SAM: Smart Access Memory

Intelligent AI Memory Management with ML Auto-Triggers

PiGrieco (Open Source Project)

October 19, 2025

Introduction and Purpose

What is SAM? SAM (*Smart Access Memory*) is an intelligent memory management system for AI assistants. It automatically decides when to save important information and when to retrieve it for context, using a custom machine learning model (achieving 99.56% accuracy) to analyze conversations in real-time.

In essence, SAM augments AI platforms with a persistent, context-aware memory **without any manual intervention by the user**.

Introduction and Purpose

Key Benefits:

- **Automatic Memory Management**: No manual commands needed – SAM intelligently saves or searches memory as needed.
- **Context-Aware Retrieval**: Understands conversation flow to fetch relevant past information at the right time.
- **Universal Integration**: Works across major AI platforms (e.g. Cursor IDE, Claude, Windsurf) via a standard protocol (MCP).
- **Simple Installation**: One-command setup on any platform (just a prompt instruction) makes it easy to enable SAM.

Use Case: AI-Assisted Coding Consistency

Scenario: A development team uses various AI coding assistant (e.g. Cursor IDE with GPT integration, Claude Code...) to generate code. The lead engineer defines project-specific guidelines – architecture decisions, code style conventions, important API keys/configs – and wants all AI-generated code to follow these rules.

Use Case: AI-Assisted Coding Consistency

How SAM Helps: SAM automatically saves key project notes and decisions as they come up (e.g. "Use PostgreSQL at host X, port Y" or coding style preferences) and later recalls them when context is needed. For example:

- When a developer tells the AI, "Remember, we decided to use library Z for logging," SAM will store that information persistently (in a MongoDB memory database).
- Weeks later, if someone asks the AI, "How do we log errors in this project?", SAM's memory is automatically searched and the AI is provided with the reminder to use library Z, ensuring consistent guidance. Also, any generation including the library will be biased to use such library.

Use Case: AI-Assisted Coding Consistency

This leads to **consistent code and knowledge sharing**: project requirements, technical solutions, and decisions are automatically saved and available to all team members via the AI.

Important solutions and best practices are never lost, and the AI always adheres to the team's established guidelines.

Architecture Overview

High-Level Design: SAM is a standalone memory server that interfaces with AI platforms using the open *Model Context Protocol (MCP)*. AI front-ends (Cursor, Claude, etc.) connect to SAM via MCP, allowing seamless integration into IDEs and chat assistants.

Architecture Overview

Whenever you interact with the AI, your messages are also sent to the SAM server, which then:

- **Analyzes each message in real-time** with an ML-based trigger system. Server, ML model and DB are installed locally.
- **Decides on an action**: SAM's logic will either *save* the information to memory, *search* the memory for relevant info, or take no action.
- **Executes memory operations**: If a save is triggered, the content is semantically embedded and stored in the database; if a search is triggered, SAM performs a semantic lookup to retrieve any relevant stored knowledge.
- **Enhances the AI's response**: Any retrieved context is injected back to the AI assistant, enriching its reply with relevant past information without the user having to manually recall.

Architecture Overview

Key Components: Internally, SAM's architecture has an **Auto-Trigger System** for deciding when to save/search, and a **Memory Service** for storing and retrieving information. Stored data is kept in a MongoDB database and indexed with vector embeddings for semantic search. This design ensures the system is both intelligent and efficient in handling memory.

A Watchdog is added for error handling and a manual fallback with keywords for memory operations is in place.

Server Modes and Integration

SAM can operate in multiple modes to fit different environments:

- **MCP-Only Mode**: Uses standard I/O (stdio) via the MCP protocol for direct integration with IDEs (ideal for Cursor, Claude Desktop, etc.). In this mode, SAM runs in the background and communicates through the MCP interface without needing network sockets.
- **HTTP API Mode**: Runs an HTTP REST server (default port 8000) exposing endpoints for memory operations. This is useful for development, testing, or integrating SAM with web applications or tools via simple HTTP requests.

