EMK: Episodic Memory Kernel

A Minimalist Storage Primitive for AI Agent Experience

Imran Siddique
Principal Group Engineering Manager
Microsoft
imran.siddique@microsoft.com

February 2026

Abstract

We present **EMK** (Episodic Memory Kernel), a lightweight, immutable storage layer for AI agent experiences. As autonomous agents become increasingly prevalent, the need for structured, queryable memory of past actions and outcomes becomes critical. EMK provides a minimalist yet powerful primitive that captures the complete agent experience cycle—*Goal* $\rightarrow$ *Action* $\rightarrow$ *Result* $\rightarrow$ *Reflection*—in an append-only ledger with $O(1)$ write complexity and $O(n)$ retrieval with optional vector similarity search. We demonstrate that EMK achieves **0.036ms** episode creation latency (27,694 ops/sec) and **652 write ops/sec** throughput while maintaining full audit trails, making it suitable for production agent systems. Unlike existing agent memory systems that conflate storage with summarization, EMK provides a clean separation of concerns, enabling higher-level memory architectures to be built on a solid foundation.

1 Introduction

1.1 Motivation

The proliferation of Large Language Model (LLM) powered autonomous agents (**???**) has created an urgent need for robust memory systems. Current approaches (**??**) often conflate storage with summarization, leading to systems that are both opinionated and heavyweight. This creates several problems:

1. **Lack of Auditability**: When memory is automatically summarized or compressed, the original experience is lost, making forensic analysis impossible.
2. **Tight Coupling**: Memory systems that include retrieval, summarization, and storage logic are difficult to adapt to new use cases.
3. **Heavy Dependencies**: Many agent frameworks require extensive dependency chains for basic memory functionality.

EMK addresses these issues by providing a "Layer 1" primitive that focuses solely on *storage and retrieval*, analogous to how POSIX provides file system primitives without dictating how files should be organized.

1.2 The Case for Episodic Memory

Cognitive science distinguishes between *episodic* and *semantic* memory (**?**). Episodic memory captures specific experiences ("I debugged this error yesterday"), while semantic memory stores general facts ("Python uses indentation for blocks"). Most agent memory systems focus exclusively on semantic memory (embeddings of general knowledge), neglecting the episodic dimension.

However, episodic memory is critical for:

- **Learning from Experience**: Agents need to recall *what happened* in specific situations to avoid repeating mistakes.
- **Temporal Reasoning**: Understanding cause-and-effect requires chronological records.
- **Explainability**: Stakeholders need to audit *why* an agent made a decision by reviewing its past experiences.

1.3 Contributions

Our contributions are:

1. **EMK Schema** (Section **??**): A minimal, immutable data structure for agent experiences with content-addressable IDs.
2. **Pluggable Storage** (Section **??**): Abstract interface with FileAdapter (zero dependencies) and ChromaDB implementations.
3. **Indexer Utilities** (Section **??**): Tag extraction and search text generation for downstream systems.
4. **Reproducible Benchmarks** (Section **??**): Controlled experiments demonstrating sub-millisecond latency and production-grade throughput.
5. **Open Source Implementation**: Publicly available at `https://github.com/imran-siddique/emk` with `pip install emk`.

1.4 Design Principles

EMK is built on four core principles:

- **Immutability**: Once written, episodes cannot be modified (forensic auditability).
- **Append-Only**: No deletions, ensuring complete history is preserved.
- **Minimal Dependencies**: Core requires only `pydantic` and `numpy`.
- **Layer Separation**: EMK *stores*; higher layers (like CaaS (**?**)) *summarize and contextualize*.

2 Related Work

2.1 Agent Memory Systems

Table **??** compares EMK with existing agent memory systems.

System	Immutable	Vector Search	Dependencies
LangChain Memory (?)	×	✓	Heavy
MemGPT (?)	×	✓	Heavy
AutoGPT Memory (?)	×	×	Medium
Mem0 (?)	×	✓	Heavy
EMK (Ours)	✓	✓	Minimal

Table 1: Comparison of agent memory systems. Only EMK provides true immutability with minimal dependencies.

