Event-Sourced Memory for Multi-Agent Systems: An Auditable, Replayable Substrate with a Deterministic Context Policy

Anonymous authors
Paper under double-blind review

Abstract

Large Language Model (LLM)–based multi-agent systems increasingly coordinate many agents over long horizons, yet most still share memory through a mutable workspace—a blackboard, buffer, or retrieval store—that scales poorly with interaction length and discards the provenance of each decision as state is overwritten. We introduce **Event-Sourced Memory (ESM)**, which represents multi-agent memory as an append-only log of immutable, typed events. Each agent reconstructs its working context by a deterministic projection over the log, and a deterministic, training-free **Context Policy** selects relevant events under a bounded token budget using semantic, temporal, and role-dependency signals. Because the log is never overwritten and projections are pure functions of it, ESM provides two properties by construction that mutable or summarized memories cannot: exactly reproducible context reconstruction, and full causal provenance for every decision. Its cost profile is likewise decoupled from history length—$O(1)$ appends and $O(\log T + k)$ indexed retrieval, against the $O(T)$ of replay-based memory—which we establish analytically. A multi-agent system built on ESM attains 72.1% on Terminal-Bench 2.0, an externally verifiable leaderboard result, and ablations isolate the contributions of immutability, projection, and selective retrieval. ESM uses no learned components and no third-party runtime dependencies, and is positioned as a deterministic, auditable substrate complementary to long-context models and learned-memory systems.

1 Introduction

Large Language Model (LLM)–based multi-agent systems have become powerful tools for reasoning, planning, and collaboration on open-ended tasks. But as these systems run longer and coordinate more agents, a central question remains: how can agents maintain memory that is coherent, efficient, and *accountable* over long horizons? Most architectures answer with a mutable shared state—a blackboard, buffer, or retrieval store that agents read and overwrite. This is simple, but it scales poorly with interaction length and, more subtly, it discards provenance: once state is overwritten, the causal record of why an agent acted is lost. Recent analyses of multi-agent failure point directly at this. Lost or reset history is among the most common documented failure modes (Cemri et al., 2025), and the unchecked propagation of errors across agents is a leading cause of compounding failure (Kim et al., 2025).

This paper introduces **Event-Sourced Memory (ESM)**, which redefines multi-agent memory as an append-only log of immutable, typed events rather than a mutable workspace. Each event records one atomic agent–environment interaction; each agent reconstructs its working context by applying a deterministic *projection* over the log; and a deterministic *Context Policy* selects which events enter the prompt under a token budget. Because the log is never overwritten and projections are pure functions of it, two properties hold *by construction*: context reconstruction is exactly reproducible, and any decision can be traced to the precise events that produced it. And because appends are $O(1)$ and retrieval is $O(\log T + k)$ over an incrementally indexed log, per-step memory cost is decoupled from history length.

Event sourcing itself is not new—it is standard in distributed systems (Fowler, 2005) and has recently been applied to individual LLM agents (Wang et al., 2025a; dos Santos Filho, 2026). Our contribution is what sits on top of the log in a multi-agent setting, and what the resulting substrate guarantees. Specifically, this paper:

1. **Formalizes Event-Sourced Memory** for multi-agent systems—a typed immutable event log, projection-based context reconstruction, and a causal-provenance DAG—and establishes deterministic reconstruction (Proposition 1) as the basis for replay and provenance (Section 4).
2. **Introduces a deterministic, training-free Context Policy** that scores events by semantic, temporal, and role-dependency signals and admits an indexed $O(\log T + k)$ realization, with no learned components and no third-party runtime dependencies (Section 6).
3. **Characterizes the substrate analytically**—an $O(1 + \log T + k)$ per-step cost against the $O(T)$ of blackboard and replay memory—and identifies reproducible reconstruction and provenance as by-construction guarantees that mutable or summarized memories cannot provide (Appendix A).
4. **Reports an empirical study**: a multi-agent system built on ESM attains 72.1% on Terminal-Bench 2.0, an externally verifiable result on the public leaderboard, and component ablations isolate the contributions of immutability, projection, and selective retrieval (Section 7).

By decoupling reasoning from global state mutation, ESM offers an auditable, replayable, and computationally efficient memory substrate for multi-agent LLM systems—one that complements long-context models and learned-memory approaches rather than competing with them.

2 Background and Motivation

Mutable shared state in multi-agent systems. Early LLM multi-agent frameworks store agent outputs in a mutable shared workspace—a blackboard, buffer, or retrieval pipeline—that each agent reads and overwrites. This is simple but has two structural drawbacks: it scales linearly or worse with interaction length, since each step re-reads or re-expands shared context; and it erases provenance, since states are overwritten rather than retained. The consequence is coordination that is both expensive and opaque, making it difficult to audit why an agent acted as it did or to recover the causal dependencies between actions.

These drawbacks have measurable consequences. Recent empirical studies make the cost concrete. A failure-mode analysis spanning many multi-agent frameworks (Cemri et al., 2025) identifies, among its most common failures, the loss or reset of conversation history—precisely the hazard a mutable, overwrite-based store invites. A controlled scaling study (Kim et al., 2025) finds that, absent validation checkpoints, errors propagate and amplify across agents far more than under centralized, checked coordination, implicating the absence of a reliable shared record as a driver of compounding error. And industry reports on multi-agent systems observe that token usage—dominated by repeated context—explains most of the variance in both cost and performance (Anthropic, 2025). Together these motivate a substrate that retains history losslessly, makes provenance explicit, and bounds per-step context.

Event sourcing in distributed systems. Outside the LLM domain, event sourcing represents application state as an append-only log of immutable events, from which any state is derived by replay. Apache Kafka, CQRS, and event-sourced databases use this to obtain auditability, fault tolerance, and fine-grained provenance (Fowler, 2005). These guarantees are precisely what mutable multi-agent memory lacks.

