [19], but the feedback is grounded in DSL validation and CEP specific static checks.

The loop terminates when validation succeeds, when the repaired pattern no longer changes, when required user input is unavailable, or when a fixed round limit is reached. The final output is a verified runtime DSL pattern together with its event catalog. The pattern is passed to the `sem_pattern` compiler for NFA lowering and stream execution, while the catalog guides the upstream semantic extraction layer. By restricting natural language to the construction and refinement of an explicit intermediate representation, the `sem_pattern` operator preserves interpretability, enables deterministic validation before execution, and allows generated rules to run on the same runtime as manually written patterns.

## 3.3 Ground Truth Synthesis Pipeline

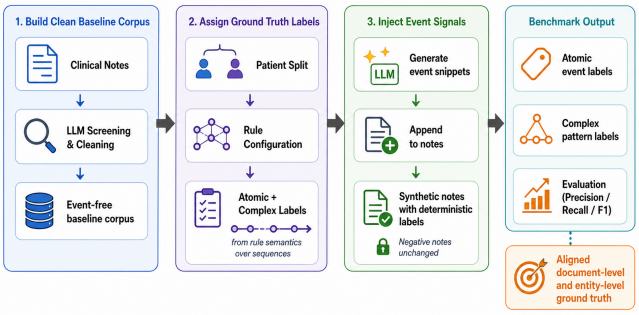


**Figure 2: The ground truth synthesis pipeline, illustrating the progression from raw clinical notes to aligned document and entity level labels.**

Evaluating the `sem_pattern` operator requires ground truth at two granularities. The system must know whether each document contains a target atomic event, and whether the ordered document sequence for an entity satisfies a complex temporal rule. Manual annotation of such labels is expensive and difficult to audit, while direct synthetic generation may produce unrealistic clinical narratives. We therefore construct a controlled benchmark that preserves real clinical context from MIMIC IV notes while assigning deterministic labels for both atomic events and complex event patterns. Figure 2 illustrates the overall architecture of this synthesis pipeline.

The pipeline consists of three stages. First, as shown in Step 1 of Figure 2, we construct a clean baseline corpus in which target events are absent. Given a set of atomic event definitions, an LLM based screener examines each note and identifies whether it contains any event trigger. Notes without target signals are retained. Notes with localized triggers are rewritten to remove the event specific content while preserving the surrounding clinical narrative. Notes dominated by target event descriptions are discarded because neutralization would substantially alter their meaning. To reduce residual leakage, each rewritten note is passed through a second verification loop. A lenient validator checks whether any target signal remains, and notes that fail validation are rewritten again with neutral descriptions rather than simple deletion. This process produces an event free baseline while maintaining realistic document structure and clinical context.

Second, as depicted in Step 2 of Figure 2, we assign ground truth labels through a hierarchical configuration process. Patients are divided into a control group with no target events and an experimental group assigned to complex event rules. For each patient in the experimental group, the assigned rule determines which atomic events must be injected into which notes. Events required by positive rule paths are labeled as present, events forbidden by negation constraints are forced to remain absent, and unrelated events are labeled as absent. These document level labels are then aggregated into patient level complex event labels by evaluating the rule semantics over the labeled sequence. A patient is positive for a complex pattern only if the sequence satisfies all predicate, ordering, negation, and event time constraints.

Third, corresponding to Step 3 of Figure 2, we convert the logical labels into unstructured text through targeted event injection. For each positive atomic event label, an LLM generates a short clinical style snippet that implies the corresponding event. Instead of regenerating the entire note, the snippet is appended to the neutralized baseline note as an addendum. This additive construction preserves the original clinical narrative, limits hallucination, and ensures that each injected signal corresponds exactly to the predefined label assignment. Negative notes are left unchanged after neutralization.

This synthesis pipeline yields a benchmark with aligned document level atomic labels and entity level complex pattern labels (Benchmark Output in Figure 2). The resulting data supports quantitative evaluation of extraction quality, temporal pattern detection, and end to end performance using precision, recall, and F1 score.