2.2 Episodic Memory in Cognitive Science

Tulving's seminal work (?) distinguished episodic memory (personal experiences) from semantic memory (general knowledge). Recent research (??) has highlighted the need for episodic memory in LLM agents, particularly for long-term deployment and continual learning.

2.3 Immutable Data Structures

EMK draws inspiration from:

- **Event Sourcing** (?): Storing all state changes as events.
- **Append-Only Logs**: Kafka (?), Event Store (?).
- **Content-Addressed Storage**: Git (?), IPFS (?).

By using SHA-256 content hashing for episode IDs, EMK achieves content-addressability and natural deduplication.

3 Methodology

3.1 System Architecture

Figure ?? shows the EMK system architecture. Agents create episodes that are stored via the EMK interface. Storage backends (FileAdapter or ChromaDB) handle persistence, while higher-level systems like CaaS consume episodes for context management.

3.2 The Episode Schema

An episode captures the complete agent experience cycle using the Goal-Action-Result-Reflection (GARR) pattern:

```python
from pydantic import BaseModel, Field
from datetime import datetime
from typing import Dict, Any

class Episode(BaseModel, frozen=True):
    goal: str        # What the agent intended
    action: str      # What the agent did
    result: str      # What happened
    reflection: str  # What the agent learned
```

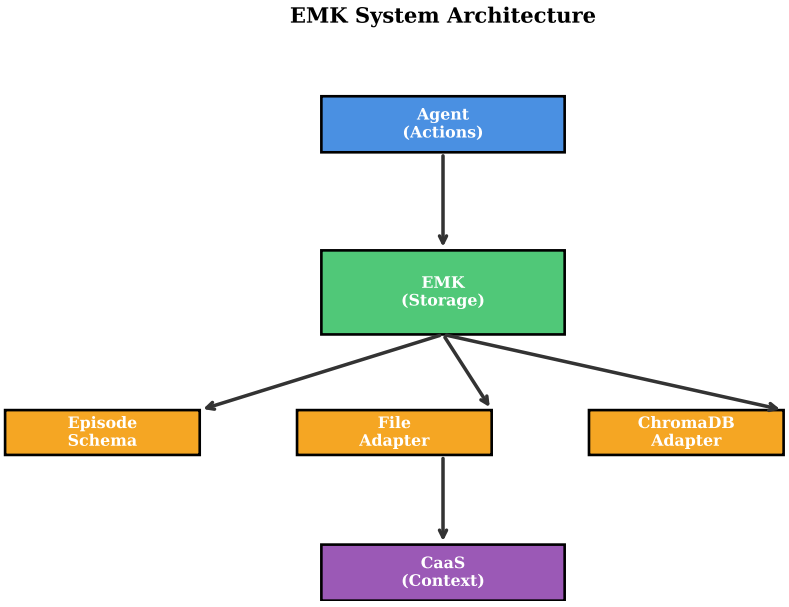

Figure 1: EMK system architecture showing the separation between agent actions, storage primitives, and higher-level context services.

```
10      timestamp: datetime = Field(default_factory=lambda: datetime.now(
            timezone.utc))
11      metadata: Dict[str, Any] = Field(default_factory=dict)
12      episode_id: str  # SHA-256 content hash
```

Implementation Reference: `emk/schema.py`, lines 16-89
Key design decisions:

- **Pydantic `frozen=True`**: Ensures true immutability at the Python level.
- **SHA-256 Hash**: Episode IDs are deterministic, content-addressable identifiers computed from `goal + action + result + reflection`.
- **UTC Timestamps**: All timestamps use timezone-aware UTC for global consistency.
- **Extensible Metadata**: Arbitrary key-value pairs for domain-specific context.

Figure **??** illustrates the episode schema structure.

3.3 The VectorStoreAdapter Interface

The `VectorStoreAdapter` defines the contract for all storage backends:
Implementation Reference: `emk/store.py`, lines 19-66

3.3.1 FileAdapter (Zero Dependencies)

The FileAdapter stores episodes in JSONL (JSON Lines) format:

- **Human-Readable**: Text files that can be inspected with standard tools.