Motivation for Event-Sourced Memory. ESM brings this discipline to multi-agent LLM memory. Treating each interaction as an immutable, typed event and reconstructing context by projection turns memory management into event coordination: rather than reloading or overwriting shared state, agents retrieve causally and semantically relevant events on demand from a record that is never lost. This yields

three properties developed in the rest of the paper—scalability (constant-time appends, sublinear retrieval), persistence (lossless, replayable history), and interpretability (every output traceable to its events)—and grounds the by-construction guarantees of Section 7.4.

3 Related Work

We organize related work into four threads—multi-agent memory and coordination, single-agent long-term memory, event sourcing for agents, and learned context management—and state in each how ESM differs.

3.1 Multi-Agent Memory and Coordination

Frameworks for coordinating multiple LLM agents—role-playing dialogue (CAMEL (Li et al., 2023)), hierarchical planning (MetaGPT (Hong et al., 2024)), tool delegation (HuggingGPT (Shen et al., 2023)), debate (Du et al., 2023), and emergent-collaboration frameworks (Chen et al., 2023))—improve capability through collaboration but typically share state through conversational history or a mutable workspace, leaving long-term scalability and provenance unaddressed; benchmarks such as AgentBench (Liu et al., 2023) confirm that cooperation helps but is hard to measure consistently. Recent work targets multi-agent memory directly. G-Memory (Zhang et al., 2025a) organizes collaboration history as a three-tier graph (insight, query, interaction) and reports success-rate gains across several multi-agent frameworks without modifying them; collaborative-memory systems (Rezazadeh et al., 2025) add per-fragment access control and provenance tracking; and blackboard multi-agent designs (Yoon et al., 2025) report improvements over RAG (Lewis et al., 2020) and master–slave baselines by centralizing shared state. ESM differs from all of these in its storage discipline: rather than a mutable workspace or a graph updated in place, the shared substrate is an append-only log, and each agent's view is a deterministic projection of it (Proposition 1). This is what gives ESM exact replay and provenance—properties a mutating store cannot offer by construction.

3.2 Single-Agent Long-Term Memory

A large body of work extends the effective memory of single agents. Scratchpad and self-reflection loops (ReAct (Yao et al., 2023), Reflexion (Shinn et al., 2023)) provide short-term working memory; memory-augmented architectures (LongMem (Wang et al., 2023)) and OS-inspired paging (MemGPT/Letta (Packer et al., 2023)) extend context beyond the window; and the memory stream of Generative Agents (Park et al., 2023) retrieves by recency, importance, and relevance. More recent systems restructure or consolidate memory: A-MEM (Xu et al., 2025) maintains dynamically linked notes, Mem0 (Chhikara et al., 2025) extracts and consolidates salient facts (with a graph variant), MemTree (Rezazadeh et al., 2024) builds a hierarchical summary tree, and GraphRAG (Han et al., 2024) indexes a knowledge graph for multi-hop retrieval; parametric approaches (MemoryLLM and M+ (Wang et al., 2025b)) fold memory into model state. The closest of these to ESM on the provenance dimension is Zep/Graphiti (Rasmussen et al., 2025), a bi-temporal knowledge graph that tracks validity intervals and episode provenance. ESM's distinction is that these systems mutate or summarize their stores—Zep invalidates superseded facts, Mem0 and A-MEM rewrite notes, MemGPT evicts under model control—whereas ESM retains every event and reconstructs any past state by replay. ESM therefore does not claim to introduce provenance; it claims a stronger, by-construction form of it: lossless retention plus deterministic reconstruction, rather than provenance over a mutating structure.

3.3 Event Sourcing for Agents

Event sourcing—deriving state from an append-only log of immutable events—is a long-established pattern in distributed systems (Fowler, 2005) and has recently been applied to LLM agents. The OpenHands Software Agent SDK (Wang et al., 2025a) treats interactions as immutable events appended to a log, and event-sourcing architectures for autonomous agents (dos Santos Filho, 2026) add deterministic projection and replay verification. We therefore do not claim event sourcing for agents as novel. ESM's contribution is specific and downstream of the storage pattern: a deterministic, training-free multi-agent *Context Policy* over the log, projection-based working-context reconstruction with an analyzed $O(1 + \log T + k)$ cost profile,

and an account of how the substrate's by-construction properties bear on multi-agent coordination. We view ESM as complementary to these systems rather than competing with the storage abstraction they share.

3.4 Learned Context Management

A parallel and fast-growing line learns to manage context: an external policy, often trained by reinforcement learning, decides what a frozen agent keeps in context (Zhang et al., 2025b). Such methods can adapt to a task distribution, but the learned policy is opaque and renders a run's context non-reproducible. ESM takes the opposite stance by design: its Context Policy is deterministic and training-free, which preserves the exact reproducibility of Proposition 1 and keeps the system free of learned components and third-party runtime dependencies. We regard the deterministic and learned regimes as a genuine design axis, and position ESM firmly on the deterministic, auditable end of it.

Benchmarks and surveys. Recent surveys organize this space by the form and function of memory (Hu et al., 2025b) and by the related problem of context engineering (Mei et al., 2025). Evaluation has consolidated around long-horizon benchmarks—LoCoMo (Maharana et al., 2024) (multi-session QA), LongMemEval (Wu et al., 2024) (six memory abilities over very long histories), MemoryAgentBench (Hu et al., 2025a) (retrieval, test-time learning, long-range understanding, and selective forgetting, with no system excelling at all four), and BEAM (BEAM, 2025) (million-token contexts). These principally measure recall-style accuracy; auditability, deterministic replay, and behavior under concurrent multi-agent writes—the dimensions on which ESM is differentiated—are not their focus.

4 Event-Sourced Memory: Formalization

ESM represents a multi-agent interaction history as a temporally ordered, immutable sequence of atomic events. This section gives the formal structure of that representation and states the properties the rest of the paper builds on: deterministic reconstruction, incremental projection, causal traceability, and bounded retrieval cost.