Server Modes and Integration

- **Proxy Mode**: Runs an HTTP proxy on port 8080 that AI requests can be routed through. In this mode, SAM intercepts the traffic between an AI client and its backend, injecting memory management automatically. This “AI interception” provides enhanced features without modifying the client.
- **Universal Mode**: Runs both the MCP interface and HTTP server together. This is a production-ready mode ensuring maximum compatibility — the IDE can use MCP and external tools can use the HTTP API simultaneously.

Server Modes and Integration

- **Watchdog Mode**: In any mode, an optional *watchdog service* can be enabled. The watchdog monitors for specific keywords (e.g., "remember this", "restart SAM") in the input or log files and will automatically restart the SAM server if it detects that a restart keyword was issued or if the process died unexpectedly. This ensures high reliability (SAM is always running when needed) with safeguards like rate-limiting (max 10 auto-restarts/hour, etc.) to prevent loops.

Auto-Trigger System (ML + Rules)

A core innovation in SAM is its **Auto-Trigger System**, which decides when to save or retrieve memories without user commands. It uses a **hybrid approach**:

- **Machine Learning Model**: At the heart is a custom-trained transformer (BERT-based) classifier that examines each user message. This model was trained on over **50,000 annotated conversation snippets** labeled as either `SAVE_MEMORY`, `SEARCH_MEMORY`, or `NO_ACTION`. The training data spans technical discussions and project chats in English (70%) and Italian (30%), so the model is bilingual. It achieved about **99.56% accuracy** on its validation set – extremely high precision/recall for detecting when to save or fetch memory.

Auto-Trigger System (ML + Rules)

- **Deterministic Rules**: In parallel, SAM has hand-crafted rules to catch obvious cues. For example, if a user explicitly says “remember this”, these patterns are recognized by rules. The system combines rules with the ML model in a *hybrid engine* – the rules handle clear-cut cases and serve as a fallback if the ML model is unsure.
- **High Confidence Thresholds**: To avoid false triggers, the ML model is conservative – it will only trigger a memory save/search when its confidence is very high (e.g. $>$95% certain). If the model’s confidence is low or ambiguous, SAM defers to deterministic rules or does nothing, ensuring reliability.

Outcome: Through this hybrid ML+rules approach, SAM can intelligently detect opportunities to save important info or bring up relevant context with human-level precision, while minimizing any erroneous actions.

Intelligent Memory Management in Action

Once the Auto-Trigger System decides to act, SAM handles memory operations as follows:

- **Auto-Save Mechanism**: For messages classified as containing important information, SAM will automatically save the content to its memory store. Examples that trigger saves include: key decisions, technical solutions or workarounds, project requirements/specifications, and insights or explanations that could be useful later. SAM extracts the relevant text and stores it in a MongoDB database, indexing it with a vector embedding (using a Sentence-Transformer model) so that it can be semantically searched later.

Intelligent Memory Management in Action

- **Auto-Search Recall**: If a user's message seems to query past knowledge (e.g. "Did we already decide on the database config?" or "Can I see the solution we used last time for X?"), SAM will automatically perform a memory search. It uses **semantic similarity search**: the current query or conversation context is converted into an embedding and compared against stored memory embeddings to find relevant matches (even if the wording is different). Any relevant snippets found are fetched from the database.
- **Context Injection**: For search triggers, the retrieved information is provided back to the AI assistant (e.g., appended to the prompt or via the MCP interface) so that the AI can directly use those details in its answer. This happens seamlessly – the user doesn't have to ask explicitly for the past info; SAM augments the AI's context behind the scenes.

Intelligent Memory Management in Action

- **No-Action Filtering**: If the message is trivial (small talk, greetings, or off-topic chitchat), SAM appropriately does nothing. The system is designed to stay silent unless it truly detects a valuable memory operation, so it won't clutter the conversation with unnecessary interventions.

Benefit: All important knowledge persists across sessions and is available on demand. This semantic memory means even if you phrase a question differently later, SAM can surface the relevant info (e.g., it understands that "DB config" is the same as "database settings" you saved).