## 3.4 Embedding-Based Similarity Filtering for Extraction Efficiency

The semantic extraction layer invokes an LLM over clinical notes to materialize atomic events. Input text often contains some sentences that are unrelated to the event detection task, so applying extraction to the entire document can waste tokens and increase latency. This part adds an optional embedding-based filtering stage before LLM extraction. The filter is used as a lightweight retrieval step: it reduces the textual context passed to the extractor while preserving sentences that are semantically close to the event definitions used by the active rules. Specifically, this filtering mechanism invokes a privately deployed instance of the text-embedding-3-small model [20] on Azure to generate dense vector representations for both the event descriptions and the document sentences.

For each rule, the system first constructs query texts from the registered atomic event definitions. Concretely, each event is converted into a short natural language description containing the event name and its semantic description. These descriptions are embedded once and reused as rule level targets. For each input document, the source text is split into sentence level candidates using a deterministic sentence chunker that removes very short or noninformative fragments. The candidate sentences are embedded in batch, and the system computes cosine similarity between every candidate sentence and every target event description.

Let $Q = \{q_1, \ldots, q_m\}$ denote the embeddings of event descriptions and let $S = \{s_1, \ldots, s_n\}$ denote the embeddings of candidate sentences in a document. A sentence $s_i$ is retained if

$$\max_{q_j \in Q} \frac{s_i^\top q_j}{\|s_i\|_2 \|q_j\|_2} \geq \tau,$$

where $\tau$ is a configurable similarity threshold. The implementation performs this operation as a matrix computation by normalizing target and candidate embeddings and taking the maximum similarity across all event targets. The retained sentences are concatenated to form a filtered document. Documents with no retained sentence are skipped for the corresponding extraction pass.

This filtering stage is applied before the LLM sees the document. It only changes the evidence context supplied to the semantic extractor. The threshold $\tau$ controls the tradeoff between efficiency and recall. A lower threshold keeps more context and reduces the risk

of discarding weakly phrased evidence; a higher threshold sends fewer tokens to the LLM but may remove indirect mentions.

The design follows the same intuition as dense retrieval methods, where semantic similarity in an embedding space is used to select candidate passages before expensive reasoning or generation steps [15]. In VectraFlow's `sem_pattern` operator, retrieval is not used to answer the rule itself. Instead, it serves as a transparent preprocessing phase that restricts the extraction context before pattern construction.

# 4 EVALUATION

## 4.1 Datasets

The `sem_pattern` operator is evaluated on two document stream sources that stress different parts of the pipeline. The first source is MIMIC IV discharge reports. MIMIC IV is a deidentified electronic health record resource collected from Beth Israel Deaconess Medical Center, and MIMIC IV Note provides deidentified free text clinical notes linked to the clinical database [13, 14]. Discharge reports are utilized because they are long clinical narratives that describe the reason for admission, hospital course, diagnoses, procedures, medications, discharge instructions, and follow up plans. They provide realistic unstructured evidence for evaluating whether semantic event definitions can be materialized into atomic events from clinical text.

The second source is TCELongBench, a benchmark for temporal complex events over online news [27]. TCELongBench defines a temporal complex event as a collection of related news articles that unfold over an extended period, and evaluates temporal and long context understanding through reading comprehension, temporal sequencing, and future event forecasting tasks. The benchmark contains 2289 temporal complex events and 88821 question answering pairs. For these experiments, TCELongBench articles are converted into the exact document stream schema utilized by the `sem_pattern` operator, where each article functions as a timestamped document keyed by a complex event identifier. This setting is suitable for evaluating long horizon complex event detection because relevant evidence may be distributed across many articles, timestamps, and semantic event types.

MIMIC IV discharge reports evaluate atomic semantic event extraction from real clinical documents. TCELongBench evaluates whether extracted events can be composed across document boundaries and time into complex temporal patterns. Together, these two benchmarks span the full execution pipeline, from raw text to atomic event extraction to automaton-based temporal pattern matching.

