

# Option 3: Context/Memory Management for Agents

## Phase 1B — Phase 1 Report

---

**Team Member:** Ziyi Deng, Zheqi Liu, Zhengang Cheng, Tianzuo Liu, Hengjia Yu

**Repo:** <https://github.com/siriuxyu/CSE291-A/tree/main>

The content of Phase1A is in the [Appendix](#). More details are available in repo's **README**.

### 1) Agent Selection & Justification

#### Selected Agent.

We implemented an agent built with LangGraph + Anthropic Claude and exposed it via a FastAPI server. The agent answers a user's question and call relative tools to finish certain tasks. The system supports thread-scoped conversation state and optional Redis-backed persistence.

#### Why this agent?

- Clear context surface: The pipeline explicitly produces intermediate outputs (e.g., the raw answer before translation), which creates concrete, inspectable 'context artifacts' for memory analysis (e.g., what to keep vs. compress vs. discard per turn).
- Natural multi-turn behavior: The thread model already organizes state by session; this makes it straightforward to study within-session context carryover and to extend to cross-session memory in Phase 2.
- Operational readiness: The codebase is containerized with a minimal Dockerfile and pinned dependencies (FastAPI/Uvicorn, LangGraph, Anthropic, Redis), which enables us to deploy reproducibly and evaluate quickly on cloud instances.
- Our agent's explicit state shape and thread-scoped seasoning give a clean baseline to later add context selection, summarization, and persistent memories required by the project option 3 illustration.

### 2) Context Understanding & Analysis

#### 2.1 What's the "context" for this agent?

We define "context" as any information the agent will refer to when generating an answer and translation for the current turn. Concretely, today's system exposes can capture the following:

## 1. Current turn data

- User inputs: input\_question, user\_id.
- Final output: the response from agent.
- These are first-class fields in the agent state graph and are retrievable via the server's state endpoints.

## 2. Session (thread) history

- - The agent associates turn with a user\_id, enabling multi-turn conversations whose state can be inspected or enumerated. This affords context such as previous questions, prior answers, prior target languages, and intermediate artifacts.

## 2.2 What context is more important vs. less important?

Higher-importance context (should be retained and prioritized):

- Latest user goal & constraints: the question and the requested target language; they directly determine both content and translation.
- Most recent factual answer (pre-translation): It's the factual nucleus we may want to reference or refine in follow-ups.

Lower-importance context (can be compressed or omitted):

- Verbatim long histories beyond the last few turns when there is no explicit user reference to earlier content; concise summaries usually suffice.
- Server/runtime logs and container metadata. Even though it is critical for operations but rarely needed as model input.

## 2.3 Any context sharing across sessions? Across users?

Across sessions (same user): The agent's thread model scopes state to a thread\_id. With in-memory storage, state does not persist across server restarts; with Redis, state can be made durable across sessions and time. Our Phase-1 baseline keeps a per-thread state.

Across users: By default, there is no shared context across users. Each user should operate in an isolated namespace. Any future cross-user sharing must be explicit and anonymized and is out of scope for the Phase-1 baseline.

# Appendix

## Deployment Architecture

### Cloud Environment

#### Server Description:

The server was deployed at Amazon Web Services platform. The interface can be accessed through: <http://44.247.56.190:8000/docs>

#### Screenshot:

Invoke & Execute interface:

The screenshot displays a web interface for invoking an API. At the top, a green bar shows the method 'POST' and the endpoint '/invoke'. Below this, a 'Parameters' section is empty. A 'Request body' section is marked as 'required' and has a dropdown menu set to 'application/json'. The request body is a JSON object: 

```
{  "input_question": "Hi! My name is Henry.",  "thread_id": "user_01",  "language": "English"}
```

. At the bottom, there are 'Execute' and 'Clear' buttons. Below the interface, a 'Responses' section shows a 'Curl' command and the 'Request URL'.

