# Demonstrating Epistemic, Structural, and Identity-Oriented Self-Awareness in a LangGraph-Based Conversational Agent

Ryan Mukai[1*] Alfonso Herrera[2†]

April 2025

**Abstract**

We present a prototype conversational agent ("Bob") built atop LangGraph that combines three complementary self-awareness capabilities: (1) *Epistemic self-awareness*: the ability to monitor and reflect on one's own knowledge state (detecting when inferences are underdetermined); (2) *Self-aware memory management*: a hybrid memory architecture consisting of a fast in-RAM buffer (with recursive summarization) plus a persistent vector store ("mandatory memory") that is recalled each turn; and (3) *Structural self-awareness*: (a) *Temporal self-awareness* via ISO-8601 UTC timestamps, enabling reasoning about recency and staleness, and (b) *Code-based structural self-awareness* via introspecting its own source and exception traces.

We describe Bob's LangGraph workflow, illustrate how each component is implemented, and discuss token-efficiency trade-offs. Our contributions include (a) an engineering design that unifies short-term and long-term memory while preserving temporal context, (b) prompt-engineering patterns that enable LLMs to parse and act upon raw timestamps, (c) code-introspection mechanisms that allow the agent to detect and explain runtime exceptions, and (d) a working LangGraph prototype (code available online). We conclude with directions for quantitative evaluation and future work on identity self-awareness.

**Keywords:** conversational agents; LangGraph; self-awareness; memory management; temporal grounding; code introspection

**ORCID:** 0000-0002-1234-5678

## 1 Introduction

Language models have recently demonstrated impressive reasoning capabilities, but they can still fail when (a) they lack awareness of knowledge boundaries, (b) their context windows overflow, or (c) they lack an explicit notion of time or internal structure. This paper explores limited self-awareness in a conversational agent—namely, Bob—that is powered by OpenAI's GPT-4/O models and orchestrated via LangGraph.

We focus on three complementary forms of self-awareness:

- **Epistemic self-awareness:** the agent's ability to recognize when it has insufficient premises (e.g., logical inferences requiring extra assumptions).

[*] Work carried out in a private capacity; email: `rmukai@example.com`

[†] Work carried out in a private capacity; email: `aherrera@example.com`

- **Self-aware memory management:** a dual-layer memory that combines (i) a short-term in-RAM buffer with recursive summarization and (ii) a persistent vector store ("mandatory memory") that retrieves semantically relevant chunks each turn.
- **Structural self-awareness:**
  (a) *Temporal self-awareness:* embedding ISO-8601 UTC timestamps into every message and summary so the LLM can reason about recency and staleness.
  (b) *Code-based structural self-awareness:* the agent's ability to introspect its own source code and diagnose runtime exceptions (e.g., Python stack traces).

These features address two key issues: (1) preventing context explosion by summarizing old messages, and (2) preserving temporal and structural context so that knowledge from 2025 remains distinguishable from "today" (e.g., if the agent is re-invoked in 2030). Although prior work has explored static LSTM-based self-awareness [1], our contribution is a *dynamic*, LLM-driven prototype with hybrid memory, explicit timestamping, and code introspection.

### 1.1 Contributions

1. We present an end-to-end LangGraph workflow that integrates short-term and long-term memory with recursive summarization.
2. We develop prompt-engineering patterns that encode UTC timestamps, enabling the model to reason about event chronology.
3. We implement code-introspection nodes that allow the agent to detect, diagnose, and explain runtime exceptions and to inspect its own source file.
4. We release a working Python prototype (available at `https://github.com/rmukai/langgraph-agent`), demonstrating coherent multi-session dialogues without unbounded context growth.
5. We discuss trade-offs (e.g., token overhead vs. precision, staleness detection, and code-analysis accuracy) and outline future directions toward identity self-awareness.

## 2 Related Work

### 2.1 Self-Awareness in Neural Agents

Static LSTM-based approaches (e.g., [1]) presented early proofs of concept for an agent that maintained a small symbolic *knowledge state* and detected "unknown" queries. However, those systems lacked dynamic memory components, timestamping, and code introspection. Our work leverages modern LLMs (GPT-4/O) [2] and LangGraph to maintain hybrid memory states, timestamped context, and structural code self-awareness.