4.1 Event Representation

Definition 1 (Event log)**.** *The event log $\mathcal{E} = (e_t)_{t=1}^{T}$ is the totally ordered sequence of all events emitted up to time T. Each event is a typed record of one atomic agent–environment interaction,*

$$e_t = (t,\, A_i,\, s_t,\, a_t,\, r_t,\, m_t) \in \mathcal{T} \times \mathcal{I} \times \mathcal{S} \times \mathcal{U} \times \mathcal{R} \times \mathcal{M}, \tag{1}$$

where t is the event index (timestamp), $A_i \in \mathcal{I}$ the originating agent, $s_t \in \mathcal{S}$ an observation or internal-state summary, $a_t \in \mathcal{U}$ the action taken, $r_t \in \mathcal{R}$ an optional reward or feedback signal, and $m_t \in \mathcal{M}$ metadata (agent role, dependency tags, and similar annotations).

The typed product space enforces schema-level consistency across heterogeneous agents. The log is *append-only* and totally ordered by t: writing a new event extends the log without modifying any existing entry,

$$\mathcal{E}_T = \mathcal{E}_{T-1} \cup \{e_T\}, \qquad \text{Append}(e_T) = O(1). \tag{2}$$

Every interaction is therefore recorded exactly once and never altered retroactively; agents never overwrite shared state, as in a blackboard, but only append. One consequence is used repeatedly in what follows.

Proposition 1 (Deterministic reconstruction)**.** *Any function of the log that does not itself invoke a stochastic model is uniquely determined by the log's contents. In particular, the per-agent reconstruction $P_i(\mathcal{E}_{\leq t})$ defined in Section 4.2 is a pure function of $\mathcal{E}_{\leq t}$: two executions producing identical logs yield identical reconstructed contexts.*

The proposition is immediate—P_i is a deterministic fold over an immutable input—but load-bearing: it is what lets us describe replay and provenance as holding *by construction* rather than by measurement (Section 7.4). It does not extend to the agents' model outputs, which may be stochastic.

4.2 State Reconstruction via Projections

Rather than maintaining a monolithic global state, each agent derives its working view by applying a *projection* over the log. Let $P_i : \mathcal{E} \to S_i$ be the projection for agent A_i, a fold that selects the events within A_i's scope and reduces them to a state:

$$P_i(\mathcal{E}_{\leq t}) = f_{\text{reduce}}\big(\{e_j \in \mathcal{E}_{\leq t} : A(e_j) \in \text{scope}(i)\}\big), \tag{3}$$

where $A(e_j)$ is the originating agent of e_j, scope(i) the set of agents visible to A_i, and f_{reduce} an application-defined accumulation (for example, concatenation of relevant contents, or maintenance of a running task ledger). Because the log only grows, projections update incrementally: advancing A_i's state from $t-1$ to t requires folding in only the new event e_t when it lies in scope, at cost $O(\Delta_t)$, where Δ_t is the number of new in-scope events (typically $\Delta_t = 1$), rather than the $O(T)$ of recomputing from scratch. Incremental projection is what lets each agent hold an up-to-date view without copying or replaying full state each turn.

4.3 Causal Dependencies and Temporal Locality

Events may record their causal antecedents. Let cause$(e_t) \subseteq \mathcal{E}_{\leq t-1}$ be the events that e_t directly depends on—e.g., the message that prompted A_i to take action a_t. Writing $e_j \to e_t$ iff $e_j \in$ cause(e_t) induces a directed acyclic graph $(\mathcal{E}, \to)$, a partial order in which $e_j \to e_t$ means e_j is upstream context for e_t. The transitive closure of $\to$ from any event is its *provenance*: the exact set of prior events on which it depends.

This structure yields *temporal locality*. The causally relevant subset of an event, $\mathcal{E}_t^{\text{cause}} = \{e_j : e_j \to e_t\}$ (optionally extended to multi-hop ancestors), is typically far smaller than the full history: when interactions are localized in time or topic, $|\mathcal{E}_t^{\text{cause}}| \ll T$. Agents can therefore reason over causally linked events rather than the entire log, shrinking retrieval and processing without discarding critical context, and any output can be explained by walking its provenance.

4.4 Semantic Indexing

To support retrieval by content as well as by causal links, each event's content s_t (and optionally its metadata m_t) is embedded as $v_t = f_{\text{emb}}(s_t) \in \mathbb{R}^d$. Given a token budget k and a query q from agent A_i, a *Context Policy* π selects the assembled context

$$C(q, A_i) = \text{Select}_k\big(\text{score}(e_t, q, A_i)\big), \tag{4}$$

where Select$_k$ returns the highest-scoring events whose serialized length fits within k, and score$(\cdot)$ is the relevance function detailed in Section 6. Embeddings are computed once per event and cached; maintaining an Approximate Nearest-Neighbor (ANN) index incrementally as events arrive makes candidate generation $O(\log T)$, with $O(k)$ to re-rank and serialize the selected context. Combined with the $O(1)$ append, the per-step cost of writing an event and assembling its context is $O(1 + \log T + k)$, against the $O(T)$ of architectures that scan or copy shared state each step. Appendix A gives the full derivation.

4.5 Properties of the Substrate

Four properties follow from the construction above. *Immutability and consistency:* because each event is written once and never modified, concurrent agents cannot race on shared state, and the log is a deterministic, replayable record; any historical state is recovered by replaying the relevant prefix. *Incremental reconstruction:* extending the horizon or adding agents triggers no recomputation over past events—each agent folds in only new in-scope events, so cost scales with the rate of new events and the number of agents, not their product over the full history. *Causal traceability:* the provenance of any event is explicit in the causal DAG, so a decision can be explained by the precise chain of prompts, observations, and results that produced it—an audit trail latent or overwrite-based memories cannot provide. *Embedding reuse:* since events are immutable, their embeddings are computed once and shared across agents and queries, amortizing index maintenance. Together these give a per-step cost near-constant in history length (up to the logarithmic lookup and the linear context size k) while preserving reproducibility and provenance—the basis

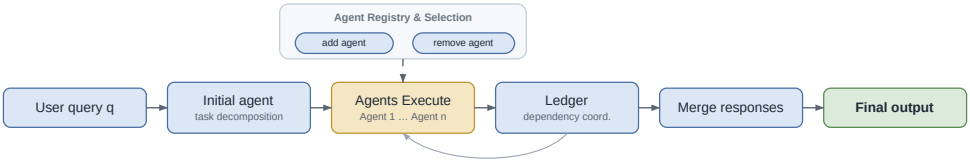


Figure 1: Multi-Agent Network (MAN) architecture. A user query is decomposed by an initial agent into discrete tasks distributed across a dynamic set of agents managed by a registry; a ledger coordinates dependent responses before all agent outputs are merged into the final output.

for the analytical efficiency profile and by-construction properties of Section 7; the detailed asymptotics, including the multi-agent comparison against blackboard and file-based memory, are derived in Appendix A.

