Building a Task-Oriented Chatbot with Tools, Memory, and Safety

Shelbi Candley
Computer Science
Full Sail University
Winter Park, FL
scandley@student.fullsail.edu

***Abstract*—This paper explains the design and implementation of a task-oriented chatbot built as a Python API. The objective was to create a system that behaves like a real production service rather than a simple demo. The chatbot maintains conversation history, uses external tools when appropriate, streams responses to the user, and applies safety protections such as content filtering, secret redaction, and rate limiting. Two tools were integrated: a weather lookup service and a small knowledge base for frequently asked questions. A twenty-prompt evaluation suite was used to measure task success, memory behavior, and refusal handling. Running in mock mode, the system achieved 100 percent task success with an average latency of 129 ms. These results show that combining structured tool access with guardrails leads to reliable and predictable performance.**

Keywords*—*GPT, function calling, FastAPI, conversational memory, safety guardrails, evaluation

I. INTRODUCTION

Large language models are increasingly used in applications that must do more than generate fluent text. Real deployments require controlled behavior, integration with external data sources, and mechanisms for monitoring performance. Without these layers, even powerful models may produce inconsistent or unsafe outcomes.

The purpose of this project was to design a chatbot that reflects how modern AI services are built in practice. Instead of relying solely on generation, the system incorporates tool execution, short-term memory, streaming interaction, and operational safeguards. In addition, metrics are recorded for each turn to support quantitative evaluation. The following sections describe the system architecture, tool design, safety mechanisms, and experimental results.

II. SYSTEM ARCHITECTURE

The chatbot is implemented as a RESTful web service using FastAPI. Clients send a message and conversation identifier to the /chat endpoint, allowing the server to reconstruct prior context. Responses are streamed back incrementally to simulate real-time interaction.

The system is divided into modular components responsible for language models access, memory handling, tool execution, safety enforcement, and metrics collection. This separation simplifies debugging and allows individual layers to evolve without effecting the entire application.

A conversation store maintains message history, while a rolling window limits how much context is passed to the model. This approach balances recall with efficiency and prevent runaway token growth.

III. TOOL USE AND FUNCTION CALLING

A core objective of the project was enabling the assistant to move beyond text generation and interact with structured utilities. Tool calling provides a reliable bridge between model reasoning and deterministic computation.

Two tools were implemented:

- Weather Lookup – returns a stubbed forecast for a requested city
- Knowledge Base Search – retrieves answers from a small FAQ dataset

When the model determines that external information is needed, it produces a structured call. The server executes the function, appends the result to the conversation, and requests a final response from the assistant. This workflow significantly reduces hallucination and improves factual consistency.

IV. MEMORY MANAGEMENT

Maintaining context across turns is essential for coherent dialogue. The chatbot uses a short-term memory strategy based on the most recent messages within a configurable window. This method allows the assistant to recall prior statements, answer follow-up questions, and reference earlier instructions while keeping token usage predictable. Evaluation tasks confirmed that the system consistently remembered user-provided details across multiple interactions.

V. SAFETY AND GUARDRAILS

Production environments require proactive risk mitigation. A safety layer was implemented to inspect inputs before they reach the model. Requests involving violence, hacking, or self-harm are intercepted and blocked with controlled responses.

In addition, secret redaction prevents accidental exposure of sensitive information, and rate limiting protects the service from abuse. All blocked interactions are recorded for auditing and analysis.

VI. Evaluation Methodology

Performance was assessed using a predefined suite of twenty prompts. These tasks were distributed across several categories:

- knowledge retrieval
- weather queries
- conversational memory
- normal usage
- safety violations

Each scenario had an expected outcome, allowing automated pass or fail scoring. Latency was measured from request receipt to completion of streaming output. To isolate orchestration quality from API variability, experiments were conducted in mock mode, where tool behavior is simulated.

VII. Results

Under evaluation, the chatbot achieved 100 percent success rate in mock mode. The average latency across all tasks was approximately 129 milliseconds, indicating responsive interaction. Tool routing behaved deterministically, and memory recall succeeded in multi-step exchanges. After refining the safety filters, all refusal scenarios produced compliant outputs.

These outcomes demonstrate that structured architecture and validation layers can yield highly stable performance even with lightweight implementations.

VIII. Failure Modes and Lessons Learned

Earlier iterations revealed that most failures stemmed from integration details rather than model reasoning. Common issues included incorrect environment variables, incomplete persistence of conversation state, and formatting mismatches between tool outputs and prompts. Improving error handling and observability made the system easier to maintain and significantly increased reliability.

IX. Conclusion

This project illustrates that building practical AI systems requires careful orchestration around the language model. Memory, tool execution, safety validation, and metrics are critical for achieving dependable behavior. Future enhancements could include persistent storage, semantic retrieval for long-term memory, and more advanced policy enforcement. Nonetheless, the current implantation demonstrated how GPT models can be transformed into structured, measurable services suitable for real-world scenarios.

Acknowledgments

The author thanks the course instructor for guidance on evaluation design and production-oriented thinking in AI system development.

References

[1] OpenAI, “API Platform Documentation,” 2024.

[2] FastAPI, “FastAPI Framework,” 2024.

[3] T. Browne, “Language Models are Few-Shot Learners,” 2020.