DESIGN MEMORY THAT MAKES AI AGENTS RELIABLE

ENGINEERING AGENT MEMORY

BUILDING LONG-RUNNING, SCALABLE AI AGENTS THAT REMEMBER WHAT MATTERS

REMEMBER WHAT MATTERS

FORGET WHAT DOES NOT

RETRIEVE ACCURATELY UNDER PRESSURE

SCALE WITHOUT EXPLODING COSTS

MULTI-AGENT STATE SHARING

RELIABLE ACROSS DAYS OF CONTINUOUS USE

FROM FIRST PRINCIPLES TO PRODUCTION SYSTEMS

SQLITE
EMBEDDED.
EFFICIENT.
RELIABLE.

POSTGRESQL
POWERFUL.
ACID COMPLIANT.
BUILT TO SCALE.

VECTOR INDEXES
SEMANTIC SEARCH.
HIGH RECALL.
LOW LATENCY.

EDWIN HOLLISTER

PRACTICAL ARCHITECTURES FOR AUTONOMOUS AGENTS THAT RUN FOR THE LONG HAUL

Engineering Memory for AI Agents

From First Principles to Production Systems

Steve Publications

This book is available at
https://leanpub.com/engineeringmemoryforaiagents

This version was published on 2026-08-14

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2026 Steve Publications

Contents

From First Principles to Production Systems **1**

Introduction **2**
The Problem with "Just Use a Larger Context Window" 2
What Agent Memory Actually Is 3
How This Book Is Organized 3
Prerequisites 4

Chapter 1: Why Agents Need Memory **5**
The Context Window Is Not Memory 5
What Happens When Agents Forget 5
Defining Agent Memory vs Related Concepts 6
The Cost of Forgetting in Real Systems 7

Chapter 2: A Taxonomy of Agent Memory **9**
Working Memory and Short-Term State 9
Episodic Memory for Interaction Histories 10
Semantic Memory for Facts and Knowledge 11
Procedural Memory for Skills and Patterns 12
Entity, User, Task and Environmental Memory 13
Shared Multi-Agent Memory 15

Chapter 3: The Memory Lifecycle from Capture to Retrieval **17**
Observing and Capturing Candidate Memories 17
Deciding What Is Worth Remembering 17
Extracting Facts, Entities and Relations 17
Structuring and Enriching Raw Observations 17
From Capture to Storage: The Memory Pipeline 17

Chapter 4: Representing Memories: Schemas and Data Structures . . . **18**
Designing a Memory Schema 18

Generating Embeddings for Semantic Search 18
Importance, Confidence and Decay Scores 18
Deduplication and Consolidation Strategies 18
Temporal Versioning and History Tracking 18
Forgetting Policies and Expiration 18

Chapter 5: Storage Foundations from Files to Databases 20
In-Memory Structures and File-Based Storage 20
SQLite with Full-Text Search (FTS5) 20
PostgreSQL for Structured Memory at Scale 20
Vector Indexes and Approximate Nearest Neighbor 20
Document Stores, Key-Value Stores, and Object Storage 20
Graph Databases for Relational Memory 20

Chapter 6: Retrieval Systems for Finding What Matters 22
Lexical Retrieval with BM25 22
Vector Similarity Search Internals 22
Hybrid Retrieval: Combining Lexical and Semantic 22
Metadata Filtering and Constrained Search 22
Reciprocal Rank Fusion and Multi-Signal Ranking 22
Rerankers and Cross-Encoders 22
Temporal and Graph-Based Retrieval 23

Chapter 7: Context Engineering for Injecting Memories into Prompts . 24
Token Budgets and Context Constraints 24
Relevance vs Recency in Memory Selection 24
Dynamic Context Assembly Strategies 24
Hierarchical and Recursive Summarization 24
Memory Scopes, Namespaces, and Isolation 24
Prompt Construction with Retrieved Memories 24