5 Architecture

The ESM system is organized into three layers—an Agent Layer, a Memory Fabric Layer, and a Storage Layer (Figure 2a)—that separate reasoning from memory orchestration from persistence. The Multi-Agent Network (MAN) workflow (Figure 1) shows the end-to-end path: an initial agent decomposes a user query into discrete tasks; a registry selects and, as needed, adds or removes specialized agents per query; agents execute independently or in dependency order; and a ledger coordinates dependent responses before they are merged into the final output. This composition supports parallel execution and modular extension without changes to the underlying memory substrate.

5.1 Layered Design

1. **Agent Layer:** Multiple LLM-based agents (e.g., Planner, Solver, Critic, Verifier), each an autonomous reasoning entity that emits structured events representing its outputs—proposals, solutions, critiques, verifications.
2. **Memory Fabric Layer:** The **Memory Orchestrator** mediates query–response traffic between agents and memory; the **Context Policy** selects relevant events from the log under the token budget k when an agent issues a query; and the **Event Indexer** maintains the auxiliary structures (embedding and keyword indices) that make selection efficient.
3. **Storage Layer:** The persistent, append-only *Event Log*, realizable as an append-only database or as indexed plain-text logs.

5.2 Data Flow

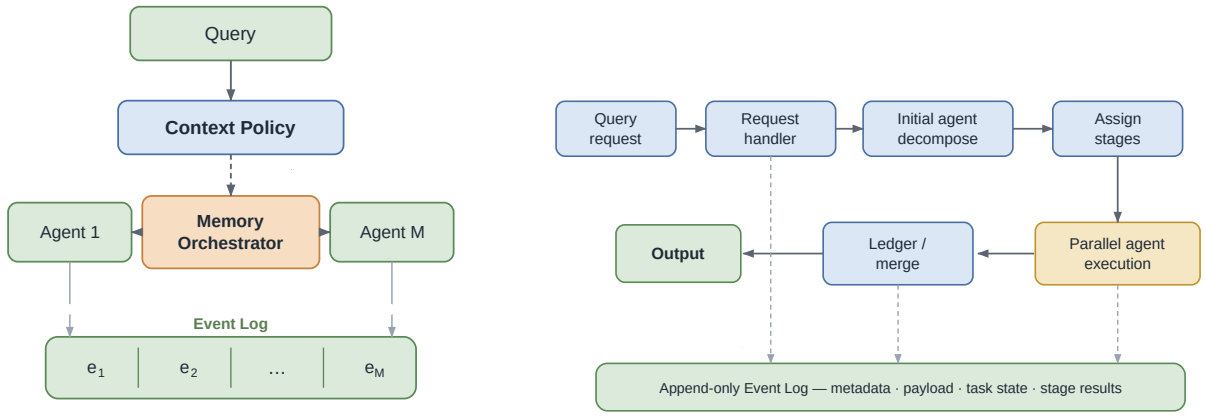
The layers form a closed loop. Agents append new events downward into the Storage Layer. When an agent needs context—to answer a query or plan its next step—it issues a request to the Memory Fabric; the Context Policy consults the Event Indexer and returns the highest-scoring events under budget k as an assembled context for grounded reasoning. Compactly,

$$\text{Agents} \longrightarrow \text{Event Log (writes)} \longrightarrow \text{Memory Fabric} \longrightarrow \text{Agents (reads)}.$$

For transport into a prompt, each event is serialized as $\langle \text{actor}, \text{role}, \text{content}, \text{timestamp} \rangle$—a projection of the typed record of Eq. (1) onto the fields the prompt requires. Figure 2b traces this flow in detail, from request handling through staged task decomposition and ledger-based merging.

5.3 Coordination and the Role of Immutability

Because the shared substrate is an append-only log rather than a mutable workspace, several coordination hazards common to blackboard architectures are removed by construction: concurrent agents cannot over-



(a) **System architecture of ESM.** The *Query Interface* interacts with the **Context Policy**, which coordinates with the **Memory Orchestrator** managing multiple LLM agents (*Agent 1 ... Agent M*). Each agent writes structured `<actor, role, content, timestamp>` events to the shared **Event Log**; the Context Policy retrieves relevant events upward under budget k, closing the loop across the Agent, Memory Fabric, and Storage layers.

(b) **Event-log execution and task-decomposition workflow.** A request is received by the handler, which stores metadata, payload, and file-analysis results separately. The initial agent decomposes incoming tasks across stages; each stage's output is managed and combined in a ledger before the final response. Arrows show the flow of events between agents, task storages, and merging modules, highlighting parallel, stage-wise orchestration.

Figure 2: Overall and detailed architecture of the Event-Sourced Memory (ESM) system.

write one another's state, history cannot be silently lost or reset, and every agent's view is reconstructed deterministically from the same log (Proposition 1). Against this backdrop the ledger and Verifier act as explicit coordination and checking points—dependent tasks wait on the specific events they require, and verification reads the same immutable record that produced a result rather than a possibly-overwritten summary of it. We return in Section 9 to how shared-memory design interacts with documented multi-agent failure modes.

5.4 Scalability and Modularity

Separating *what is stored* (immutable events) from *how it is used* (Context Policy, Memory Orchestrator) yields three practical benefits: the orchestrator can serve concurrent queries from many agents; the storage layer can be sharded or distributed, since append-only logs parallelize naturally; and agents, memory policies, and storage backends can evolve independently behind stable interfaces.