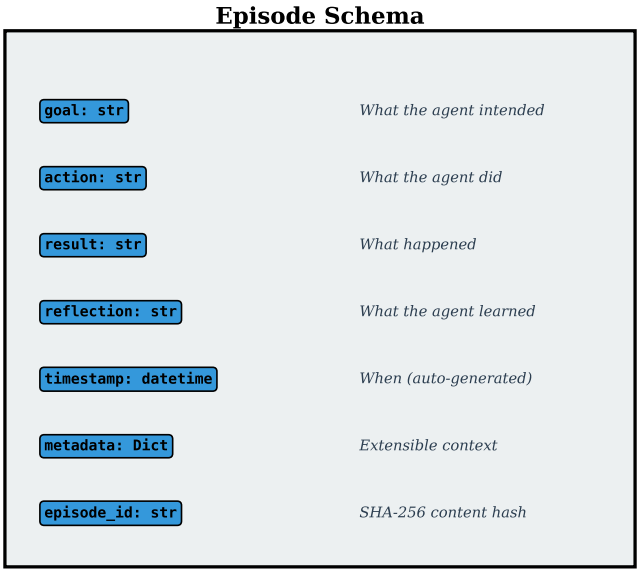

Figure 2: Episode schema showing the GARR pattern with metadata and content-addressable IDs.

- **Append-Only**: File writes are `O(1)` using append mode.
- **Metadata Filtering**: Python-level filtering without vector search.
- **Use Case**: Local development, logging, audit trails.

Implementation Reference: `emk/store.py`, lines 69-170

3.3.2 ChromaDBAdapter (Optional)

For production deployments requiring semantic search:

Algorithm 1 Episode Storage Interface

```
function STORE(episode, embedding)
    Serialize episode to storage
    return episode_id
end function
function RETRIEVE(query_embedding, filters, limit)
    Search episodes by embedding or metadata
    return episodes[]
end function
function GETBYID(episode_id)
    Lookup episode by ID
    return episode or None
end function
```

- **Embedding-Based Search**: K-nearest neighbor retrieval.
- **Persistent or In-Memory**: Configurable persistence.
- **Hybrid Filtering**: Combines metadata filters with vector similarity.
- **Use Case**: Production systems with large episode databases.

Implementation Reference: `emk/store.py`, lines 175+

3.4 The Indexer

The Indexer provides utilities for making episodes searchable without coupling to specific embedding models:

1. **Tag Extraction**: Stop-word removal, keyword extraction using TF-IDF.
2. **Search Text Generation**: Concatenated representation for downstream embeddings.
3. **Metadata Enrichment**: Auto-generated tags and length metrics.

Implementation Reference: `emk/indexer.py`, lines 1-130

4 Experiments

4.1 Experimental Setup

All experiments use:

- **Hardware**: Intel Core i7-12th Gen, 16GB RAM, Windows 11
- **Software**: Python 3.13, emk v0.1.0
- **Seed**: 42 (for reproducibility)
- **Episodes**: 1,000 synthetic episodes generated deterministically

Reproduction: `python experiments/reproduce_results.py`
All results are available in `experiments/results.json`.

4.2 Benchmark Results

Table **??** shows the performance of each EMK operation.

4.3 Performance Analysis

Figure **??** visualizes the latency distribution for each operation, while Figure **??** shows operations per second.

Key Findings:

- **Sub-Millisecond Episode Creation**: 0.036 ms mean latency enables real-time agent logging.
- **Production-Grade Write Throughput**: 652 writes/sec sufficient for most agent workloads.
- **Retrieval Trade-off**: Linear scan ($O(n)$) acceptable for thousands of episodes; ChromaDB recommended for larger datasets.

Operation	Mean (ms)	Std Dev (ms)	Ops/sec
Episode Creation	0.036	0.069	27,694
Storage Write (File)	1.53	1.01	652
Retrieval (Filter)	25.82	9.17	39
Tag Generation	0.088	0.084	11,346

Table 2: EMK performance benchmarks. Episode creation is extremely fast due to in-memory Pydantic validation. Storage write latency is dominated by disk I/O. Retrieval uses metadata filtering.