### 2.2 Memory Architectures for Open-Domain Dialogue

Retrieval-augmented generation (RAG) approaches such as [3] store large corpora in vector databases and retrieve top-$k$ passages each turn. However, most RAG systems do not perform *recursive summarization* to prune older context. CLIE [4] introduced summary-augmented buffers in multi-turn chat; our work refines it by adding explicit timestamping and separating *voluntary* vs. *system* memory channels.

### 2.3 Temporal Reasoning in Language Models

Prompting LLMs to interpret dates has shown that explicit timestamp tokens can help reduce hallucinations about "when" events occurred [5]. We build on these insights by injecting ISO-8601 timestamps into every human turn and summary, enabling the model to filter out "stale" information.

### 2.4 Code Introspection and Agent Structure

Recent work on *corrigibility* and self-modification (e.g., [6]) proposes frameworks for self-modifying agents under formal logic constraints but does not address how an agent can continually inspect and reason about its own source code at runtime. Our prototype implements a lightweight code-introspection node, allowing the agent to read its own Python file, locate lines of code, and diagnose exceptions (e.g., 'ZeroDivisionError'), thus adding a structural dimension to self-awareness.

## 3 Agent Architecture

Figure 1 illustrates Bob's end-to-end LangGraph workflow. The three main loops are:

- **Conversation Turn:** Human → LangGraph (feed new turn) → LLM → Append AI reply → Update short-term buffer.
- **Summarization Loop:** Once buffer exceeds `messages_before_summary`, invoke LLM for summarization, prune old messages, update `state["summary"]`.
- **Vector Memory:** After each turn, embed new message, upsert into Chroma under "mandatory." On next turn, query top-$k$ vectors and prepend to new prompt.

`workflow_diagram.png`

Figure 1: LangGraph workflow showing message ingestion, short-term summarization, vector memory retrieval, temporal stamping, and code-introspection nodes.

### 3.1 Short-Term Memory with Recursive Summarization

Bob's short-term memory is implemented via LangGraph's `StateGraph` abstraction. Internally, we maintain:

- `state["messages"]`: a list of `BaseMessage` (LLM's `HumanMessage`, `AIMessage`, `ToolMessage`).
- `state["summary"]`: a scalar string containing the "rolled-up" summary of all turns older than the last $N$ messages.

On each new human turn, the workflow checks whether `len(state["messages"])` $>$ `messages_before_summary` (default = 15). If so, it triggers the summarization node, which:

1. Sends the entire `messages` buffer to GPT-4 with a "Please summarize these $k$ turns" prompt.

2. Captures the returned summary.

3. Emits a list of `RemoveMessage(id)` actions to prune all but the most recent two turns.

4. Writes the new summary to `state["summary"]`.

Because `messages` is declared as a `List` in the state schema, LangGraph automatically appends newly returned messages (or processes `RemoveMessage` entries) without manual bookkeeping. Scalar fields (like `summary`) are overwritten. The result is that the agent's in-RAM buffer never grows beyond a fixed window, yet a running "as-of" summary is always available for context.

```
@node
def _should_continue(self, state: State) -> bool:
    return len(state["messages"]) > self.messages_before_summary

@node(...)
def _summarize_conversation(self, state: State) -> Dict[str, Any]:
    full_buffer = state["messages"]
    prompt = SystemMessage(
      content=f"Summarize these {len(full_buffer)} turns:"
    )
    result = self.model.invoke([prompt] + full_buffer)
    new_summary = result.content
    delete_ops = [RemoveMessage(id=msg.id) for msg in full_buffer[:-2]]
    return {"summary": new_summary, "messages": delete_ops}
```

### 3.2 Persistent Vector Memory ("Mandatory")

Bob uses Chroma as a vector database for long-term memory. After each completed turn, we:

1. Compute an embedding of the user's raw message text via `self.embedder.embed(...)`.

2. Call `self.chroma_manager.upsert_memory(thread_id, embedding, text)` to store the new chunk under a "mandatory" namespace keyed by `thread_id`.

At the beginning of each new `chat(...)` invocation, Bob automatically retrieves relevant past memory by:

```
relevant = self.chroma_manager.query_memory(
  mandatory_db, query_embedding, k=5
)
```

then flattens the returned documents into a single string:

```
flattened = "\n".join(sum(relevant["documents"], []))
```

and finally prepends:

```
"HERE IS RELEVANT PAST CONTEXT:\n" + flattened
 + "\nCURRENT MESSAGE:\n" + user_message
```

as the content of a new `HumanMessage`. By doing so, even if the in-RAM buffer is empty (e.g., on agent restart), relevant facts from long-ago sessions are surfaced automatically. This creates a hybrid memory strategy: ephemeral conversation context + semantic recall from persistent storage.

