Context Engineering & Multi-Agent Systems

What is Context Engineering?

Context engineering is how we give AI agents the right information at the right time so they can work effectively together. Unlike prompt engineering (which focuses on crafting instructions), context engineering manages the flow of memory, knowledge, and tools across multiple agents.

Think of it as designing the information architecture for collaborative AI systems.

Core Principles

1. Dynamic Context Assembly

Agents get context on-demand based on what they're doing:

- Task instructions and parameters
- Memory of past interactions
- Retrieved documents or external data
- Outputs from other agents

2. Memory Management

Short-term memory: Current conversation, active tasks, temporary data
Long-term memory: User preferences, historical patterns, learned knowledge
Memory isolation: Each agent maintains role-specific memory to prevent overload

3. Retrieval-Augmented Generation (RAG)

Agents query external sources (databases, documents, APIs) and inject relevant information into their context. This reduces hallucinations and enables real-time data access beyond training data.

4. Tool Integration

Agents can call external tools during decision-making:

- Document processors
- Calendar/scheduling APIs
- Database queries
- Third-party services

5. Context Prioritization

Smart filtering ensures agents stay within token limits while working with the most relevant information.

Multi-Agent System Design

Architecture Principles

Each agent has:

- **Defined responsibilities:** Specific tasks or domains
- **Shared context access:** Read/write to common memory
- **Independent decision-making:** Autonomy within their scope
- **Coordination protocols:** Rules for inter-agent communication

Context Flow Pattern

User Input

↓

Agent 1: Input Processing

↓ (updates shared memory)

Agent 2: Core Analysis

↓ (retrieves context + adds findings)

Agent 3: Task Coordination

↓ (orchestrates workflow)

Agent 4: Output Generation

↓

Final Output

Key Components:

- **Shared Memory Store:** Central repository for context
- **Retrieval Mechanisms:** Query past results or external knowledge
- **Role-Specific Memory:** Isolated memory per agent
- **Handoff Protocols:** Clear rules for passing control
- **Error Handling:** Failure management and help requests

Collaboration Patterns

- **Sequential:** Agents work in pipeline, building on previous outputs
- **Parallel:** Multiple agents work simultaneously, results merged later
- **Hierarchical:** Coordinator delegates to specialists and synthesizes outputs
- **Iterative:** Agents review and refine each other's work

Framework Analysis

I evaluated three open-source frameworks based on licensing, context management, multi-agent support, and ease of implementation.

Framework Comparison

Framework	License	Strengths	Weaknesses	Best For
CrewAI	MIT	• Role-based design • Built-in memory per agent • Easy tool integration • Clear responsibilities	• Learning curve • Smaller community	Well-defined agent roles with memory isolation
LangGraph	MIT	• Visual workflow mapping • Flexible branching • Strong state management	• Requires upfront graph definition • More setup overhead	Complex pipelines with dependencies and conditional logic
AutoGen	Apache 2.0	• Natural for conversations • Automatic context sharing • Human-in-the-loop	• Less structured • Can be verbose	Conversational AI and collaborative sessions

Recommendation: CrewAI

Why CrewAI:

- Role-based design aligns with specialized agent requirements
- Built-in memory management reduces complexity
- MIT license provides full freedom to use and modify
- Easy to reason about agent responsibilities and boundaries

Alternatives:

- **LangGraph:** Consider if you need complex workflow dependencies or visual management
- **AutoGen:** Consider if the use case shifts toward conversational AI or requires frequent human interaction

Next Steps

I'm ready to begin implementation. Specific details will be determined based on:

- Your project requirements
- Specific use cases or problem statements
- Feedback on this research

This architecture is flexible and can adapt to various use cases—from document processing pipelines to complex orchestration workflows.

Prepared by: Pranav Pillai
Contact: ppillai294@gmail.com