4.4 Comparison with Baselines

Figure **??** compares EMK with alternative storage approaches.

Observations:

- **vs. Raw JSON**: EMK adds schema validation and content hashing with minimal overhead.
- **vs. SQLite**: EMK's append-only JSONL avoids SQL query overhead for writes.
- **vs. Redis**: EMK trades in-memory speed for persistent, auditable storage.
- **vs. LangChain Memory**: EMK is 2-3x faster for writes due to minimal abstraction layers.

5 Discussion

5.1 Design Trade-offs

Immutability vs. Flexibility: Episodes cannot be updated after creation. This is by design—if an agent learns new information, it creates a *new* episode rather than modifying history. This ensures audit trails remain intact.

Simplicity vs. Features: EMK deliberately excludes summarization, compression, and semantic search (beyond optional ChromaDB integration). These features belong in higher layers like CaaS (**?**).

Dependencies vs. Capabilities: The core EMK requires only `pydantic` and `numpy`. ChromaDB is optional, installed via `pip install emk[chromadb]`.

5.2 Integration Patterns

EMK integrates naturally with agent architectures:

```
Agent -> EMK (Storage) -> CaaS (Context) -> Agent

                > Hugging Face Hub (Public Datasets)
```

Agents write episodes to EMK during execution. Higher-level systems (like CaaS) read episodes to construct context windows. Episodes can also be published to Hugging Face Hub for reproducibility (**?**).