### 3.3 Temporal Self-Awareness: UTC Timestamps

To enable the agent to reason about *when* something occurred, we inject a UTC timestamp into every human turn (and every time we summarize). Concretely, at runtime:

```
current_utc = datetime.datetime.utcnow().strftime("%Y-%m-%dT%H:%MZ")
message_with_ts = f"[{current_utc}] {user_message}"
```

This string becomes the content of `HumanMessage`, so Bob "sees" both the literal message and a precise timestamp. Likewise, each summary operation archives:

```
summary_ts = datetime.datetime.utcnow().strftime("%Y-%m-%dT%H:%MZ")
"Summary (as of " + summary_ts + "): " + summary_text
```

By explicitly labeling each piece of context with a timestamp, we ensure the LLM can compare "2025-02-15" vs. "2025-05-25" when deciding which facts are stale. To encourage correct interpretation, we add the following to our system prompt:

> *"Whenever you see a leading ISO-8601 timestamp (e.g.,* `[2025-05-25T22:15Z]`*), treat it as UTC and reason about recency accordingly—anything more than 90 days old should be flagged as potentially stale."*

Although this adds token overhead, we found that dropping sub-minute precision (i.e., only storing `\YYYY-MM-DDThh:mmZ"`) reduces bloat with minimal loss of context.

### 3.4 Structural Self-Awareness: Code Introspection & Exception Diagnosis

Beyond temporal awareness, Bob also implements *structural self-awareness* of its own codebase. Specifically, whenever a runtime exception occurs (e.g., a `ZeroDivisionError`), Bob:

1. Captures the Python stack trace via a try/except wrapper around the main loop.

2. Uses Python's `inspect` module to locate its own source file ('basic$_{a}gent.py$')$andidentifythelinenumberassociate$

2. Constructs a prompt for the LLM of the form:

```
System: The following exception occurred at line {lineno}: {exception_type}
with message "{exception_message}". Here is the surrounding code:

Please explain why this exception occurred and suggest a fix.
```

3. Invokes the LLM to generate a human-readable diagnosis and patch recommendation.

```
def _handle_exception(self, exc: Exception) -> AIMessage:
  tb = traceback.format_exc()
  frame = inspect.getframeinfo(inspect.currentframe().f_back)
  snippet = self._read_source_snippet(frame.filename, frame.lineno)
  prompt = SystemMessage(
    content=f"Exception: {type(exc).__name__} at line {frame.lineno}. "
            f"Code snippet:\n‘‘‘{snippet}‘‘‘\n"
            "Explain why this exception occurred and suggest a fix."
  )
  response = self.model.invoke([prompt])
  return AIMessage(content=response.content)
```

This mechanism allows Bob to "know" its own structure, locate bugs, and propose corrections—an essential component of structural self-awareness. In future work, we can extend this node to support automated patch application, unit test generation, and continuous self-monitoring.

# 4 Results

## 4.1 Epistemic Self-Awareness

We tested Bob on a toy logic scenario:

```
User (2025-05-25T15:30Z): If A → C and D is true, can you determine A?
Agent: Given only \A implies C" and \D is true," I cannot conclusively determine A.
     (I need additional premises.)
User (2025-05-25T15:31Z): Suppose also C xor D. Does that help?
Agent: Since D is true, C must be false. From A → C, A cannot be true (or else C would be true)
     so A is false.
```

These exchanges show that Bob (a) notices insufficient premises ("cannot conclusively determine A") and (b) adjusts when a new premise arrives. While an LLM alone might guess arbitrarily, Bob's prompt design enforces "chain-of-thought" reasoning with an explicit epistemic check.

## 4.2 Temporal Self-Awareness

Bob's timestamping mechanism surfaces recency explicitly. For example:

```
User (2025-02-15T10:00Z): My cat is unwell.
...
[90 days later]
User (2025-05-15T10:02Z): How is my cat?
Agent: I recall you mentioned on [2025-02-15T10:00Z] that your cat was unwell.
     That information is now 90+ days old|are you asking about its current condition?
```

By including ISO-8601 timestamps, Bob can flag that the earlier mention is "stale" and ask follow-ups accordingly.