Chapter 8: Memory Architectures for Agent Types 26
Memory for Tool-Using Agents 26
Coding Agents and Repository-Aware Memory 26
Research Agents and Long-Horizon Reasoning 26
Personal Assistants and User-Centric Memory 26
Multi-Agent Systems and Shared Memory 26
Autonomous Workflows, Plans, and Checkpoints 26

Chapter 9: Frameworks, Integrations, and Interoperability 28

Vendor-Independent Memory Architecture 28
Integrating with Model APIs and Tool Calling 28
The Model Context Protocol (MCP) for Memory Tools 28
Plugging into Agent Frameworks Without Lock-In 28
Designing a Memory Service API 28

Chapter 10: Correctness, Reliability, and Consistency 29
Stale Memories and Knowledge Drift 29
Contradictions and Conflicting Information 29
Provenance, Attribution, and Trust 29
Deterministic vs Probabilistic Retrieval Guarantees 29
Concurrency, Race Conditions, and Consistency 29
Backups, Migrations, and Disaster Recovery 29

Chapter 11: Security, Privacy, and Access Control 31
Access Control and Tenant Isolation 31
Encryption at Rest and in Transit 31
Handling Secrets and Sensitive Data 31
Prompt Injection Through Stored Memories 31
Memory Poisoning and Malicious Content 31
Deletion, Retention, and Auditability 31

Chapter 12: Evaluating Agent Memory 33
Retrieval Metrics: Precision, Recall, and Ranking 33
Task-Success Measurement with Memory 33
Long-Horizon Benchmarks and Stability Tests 33
Cost, Latency, and Efficiency Metrics 33
Ablation Testing and Causal Attribution 33
Building Repeatable Evaluation Pipelines 33

Chapter 13: Observability, Debugging, and Diagnostics 35
Memory Traces and Retrieval Logs 35
Ranking Explanations and Decision Auditing 35
Token Accounting and Context Visibility 35
Interactive Memory Inspection Tools 35
Diagnosing Common Failure Modes 35

Chapter 14: Performance, Cost, and Scale Engineering 36
Embedding Costs and Optimization Strategies 36
Query Latency and Indexing Performance 36

Storage Growth Modeling and Cost Projections 36
Caching, Batching, and Throughput 36
Partitioning, Sharding, and Replication 36
Token Cost Engineering 36

Chapter 15: Building a Production Memory System via a Staged Roadmap 38
Stage 1: Minimal Memory with Conversation History 38
Stage 2: Persistent Structured Memory in SQLite 38
Stage 3: Lexical and Semantic Retrieval with Hybrid Ranking 38
Stage 4: Consolidation, Forgetting, and Temporal Reasoning 38
Stage 5: Provenance, Evaluation, Observability, and Security 38
Stage 6: Scalable Multi-Agent Production Architecture 39

Conclusion: The Engineering Discipline of Agent Memory 40
What We Know 40
What Remains Uncertain 40
When Simple Is Better 40
The Path Forward 40

References 41

From First Principles to Production Systems

Most AI agents are built without a deliberate memory architecture. They accumulate conversation history until context windows overflow, retrieve documents with brittle vector searches and fail in unpredictable ways when information changes or contradicts itself. This book treats agent memory as an engineering discipline rather than a feature. You will learn how to design systems that remember what matters, forget what does not, retrieve accurately under pressure, scale without exploding costs and operate reliably across days of continuous use with multiple agents sharing state. The material progresses from first principles through working implementations in SQLite, PostgreSQL and vector indexes to production architectures for long-running autonomous agents. No single model, vendor or framework is assumed correct; the emphasis is on understanding mechanisms deeply enough to make your own trade-offs with confidence.

Introduction

An AI agent without memory is a person who forgets everything between sentences. It can be clever in the moment but cannot build relationships, maintain projects, learn from mistakes or accumulate expertise. The promise of agentic systems depends entirely on their ability to remember.

Yet most teams approach memory as an afterthought. They start by stuffing conversation history into prompts until latency spikes and costs balloon. They add a vector database because tutorials say so, without understanding why BM25 full-text search might be better for their use case. They watch agents repeat themselves, contradict earlier statements or hallucinate facts they never learned, then blame the model rather than the architecture.