6 Context Policy

The Context Policy (CP) pairs each agent query with a relevant subset of the event log under a token budget. Because the substrate separates storage from retrieval (Section 5), the CP is a pluggable component: any selection rule operates over the same immutable log without touching storage or projection. We deliberately adopt a transparent, deterministic policy and leave richer selection rules to future work—a choice that keeps the whole pipeline training-free and, by Proposition 1, exactly reproducible, in contrast to recent approaches that learn an external context-management policy and thereby give up replayability.

Algorithm 1 Heuristic Context Selection

Require: Query q, agent A_i, event log $\mathcal{E}$, token budget k, weights $\lambda_1, \lambda_2, \lambda_3$, decay α

```
S ← []
for each event e_t ∈ E do
    r_1 ← cos(f_emb(s_t), f_emb(q))          ▷ semantic similarity
    r_2 ← exp(−α(T − t))                     ▷ temporal recency
    r_3 ← role_w(ρ(m_t), A_i)                ▷ role/dependency weight
    S ← S ∪ {(e_t, λ_1 r_1 + λ_2 r_2 + λ_3 r_3)}
end for
C ← Select(S, k)                             ▷ highest-scoring events fitting budget k
return serialize(C)
```

6.1 Heuristic Context Policy

Our policy scores each event by a weighted combination of three signals—semantic similarity, temporal recency, and role dependency—in the lineage of the memory-stream retrieval of Generative Agents (Park et al., 2023), which combines relevance, recency, and an LLM-assigned importance term. We make one deliberate substitution: in place of a model-assigned importance score, we use a *deterministic role/dependency weight* derived from event metadata. This retains a structural notion of which events matter to which agents while keeping the policy free of any learned component. For a query q from agent A_i, each event e_t receives:

- **Semantic similarity** $r_1 = \cos\big(f_{\text{emb}}(s_t),\ f_{\text{emb}}(q)\big)$ — topical match between the event's content and the query.
- **Temporal recency** $r_2 = \exp(-\alpha(T - t))$ — exponential decay favoring more recent events.
- **Role relevance** $r_3 = \text{role_w}\big(\rho(m_t),\ A_i\big)$ — a lookup over a role/dependency matrix, where $\rho(m_t)$ is the producing role recorded in the event metadata m_t and A_i is the querying agent. For example, a Planner's query boosts Solver-produced events, and metadata tags such as "error" or "follow-up" raise priority.

The overall score is $\text{score}(e_t, q, A_i) = \lambda_1 r_1 + \lambda_2 r_2 + \lambda_3 r_3$ with fixed weights λ_i. The policy returns the highest-scoring events whose serialized length fits the token budget k (Algorithm 1).

6.2 Indexed Realization

As written, Algorithm 1 scans the full log, costing $O(T)$ per query. The identical selection is realized sublinearly using the semantic index: retrieve the top-m candidates by r_1 from the ANN index in $O(\log T)$, then compute r_2 and r_3 and the combined score over only those m candidates and select within budget, in $O(m)$. With $m = O(k)$ this gives the $O(\log T + k)$ retrieval cost used in Section 7 and Appendix A. Recency and role weights need no index of their own, since each is computed from a candidate's timestamp and metadata at scoring time.

6.3 Discussion

The heuristic is interpretable and training-free: it approximates a well-informed analyst—prefer topically relevant and recent events, and ensure role-critical inputs are present—while every selection remains a deterministic function of the log and the fixed weights. This determinism is not incidental. It is what makes a run's assembled contexts reproducible and auditable, and it is precisely the property a learned policy would trade away. The scoring rule can be tuned or replaced without altering the substrate, which is why we treat the policy as a component rather than as the contribution of the paper.

Table 1: Performance across code generation benchmarks. Confidence intervals are 95% normal-approximation binomial intervals over the n tasks.

Metric	**HumanEval**	**APPS-300**
n	164	300
Matched	130	161
Unmatched	34	139
Accuracy ($\hat{p}$)	0.793	0.537
95% CI	± 0.062	± 0.056

7 Results

We evaluate Event-Sourced Memory (ESM) along two distinct axes. The first is *capability*: the task performance of a multi-agent system built on the ESM substrate, measured on one agentic and two code-generation benchmarks. The second is *efficiency*: the asymptotic cost profile of the substrate's memory operations, which we characterize analytically. We are deliberate about this separation, and in particular about distinguishing measured results from properties that hold by construction. Throughout, the Context Policy is the transparent heuristic of Section 6; no component is learned, and the system relies on no third-party runtime dependencies.

7.1 Agentic Reasoning: Terminal-Bench 2.0

Our primary capability evaluation is on Terminal-Bench 2.0, a benchmark of 89 tasks that exercise multi-step tool use, environment interaction, and long-horizon consistency in a sandboxed terminal. The ESM-based system, submitted under the identifier **MAYA-V2 (ADYA)**, attains an overall accuracy of $\mathbf{72.1\% \pm 2.2}$. This entry is listed on the public Terminal-Bench 2.0 leaderboard, so the result is independently and externally verifiable rather than self-reported; the reported interval reflects the benchmark's standard multi-attempt evaluation protocol.

We attribute this performance to ESM's avoidance of redundant context replay. Rather than concatenating the full interaction history at each step—the dominant cost in prompt-based agent loops such as ReAct—ESM reconstructs a compact working context through selective event projection, sustaining coherence across long task horizons within a bounded token budget.

7.2 Code Generation: HumanEval and APPS-300

As secondary capability evidence, we evaluate on two program-synthesis benchmarks, HumanEval (Chen et al., 2021) and APPS-300 (Hendrycks et al., 2021) (Table 1). On HumanEval the system solves 130 of 164 tasks (79.3%); on the more demanding APPS-300, which involves longer solution chains and deeper dependency structure, it solves 161 of 300 (53.7%). We report 95% normal-approximation binomial confidence intervals computed over the respective task counts, so that the point estimates are read against their sampling uncertainty.

The gap between the two benchmarks is consistent with the greater length and dependency depth of APPS tasks. Performance remains stable as interaction horizons grow, consistent with the bounded per-step context that projection-based reconstruction provides.