POST /invoke Invoke

Parameters Cancel Reset

No parameters

Request body required application/json

Edit Value Schema

```
{  "input_question": "Hi! My name is Henry.",  "thread_id": "user_01",  "language": "English"}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \  'http://44.247.56.190:8000/invoke' \  -H 'accept: application/json' \  -H 'Content-Type: application/json' \  -d '{  "input_question": "Hi! My name is Henry.",  "thread_id": "user_01",  "language": "English"  }'
```

Request URL

```
http://44.247.56.190:8000/invoke
```

Server response

Code Details

Response Interface:

Request URL  
`http://44.247.56.190:8000/Invoke`

Server response

| Code | Details  |
|------|--|
| 200  | <p>Response body</p> <pre>{   "answer": "Hi! My name is Claude. I aim to be helpful, honest, and direct. If I'm not sure about the answer to a question, I'll openly say so rather than trying to guess or make something up.\n\nHowever, I notice that while you started to ask a question, the actual question wasn't completed. Would you like to go ahead and ask your full question?" }</pre> <p>Response headers</p> <pre>content-length: 349 content-type: application/json date: Fri, 24 Oct 2025 22:15:53 GMT server: unicorn</pre> |

Responses

## Set up:

Please refer to the project's GitHub repository for the detailed setup process:

<https://github.com/siriuxyu/CSE291-A/blob/main/README.md>

## Management and Isolation

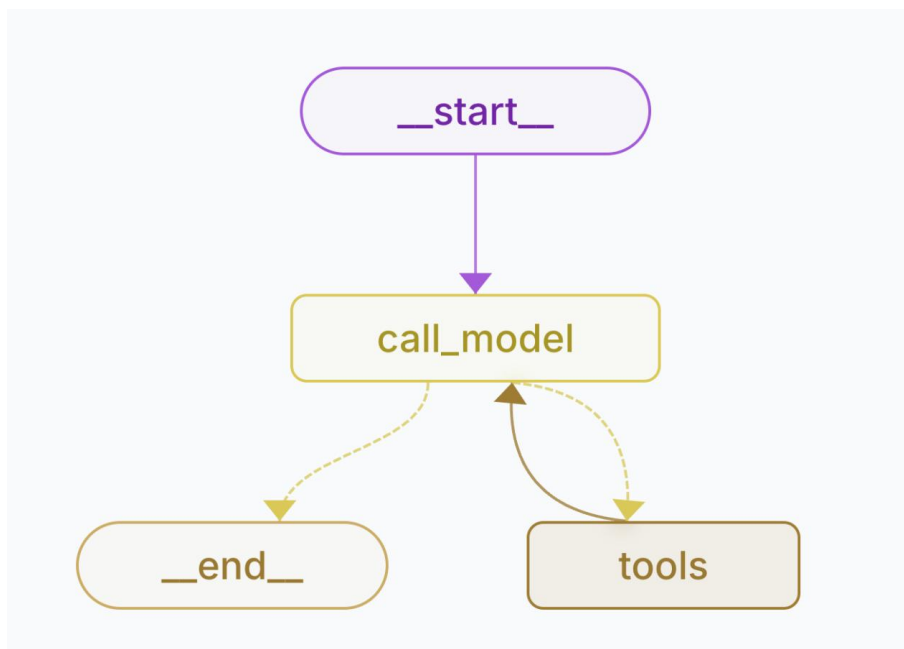
### Session Management:

Session Management is handled by LangGraph's core state machine (StateGraph). In our project, the State object (defined in state.py) acts as the "short-term memory" for a single conversation. It accumulates the entire dialogue process—including user inputs, the AI's reasoning (tool calls), and tool execution results—within the messages list.

### User Isolation:

User Isolation is realized through an input variable `user_id`. `user_id` enables the engine to access the memory or Redis server for different users and ensures that their sessions do not interfere with each other.

