Product Requirements Document (PRD)

🧠Project Title: MemoryService – A Modular Cognitive Memory Layer for Autonomous Agents

---

1. Overview

MemoryService is an open-source, plug-and-play memory management system designed for autonomous agents, agent frameworks, and standalone LLM workflows. It mimics cognitive memory layers and allows agents to store, retrieve, summarize, and evolve their memory over time. The service is modular, extensible, and optimized for both prototyping and production use.

---

2. Objectives

- Provide a unified memory manager compatible with LangChain, CrewAI, AutoGen, and vanilla scripts.
- Enable multi-layered memory: raw logs, summarizations, vector embeddings, and graph-based relationships.
- Support self-evolving memory via summarization, scoring, decay, and relevance routing.
- Serve as the base memory engine for intelligent, long-running, goal-driven agents.

---

3. Target Users

- AI/LLM developers and researchers
- Builders of autonomous agent systems
- Open-source AI tool creators
- Academic researchers in cognitive architectures

---

4. Features

4.1 Core Features

Feature	Description
MemoryManager API	Unified interface to save, retrieve, link, and prune memory
Multi-format persistence	Support JSON, SQLite, Vector DB (FAISS), Graph DB
Memory schema	Unified memory chunk format with tagging and scoring

Feature	Description
Retrieval Router	Intent- and context-based memory retrieval logic
LLM-based Summarizer	Optional component to generate memory summaries
Priority and decay	Scoring system to evolve/prune memory dynamically
Query DSL	DSL/API to query memory across filters, time, importance
Metadata & tagging	Add and query memory chunks by importance, topic, context
Pluggable architecture	Add or remove backends and logic layers independently

4.2 Agent Adapters

MemoryService is designed as a framework-agnostic memory backend. Here's how it integrates with popular ecosystems:

Framework	Integration Type	Use Cases & Value
LangChain	Implements `BaseMemory` adapter	Drop-in replacement for LangChain's memory: supports raw, vector, and summary layers
CrewAI	Injects context + memory scoring	Supplies recent memory + goals + summaries into agent task runs
AutoGen	Hooks into `history` and `postprocess`	Maintains ongoing chat history, enhances reactivity via scored memory
ADK (LlamaIndex)	Exposes `MemoryComponent` plugin	Adds custom memory as a source + destination for data graphs
Vanilla Scripts	SDK + CLI usage	Developers can use MemoryService independently of any framework
Future: LangGraph	Graph-based LLM workflow planner	Memory as a first-class node in LangGraph-style pipeline
Integration	Description	
---	---	
LangChain (optional)	Custom memory adapter implementing BaseMemory	Custom memory adapter implementing BaseMemory
CrewAI (optional)	Context injection and result tracking	Context injection and result tracking

Framework	Integration Type	Use Cases & Value
AutoGen (optional)	Hook into `postprocess` and `history`	Hook into `postprocess` and `history`
ADK (optional)	Pluggable MemoryComponent interface	Pluggable MemoryComponent interface

4.3 Developer Tools

- CLI: `memory save`, `memory query`, `memory graph`
- Local viewer: Visualize memory timelines and clusters
- REST API: `POST /memory`, `GET /memory?tags=projectX`

---

5. Technical Architecture

5.x Performance Optimization Plan

To reduce latency introduced by the multi-layered architecture, the following strategies will be implemented:

Retrieval Optimization

- **Layer Prioritization:** Early-exit logic in the MemoryRouter to return results from the first layer that produces relevant content.
- **Pre-filtering by Tags/Context:** Limit the search space of vector stores or summaries using metadata tags.
- **Importance-Based Filtering:** Prioritize retrieval from high-importance or frequently accessed memory chunks.

System Performance

- **LRU/Redis Caching:** Cache embeddings, top queries, and recent retrieval results.
- **Asynchronous Retrieval:** Use asyncio/threading to pre-fetch memory in parallel.
- **Time-Constrained Graph Traversal:** Set hop/time limits when querying the graph memory layer.
- **Summary Caching:** Avoid redundant summarization by caching summaries and only updating them incrementally.
- **Memory Zone Partitioning:** Use hot/cold storage zones to distinguish between recent and archived memory.