## 4.2 Benchmark Construction

Both datasets are used with the ground truth synthesis pipeline described in Section 3.3. In total, the evaluation benchmark contains 256 MIMIC-IV discharge reports and 505 TCELongBench article documents. This scale is sufficient to exercise both document-level extraction and entity-level temporal pattern detection while keeping repeated LLM-based evaluation tractable.

An event-free baseline is first constructed by screening and neutralizing documents that contain target event evidence. Labels are subsequently assigned through a hierarchical configuration process detailing which atomic events must occur in each document and which entity-level complex rules must be satisfied. Finally, targeted event injection converts these logical labels into unstructured text by inserting short natural language evidence into selected documents.

This construction gives two aligned label spaces. At the document level, it provides labels for evaluating atomic event extraction. At the entity level, it provides labels for evaluating whether the `sem_pattern` runtime detects the intended temporal rule over the ordered document stream. For MIMIC-IV, the injected evidence is placed into realistic discharge report contexts. For TCELongBench, the same procedure produces long article sequences in which required and forbidden events can be separated by substantial temporal distance.

## 4.3 Baselines

The `sem_pattern` operator is compared against multiple baselines implemented within the experimental harness.

The direct LLM baseline receives the rule description, event definitions, and source text, and directly predicts whether the complex rule is triggered. It does not expose intermediate atomic events and does not use automaton based temporal matching. In the multi document setting, the implementation supplies cumulative entity history to the model before producing the final entity level decision.

The direct LLM with retrieval baseline applies the embedding filter before the direct decision prompt. It uses the same rule and event descriptions as retrieval queries, keeps only relevant text fragments, and asks the LLM to decide the complex event label from the reduced context. This baseline tests whether retrieval alone can compensate for long contexts without explicit event state or temporal automata.

The offline batch LLM baseline simulates periodic monitoring. At fixed time intervals, it evaluates each rule over the cumulative context available for an entity and stops after a positive decision when configured to do so. This baseline uses the same direct detection prompt semantics as the direct LLM baseline, but changes the execution policy from per document evaluation to scheduled batch evaluation.

The `sem_pattern` system first extracts atomic events with the semantic extraction layer and then applies the compiled pattern over a keyed event stream. The retrieval variant inserts the embedding filter before extraction, reducing the text passed to the LLM while preserving the same downstream automaton. The separated extraction variant runs rule specific extraction passes before the same pattern runtime. These variants isolate the effect of retrieval and extraction organization while keeping the automaton based execution semantics fixed.

## 4.4 Online Streaming `sem_pattern` Detection

Online complex event detection is first evaluated under four execution settings: Direct LLM, Direct LLM with embedding filtering, `sem_pattern`, and `sem_pattern` with embedding filtering. All settings use the same rule text, event definitions, input documents, entity keys, timestamps, and ground truth labels. The only difference is the execution strategy. Direct LLM asks the model to decide whether a complex event is triggered by the available document