This book exists to fix that gap. It treats agent memory as a first-class engineering problem with its own requirements, trade-offs and failure modes. By the end, you will be able to architect memory systems from scratch: decide what to store and in what form, design retrieval pipelines that actually find relevant information, manage context windows so useful memories are not drowned out by noise, prevent security vulnerabilities introduced by stored data, measure whether memory improves your agent rather than merely adding complexity and scale the whole thing without burning through budget.

The Problem with "Just Use a Larger Context Window"

Large language models have grown dramatically in their context window sizes. Models that once handled 4,096 tokens now support 128,000 or more. This creates an illusion that memory is solved: just keep everything in the prompt and let the model figure it out.

The problem is structural. A context window is not memory; it is a stateless scratchpad. Every API call starts at step zero. The model does not retain anything between invocations. To simulate continuity, you must resend the entire history every time. This means every LLM call reprocesses all tokens in the window, so a 50,000-token context costs more than a 10,000-token context even if only 500 tokens changed since the last turn [1]. Costs grow

roughly linearly with history length, and attention mechanisms degrade on information buried in the middle of long contexts [2].

Even worse, naive history accumulation causes what engineers call "context dilution": as irrelevant or low-value content fills the window, it competes with genuinely important information for the model's attention. The agent starts missing details that were technically present but drowned out by noise. Simply paying for a larger window does not fix this; it postpones it while increasing cost.

What Agent Memory Actually Is

Agent memory is external storage that persists across sessions and survives context window resets, combined with deliberate retrieval mechanisms that inject relevant information into prompts when needed. It is not the same as:

- **The context window**, which is temporary working space cleared between calls.
- **Conversation history**, which is raw interaction logs that may or may not be useful to remember.
- **Retrieval-augmented generation (RAG)**, which fetches documents for answering questions but does not necessarily track what the agent learned from its own actions.
- **Model training**, which encodes knowledge into weights permanently and cannot be updated incrementally in most hosted API setups.

Memory sits between these concepts. It captures observations during interactions, decides which ones are worth keeping long-term, structures them for efficient retrieval and feeds them back into the agent's context when they matter.

How This Book Is Organized

The book builds progressively from foundation to mastery. The first section establishes why agents need memory, distinguishes memory from related concepts and provides a taxonomy for classifying different memory types. The second section covers the complete lifecycle: how memories are captured,

structured, stored, retrieved, maintained and eventually forgotten. The third section dives into concrete storage technologies and retrieval algorithms with working implementations. The fourth section addresses context engineering: how to assemble retrieved memories into prompts without overwhelming the model. The fifth section covers specialized architectures for different agent types and multi-agent systems. The final sections treat correctness, security, evaluation, observability, performance and cost as first-class concerns rather than afterthoughts.

Each chapter includes working code examples, schemas and architectural diagrams where they clarify the material. The book culminates in a staged roadmap showing how to evolve from a minimal memory-enabled agent into a scalable production architecture supporting long-running multi-agent workloads. Throughout, the emphasis is on understanding why each design choice works, what can go wrong, how to measure outcomes and when a simpler approach is better than a sophisticated one.

Prerequisites

You should be comfortable with basic programming concepts and have encountered LLM APIs. No prior expertise in vector databases, agent frameworks or information retrieval is assumed. The book introduces necessary concepts as it goes and explains jargon rather than glossing over it.

Chapter 1: Why Agents Need Memory

The Context Window Is Not Memory

When you send a request to an LLM API, the model receives your prompt, generates a response and then forgets everything. There is no internal state carried forward. If you want the model to remember what happened in previous turns, you must include that history in every subsequent prompt. This requirement reveals the fundamental limitation: the context window is temporary working space, not persistent storage.

Consider an agent tasked with helping a developer debug a distributed system over several days. On day one, the agent learns about the architecture, identifies a race condition and suggests a fix. On day two, the user reports that the fix introduced a deadlock under certain conditions. Without memory, the agent has no way to recall what it suggested yesterday unless the entire conversation history is resent in full. Even then, as days pass and the conversation grows to thousands of messages, the context window becomes expensive to maintain and unreliable for recalling specific details mentioned weeks ago.