Benchmarks & KPIs

- **Query Latency per Layer:** Time to respond from vector, summary, and raw layers.
- **Cache Hit Ratio:** Percentage of cache hits vs total memory requests.
- **End-to-End Retrieval Latency:** Total time to return memory given a goal or task.
- **Graph Traversal Time:** Time to explore connected nodes in graph memory.

These optimizations will ensure the system maintains fast access times even as memory scales and complexity increases.

5.0 Dependencies

Each dependency below is tagged with the concept(s) or capability it helps enable:

Dependency	Purpose	Supports Concepts
Python 3.10+	Core runtime	General Infrastructure
FAISS	Vector similarity search (optional plugin)	Semantic Recall, Similarity Retrieval
Chroma	Alternative vector store backend	Vector Search, Plug-n-play Storage
SQLite	Lightweight persistent structured storage	Raw Store, Zero Dependency, Long-Term Memory
DuckDB	Metadata scoring and in-memory analytics	Adaptive Behavior, Memory Scoring, Summary Analysis
OpenAI / HF APIs	Embedding + Summarization APIs	Summary, Semantic Memory, LLM Scoring, Feedback Loop
Neo4j	Graph memory backend	Graph + Timeline Memory, Intent Routing, Multi-Agent Context
networkx	In-memory graph operations	Graph Memory, Memory Traversal, Lightweight Graphing
LangChain	Agent framework integration	Adapter Layer, Optional Ecosystem Plug-in
CrewAI	Agent task memory injection	Multi-Agent Collaboration, Task-Based Recall
AutoGen	Hook memory via postprocess/ history	Self-Evolving Memory, Feedback Loop
ADK (LlamaIndex)	Modular memory plug-in support	Adapter for LlamaIndex-style workflows
FastAPI	REST API interface	API-based Access, Server Interfaces
React/Svelte	UI for graph dashboard (Phase 3)	Visual Memory Maps, Inspection Tools
Pytest	Unit testing	QA and Stability
Black/Flake8	Code linting and formatting	DevX & Maintainability

Dependency	Purpose
Python 3.10+	Core runtime
FAISS	Vector similarity search (optional plugin)

Dependency	Purpose
Chroma	Alternative vector store backend
SQLite	Lightweight persistent structured storage
DuckDB	Metadata scoring and in-memory analytics
OpenAI / HF APIs	Embedding + Summarization APIs
Neo4j	Graph memory backend
networkx	In-memory graph operations
LangChain	Agent framework integration
CrewAI	Agent task memory injection
AutoGen	Hook memory via postprocess/history
ADK (LlamaIndex)	Modular memory plug-in support
FastAPI	REST API interface
React/Svelte	UI for graph dashboard (Phase 3)
Pytest	Unit testing
Black/Flake8	Code linting and formatting

5.1 Memory Chunk Format

```
{
  "id": "mem_001",
  "type": "tool_result",
  "timestamp": "2025-07-19T14:00",
  "agent": "researcher",
  "content": "ToolX failed due to timeout",
  "importance": 8,
  "tags": ["toolX", "failure", "critical"],
  "embedding": [...],
  "linked_to": ["mem_000"]
}
```

5.2 Layered Memory Stores

Layer	Backend	Purpose
Raw	JSON, SQLite	Full message history
Summary	JSON, LLM API	Summarized memory chunks

Layer	Backend	Purpose
Vector	FAISS, Chroma, SQLite	Semantic search support
Graph	Neo4j, networkx	Causal/relational reasoning
Metadata Store	DuckDB, SQLite	Filter, tag, and score memory

5.3 Memory Router

- Determines the best store(s) to retrieve from based on agent query, tags, goal, and priority.
- Supports fallback chains (e.g., vector → summary → raw)

---

6. Milestones & Roadmap

😺Usability-Centric Memory Capabilities

To fully capture real-world usability of memory in agent scenarios, we define how MemoryService supports query-specific and behavior-specific use cases.

