<u>Whitepaper "Context Engineering: Sessions & Memory" main points</u>

Introduction

- **Context Engineering** is the process of dynamically assembling and managing information in an LLM's context window to create stateful, intelligent agents.
- The two core components are **Sessions** (the container for the current conversation's history and working memory) and **Memory** (the mechanism for long-term persistence and personalization across multiple sessions).

Context Engineering

- LLMs are inherently stateless; Context Engineering makes them stateful by constructing a dynamic, state-aware prompt for every turn.
- It is an evolution of Prompt Engineering, dealing with the entire payload (system instructions, tools, history, memory, RAG data).
- The goal is to provide **only the most relevant information** to manage cost, latency, and "context rot."
- The process follows a continuous cycle: **Fetch Context, Prepare Context** (blocking), **Invoke LLM and Tools**, and **Upload Context** (often asynchronous).

Sessions

- A session is a self-contained, chronological record of a single, continuous conversation tied to a specific user.
- It includes
    - **Events** (user input, agent response, tool calls/output) and
    - **State** (structured working memory, like a scratchpad).
- Production systems require persistent storage, demanding **Security and Privacy** (strict isolation, PII redaction), **Data Integrity** (deterministic event order, TTL policies), and **Performance** (fast read/write times).
- **Variance across frameworks** exists (e.g., ADK uses explicit Session and Event objects; LangGraph's mutable state acts as the session).
- **Multi-agent systems** use either a **shared, unified history** (for tightly coupled tasks) or **separate, individual histories** (communicating via messages/tools).

Managing Long Context Conversation: Tradeoffs and Optimizations

- Long conversation history increases **Cost, Latency, and Context Rot**.
- **Compaction strategies** (e.g., **Keep the last N turns**, **Token-Based Truncation**, and **Recursive Summarization**) are used to shrink history while preserving context.
- Expensive compaction operations (like summarization) should be run **asynchronously** in the background and their results persisted.

Memory

- **Memory** is a snapshot of **extracted, meaningful information** persisted across sessions for continuous, personalized experience.
- It is crucial for **Personalization, Context Window Management** (compaction), **Data Mining**, and **Agent Self-Improvement** (procedural memory).
- A **Memory Manager** is a specialized, decoupled service that handles the memory lifecycle (Extraction, Consolidation, Storage, Retrieval) and often uses

framework-agnostic data structures to enable multi-agent interoperability.
- Memory is complementary to **RAG**: RAG provides static, global facts (expert on the world), while Memory provides dynamic, user-specific context (expert on the user).

Types and Organization of Memory
- Memories are categorized by knowledge type:
  - **Declarative memory** ("knowing what"): Facts, figures, and events (e.g., semantic or episodic).
  - **Procedural memory** ("knowing how"): Skills and workflows (e.g., sequence of tool calls).
- **Organization patterns** include **Collections**, **Structured User Profile**, and **Rolling Summary** (used for session compaction).
- **Storage Architectures** rely on **Vector Databases** (for semantic similarity) and/or **Knowledge Graphs** (for relational structure).
- **Creation mechanisms** are **Explicit** (user command) or **Implicit** (agent inference), and managed **Internally** (by the agent framework) or **Externally** (by a dedicated service).
- **Memory Scope** can be **User-Level** (persists across sessions), **Session-Level** (for compaction), or **Application-Level/Global** (shared context).
- **Multimodal Memory** most commonly handles non-textual sources (images, audio) but stores the resulting memory as **textual insight**.

Memory Generation: Extraction and Consolidation
- Memory Generation is an **LLM-driven ETL pipeline** (Extract, Transform, Load).
- **Extraction** is a targeted filtering process to find information "meaningful" to the agent's purpose, often guided by schemas or few-shot examples.
- **Consolidation** is the most sophisticated stage, where an LLM compares new insights with existing memories to handle **Duplication, Conflicts, and Evolution**, deciding to **UPDATE, CREATE, or DELETE/INVALIDATE** memories.
- **Memory Provenance** (lineage) must be tracked to assess the memory's trustworthiness and inform conflict resolution during consolidation.
- Memory generation should be an **asynchronous background process** (non-blocking) to ensure a fast user experience.
- The agent must choose a trigger mechanism (e.g., **Session Completion**, **Turn Cadence**, or **Memory-as-a-Tool**) to initiate generation.

Memory Retrieval and Inference
- **Intelligent Retrieval** scores memories across **Relevance** (semantic similarity), **Recency**, and **Importance** to find the best fit.
- Retrieval can be **Proactive** (at the start of every turn) or **Reactive** (**Memory-as-a-Tool**), where the agent decides when to query its memory.
- Memories are strategically placed in the context window by either injecting them into the **System Instructions** (for stable context) or the **Conversation History** (as tool output or injected dialogue).

Testing, Evaluation, and Production

- **Evaluation** uses **Quality Metrics** (Precision, Recall, F1-Score) for memory generation and **Performance Metrics** (Recall@K, Latency) for retrieval.
- **Production considerations** demand **Decoupling** (memory processing as a separate, non-blocking service) and **Resilience** (handling race conditions, retries, and multi-region replication).
- **Privacy and security** are critical, requiring **Strict Data Isolation** (ACLs), **User Control** (opt-out), and **Sanitization** using safeguards like **Model Armor** to prevent memory poisoning.

Sources:

- Context Engineering: Sessions & Memory.pdf