The problem compounds for autonomous agents that operate over extended periods. An agent managing deployment pipelines, monitoring systems or conducting research may execute hundreds or thousands of steps across multiple sessions. Each step produces observations, decisions and artifacts that future steps depend on. If all this information must live in the context window, costs become unsustainable and reliability collapses.

What Happens When Agents Forget

Memory failures manifest in predictable patterns that degrade user experience and trust:

Repetition loops. An agent without memory of what it has already suggested will offer the same advice repeatedly. A customer service bot might ask for an order number five times because it cannot recall receiving it three turns ago. This is not a model failure; it is an architectural one.

Inconsistent behavior. Without access to earlier decisions, agents contradict themselves. An agent that recommended a particular library on Monday might suggest a different one on Friday with no explanation of why the recommendation changed, even though nothing in the environment actually did.

Lost context in long horizons: research agents, coding assistants and autonomous workflows operate across extended timelines. Without persistent memory, they cannot build on earlier work, verify consistency with past decisions or avoid re-exploring paths already proven unproductive.

Failure to learn from feedback. Users provide corrections and preferences constantly: "I prefer concise answers," "Don't use that framework," "Remember I'm working in Python 3.11." Without a mechanism to capture and retrieve these signals, the agent ignores them after they leave the context window, appearing stubborn or inattentive.

Inability to coordinate across agents: multi-agent systems depend on shared memory for coordination. If one agent discovers that a particular API endpoint is down, other agents should know. Without shared state, each agent independently encounters the same failure, retries the same broken operation and generates redundant error logs.

These failures are not theoretical. Production teams report that 79 percent of multi-agent system failures stem from coordination and memory issues rather than technical bugs [4]. The gap between what users expect from persistent agents and what naive implementations deliver is substantial.

Defining Agent Memory vs Related Concepts

To design memory systems effectively, you must distinguish them from related mechanisms that serve different purposes:

Context window. The finite token budget available during a single inference call. Everything the model can see right now: system instructions, recent conversation history, retrieved documents, tool definitions. It is fast and directly accessible but temporary and expensive to maintain at scale. Think of it as RAM: valuable working space that must be managed carefully.

Conversation history: the raw log of interactions between user and agent. Valuable for audit trails and debugging but not all of it should be treated as memory.

Retrieval-augmented generation (RAG): a pattern where documents are fetched from an external store and injected into prompts to ground the model's responses. RAG solves a specific problem: giving agents access to information not in their training data. But RAG is not agent memory per se. It fetches static documents; it does not track what the agent learned from its own actions, decisions or interactions over time.

Knowledge bases: curated collections of facts, policies and procedures that agents can query. Like RAG sources, they are external to the agent's experience. Memory differs in that it is generated by the agent itself through observation and reflection, not pre-populated by humans.

Model training. Encoding knowledge directly into model weights via fine-tuning. This creates permanent memory but lacks flexibility: you cannot incrementally update a hosted API model's weights based on today's interactions. Training is appropriate for stable, generalizable knowledge; memory is appropriate for dynamic, agent-specific information.

Caches: temporary storage that speeds up repeated operations. Caches improve performance but do not constitute memory in the cognitive sense. An LLM cache might store embeddings to avoid recomputation but it does not help the agent remember what it decided last week.

Agent memory combines elements of all these: it persists like a database, retrieves selectively like RAG, is generated dynamically unlike static knowledge bases, updates incrementally unlike training and serves cognitive functions unlike caches.

The Cost of Forgetting in Real Systems

The consequences of inadequate memory extend beyond user frustration to measurable economic impact:

Token waste. Agents that accumulate full conversation history in context windows pay for every token on every turn. A production session with poor context management can cost $10 to $100 or more, with 60 to 70 percent of spend attributable to unnecessary context [5]. Retrieval-based memory reduces this by 60 to 80 percent by loading only relevant information per turn [5].

