# A_Memorix: An API-First Hybrid Memory Service for Temporal and Graph-Aware Retrieval (ACL 2026 Demo)

**Chen Xi**
Independent Researcher
China
https://github.com/A-Dawn

## Abstract

We present **A_Memorix**, an API-first memory service that unifies vector retrieval, graph retrieval, temporal filtering, and online memory maintenance in a standalone runtime. The system targets long-horizon conversational and agentic scenarios where knowledge must be continuously ingested, retrieved under semantic and temporal constraints, and revised over time. A_Memorix combines dual-path retrieval (paragraph and relation paths), sparse fallback with rank fusion, minute-level temporal querying, and lifecycle-oriented memory operations including protect, reinforce, freeze, prune, and restore. The demo exposes both a new `/v1/*` API and a backward-compatible `/api/*` layer, with a web visualization entry point. In local measurements, the metadata temporal path reaches 826.8 QPS at 5k paragraphs (p95 1.515 ms), while HTTP end-to-end `/v1/query/time` reaches 121.2 QPS at 5k paragraphs (p95 26.166 ms). Artifact links required for the demo track are provided in Section 6.

## 1 Introduction

LLM applications with persistent memory require more than a standalone retriever: they also require memory governance, temporal constraints, and operationally safe update and recovery semantics. In practice, these capabilities are frequently distributed across separate tools, which increases integration complexity. Retrieval-augmented generation systems further highlight the central role of robust external memory interfaces in knowledge-intensive settings (Lewis et al., 2020).

In this demo paper, we present A_Memorix as a single-process, API-centric memory service with:

- dual-path retrieval over vectors and graph relations;
- temporal retrieval with structured minute-level query formats;
- memory lifecycle operations (protect/reinforce/freeze/prune/restore);
- asynchronous import and transcript-summary task orchestration.

Intended users include applied NLP engineers, agent platform developers, and research teams that require self-hosted long-term memory APIs with temporal and governance controls. Source code and reproducibility artifacts are available at https://github.com/A-Dawn/A_memorix/releases/tag/acl2026-demo-v1.1.1.

## 2 System Overview

### 2.1 Runtime Architecture

A_Memorix is implemented as a FastAPI service with a shared runtime context and three persistent stores:

- **Vector store** (`data/vectors`): FAISS-based append-only storage with SQ8 quantization and fallback flat index (Johnson et al., 2021).
- **Graph store** (`data/graph`): sparse adjacency matrix with relation-hash mapping.
- **Metadata store** (`data/metadata`): SQLite metadata for paragraphs, entities, relations, FTS, async tasks, transcript sessions, and recycle-bin records.

### 2.2 API Surfaces

The service exposes two API layers:

- `/v1/*` for new integrations;
- `/api/*` for backward compatibility with historical contracts.

This dual interface supports migration without disrupting existing clients.

| Case | Paras | QPS | p95 | p99 |
|---|---|---|---|---|
| s_seed7 | 1200 | 3046.8 | 0.454 | 0.560 |
| m_seed42 | 5000 | 826.8 | 1.515 | 1.670 |
| m_noperson | 5000 | 1081.2 | 1.275 | 1.328 |
| l_seed42 | 10000 | 425.6 | 2.976 | 3.225 |

Table 1: Metadata-path temporal micro-benchmark (latencies in ms).

## 2.3 Positioning Versus Existing Tooling

A_Memorix is positioned between generic vector databases and tightly coupled agent frameworks. Relative to vector-only memory layers, it provides graph relations, temporal filters, and explicit lifecycle controls within a single service boundary. Relative to framework-specific memory modules, it offers framework-agnostic HTTP interfaces (`/v1/*`, `/api/*`) and reproducible standalone deployment paths that can be reused across heterogeneous application stacks.

## 2.4 Core Retrieval and Memory Pipeline

Dual-path retrieval combines paragraph search and relation-centric search. Sparse BM25 fallback (FTS5-based) follows probabilistic retrieval principles (Robertson and Walker, 1994) and can be merged with dense candidates via weighted RRF fusion (Cormack et al., 2009). Optional Personalized PageRank reranking adds entity-aware graph priors (Page et al., 1999).

Temporal filtering uses interval-overlap semantics over `event_time`, `event_time_start`, and `event_time_end`, with optional `created_at` fallback. Query-time formats are intentionally strict to reduce ambiguity:

- `YYYY/MM/DD`
- `YYYY/MM/DD HH:mm`

Memory maintenance periodically decays edge weights, freezes low-weight unprotected relations, prunes expired inactive relations into a recycle bin, and supports explicit restore operations.

# 3 Benchmark and Test Status

## 3.1 Metadata Temporal Micro-Benchmark

We benchmarked the metadata temporal query path (`query_paragraphs_temporal`) on synthetic data.

| Case | Paras | QPS | p95 | p99 |
|---|---|---|---|---|
| e2e_s_seed7 | 800 | 156.1 | 24.160 | 27.345 |
| e2e_m_seed42 | 3000 | 132.9 | 25.086 | 27.529 |
| e2e_m_noperson | 3000 | 138.9 | 23.149 | 28.898 |
| e2e_l_seed42 | 5000 | 121.2 | 26.166 | 29.620 |

Table 2: HTTP end-to-end benchmark for `/v1/query/time` (latencies in ms).