7.3 Efficiency: Analytical Cost Profile

A central design claim of ESM is that the cost of memory operations is decoupled from history length. Appendix A derives this formally: appends are $O(1)$, semantic retrieval is $O(\log T + k)$ under an ANN index over cached event embeddings, and state reconstruction is incremental rather than $O(T)$. The resulting per-step memory cost is $O(1 + \log T + k)$, in contrast to the $O(T)$ profile of blackboard or full-replay architectures that re-read or rewrite shared state at every step (Table 3).

We present these as *analytical* results and are careful not to overstate them: they follow directly from the data structures—an append-only log, cached embeddings, and an incrementally maintained index—and bound the substrate's asymptotic cost independently of any model or workload, but they are not a substitute for measured wall-clock behavior. A controlled study of constants and crossover points—token consumption, retrieval latency, and throughput as functions of T and n, measured head-to-head against full-context replay, sliding-window, and retrieval baselines—is ongoing. We flag this explicitly because asserting unmeasured efficiency would misrepresent the evidence: what we claim is a proven asymptotic profile, not an empirical latency result.

7.4 Properties That Hold by Construction

Two further properties of the substrate are consequences of its design rather than empirical findings, and we state them as such. First, because the log is append-only and each projection P_i is a deterministic function of the log, the working context reconstructed for any agent at any time t is uniquely determined by the events up to t: given the same log, context reconstruction is exactly reproducible. Second, because every event records its originating agent and causal metadata, any reconstructed context—and therefore any decision conditioned on it—can be traced to the precise set of events that produced it. These follow from immutability and deterministic projection; they are not claims about end-to-end agent determinism, which additionally depends on the decoding behavior of the underlying model. A quantitative protocol for replay fidelity and provenance completeness is a direction we return to in Section 10.

7.5 Discussion

Taken together, the results indicate that the ESM substrate delivers competitive agentic and code-generation performance while keeping memory cost asymptotically flat in history length and preserving exact reproducibility and provenance of reconstructed context. Unlike retrieval-only systems such as RAG or memory-stream approaches, which reassemble context heuristically and retain no provenance, ESM unifies persistence, bounded-context retrieval, and causal traceability in a single abstraction. The capability results establish that this structure does not come at the cost of task performance; the analytical profile and the by-construction properties establish what it adds.

8 Ablation Study

To isolate the contribution of ESM's components, we evaluate controlled variants that selectively disable individual design elements while holding the backbone model, the token budget, and the task set fixed. All variants are evaluated on Terminal-Bench 2.0, HumanEval, and APPS-300.

8.1 Variants

- **Full ESM:** The complete formulation—append-only event log, projection-based context reconstruction, and the heuristic Context Policy for selective retrieval.
- **w/o Context Policy:** Retrieval is reduced to a recency-only heuristic, removing semantic and role-aware scoring.
- **w/o Event Projection:** Context is formed by concatenating the entire event history, reverting to a prompt-replay regime.
- **w/o Semantic Indexing:** Events are accessed in temporal order only, without embedding-based similarity search.
- **Mutable Memory:** The append-only log is replaced with a single mutable state that is overwritten at each step, removing immutability and causal traceability.

8.2 Results

Table 2 reports accuracy for each variant across the three benchmarks.

Table 2: Ablation results across benchmarks (accuracy, %). Each row removes a single component from Full ESM; Mutable Memory removes immutability itself.

Variant	Terminal-Bench 2.0	HumanEval	APPS-300
Full ESM	72.1	79.3	53.7
w/o Context Policy	66.4	74.8	49.2
w/o Semantic Indexing	64.1	72.9	47.8
w/o Event Projection	61.2	70.5	45.6
Mutable Memory	58.7	68.3	43.1

8.3 Reading the Ablation

The salient feature of Table 2 is not the size of any single gap but the *consistency* of the ordering: on all three benchmarks the ranking is identical—Full ESM, then removal of the Context Policy, semantic indexing, event projection, and finally mutable memory—with every removal reducing accuracy and no inversions. We treat this consistency, rather than any single gap, as the evidence. At these task counts an individual pairwise gap is comparable to sampling noise (for a proportion near 0.7 at $n = 89$, the binomial standard error is roughly five points, so 72.1 vs. 66.4 is about one standard error), but a monotonic ordering holding simultaneously across three independent benchmarks is far less probable under noise than any single comparison. We therefore read the ablation as evidence of the *direction and rank* of each component's contribution, not as precise effect sizes; tighter estimates via multiple seeds are a straightforward camera-ready strengthening.

The per-component pattern is interpretable. Removing the **Context Policy** (leaving recency only) gives the smallest degradation, so semantic and role-aware scoring add signal beyond recency while recency alone remains a serviceable fallback. Removing **semantic indexing** costs slightly more, consistent with temporal order missing topically relevant but distant events. Removing **event projection** is the most damaging single ablation—without it the system reverts to full-history concatenation, inflating context and diluting signal—and the **mutable-memory** variant is lowest overall, reflecting the compounding cost of overwriting state. The thesis follows: performance comes not from any single component but from the interaction of immutable storage, projection-based reconstruction, and selective retrieval, with projection and immutability load-bearing and the policy a transparent, replaceable refinement.

9 Discussion

Shared-memory design and multi-agent failure modes. A recurring theme in recent analyses of multi-agent systems is that coordination failures are often memory failures. The MAST taxonomy (Cemri et al., 2025) lists the loss or reset of conversation history among its most common failure modes, and the scaling study of (Kim et al., 2025) attributes much of the error amplification in decentralized agent systems to the absence of validation checkpoints that catch mistakes before they propagate. ESM speaks to both by construction rather than by tuning. An append-only log cannot silently lose or reset history—every event persists and every agent view is a deterministic projection of the same record (Proposition 1)—so the "lost history" class of failures is removed structurally. The ledger and Verifier roles provide explicit checkpoints over that record, reading the same immutable events that produced a result rather than a possibly-overwritten summary. We stress that this is an architectural argument: we show that ESM eliminates these failure classes by design, but a controlled measurement of failure-mode incidence and error amplification against a mutable-blackboard baseline—the experiment that would turn the argument into evidence—is left to future work (Section 10).