Redundant work: without memory of past actions, agents repeat searches, re-read documents already analyzed and re-derive conclusions they estab-

lished earlier. This wastes both compute and time in latency-sensitive applications.

Error propagation. An agent that forgets it already encountered a particular error will retry the same failing operation indefinitely or escalate issues unnecessarily. Memory of past failures enables smarter recovery strategies.

Lost expertise. Over weeks of use, an agent can learn domain-specific patterns: which APIs are reliable, which code patterns cause bugs, which users prefer detailed explanations versus summaries. Without memory to preserve these insights, each new session discards accumulated expertise.

The economic argument for memory is straightforward: agents that remember correctly do more useful work per token spent. They reduce redundant operations, maintain context across sessions without loading everything every time and improve task success rates by building on earlier decisions rather than starting from scratch.

Memory is not a luxury feature for AI agents. It is the difference between a clever chatbot that entertains in isolated conversations and a reliable system that accumulates value over time. The rest of this book explains how to build it correctly.

Chapter 2: A Taxonomy of Agent Memory

Working Memory and Short-Term State

Working memory holds information actively being manipulated during the current task. In human cognition, it is limited in capacity; in AI agents, it maps directly to the context window. Everything the model can see during a single inference call lives in working memory: system instructions, recent turns of conversation, tool definitions, retrieved documents, intermediate reasoning steps.

Working memory has three defining properties:

Direct accessibility. The model reads and reasons over working memory content without any retrieval step. There is no latency cost for accessing information already in the prompt; the trade-off is that every token in the window costs something on every call.

Volatility: working memory disappears between calls unless explicitly preserved. If you do not include conversation history in the next prompt, it is gone. This volatility requires deliberate management: you must decide what to keep, what to compress and what to discard as tasks evolve.

Capacity constraints: even models with large context windows have practical limits. A 128,000-token window sounds generous until an autonomous agent executing a multi-hour research task accumulates tool outputs, file contents, API responses and intermediate reasoning that easily exceed it. Moreover, models perform worse on information buried deep in long contexts due to attention degradation [6].

In practice, working memory management involves:

- Maintaining a sliding window of recent conversation turns for coherence
- Including only the retrieved documents relevant to the current step

- Using scratchpads or temporary variables within prompts to hold intermediate results
- Periodically compressing older content into summaries that preserve decision-relevant information while freeing space

Working memory is the foundation upon which all other memory types operate. Long-term memories are useful only if they can be retrieved and loaded into working memory when needed.

Episodic Memory for Interaction Histories

Episodic memory stores specific events and interactions: what happened, when, with whom and in what context. It corresponds to the human ability to recall personal experiences. For AI agents, episodic memory captures discrete moments from their operational history that may be relevant to future decisions.

Examples of episodic memories:

- "On March 15 at 2:30 PM, the user reported a timeout error when calling the payments API with request IDs starting with 'tx-'."
- "During the code review on April 2, the agent discovered that module X had circular dependencies and recommended refactoring."
- "The user corrected the agent's assumption about timezone handling: they operate in UTC, not local time."

Episodic memories differ from raw conversation logs in two ways. First, they are curated: not every turn of conversation is stored as an episode; only interactions containing meaningful events, decisions, corrections or observations. Second, they are structured with metadata: timestamps, participants, topics and confidence levels make them searchable and filterable.

Episodic memory serves several purposes:

Continuity across sessions. When a user returns after days or weeks, the agent can retrieve relevant past interactions rather than asking them to repeat context. "Last time we discussed your deployment pipeline, you mentioned that staging uses Kubernetes but production still runs on EC2. Has that changed?" This kind of continuity builds trust and efficiency.

Learning from outcomes. If an agent's recommendation led to a successful deployment or a production incident, recording that episode allows future decisions to incorporate historical outcomes. "We tried approach A three times; it worked twice and failed once under high load. Approach B has not been tested yet."

Auditability and debugging. Episodic logs provide traceable records of what the agent knew when it made decisions, which is essential for diagnosing failures and meeting compliance requirements.