Memory Use Case	Supported By	Description
Query Relevance Tracking	Retrieval Router, Metadata Scoring	Ensures memory chunks retrieved are relevant to the agent's context
Temporal Awareness	Memory Schema (timestamps, linked chunks)	Enables time-aware queries and decaying recall
Feedback-Loop Based Adaptation	Decay Engine, Memory Scoring	Adjusts importance based on agent feedback or task outcomes
Intent-Based Context Resolution	Goal Tags, Retrieval DSL	Allows agents to retrieve memories aligned with goal/task intent
Session-Based Isolation	Metadata Tagging, CLI	Filter memories by session/user/agent role
Multi-query Resolution	MemoryRouter + Graph Traversal	Fuses multiple memory layers to provide coherent answers
User Behavioral Pattern Detection	Scoring + Graph Memory	Flags shifts in interaction or task evolution
Adaptive Recall	Scoring, Fading, and Summary Refresh	Refreshes memory over time based on priority and recent relevance
Raw + Summary Fusion	Retrieval Chain Logic	Allows blended answers from summarized + original memory

Memory Use Case	Supported By	Description
Cross-Agent Memory Sync	Adapter Layer + Metadata Tagging	Supports shared memories or views across agent roles
Layered Retrieval Traceability	CLI + Logs + Graph Traversal	Enables developers to trace which layer contributed to what answer

These ensure MemoryService doesn't just store data—but behaves in a usable, goal-aligned, evolving manner across agent use cases.

👹Comparison with Existing Memory Services

Feature / Capability	MemoryService	LangChain Memory	CrewAI Memory	AutoGen Memory	LlamaIndex Memory	Notes
Multi-layered Architecture	✅	⚠️Single-layer	⚠️ Custom	⚠️Custom	⚠️ Component-only	MemoryService separates raw, vector, summary, graph
Vector Store Support	✅Pluggable	⚠️Built-in	❌	❌	✅	MemoryService supports FAISS, Chroma, SQLite
Graph Memory	✅Neo4j/ NetworkX	❌	❌	❌	⚠️WIP	Explicit support for relational memory & timeline
Summarization Layer	✅Optional LLM	⚠️LLM-Only	❌	⚠️ Hardcoded	✅	LLM-based summarizer, local or remote
Score + Decay Engine	✅Priority/ Decay	❌	❌	❌	❌	Importance-based auto pruning and fading
DSL for Querying	✅	❌	❌	❌	⚠️API only	Developer-friendly query language
CLI Tooling	✅Full CLI	❌	❌	❌	❌	Use memory without any framework

Feature / Capability	MemoryService	LangChain Memory	CrewAI Memory	AutoGen Memory	LlamaIndex Memory	Notes
REST API	✅	❌	❌	⚠️ Experimental	⚠️Custom	Accessible from any platform/ language
Framework Independence	✅Core Neutral	❌	❌	❌	❌	Designed to be adopted by any system
Feedback Loop Awareness	✅	❌	❌	⚠️Manual	❌	Memory improves over time based on usage
Intent/Goal-Based Retrieval	✅	❌	⚠️ Static	⚠️Prompt logic	❌	Retrieval logic linked to user/ agent intent
Telemetry Support	✅Optional	❌	❌	❌	❌	Benchmarking and memory usage analytics
Zero Dependency Option	✅Planned	❌	❌	❌	❌	SQLite + On-device embeddings/ summarization

✅= Full support | ⚠️= Limited/Manual | ❌= Not available

Benchmark & Telemetry Plan

To monitor and optimize memory performance and usability:

Key Benchmarks:

- 🩷*Layer Latency Benchmarks:* Measure retrieval speed across raw, vector, summary, and graph layers
- 🔁*Cache Hit Ratio:* Track efficiency of recent memory/cache layers
- *Recall Accuracy:* Verify if correct memory is retrieved for a given prompt and goal
- ⚠️*Intent Routing Latency:* Time taken to resolve retrieval path based on goal/intent

Telemetry Goals (Optional Opt-in):

- 🔄Popular query tags, layer usage ratio (e.g., vector vs summary), store sizes
- 🔕Memory scoring usage trends (decay frequency, importance scores)
- 🆚Graph traversal depth and time

All telemetry will be opt-in, anonymized, and focused on improving memory performance across agentic frameworks.