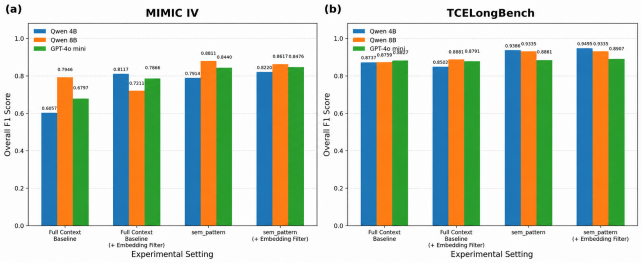


**Figure 3: Overall F1 scores on MIMIC-IV and TCELongBench across four execution settings.**

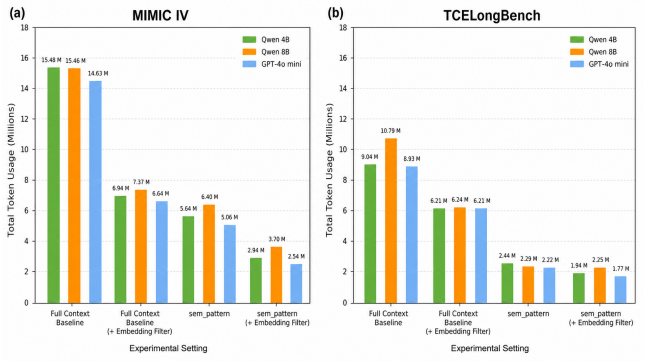


**Figure 4: Total token consumption across datasets and execution settings.**

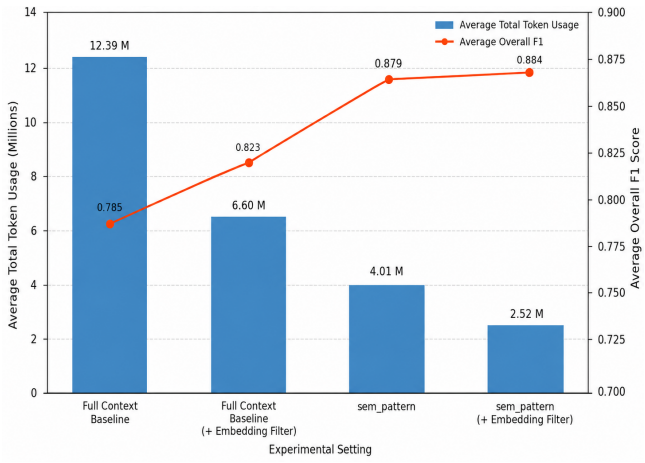


**Figure 5: Average Token Usage and F1 Across Experimental Settings**

context. Direct LLM with embedding filtering first reduces the context using the sentence filtering method described in Section 3.4 with a threshold of 0.3, and then applies the same direct decision prompt. This filtering step is embedding based and does not consume LLM inference tokens. In contrast, `sem_pattern` first materializes atomic semantic events from documents and then evaluates the compiled rule with the NFA runtime described in Section 3.1. Finally, `sem_pattern` with embedding filtering applies the same sentence filter before atomic event extraction, while preserving the downstream automaton-based execution. This controlled comparison isolates the effect of explicit event extraction and automaton-based temporal matching from differences in task specification or data construction.

Figure 3 evaluates execution accuracy, demonstrating that the `sem_pattern` operator consistently outperforms monolithic full-context reasoning. This architectural advantage is most pronounced on MIMIC IV, where accumulating evidence across lengthy clinical notes overwhelms direct language model judgment. For instance, the Qwen 3 4B [22] full context baseline achieves a 0.606 F1 score. While embedding filtering improves this to 0.812, `sem_pattern` with filtering attains the highest observed score at 0.822. The same phenomenon occurs with GPT 4o mini [1] on MIMIC IV, where filtered `sem_pattern` reaches 0.848 compared to the 0.680 baseline.

Results on the longitudinal news articles of TCELongBench corroborate these findings. The `sem_pattern` architecture with embedding filtering achieves the highest Qwen F1 at 0.934, significantly improving upon the 0.876 baseline. While GPT 4o mini exhibits a narrower performance gap due to stronger baseline capabilities, the filtered `sem_pattern` configuration still delivers the peak F1 score of 0.891. Ultimately, decoupling atomic semantic materialization from stateful temporal execution substantially improves detection performance. Embedding filtering further contributes by pruning irrelevant context before extraction, suggesting that explicit automaton based matching can offer advantages for long-horizon complex event detection.

Figure 4 quantifies the token cost of different execution strategies. The full context baseline is expensive because each decision requires the model to reread accumulated entity history, averaging 15.2M tokens on MIMIC IV and 9.6M tokens on TCELongBench. Embedding filtering reduces this cost by pruning irrelevant text before prompting, lowering the averages to 7.0M and 6.2M tokens, respectively. However, filtering alone still leaves the LLM responsible for complex event reasoning over the remaining context.