Implementation patterns for episodic memory typically involve:

- Storing episodes as structured records with fields like timestamp, summary, participants, topics and outcome
- Indexing them by time range, topic keywords and entities mentioned
- Retrieving them using temporal filters ("events in the last 7 days") combined with semantic or lexical search on summaries
- Summarizing long sequences of related episodes into higher-level narratives to control storage growth

Episodic memory is particularly important for personal assistants, customer service agents and any system where relationship continuity matters.

Semantic Memory for Facts and Knowledge

Semantic memory stores general facts, concepts and world knowledge independent of when or how they were learned. It answers questions like "What is the user's preferred programming language?" or "What are the company's deployment policies?" Unlike episodic memory, semantic memories do not carry rich contextual detail about the specific interaction where they originated; they are distilled into reusable facts.

Examples:

- "User prefers Python over JavaScript for data processing tasks."
- "The staging environment uses PostgreSQL 15 with connection pooling via PgBouncer."
- "API rate limits are 100 requests per minute for standard endpoints, 1,000 for premium."

Semantic memory is the closest analog to a knowledge base but with a critical distinction: it is generated and maintained by the agent through its own experience rather than pre-populated by humans. Over time, an agent accumulates semantic facts about its users, environment, domain conventions and operational procedures that no static documentation could anticipate.

Semantic memory supports:

Personalization: agents that remember user preferences, working styles and historical decisions can tailor their behavior without requiring explicit configuration each session.

Domain expertise: a coding agent that has worked on a particular codebase for months accumulates knowledge about its architecture patterns, known pain points and team conventions that make it more effective than a generic assistant.

Consistency. Semantic facts provide stable reference points that prevent agents from contradicting established information or reinventing decisions already made.

The engineering challenge with semantic memory is keeping it accurate as the world changes. Facts have lifespans: API endpoints move, policies update, preferences shift. A semantic memory system must support updates, track provenance so facts can be verified against their sources and allow expiration or correction when information becomes stale.

Semantic memory integrates naturally with RAG systems. Domain-specific knowledge from documentation can populate the initial semantic store, while agent-generated facts add personalized and operational knowledge on top. The retrieval mechanisms are similar; the difference lies in provenance and update semantics.

Procedural Memory for Skills and Patterns

Procedural memory encodes how to do things: workflows, strategies, tool usage patterns and problem-solving approaches that have proven effective. It corresponds to human skills like riding a bicycle or typing: once learned, they become automatic and are applied without deliberate reasoning each time.

For AI agents, procedural memory captures reusable knowledge about processes:

- "When the user asks for deployment assistance, first check the CI/CD pipeline status, then verify environment configurations, then review recent changes."
- "To debug timeout errors in this codebase, always check the database connection pool settings first because they are a known source of intermittent failures."
- "The testing framework requires tests to be run with the –isolated flag to avoid flaky behavior from shared state."

Procedural memories can be stored as:

Explicit workflows: structured sequences of steps that the agent follows for recurring task types. These might be encoded as checklists, decision trees or even executable scripts.

Patterns and heuristics. General rules of thumb derived from experience: "When X happens, try Y first because it works 80 percent of the time."

Tool usage knowledge: details about how to use specific APIs, libraries or systems effectively: required parameters, common pitfalls, performance characteristics.

Procedural memory is particularly valuable for agents that operate repeatedly in the same domain. A customer support agent learns which troubleshooting steps resolve which issues fastest. A research agent discovers which sources are most reliable for particular topics. A coding assistant internalizes team conventions and architectural patterns.

Implementing procedural memory often involves:

- Extracting successful strategies from episodic memories through reflection or summarization
- Storing procedures as structured templates that can be retrieved and instantiated for new tasks
- Maintaining confidence scores or success rates so the agent prefers proven approaches over untested ones
- Allowing procedures to evolve as new evidence suggests improvements

Procedural memory connects closely with experiential learning patterns like Reflexion, where agents analyze their past attempts and refine their strategies accordingly [7]. The key engineering insight is that procedural knowledge should be explicit and retrievable rather than implicit in prompt templates; this enables dynamic adaptation based on context.