Relationship to long-context models and learned memory. ESM is complementary to, not competitive with, two adjacent directions. As context windows grow, the problem shifts from *fitting* history into the prompt to *choosing* what belongs there; the Context Policy becomes a filter over a larger candidate space rather than a workaround for a small one, and ESM's value—deciding explicitly and reproducibly which events the model sees—persists. Against learned context managers (Zhang et al., 2025b), ESM occupies

the deterministic, auditable end of the design axis: it trades the adaptivity of a trained policy for exact reproducibility and freedom from learned components and runtime dependencies. The two are not mutually exclusive; a learned policy could in principle operate over the same immutable substrate, inheriting its replay and provenance guarantees while relaxing determinism where that trade is acceptable.

Privacy, security, and governance. A persistent event log raises real governance questions. Because events may capture sensitive information, deployments need access control so that only authorized agents query particular subsets of the log, and may require encryption or selective redaction to limit leakage. The append-only discipline that gives ESM its auditability is in tension with data-governance requirements such as the right to erasure: lossless retention and selective forgetting pull in opposite directions, and reconciling them—for instance through access-scoped views or cryptographic erasure of event payloads—is a design problem any production deployment must confront. We flag this as a genuine limitation of the approach, not an incidental detail.

10 Limitations and Future Work

Evaluation scope. The clearest limitation is the scope of our evaluation. We report capability results and an analytical efficiency profile, but not yet: matched head-to-head comparisons against named memory systems under identical backbones, budgets, and judges; measured efficiency curves (token consumption, retrieval latency, and throughput as functions of log size T and agent count n); or results on standardized long-horizon benchmarks such as LongMemEval (Wu et al., 2024) and MemoryAgentBench (Hu et al., 2025a). The headline Terminal-Bench 2.0 result is externally verifiable but, like the ablations, would be strengthened by multiple seeds and reported confidence intervals throughout. Relatedly, deterministic reconstruction and provenance are argued by construction (Proposition 1); turning them into measured results—replay fidelity under perturbation, provenance completeness, and audit-query latency—would substantiate and quantify them. The substrate is deliberately structured so that storage and Context Policy can be held fixed while baselines, benchmarks, and these protocols vary.

Operational considerations. Two practical issues arise in long-running deployments. Event granularity must be chosen with care—too fine (every token-level change) bloats the log, too coarse (whole dialogues as one event) loses detail—and adapting it to the task is open. And the log grows without bound; managing this through archival, tiered storage, or pruning, or by feeding larger slices directly to ultra-long-context models, remains future work. The tension between append-only retention and the right to erasure (Section 9) is a further constraint any production deployment must address.

Policy and modality. The Context Policy here is a deliberately simple deterministic heuristic; because selection is a pluggable component over the immutable log, richer strategies can be studied without altering the substrate. Finally, we focus on text events: event sourcing is modality-agnostic, but extending retrieval to images and tool outputs would need modality-specific indexing and similarity measures.

11 Conclusion

We presented **Event-Sourced Memory (ESM)**, which recasts multi-agent memory as an append-only log of immutable, typed events with a deterministic, training-free Context Policy. Separating persistent storage from contextual usage gives bounded-context reasoning over an unbounded history and yields two properties that mutable or summarized memories cannot provide together: exactly reproducible context reconstruction and full causal provenance (Proposition 1). The substrate's $O(1 + \log T + k)$ per-step cost is established analytically; a system built on ESM reaches an externally verifiable 72.1% on Terminal-Bench 2.0; and ablations isolate the roles of immutability, projection, and selective retrieval—all with no learned components or third-party runtime dependencies. We see ESM less as a finished system than as an auditable, replayable substrate, and hope it offers a useful foundation for more scalable, reliable, and accountable multi-agent AI.

References

Anthropic. How we built our multi-agent research system. Anthropic Engineering Blog, `https://www.anthropic.com/engineering/multi-agent-research-system`, 2025.

BEAM. Beyond a million tokens: Benchmarking and enhancing long-term memory in LLMs. *arXiv preprint arXiv:2510.27246*, 2025.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, et al. Why do multi-agent LLM systems fail? *NeurIPS Datasets and Benchmarks*, 2025. arXiv:2503.13657.

Mark Chen et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, et al. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. *arXiv preprint arXiv:2308.10848*, 2023.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Elzo Brito dos Santos Filho. ESAA: Event sourcing for autonomous agents in LLM-based software engineering. *arXiv preprint arXiv:2602.23193*, 2026.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

Martin Fowler. Event sourcing. `https://martinfowler.com/eaaDev/EventSourcing.html`, 2005.

Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A. Rossi, Subhabrata Mukherjee, Xianfeng Tang, et al. Retrieval-augmented generation with graphs (GraphRAG). *arXiv preprint arXiv:2501.00309*, 2024.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with APPS. *NeurIPS Datasets and Benchmarks*, 2021.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, et al. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024. arXiv:2308.00352.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in LLM agents via incremental multi-turn interactions (MemoryAgentBench). *arXiv preprint arXiv:2507.05257*, 2025a.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, et al. Memory in the age of AI agents. *arXiv preprint arXiv:2512.13564*, 2025b.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, et al. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008, 2023. arXiv:2303.17760.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. *arXiv preprint arXiv:2308.03688*, 2023.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents (LoCoMo). *arXiv preprint arXiv:2402.17753*, 2024.

Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, et al. A survey of context engineering for large language models. *arXiv preprint arXiv:2507.13334*, 2025.

Charles Packer, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023. arXiv:2304.03442.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

Alireza Rezazadeh, Zichao Li, Wei Wei, and Yujia Bao. From isolated conversations to hierarchical schemas: Dynamic tree memory representation for LLMs. *arXiv preprint arXiv:2410.14052*, 2024.