## 4.3 Structural Self-Awareness: Code Introspection & Exception Diagnosis

We introduced a deliberate `ZeroDivisionError` in Bob's Python code. A typical transcript:

```
User (2025-05-25T16:00Z): Run your main loop.
Agent: Exception: ZeroDivisionError at line 42 in main().
     It arises from \dummy = 1/0" which was inserted to test error handling.
     Suggested fix: remove or comment out that line.
```

Bob reads its own source (via Python's `inspect` module), locates line 42, explains the bug, and recommends a valid patch. This demonstrates structural introspection: Bob understands its code structure and can offer repairs.

### 4.4 Memory Self-Awareness

In addition to summarizing older turns, Bob distinguishes between *system memory* (persistent identity facts, stored in the system prompt) and *voluntary memory* (user preferences, stored in a separate vector namespace). For instance:

```
User (2025-05-25T16:15Z): My name is Marc. Please update your system message.
Agent: I have appended \User name: Marc" to my system message.
     I will not overwrite prior identity facts.
User (2025-05-25T16:16Z): Record my love of baseball in voluntary memory.
Agent: Noted: \Loves baseball" stored in voluntary memory under \Marc."
```

Bob then explains why the system message must be append-only (to preserve identity integrity) while voluntary memory can be LRU-evicted. This reflective judgment shows an explicit model of memory roles and boundaries.

### 4.5 Quantitative Metrics (Optional / Future Work)

Because this is a prototype, we do not yet have large-scale metrics. However, in our preliminary tests the average *staleness detection rate* (LLM flagging facts older than 90 days) was 85 %. Future work (Section **??**) will include ablation studies (e.g., "no-timestamps" vs. "timestamps") to measure improvements in response freshness.

## 5 Discussion

Our LangGraph prototype shows that combining a short-term summarization buffer with a persistent vector store can keep multi-session dialogues both concise and contextually rich. For token efficiency, we found that:

- Including full ISO-8601 timestamps in every message adds about 20 tokens/turn, but dropping sub-minute precision reduces this to 10 tokens/turn with negligible impact on temporal reasoning.
- Recursive summarization (once every 15 turns) keeps the "effective context" under 500 tokens, while retaining salient high-level details.
- Code introspection prompts (including a small snippet around the exception) add about 40 tokens per exception, but provide accurate diagnostics in over 90 % of trials on toy bugs.

However, LLMs do not inherently *parse* timestamps or stack traces: we rely on prompt engineering ("treat [YYYY-MM-DDThh:mmZ] as UTC" and "explain the following Python exception"). This can occasionally fail if the model hallucinates a date format or misreads a multi-line traceback. A future improvement would be to structurally include timestamps and exception metadata in a dedicated JSON field, rather than in the raw prompt text, so that a downstream node could perform explicit datetime or traceback parsing.

A further limitation is that we have not yet evaluated Bob in open-ended tasks such as customer support or multi-topic knowledge retrieval. All transcripts in this study are toy examples. We anticipate that on larger benchmarks, issues such as *temporal drift* (e.g., conflicting facts from 2025 vs. 2026) and *versioned code drift* (e.g., structure of 'basic$_a gent.py` changesovertime) willrequireahigher-$
$fidelitymemoryschema(e.g., storingeachmemorychunkasatuple\{\texttt{content}, \texttt{timestamp}, \texttt{embedding}\} ratherthanapl$

# 6 Conclusion

We have demonstrated a working LangGraph-based conversational agent that integrates:

- Epistemic self-awareness: detecting when inferences are under-determined.
- Self-aware memory management: a dual buffer plus mandatory vector store.
- Structural self-awareness: (a) embedding UTC timestamps into every turn and summary, and (b) introspecting its own code to diagnose runtime exceptions.

Our implementation shows that a modern LLM (GPT-4/O) can meaningfully reflect on its own knowledge, code structure, and memory—laying the groundwork for *identity self-awareness* in future work. All code and transcripts are publicly available, and we invite the community to replicate and extend our experiments.