⬛Benchmark & Telemetry Plan

To monitor and optimize memory performance and usability:

Key Benchmarks:

- 🩷*Layer Latency Benchmarks:* Measure retrieval speed across raw, vector, summary, and graph layers
- 🔁*Cache Hit Ratio:* Track efficiency of recent memory/cache layers
- *Recall Accuracy:* Verify if correct memory is retrieved for a given prompt and goal
- ⚠️*Intent Routing Latency:* Time taken to resolve retrieval path based on goal/intent

Telemetry Goals (Optional Opt-in):

- 🔄Popular query tags, layer usage ratio (e.g., vector vs summary), store sizes
- 🔕Memory scoring usage trends (decay frequency, importance scores)
- 🆚Graph traversal depth and time

All telemetry will be opt-in, anonymized, and focused on improving memory performance across agentic frameworks.

👉🏿Conceptual Mapping Table

Concept	Implemented In	Dependency
Context Over Time	Phase 1 - Schema/Metadata	JSON/SQLite
Adaptive Behavior	Phase 2 - Scoring/Decay	Plugin Engine
Goal-Oriented Recall	Phase 1, Phase 3	MemoryRouter
Multi-Agent Collaboration	Phase 2	CrewAI Adapter
Self-Evolving Memory	Phase 1, Phase 2	Summarizer/Decay
Knowledge Graph + Timeline	Phase 3 - Graph Memory	Neo4j/NetworkX
Emotional/Biological Memory	Phase 2	Custom Decay Engine
Visual Memory Maps	Phase 3	React/Svelte
Intent-Based Retrieval	Phase 3	Traversal Engine
Feedback Loop Awareness	Phase 3	Memory Scoring
Zero-Dependency Operation	Future Plan (11.4)	CLI/SQLite/TinyML

Phase 1: MVP (1 Month) 🧠[Foundational]

(Supports: Context Over Time, Goal-Oriented Recall, Self-Evolving Memory, Summary-based Recall)

Development

- Schema design and memory chunk definition
- Core MemoryManager with basic operations
- Implement JSON and SQLite stores
- CLI (basic): save, retrieve, summarize
- Basic scoring and tagging
- Embedding integration (OpenAI API)
- Retrieval fallback logic (vector → summary → raw)
- Unit tests for all modules
- CI pipeline with lint + test

Documentation & Awareness

- README + setup docs
- Basic tutorials
- Medium launch post
- Demo video + social thread

Phase 2: Modular & Extensible (1–2 Months) 😾[Semantic + Extensibility]

(Supports: Adaptive Behavior, Multi-Agent Collaboration, Biological Memory Decay, Intent + Similarity Retrieval)

Development

- Plugin system for stores and summarizers
- Vector store adapters (FAISS, Chroma)
- Graph store adapter (Neo4j)
- Decay engine for memory pruning
- LangChain + CrewAI adapters (optional)

Documentation & Awareness

- Plugin builder guide
- LangChain/CrewAI walkthroughs
- Medium article: Semantic memory integration
- Video series: “Memory in Agents”

Phase 3: Graph Memory + Full UX (2–3 Months) 👃[Graph + Cognitive UX]

(Supports: Knowledge Graph + Timeline, Visual Memory Maps, Feedback Loops, Contextual Awareness, Intent-Based Retrieval)

Development

- Graph memory engine (networkx, Neo4j)
- Traversal and scoring engine
- Interactive dashboard (React/Svelte)
- REST API (CRUD + query)

- AutoGen + ADK adapters (optional)

Documentation & Awareness

- Full API reference
- Graph-based memory tutorial
- Integration walkthroughs
- Architecture deep-dive video
- Community webinar and AMA

---

7. Adoption Strategy

- MemoryService is a foundational package intended to be **used by frameworks**, not depend on them
- Direct integration with LLM APIs allows us to operate fully independently (e.g., OpenAI, Claude, open-source models)
- Provide optional plugins for LangChain, CrewAI, AutoGen — but do not tightly couple the project to any one ecosystem
- Position MemoryService as a general-purpose, LLM-agnostic memory layer usable via REST, CLI, and SDK
- Emphasize agent framework neutrality in branding, documentation, and architecture
- Offer adapters as convenience utilities, not core modules
- Encourage community plugins for niche frameworks without adding core dependencies
- Target developer tooling and system builders who need persistent, modular memory
- Create sample projects showing usage in vanilla Python, FastAPI, terminal, and headless modes
- Build compatibility with **any LLM or agentic framework**, not just LangChain, CrewAI, or AutoGen
- Provide generic SDKs, adapters, and REST APIs to maximize framework independence
- Avoid hard dependencies or philosophical alignment with specific frameworks
- Promote MemoryService as a standalone, pluggable memory layer usable via CLI, REST, or SDK
- Offer ready-to-use plugins for LangChain, CrewAI, AutoGen **as optional integrations**
- Contribute sample integrations to multiple ecosystems without tightly coupling to any
- Engage with broader dev communities: RAG frameworks, research tools, browser agents, etc.
- Create a neutral project brand that appeals to both experimental and production teams
- Identify 3 popular open-source frameworks to adopt (LangChain, CrewAI, AutoGen)
- Contribute ready-to-use memory plugins as PRs
- Provide sample projects with plug-n-play demos
- Collaborate with each community via GitHub/Discord/X
- Create a landing page with detailed docs, showcase videos, and project badges
- Host webinars and AMA sessions on using memory in agent development