The `sem_pattern` operator changes the cost structure more fundamentally. Without embedding filtering, it reduces average token consumption to 5.7M on MIMIC IV and 2.3M on TCELongBench, corresponding to reductions of 62.5% and 75.8% relative to full context execution. With embedding filtering, the averages further drop to 3.1M and 2.0M tokens, giving roughly 80% reduction on both datasets. The result shows that most of the efficiency gain comes from removing temporal composition from the LLM path: the model is used for bounded semantic materialization, while long horizon state progression is handled by the automaton runtime.

Figure 5 summarizes the efficiency and accuracy tradeoff across execution strategies. The monolithic full-context baseline has the highest token cost and the lowest overall F1 in this comparison, with an average F1 score of 0.785. By moving temporal reasoning to an explicit state machine, `sem_pattern` reduces token consumption by nearly 68% while improving the overall F1 score to 0.879. Embedding filtering provides an additional complementary optimization. When combined with `sem_pattern`, it reduces total token usage by 80% relative to the baseline and achieves the highest overall F1 score of 0.884. These results suggest that pruning irrelevant context is

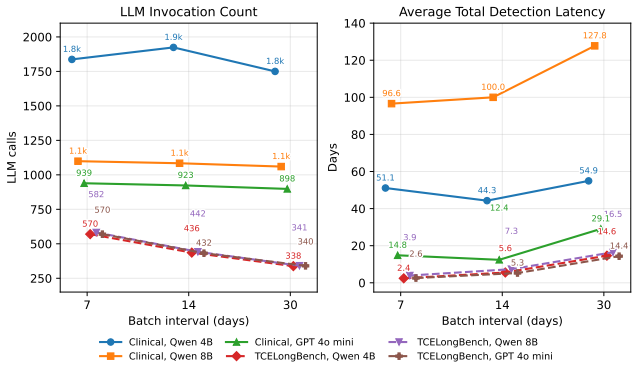


**Figure 6: Interval delay tradeoff.**

helpful, while separating temporal state progression from LLM inference accounts for the larger efficiency gain.

## 4.5 Periodic Batch Baseline

This section evaluates a periodic batch LLM baseline under long-horizon monitoring settings. This baseline groups documents by entity and evaluates each rule only at fixed batch ticks. At each tick, the LLM receives the cumulative entity context observed so far, together with the same rule definitions and event descriptions used by the other settings, and directly predicts whether the complex event has occurred.

This setting differs from the `sem_pattern` operator in its execution semantics. It does not materialize atomic events or maintain automaton states over a keyed stream. Instead, each invocation repeats direct rule detection over an expanding textual history. This baseline quantifies the cost–latency tradeoff of periodic full-context monitoring relative to online event-driven execution.

Figure 6 illustrates the effect of the batch interval on LLM invocation count and detection delay. Across these configurations, LLM invocations decrease from 933 at the 7-day interval to 788 at the 30-day interval, a 15.5% reduction. This reduction is more pronounced on TCELongBench, where the average number of calls drops from 574 to 340 because fewer batch ticks are evaluated over the shorter time horizon. The clinical dataset shows a smaller reduction, from 1292 to 1236 calls on average. Reducing the number of model invocations leads to higher detection latency. On average, latency increases by 50.1 percent across all settings. The effect is especially clear on TCELongBench, where average latency rises from 3.0 to 15.1 days as the batch interval grows from 7 to 30 days.

These results show that periodic batching offers only modest invocation savings while delaying detection. Short intervals reduce latency but require frequent repeated evaluation, whereas long intervals reduce invocation frequency but still require each LLM call to reason over accumulated entity context. This tradeoff supports the motivation for decoupling semantic materialization from temporal state progression. By using the language model for per-document atomic event extraction and delegating sequence evaluation to a deterministic automaton, `sem_pattern` reduces reliance on repeated full-context rule detection over continuous streams.

# 5 DISCUSSION

The architectural contribution of the `sem_pattern` operator lies in the separation of semantic materialization from temporal state progression. Guiding large language models to natively manage state across expanding context windows tightly couples correctness, inference latency, and financial cost within a single opaque execution step. This architecture realigns semantic monitoring with foundational stream processing principles, ensuring that temporal correctness and resource costs remain transparently tunable at the runtime layer [4]. Furthermore, this explicit design creates a natural path toward declarative query optimization, allowing model selection, prompting strategies, and operator placement to be planned dynamically rather than hardcoded [17].