Entity, User, Task and Environmental Memory

Beyond the cognitive science categories above, production systems benefit from additional memory types organized around operational concepts:

Entity memory. Tracks specific objects in the agent's domain: users, projects, files, services, APIs, organizations. Each entity accumulates attributes and relationships over time. Entity memory is often implemented as a knowledge graph where nodes represent entities and edges represent relationships. For example, an agent managing software projects might track that "Project Alpha uses React and PostgreSQL," "Developer Jane owns the authentication module," and "Service Beta depends on Service Gamma."

Entity memory enables precise lookups: "What services does Jane own?" or "Which projects use PostgreSQL?" It also supports relationship reasoning: if a database change affects PostgreSQL, the agent can identify all dependent projects. Entity recognition, or determining that "Jane Smith," "J. Smith," and "jsmith" refer to the same person, is a critical engineering challenge for entity memory systems.

User and profile memory: a specialized form of entity memory focused on individuals interacting with the agent. It stores preferences, communication styles, role information, historical interactions and context specific to each user. User memory must handle privacy carefully: some facts are sensitive, some users expect different retention policies and multi-tenant systems require strict isolation between user profiles.

Task memory. Tracks the state of ongoing work items: current status, completed steps, pending actions, blockers, decisions made, artifacts produced. Task memory is essential for agents that manage projects or execute multi-step workflows across sessions. It ensures that work can resume after interruptions without losing track of progress.

Environmental state. Captures facts about external systems the agent interacts with: API availability, configuration settings, resource utilization, recent incidents, version information. Environmental state changes independently of the agent's actions and must be refreshed periodically to remain accurate. Unlike other memory types, environmental state is often ephemeral: yesterday's deployment status may be irrelevant today.

These operational memory types complement the cognitive taxonomy. A production system typically needs all of them: episodic memories for inter-

action history, semantic memories for facts, procedural memories for skills, entity memories for domain objects and task/environmental memories for current operations.

Shared Multi-Agent Memory

When multiple agents collaborate, they need shared memory to coordinate effectively. Shared multi-agent memory provides a common ground where agents can publish information, discover what others know, avoid redundant work and maintain consistent understanding of the environment.

Shared memory introduces architectural choices:

Centralized store. All agents read from and write to a single shared database or knowledge graph. This approach ensures consistency and makes information universally accessible but requires careful access control to prevent agents from reading data they should not see or overwriting each other's work.

Federated stores. Each agent maintains its own memory, with selective sharing through explicit communication channels. This approach provides isolation and autonomy but risks duplication and inconsistency if agents do not synchronize properly.

Hierarchical tiers: information is organized into levels: global memory accessible to all agents, group or role-scoped memory for subsets of agents and private memory for individual agents. This pattern balances coordination needs with privacy and noise reduction requirements. Industry practice is converging toward hierarchical designs as the default for production multi-agent systems [8].

Shared memory must address several challenges:

Concurrency: multiple agents may read and write simultaneously, creating race conditions if not handled properly. Transactional updates, optimistic locking or conflict-resolution strategies are necessary to maintain consistency.

Noise and relevance: as more agents contribute to shared memory, it grows and becomes harder for any individual agent to find relevant information. Filtering mechanisms based on role, task or topic help agents focus on what matters to them.

Security and isolation. Not all agents should see all data. Access control policies must define which agents can read or write which memories, particularly in multi-tenant environments where different organizations' agents share infrastructure.

Provenance tracking. When multiple agents contribute information, it becomes important to know who said what and when. Provenance metadata supports accountability and helps resolve conflicts when agents disagree about facts.

Shared memory is the backbone of sophisticated multi-agent architectures like research teams where one agent gathers information while another analyzes it, or deployment pipelines where separate agents manage testing, approval and execution. Without well-designed shared memory, such systems devolve into collections of isolated agents that cannot effectively collaborate.

