

MEMORY KEEP BLUEPRINT — v2.0 (Production-Hardened)

We didn't build a system that never forgets. We built a system that knows how to remember — efficiently, deliberately, and at scale.

PART I — HUMAN EXPLANATION (HOW IT WORKS IN PRACTICE)

The Core Idea

Large Language Models do not remember. They only see what is placed in front of them at generation time.

If you continuously replay the entire conversation history: - costs explode - reasoning degrades - hallucinations increase - systems feel slow, confused, and brittle

Memory Keep solves this by separating conscious thought from long-term memory.

The AI has: - a **small, controlled conscious mind** (what it is actively thinking with) - and a **large external memory** (what it can access when needed)

Only what the AI must *actively reason over* lives in context. Everything else is stored safely and retrieved intentionally.

Conscious Capacity Is a Design Choice

Even if a model supports massive context windows (hundreds of thousands or millions of tokens), **you do not have to use them.**

Memory Keep allows you to define an **application-level context cap** based on the job:

- chat assistants → smaller windows
- blogging tools → moderate windows
- complex analysis tools → larger windows

The AI remains fully intelligent because: - identity and rules are always present - long-term memory is always searchable

More context is not more intelligence. Controlled context produces better reasoning at lower cost.

The Memory Types (Plain Language)

1. Core Memory — *Who the AI Is*

Identity, personality, tone, and boundaries.

- Stored as flat files / system prompts
 - Always loaded
 - Does not grow dynamically
-

2. Directives — *How the AI Must Behave*

Rules, constraints, and operating procedures.

- Stored as flat configuration
 - Always loaded
 - Strictly size-limited
-

3. The Stream — *Conscious Thought*

The active conversation buffer.

- This is what the AI is currently thinking about
 - It has a deliberate upper bound
 - When full, it is consolidated and cleared
-

4. Experience Memory — *What Happened Before*

Important past events and facts the AI chose to remember.

- Stored in a database
 - Never loaded wholesale
 - Retrieved only when relevant
-

5. Domain Memory — *Job Data*

Structured data required for the AI's role (leads, inventory, state, etc.).

- Stored in a database
 - Never preloaded
 - Retrieved selectively
-

Who Decides What Gets Remembered?

The AI does — continuously, in real time.

On every message, the AI decides: - is this important? - should it be remembered? - where should it live?

Memory is an *active cognitive decision*, not a side effect of summarization.

Performance Reality: Decoupled Memory Decisions

The AI's authority over memory does **not** require blocking the user.

Implementations may: - offload memory classification to a smaller sidecar model - run memory assessment asynchronously - use optimistic generation with delayed persistence

The invariant remains:

The AI decides what matters — but the user never has to wait for bookkeeping.

The Sifter — From Backup to Analyst

The Sifter runs only when the Stream is consolidated.

It does **not** hunt for missed facts.

Instead, it looks for **patterns across time**: - sentiment shifts - recurring friction - task progression - behavioral trends

The Intake Valve captures facts. The Sifter synthesizes insight.

Continuity Without Amnesia

When the Stream fills up: - the conversation is summarized - the Stream is cleared - the summary becomes the **first entry** of the new Stream

To preserve precision, the system may also carry over: - a small rolling overlap of the most recent raw messages - or pinned task-specific instructions

This prevents mid-task detail loss while keeping context small.

PART II — MACHINE-LEVEL SPECIFICATION

1. Storage Architecture

1.1 Prompt-Bound (Flat-Loaded)

- Core Memory
- Directives
- Stream (including post-flush summary and overlap)

Only these may exist in the LLM prompt.

1.2 Database-Backed (External)

- Experience Memory (unstructured)
- Domain Memory (structured)

Never preloaded. Searchable only.

2. Context Contract (Hard Rules)

Prompt may contain: 1. Core Memory 2. Directives 3. Stream 4. Retrieved snippets (if explicitly selected)

Prompt must never contain: - full Experience Memory - full Domain Memory

3. Memory Authority Model

3.1 Intake Valve (Primary Authority)

Runs on every message.

- classifies importance
- selects storage target

Execution may be async or sidecar-based.

3.2 Sifter (Secondary Authority)

Runs only during consolidation.

- analyzes full Stream snapshot
 - extracts synthesized patterns
 - does not duplicate intake work
-

4. Stream Monitoring & Context Control

Define: - `MODEL_MAX_CONTEXT` - `APP_CONTEXT_CAP`

Rule:

```
APP_CONTEXT_CAP ≤ MODEL_MAX_CONTEXT
```

Trigger:

```
IF stream_tokens > (APP_CONTEXT_CAP * 0.85)
THEN perform_memory_keep()
```

5. Memory Keep Procedure

Step 1 — Snapshot

Capture Stream.

Step 2 — Sift

Generate: - summary - synthesized patterns

Step 3 — Persist

- save patterns → Experience Memory
- update Domain Memory

Step 4 — Flush

Clear Stream.

Step 5 — Resume

Insert: - summary - optional rolling overlap

Stream resumes immediately.

6. Retrieval Protocol (Explicit)

Retrieval requires: 1. standalone semantic query generation 2. vector search 3. relevance re-ranking 4. discard low-confidence results

Better no memory than wrong memory.

7. Reference Pseudocode

```
function handleMessage(msg) {
    // Async or sidecar intake valve
    enqueueMemoryAssessment(msg);

    if (stream.tokens() > APP_CONTEXT_CAP * 0.85) {
        performMemoryKeep();
    }

    const query = ai.generateSearchQuery(msg);
    const retrieved = query ? retrieveAndRerank(query) : null;

    const prompt = {
        core: loadCore(),
        directives: loadDirectives(),
        stream: loadStream(),
        retrieved
    };

    const reply = llm.generate(prompt);
    stream.append(reply);
    return reply;
}

function performMemoryKeep() {
    const snapshot = loadStream();
    const analysis = llm.sift(snapshot);

    savePatterns(analysis.patterns);
```

```
    saveDomain(analysis.domain);

    clearStream();
    appendToStream(analysis.summary);
    appendOverlap(snapshot.tail);
}
```

8. Invariants

- Conscious context is capped
 - Memory authority is AI-driven
 - Retrieval is explicit and gated
 - Summaries are injected once
 - Overlap preserves precision
 - No database preload ever
-

FINAL NOTE

Memory Keep is not chat history management.

It is **intentional cognition under real-world constraints**.