The primary limitation of this decoupling is that upstream semantic hallucinations immediately translate into persistent stateful errors. Because the underlying automaton is strictly deterministic, a misclassified event type or hallucinated timestamp permanently corrupts downstream pattern matching. While augmenting the pipeline with retrieval mechanisms mitigates context exhaustion and localizes evidence [16], it concurrently introduces ranking volatility and expands the privacy attack surface when querying sensitive histories [26]. Consequently, robust deployment demands more than formal automaton correctness. It requires calibrated semantic extraction, principled abstention under uncertainty, and sophisticated mechanisms to revise state when subsequent documents contradict prior classifications.

One future direction is to make semantic event streams transactional rather than merely append only. Instead of committing each extracted event as the final state, the runtime should attach confidence, provenance, and version metadata to every semantic event, allowing uncertain matches to remain tentative until corroborated by later evidence. This would enable rollback, selective re-extraction, and replay when prompts, models, or event schemas change. The next generation of LLM-augmented stream processors should treat model calls as adaptive physical operators: the system should choose when to retrieve, when to invoke a small or large model, when to abstain, and when to verify based on latency budgets, uncertainty, and downstream state impact. Such an architecture would move beyond using LLMs as isolated extractors and toward a principled runtime for approximate, auditable, and revisable semantic state.

# 6 CONCLUSION

`sem_pattern` highlights a core design principle for semantic stream processing: semantic event materialization and temporal state progression should be handled by separate components. The operator keeps LLM inference focused on bounded per-document extraction and delegates long-horizon rule evaluation to deterministic automata. In the benchmark, this design achieves higher overall F1 with lower token cost than direct full-context reasoning baselines. These results suggest that combining approximate semantic extraction with formal stream semantics provides a practical foundation for auditable and scalable analytics over unstructured event streams.

# REFERENCES


[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Liane Ahmad, Ilge Akkaya, Felipe L. Aleman, Daniel Almeida, Johannes Altenschmidt, Sam Altman, Shantanu Anadkat, and Rafael Avila. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. 2008. Efficient Pattern Matching over Event Streams. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. 147–160.

[3] Tyler Akidau, Edmon Begoli, Slava Chernyak, Fabian Hueske, Kathryn Knight, Kenneth Knowles, Daniel Mills, and Dan Sotolongo. 2021. Watermarks in Stream Processing Systems: Semantics and Comparative Analysis of Apache Flink and Google Cloud Dataflow. *Proceedings of the VLDB Endowment* 14, 12 (2021), 3135–3147. https://doi.org/10.14778/3476311.3476389

[4] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam Whittle. 2015. The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1792–1803. https://doi.org/10.14778/2824032.2824076

[5] Elias Alevizos, Anastasios Skarlatidis, Alexander Artikis, and Georgios Paliouras. 2017. Probabilistic Complex Event Recognition: A Survey. *ACM Comput. Surv.* 50, 5, Article 71 (Sept. 2017), 31 pages. https://doi.org/10.1145/3117809

[6] Apache Flink. 2024. FlinkCEP — Complex Event Processing. https://nightlies.apache.org/flink/flink-docs-release-1.20/docs/libs/cep/. Accessed: 2024.

[7] Tom B. Brown et al. 2020. Language Models are Few-Shot Learners. In *NeurIPS*, Vol. 33. 1877–1901.

[8] Marco Bucchi, Alejandro Grez, Andrés Quintana, Cristian Riveros, and Stijn Vansummeren. 2022. CORE: A Complex Event Recognition Engine. *PVLDB* 15, 9 (2022), 1951–1964.

[9] Shu Chen, Deepti Raghavan, and Uğur Çetintemel. 2025. Continuous Prompts: LLM-Augmented Pipeline Processing over Unstructured Streams. *arXiv preprint arXiv:2512.03389* (2025). https://arxiv.org/abs/2512.03389

[10] John Dagdelen et al. 2024. Structured Information Extraction from Scientific Text with Large Language Models. *Nature Communications* 15 (2024), 1418.