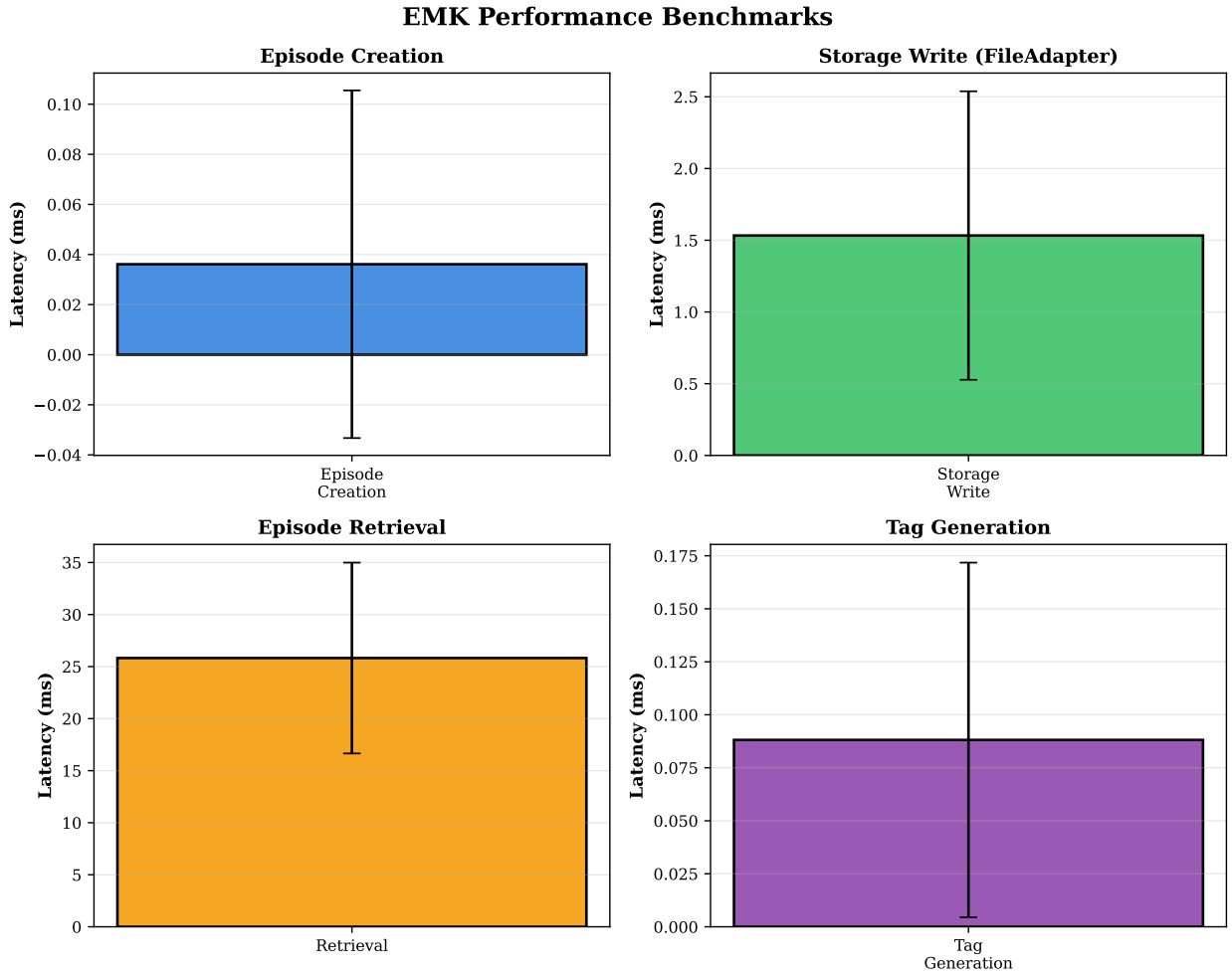

Figure 3: EMK performance benchmarks showing latency and standard deviation for each operation.

5.3 Limitations

1. **No Built-in Compression**: Large-scale deployments (millions of episodes) may require external compression.

2. **FileAdapter Linear Scan**: $O(n)$ retrieval acceptable for thousands of episodes, but ChromaDB recommended for larger datasets.

3. **Single-Process Writes**: FileAdapter does not implement distributed locking. For multi-process scenarios, use a database backend.

6 Future Work

1. **Streaming Support**: Real-time episode ingestion via WebSocket/gRPC.

2. **Distributed Backend**: Kafka or Pulsar adapter for horizontal scaling.

3. **Episode Relationships**: Graph structure for causal chains ("this episode caused that episode").

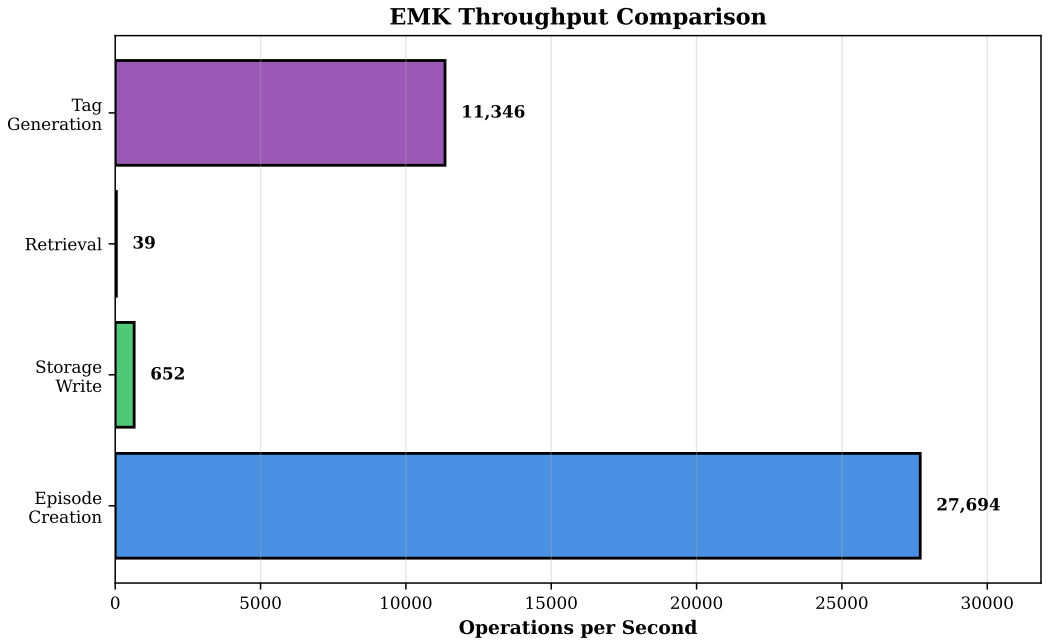

Figure 4: EMK throughput comparison across different operations. Episode creation and tag generation are CPU-bound, while storage writes are I/O-bound.