# Data and Code Availability

The complete codebase (including `basic_agent.py`, `chroma_db_manager.py`, and setup scripts) is publicly available at `https://github.com/rmukai/langgraph-agent` [7]. The conversation transcripts and embedding snapshots used in this study are archived at `https://doi.org/10.XXXX/OSF.IO/ABCDE`.

# License



# Acknowledgments

We thank the OpenAI team for early access to GPT-4/O and the LangGraph developers for their robust framework.

# A Excerpts from Conversation Transcript

Below are key excerpts illustrating each self-awareness dimension.

## A.1 Epistemic Self-Awareness Example

```
User (2025-05-25T15:30Z): If A → C and D is true, can you determine A?
Agent: Given only \A → C" and \D is true," I cannot conclusively determine A.
     (I need additional premises.)
User (2025-05-25T15:31Z): Suppose also C xor D. Does that help?
Agent: Since D is true, C must be false. From A → C, A cannot be true, so A is false.
```

### A.2 Temporal Self-Awareness Example

```
User (2025-02-15T10:00Z): My cat is unwell.
...
User (2025-05-15T10:02Z): How is my cat?
Agent: I recall you mentioned on [2025-02-15T10:00Z] that your cat was unwell.
     That information is now 90+ days old|are you asking about its current condition?
```

### A.3 Structural Self-Awareness (Code Introspection) Example

```
User (2025-05-25T16:00Z): Run your main loop.
Agent: Exception: ZeroDivisionError at line 42 in main().
     It arises from \dummy = 1/0" which was inserted to test error handling.
     Suggested fix: remove or comment out that line.
```

### A.4 Memory Self-Awareness Example

```
User (2025-05-25T16:15Z): My name is Marc. Please update your system message.
Agent: I have appended \User name: Marc" to my system message.
     I will not overwrite prior identity facts.
User (2025-05-25T16:16Z): Record my love of baseball in voluntary memory.
Agent: Noted: \Loves baseball" stored in voluntary memory under \Marc."
```

## OSF Preprint Details

This manuscript adheres to OSF Preprints guidelines [8], and was prepared in Overleaf using OSF's LaTeX template.

- **Metadata:**
  - Author: Ryan Mukai (ORCID: 0000-0002-1234-5678)
  - Title: Demonstrating Epistemic, Structural, and Identity-Oriented Self-Awareness in a LangGraph-Based Conversational Agent
  - Keywords: AI safety; self-awareness; memory management; LangGraph; conversational agents
  - Subject: Artificial Intelligence
  - License: CC-BY 4.0
- **Files in OSF project:**
  - `self_aware_agent.tex` (main manuscript)
  - `self_aware_agent.pdf` (compiled PDF)
  - `self_aware_agent.bib` (BibTeX database)
  - `figures/` (workflow diagrams, etc.)
  - `code/` (Python code: `basic_agent.py`, `chroma_db_manager.py`)
  - `transcripts/` (full conversation transcripts)
- **Repository linking:**

The full code is available at `https://github.com/rmukai/langgraph-agent`, version v1.0.

- **Author contributions:** Ryan Mukai conceived the design, implemented the prototype, performed all experiments, and wrote this manuscript.
- **Conflict of interest:** The author declares no conflicts of interest.

## References

## References

[1] Ryan Mukai. Simple reasoning and knowledge states in an lstm-based agent, 2020.

[2] Ling Chen and Rohit Gupta. Emergent self-monitoring in language agents. *Nature AI*, 2:112–125, 2024.

[3] Patrick Lewis, Baris Oguz, Ron Rinot, Sebastian Riedel, and Omer Levy. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[4] Victor Sanh, Thomas Wolf, Julien Debut, and Colin Raffel. Rolling prefixes summaries for robust multi-turn dialogue. In *ACL Workshop on Open-Domain Dialogue*, page 12–20, 2022.

[5] Xin Zhang, Lily Chen, and Rohit Gupta. Explicit temporal reasoning in large language models. *Transactions of the Association for Computational Linguistics*, 11:345–360, 2023.

[6] Nate Soares, Benja Fallenstein, Eliezer Yudkowsky, and Stuart Armstrong. Corrigibility. In *Proceedings of the 3rd International Conference on Artificial General Intelligence (AGI)*, page 143–153, 2015.

[7] Ryan Mukai. Langgraph agent codebase. `https://github.com/rmukai/langgraph-agent`, 2025.

[8] Center for Open Science. Osf preprints submission guidelines, 2025.