Chapter 3: The Memory Lifecycle from Capture to Retrieval

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Observing and Capturing Candidate Memories

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Deciding What Is Worth Remembering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Extracting Facts, Entities and Relations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Structuring and Enriching Raw Observations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

From Capture to Storage: The Memory Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Chapter 4: Representing Memories: Schemas and Data Structures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Designing a Memory Schema

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Generating Embeddings for Semantic Search

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Importance, Confidence and Decay Scores

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Deduplication and Consolidation Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Temporal Versioning and History Tracking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Forgetting Policies and Expiration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Chapter 5: Storage Foundations from Files to Databases

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

In-Memory Structures and File-Based Storage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

SQLite with Full-Text Search (FTS5)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

PostgreSQL for Structured Memory at Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Vector Indexes and Approximate Nearest Neighbor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Document Stores, Key-Value Stores, and Object Storage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Graph Databases for Relational Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Chapter 6: Retrieval Systems for Finding What Matters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Lexical Retrieval with BM25

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Vector Similarity Search Internals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Hybrid Retrieval: Combining Lexical and Semantic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Metadata Filtering and Constrained Search

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Reciprocal Rank Fusion and Multi-Signal Ranking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Rerankers and Cross-Encoders

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Temporal and Graph-Based Retrieval

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Chapter 7: Context Engineering for Injecting Memories into Prompts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Token Budgets and Context Constraints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Relevance vs Recency in Memory Selection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Dynamic Context Assembly Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Hierarchical and Recursive Summarization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Memory Scopes, Namespaces, and Isolation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Prompt Construction with Retrieved Memories

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Chapter 8: Memory Architectures for Agent Types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Memory for Tool-Using Agents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Coding Agents and Repository-Aware Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Research Agents and Long-Horizon Reasoning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Personal Assistants and User-Centric Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Multi-Agent Systems and Shared Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Autonomous Workflows, Plans, and Checkpoints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Chapter 9: Frameworks, Integrations, and Interoperability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Vendor-Independent Memory Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Integrating with Model APIs and Tool Calling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

The Model Context Protocol (MCP) for Memory Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Plugging into Agent Frameworks Without Lock-In

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Designing a Memory Service API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Chapter 10: Correctness, Reliability, and Consistency

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Stale Memories and Knowledge Drift

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Contradictions and Conflicting Information

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Provenance, Attribution, and Trust

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Deterministic vs Probabilistic Retrieval Guarantees

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Concurrency, Race Conditions, and Consistency

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Backups, Migrations, and Disaster Recovery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Chapter 11: Security, Privacy, and Access Control

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Access Control and Tenant Isolation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Encryption at Rest and in Transit

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Handling Secrets and Sensitive Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Prompt Injection Through Stored Memories

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Memory Poisoning and Malicious Content

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Deletion, Retention, and Auditability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Chapter 12: Evaluating Agent Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Retrieval Metrics: Precision, Recall, and Ranking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Task-Success Measurement with Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Long-Horizon Benchmarks and Stability Tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Cost, Latency, and Efficiency Metrics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Ablation Testing and Causal Attribution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Building Repeatable Evaluation Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Chapter 13: Observability, Debugging, and Diagnostics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Memory Traces and Retrieval Logs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Ranking Explanations and Decision Auditing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Token Accounting and Context Visibility

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Interactive Memory Inspection Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Diagnosing Common Failure Modes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Chapter 14: Performance, Cost, and Scale Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Embedding Costs and Optimization Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Query Latency and Indexing Performance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Storage Growth Modeling and Cost Projections

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Caching, Batching, and Throughput

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Partitioning, Sharding, and Replication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Token Cost Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Chapter 15: Building a Production Memory System via a Staged Roadmap

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Stage 1: Minimal Memory with Conversation History

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Stage 2: Persistent Structured Memory in SQLite

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Stage 3: Lexical and Semantic Retrieval with Hybrid Ranking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Stage 4: Consolidation, Forgetting, and Temporal Reasoning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Stage 5: Provenance, Evaluation, Observability, and Security

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Stage 6: Scalable Multi-Agent Production Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

Conclusion: The Engineering Discipline of Agent Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

What We Know

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

What Remains Uncertain

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

When Simple Is Better

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

The Path Forward

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/engineeringmemoryforaiagents.