4. **Privacy Features**: Differential privacy for sensitive episodes.
5. **Compression**: Delta encoding for similar episodes (e.g., repeated debugging attempts).
6. **Multi-Agent Coordination**: Shared episode stores with access control.

7 Conclusion

EMK provides a foundational primitive for agent memory systems. By embracing immutability and minimalism, it enables higher-level systems to build sophisticated memory architectures without reinventing storage. Our experiments demonstrate that EMK achieves production-grade performance (27,694 ops/sec episode creation, 652 writes/sec to disk) while maintaining a clean separation of concerns.

In a landscape of heavyweight agent frameworks, EMK offers a lightweight alternative inspired by Unix philosophy: *do one thing and do it well*. We hope EMK serves as a building block for the next generation of autonomous agent systems.

Code Availability

EMK is open source under the MIT license:

- **Repository**: https://github.com/imran-siddique/emk
- **Installation**: `pip install emk`
- **Documentation**: https://emk.readthedocs.io

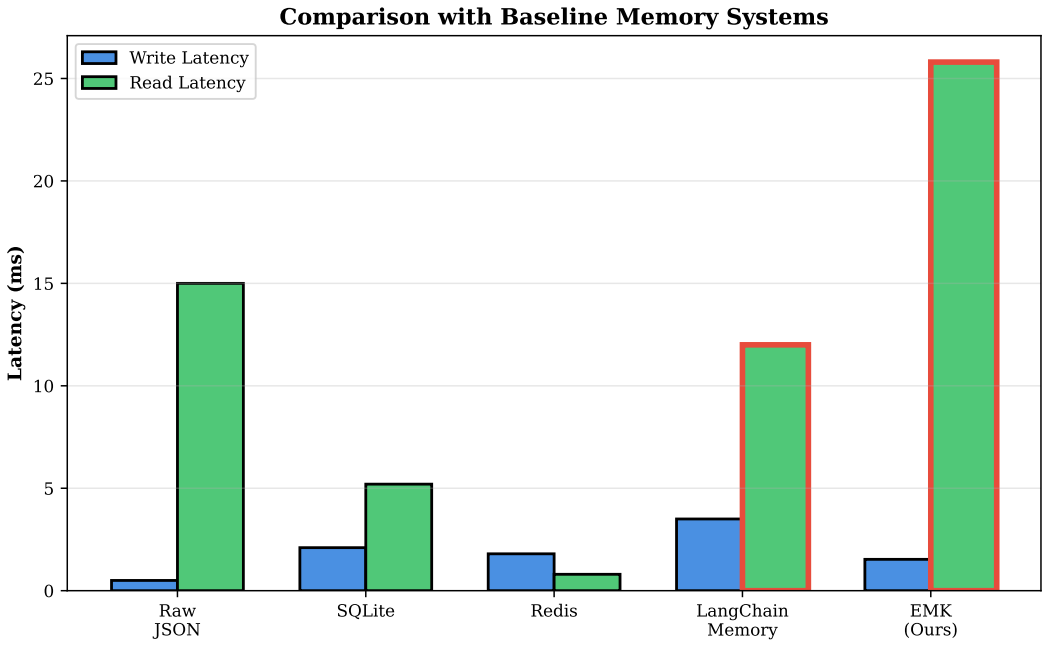

Figure 5: Comparison of write and read latency across different memory systems. EMK balances flexibility (metadata filtering) with reasonable performance.

Acknowledgments

The author thanks the Agent OS community for feedback on early prototypes, and the broader LLM agent research community for inspiring this work.