[11] EsperTech. 2023. Esper Reference Documentation — Event Pattern Operators. http://esper.espertech.com/release-9.0.0/reference-esper/html/event_patterns.html. Accessed: 2024.

[12] Alejandro Grez, Cristian Riveros, and Martín Ugarte. 2019. A Formal Framework for Complex Event Processing. In *ICDT*. 5:1–5:18.

[13] Alistair Johnson, Tom Pollard, Steven Horng, Leo Anthony Celi, and Roger Mark. 2023. MIMIC-IV-Note: Deidentified free-text clinical notes. *PhysioNet* (Jan. 2023). https://doi.org/10.13026/1n74-ne17 Version 2.2.

[14] Alistair E. W. Johnson, Lucas Bulgarelli, Lu Shen, Alvin Gayles, Ayad Shammout, Steven Horng, Tom J. Pollard, Sicheng Hao, Benjamin Moody, Brian Gow, Li Wei H. Lehman, Leo A. Celi, and Roger G. Mark. 2023. MIMIC IV, a Freely Accessible Electronic Health Record Dataset. *Scientific Data* 10, 1 (2023).

[15] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. 6769–6781. https://doi.org/10.18653/v1/2020.emnlp-main.550

[16] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, Vol. 33. https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[17] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Rana Shahout, and Gerardo Vitagliano. 2025. Palimpzest: Optimizing AI-Powered Analytics with Declarative Query Processing. In *Proceedings of the 15th Conference on Innovative Data Systems Research (CIDR)*.

[18] Duo Lu, Siming Feng, Jonathan Zhou, Franco Solleza, Malte Schwarzkopf, and Uğur Çetintemel. 2025. VectraFlow: Integrating Vectors into Stream Processing. In *Proceedings of the 15th Annual Conference on Innovative Data Systems Research*.

[19] Aman Madaan et al. 2023. Self-Refine: Iterative Refinement with Self-Feedback. In *NeurIPS*, Vol. 36.

[20] OpenAI. 2024. text-embedding-3-small. https://platform.openai.com/docs/models/text-embedding-3-small. Accessed: 2026-05-06.

[21] Liana Patel, Siddharth Jha, Melissa Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. 2025. Semantic Operators and Their Optimization: Enabling LLM-Based Data Processing with Accuracy Guarantees in LOTUS. *Proc. VLDB Endow.* 18, 11 (July 2025), 4171–4184. https://doi.org/10.14778/3749646.3749685

[22] Qwen Team. 2025. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388* (2025).

[23] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G. Parameswaran, and Eugene Wu. 2025. DocETL: Agentic query rewriting and evaluation for complex document processing. *Proceedings of the VLDB Endowment* 18, 9 (2025). https://doi.org/10.14778/3746405.3746426

[24] Eugene Wu, Yanlei Diao, and Shariq Rizvi. 2006. High-Performance Complex Event Processing over Streams. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*. 407–418.

[25] Tao Yu et al. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *EMNLP*. 3911–3921.

[26] Shenglai Zeng, Jiankun Zhang, Pengfei He, Yiding Liu, Yue Xing, Han Xu, Jie Ren, Yi Chang, Shuaiqiang Wang, Dawei Yin, and Jiliang Tang. 2024. The Good and The Bad: Exploring Privacy Issues in Retrieval-Augmented Generation (RAG). In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics, Bangkok, Thailand, 4505–4524. https://doi.org/10.18653/v1/2024.findings-acl.267

[27] Zhihan Zhang, Yixin Cao, Chenchen Ye, Yunshan Ma, Lizi Liao, and Tat Seng Chua. 2024. Analyzing Temporal Complex Events with Large Language Models? A Benchmark towards Temporal, Long Context Understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics Volume 1 Long Papers*. Association for Computational Linguistics, Bangkok, Thailand.