---

8. Success Criteria

- 1000+ GitHub stars and active contributors
- At least 3 agent frameworks using or referencing MemoryService
- Developer adoption across at least 100 independent projects
- MemoryService mentioned in major agentic repos, talks, or publications

9. Risks & Mitigation

Risk	Mitigation
Too heavy for prototypes	Design for minimal mode (JSON only)
Vector store adds complexity	Keep optional with plugin system
Privacy / safety concerns	Offer local-only mode with no external API calls
Difficult integration	Write adapters + docs for all major frameworks

10. License & Community

- **License:** MIT
- **Community Tools:** GitHub Discussions, Discord, Contributor Guide, Hackathon Support
- **Outreach:** Publish thought-leadership content monthly, community contributor recognition

11. Future Possibilities

11.1 Autonomous Agent Enhancements

- **Context Over Time**: Long-term memory with timestamps, linked summaries, and fading relevance.
- **Adaptive Behavior**: Memory scoring, event tagging, and prioritization evolve based on success/failure.
- **Multi-Agent Collaboration**: Role-specific memory slices and shared task memories with context separation.
- **Goal-Oriented Recall**: MemoryRouter aligns queries with active goals or agent-defined intents.
- **Long-Term Efficiency**: Summary pipelines, memory decay, and hot/cold zoning to control memory bloat.

11.2 Cognitive Architectures (Out-of-the-Box Concepts)

- **Multi-Layered Cognitive Memory**: Model human memory (sensory buffer, working, short-term, long-term, episodic, procedural) across different memory store layers.
- **OS-Inspired Memory**: Paging-like memory swapping, access control, shared memory pools, garbage collection.
- **Biological Reactions**: Memory tagged with sentiment, stress level, and attention-weight for emotional recall.
- **Knowledge Graph + Timeline**: Memory as a dynamically growing graph + time series (events, agents, tasks).
- **Self-Evolving Memory**: Memory updates itself based on usage patterns, aging, or LLM-scored relevance.

11.3 A New Retrieval Paradigm

Move beyond recency, similarity, and summary-based recall by retrieving memory via:

- **Intent**: What the agent is trying to do
- **Task Goal**: Match memory with ongoing objectives
- **Behavioral Change**: Detect patterns or divergences in user/agent actions
- **Feedback Loops**: Memory annotated by previous task success, failure, or reward signals
- **Contextual Awareness**: Cross-agent + cross-session correlation using tags and memory graph traversal

11.4 Zero-Dependency Roadmap

- Replace OpenAI/HF embeddings with local models (e.g., sentence-transformers, onnx, TinyML)
- Replace FAISS with hnswlib or faissqlite
- Enable offline summarization using llama.cpp and GGUF format
- Build self-contained CLI & REST tools (no FastAPI)
- Add support for in-browser memory (IndexedDB + WebAssembly)
- Enable modular plugin loading for external capabilities (embeddings, storage, graphing)
- Package minimal memory engine for edge/IoT usage
- Visual Memory Maps
- Emotion-weighted memories
- Personalization APIs (for multi-user/agent scenarios)
- Browser-native version (WebAssembly)
- Integration with LangGraph or semantic planning tools
- **Zero-dependency mode** with the following roadmap: -
- Visual Memory Maps
- Emotion-weighted memories
- Personalization APIs (for multi-user/agent scenarios)
- Browser-native version (WebAssembly)
- Integration with LangGraph or semantic planning tools