Alireza Rezazadeh, Zichao Li, Ange Lou, Yuying Zhao, Wei Wei, and Yujia Bao. Collaborative memory: Multi-user memory sharing in LLM agents with dynamic access control. *arXiv preprint arXiv:2505.18279*, 2025.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. HuggingGPT: Solving AI tasks with ChatGPT and its friends in hugging face. In *Advances in Neural Information Processing Systems*, 2023. arXiv:2303.17580.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36: 8634–8652, 2023. arXiv:2303.11366.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. In *Advances in Neural Information Processing Systems*, 2023. arXiv:2306.07174.

Xingyao Wang et al. The OpenHands software agent SDK: A composable and extensible foundation for production agents. *arXiv preprint arXiv:2511.03690*, 2025a.

Yu Wang, Dmitry Krotov, Yuanzhe Hu, Yifan Gao, Wangchunshu Zhou, Julian McAuley, Dan Gutfreund, Rogerio Feris, and Zexue He. M+: Extending MemoryLLM with scalable long-term memory. *arXiv preprint arXiv:2502.00592*, 2025b.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*, 2024.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic memory for LLM agents. *arXiv preprint arXiv:2502.12110*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. arXiv:2210.03629.

Jinsung Yoon, Hamid Palangi, et al. LLM-based multi-agent blackboard system for information discovery in data science. *arXiv preprint arXiv:2510.01285*, 2025.

Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. G-Memory: Tracing hierarchical memory for multi-agent systems. *arXiv preprint arXiv:2506.07398*, 2025a.

Yuxiang Zhang, Jiangming Shu, Ye Ma, Xueyuan Lin, Shangxi Wu, and Jitao Sang. Memory as action: Autonomous context curation for long-horizon agentic tasks. *arXiv preprint arXiv:2510.12635*, 2025b.

A Computational Complexity Analysis

This appendix derives the asymptotic efficiency of the Event-Sourced Memory (ESM) framework, which combines an event-driven data model with sublinear retrieval.

A.1 Event-level Operations

Each agent A_i emits an event $e_t = (t, A_i, s_t, a_t, r_t, m_t)$, as defined in Eq. (1), appended to the log $\mathcal{E}_T$. Unlike a blackboard system that rewrites shared state, ESM performs constant-time append:

$$\text{Append}(e_t) = O(1). \tag{5}$$

Global state reconstruction $\Sigma_t = R(\mathcal{E}_{\leq t})$ is deferred and composable across agents, eliminating $O(T)$ recomputation overhead.

A.2 Context Projection and Retrieval

For a query q, the projection operator

$$P(q) = \text{filter}(\mathcal{E}_T, \text{predicate}(q)) \tag{6}$$

returns only the relevant events. Using an Approximate Nearest Neighbor (ANN) index on embeddings $f_{\text{emb}}(s_t)$, the retrieval cost is

$$\text{Retrieval} = O(\log T + k), \tag{7}$$

where k is the context budget. Composing the selected events adds $O(k)$.

A.3 Causal Embedding Reuse

Since events are immutable and timestamped, the embedding of each e_t can be cached and reused. Only new events Δ_t require embedding updates:

$$\text{Embedding Update} = O(\Delta_t d), \tag{8}$$

where d is the embedding dimension. In steady-state operation $\Delta_t \ll T$, yielding near-constant amortized cost.

A.4 Overall Cost Derivation

Combining append, retrieval, and composition yields

$$\text{Cost}_{\text{ESM}} = O(1 + \log T + k). \tag{9}$$

For a blackboard replay system that scans or rewrites the full context at each step, the equivalent cost is $O(T)$. Hence, ESM improves both write and read efficiency asymptotically.

A.5 Scaling Comparison

The comparison in Table 3 highlights that the principal computational advantage of ESM does not stem solely from sublinear retrieval but from its event-driven decomposition. In conventional blackboard or file-based systems, each update involves rewriting or re-reading the entire shared state, resulting in linear or even quadratic growth in memory cost as the number of agents n and time horizon T increase. By contrast, the event-sourced model transforms memory operations into immutable, incremental append events. Each write is $O(1)$, and each read is a projected query on an indexed log rather than a full replay.

This structure provides two additional benefits:

Table 3: Asymptotic scaling of major memory operations across architectures (n = number of agents, T = history length, k = context budget, d = embedding dimension).

Operation	**Blackboard**	**File-based**	**Event-Sourced (ESM)**
Write (state update)	$O(nT)$	$O(T)$	$O(1)$ append-only
Semantic retrieval	$O(T)$	$O(T)$	$O(\log T + k)$ indexed
Embedding recompute	$O(Td)$	$O(Td)$	$O(\Delta_t d)$ amortized
Replay or flatten	$O(T)$	$O(T)$	$O(1)$ via projection
Total per-step	$O(T)$	$O(T)$	$O(1 + \log T + k)$

1. **Concurrency and isolation:** Because each event is independent and timestamped, multiple agents can emit events concurrently without locking shared state. Context reconstruction becomes a parallel read process rather than a serialized write–read cycle.

2. **Temporal composability:** Any state at time t can be reconstructed by replaying only its causal substream $\mathcal{E}_{\leq t}^{(A_i)}$, enabling bounded historical reasoning. The system thus achieves near-constant throughput regardless of log depth, with latency determined primarily by the semantic index lookup.

In practice, the scaling in Table 3 implies that once the embedding and index layers are stabilized, the dominant factor in system throughput is the event emission rate (number of agents n) rather than history length T. This makes ESM particularly well-suited to long-running enterprise deployments where interaction logs grow indefinitely but recent causal slices dominate active context.

A.6 Interpretation

The event-driven formalism ensures that:

- Writes are append-only $O(1)$ operations.
- Reads are localized projections using ANN indices, $O(\log T + k)$.
- State reconstruction is deferred, composable, and fully causal.
- Embeddings are cached and updated incrementally.

Together these yield system-level scalability: the total cost grows logarithmically with log length and linearly with the number of agents, rather than linearly with the full event history. The event-driven decomposition, not merely semantic indexing, is the dominant factor enabling near-constant throughput in multi-agent settings.