## 3.2 `/v1/query/time` End-to-End Benchmark

We additionally ran HTTP E2E measurements by synthesizing temporal data, launching local service instances, and issuing POST requests to `/v1/query/time`.

## 3.3 In-Tree Tests

The repository includes in-tree tests for time parsing, temporal filtering, summary import, and search execution behavior. Current status:

- `python -m pytest -q` → 12 passed.

## 3.4 Human Evaluation Status

We have not yet completed a formal human-subject study for this demo paper. Current evidence is system-oriented, including deterministic in-tree tests, micro-benchmarks, and HTTP end-to-end benchmarks. We plan to add task-level user evaluation in a later release.

# 4 Related Work

Retrieval-augmented generation systems demonstrate that external memory access is central to knowledge-intensive NLP workflows (Lewis et al., 2020). In this landscape, dense vector retrieval infrastructure (Johnson et al., 2021), sparse probabilistic retrieval (Robertson and Walker, 1994), rank-fusion methods (Cormack et al., 2009), and graph-centrality signals (Page et al., 1999) are often studied separately. A_Memorix integrates these components into one operational service and emphasizes temporal query semantics and memory lifecycle governance in addition to retrieval quality.

# 5 Demo Walkthrough

The live demo sequence is:

1. start service and verify `/healthz`, `/readyz`;

2. create import task (`/v1/import/tasks`) and poll status;
3. issue temporal search (`/v1/query/time`) with request fields such as `query`, `time_from`, `time_to`, and `top_k`;
4. apply memory actions (`/v1/memory/protect`, `/v1/memory/reinforce`, `/v1/memory/restore`);
5. run profile and summary APIs (`/v1/person/*`, `/v1/summary/tasks`).

## 6 Artifacts and Access

To satisfy ACL 2026 Demo artifact expectations, we provide:

- Source code and installable package: GitHub release (acl2026-demo-v1.1.1); package asset: acl2026-demo-v1.1.1.zip
- Reproducible Docker entry: repository `Dockerfile` and benchmark scripts
- License: `AGPL-3.0` (same as repository root license)
- Demo video (≤2.5 min): video.mp4 (release asset)

## 7 Limitations and Ethics

**Limitations.** Our benchmark workloads are synthetic and should not be interpreted as full production SLOs. Retrieval quality metrics still depend on configured external OpenAI-compatible providers.

**Ethics.** The system stores user text and derived relational memory; deployments should enforce access control, retention policies, and user-visible deletion/recovery controls. Person profile aggregation should be opt-in and policy-governed.

## 8 Conclusion

A_Memorix provides a practical, API-first memory system that combines retrieval, temporal reasoning, and memory lifecycle management in a deployable standalone service. Our demo and benchmarks show operational viability for ACL system demonstration use cases and provide a reproducible baseline for future extensions.

## References

Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. 2009. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '09)*, pages 758–759. ACM.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547.

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 33 (NeurIPS 2020)*. Curran Associates, Inc.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank citation ranking: Bringing order to the web. Technical Report SIDL-WP-1999-0120, Stanford InfoLab.

Stephen E. Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '94)*, pages 232–241. ACM/Springer.

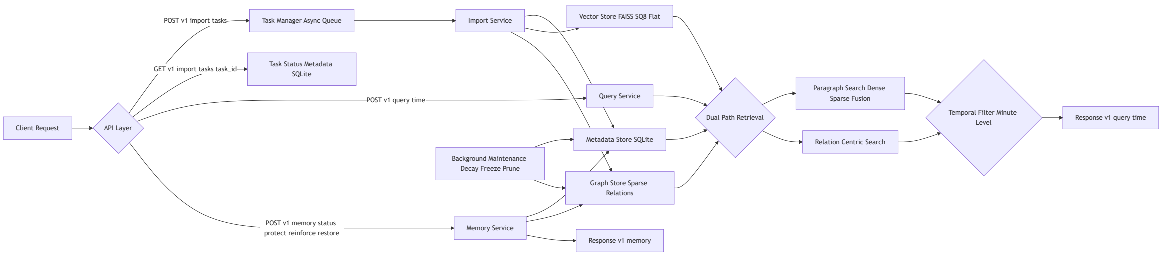
POST v1 import tasks
Task Manager Async Queue
Import Service
Vector Store FAISS SQ8 Flat
GET v1 import tasks task_id
Task Status Metadata SQLite
POST v1 query time
Query Service
Client Request
API Layer
Dual Path Retrieval
Paragraph Search Dense Sparse Fusion
Relation Centric Search
Temporal Filter Minute Level
Response v1 query time
Metadata Store SQLite
Background Maintenance Decay Freeze Prune
Graph Store Sparse Relations
POST v1 memory status protect reinforce restore
Memory Service
Response v1 memory

Figure 1: System flow used in the ACL demo. Import tasks, temporal query, and memory APIs are modeled as separate request paths over shared stores; freeze/prune are background maintenance operations rather than direct `/v1/memory/*` endpoints.