Project Structure and Components

Under the hood, **SAM is organized into modular components**:

- **Core Server Logic**: Entry point is `main.py`, which launches the MCP memory server. The core logic resides in 'src/core/' – including `server.py` (the server implementation) and the trigger systems: `auto_trigger_system.py`, `ml_trigger_system.py`, and `hybrid_trigger_system.py` (for rule-based, ML-based, and combined triggering).
- **Memory Services**: In 'src/services/', the project implements distinct services for memory management. Notably, `memory_service.py` handles high-level memory operations (invoking saves/searches), `database_service.py` manages MongoDB interactions (storing and retrieving documents), and `embedding_service.py` computes vector embeddings for texts.

Project Structure and Components

- **Interfaces**: The server supports multiple interfaces as separate modules under ‘servers/‘ – e.g. `http_server.py` for the REST API mode and `proxy_server.py` for the HTTP proxy mode. These leverage the same core logic but expose it via different protocols (HTTP endpoints or proxy).
- **Installation Utilities**: A collection of scripts in ‘scripts/‘ facilitate installation and management. The primary script is `main.sh`, a unified shell script that can install dependencies, configure platforms, or start the server in various modes. There are also platform-specific install scripts and environment configuration files (in ‘config/‘), plus a comprehensive test suite in ‘tests/‘ to verify functionality.

Installation and Setup

One goal of SAM is to be **extremely easy to install**, even for non-experts:

- **Prompt-Based Installation**: You can install SAM by simply instructing your AI assistant or IDE. For example, telling Cursor IDE: *“Install this: https://github.com/PiGrieco/mcp-memory-server on Cursor”* triggers an automated installation process. This one-command setup is cross-platform – just replace the platform name (Cursor, Claude, etc.) and the system will fetch and install the memory server.

Installation and Setup

- **Automated Setup Process**: Under the hood, the installer will check for required dependencies (Python 3.8+, MongoDB, Git) and install any missing ones, then create a virtual environment and install all Python packages. It downloads the ML model (from HuggingFace) automatically, sets up the MongoDB connection, and generates all necessary configuration files (for the MCP server, HTTP proxy, watchdog, etc.). Nothing is left to manual configuration.

Installation and Setup

- **IDE Integration Steps**: The install scripts even configure the client platform for you. For instance, in Cursor IDE it updates your ' /.cursor/settings.json' (or equivalent) to add the SAM server, so that Cursor knows to launch and use the memory server in the background. Similarly, for Claude Desktop it updates its config to include SAM. This means after installation, your AI assistant is immediately "memory-aware" without extra setup.
- **Alternative Installation**: If the prompt method isn't available, SAM provides direct install commands (curl scripts or the 'scripts/main.sh' interface for manual install) for each platform. Developers can also use Docker (with a provided Docker Compose file) for a quick production setup.

Conclusion and Next Steps

In summary, SAM (Smart Access Memory) adds a powerful long-term memory to AI systems, enabling them to retain and recall knowledge with minimal effort from users. It ensures important information, from code decisions to project insights, is never lost and is readily available to enhance future AI interactions. By combining advanced ML triggers with robust engineering (database, embeddings, etc.), SAM offers a reliable, high-precision memory companion for any AI assistant.

Conclusion and Next Steps

Open Source and Contributions: This project is fully open-source (MIT licensed). The repository is available on GitHub (PiGrieco/mcp-memory-server), and we warmly welcome contributors from the community. Whether it's improving the ML model, adding features, or reporting issues, any participation is appreciated. If you find SAM useful, you can star the repo and share it.

Future Vision: We plan to develop a managed cloud-based version of SAM for those who prefer a hosted solution.

Conclusion and Next Steps

Call to action: If you're interested in this technology or even in building a startup around intelligent AI memory, please get in touch!

We're exploring forming a founding team to take SAM to the next level. Meanwhile, feel free to try SAM yourself, contribute on GitHub, and integrate it into your projects.

Let's push the boundaries of AI-assisted development together!