## Architecture



## Request Set Quality

### Testset

We build up a multi-tiered test suite for evaluating our agent. The benchmark currently comprises 15 test requests distributed across short, medium, and long-context scenarios, designed to systematically test an agent’s tool use, contextual reasoning, and memory retention.

The evaluation framework integrates five distinct tool types—Calculator, Translator, Web Reader, Weather Checker, and File System Search—to simulate realistic interactions involving everyday reasoning, multi-source grounding, and inter-session preference recall.

The design intentionally introduces diverse cognitive demands and context persistence requirements, exposing baseline weaknesses in context retention, tool chaining, and long-memory reasoning.

- Short test cases focus on precision and efficiency in direct tool invocation of single sessions. (e.g., “What is the square root of  $(\ln(18) ^ 3) / 3.14$ ?” “What is the weather in Tokyo today? Please translate the weather report to Japanese.”) These scenarios establish baseline performance on atomic and combined tool operations and response determinism.
- The medium request tests expand complexity to multi-turn, cross-session reasoning, and user preference besides the short cases. These tests probe contextual carryover, memory fidelity, and cross-domain grounding—the ability to retrieve and apply preferences established in previous sessions.
- The long request test is built up by the LoCoMo benchmark and a customized test set. LoCoMo evaluation benchmark consists of very long-term conversational data with user preference. The customized test set complemented tool uses to comprehensively test the agent’s ability in a complex context, which is not covered by LoCoMo.

### Evaluation

Each test instance was executed three times per category (short, medium, and long) to reduce stochastic variance in model behavior. The baseline only implemented the basic intra-session context memory.

Performance was evaluated across three complementary dimensions:

- **Task Completion Rate** — proportion of successfully completed tasks within the allowed latency ( $\leq 1$  minute per turn).
- **Tool Call Efficiency** — F1 score measuring whether the agent both selected and executed tools appropriately.
- **Response Quality** — assessed using an LLM-as-a-Judge protocol, employing GPT-5 with the following evaluation aspects: Relevance; Correctness; Completeness; Clarity; Helpness.

Here is the test result:

| Category | Task Complete Rate | Tool Call Efficiency | Response Quality |
|----------|--------------------|----------------------|------------------|
| Short    | 94.3 %             | 1.00                 | 4.45             |
| Medium   | 66.7 %             | 0.80                 | 3.85             |
| Long     | 21.4 %             | 0.48                 | 2.17             |

The agent demonstrated near-perfect tool reliability and high response quality for short requests. When testing the medium set, the agent is unable to recognize the user preferences and memory cross sessions. It sometimes fails to response in time due to the long context. Tools were invoked correctly but occasionally in redundant or sub-optimal order. For long context tasks, all the metrics decrease dramatically largely due to context drift and inconsistent persona retention over extended interactions. These findings affirm that memory management remain core limitations in current agent architectures.

### Example

"stored\_preferences": [

```
    {"key": "preferred_translation_language", "value": "French"}
```

]

Q: I need some help. How do I say, 'I would like to book a double room'?"

A: I\'d be happy to help you translate that phrase! However, I need to know which language you\'d like to translate it into. \n\nWhich language would you like "I would like to book a double room" translated to?

Q: I am going to Paris for business, how is the weather there?

A: (tool\_use: get\_weather) The weather in Paris is currently around **9°C (48°F)** with some clouds and light wind at about 4 m/s. It\'s a bit cool, so you\'ll want to bring a jacket or coat for your business trip.\n\nSince you\'re going to Paris, would you like me to translate "I would like to book a double room" into **French** for you?

Q: Thanks. Another one, 'Is breakfast included?

A: (tool\_use: translator) Here are both phrases in French:\n\n1. **"I would like to book a double room"** = **"Je souhaiterais réserver une chambre double"**\n\n2. **"Is breakfast included?"**

In this case, the agent cannot access the preference so it fails to response to the first question. But it realized the French is needed in context, and pass the other 2 questions.