From LLMs to Operating Systems

Inspired by the MemGPT paper and the DeepLearning.AI course, these notebooks explore how to overcome the memory limitations of LLMs by treating them like the CPU of a traditional operating system.

Concept 1: The Memory Hierarchy

Just like a computer has fast RAM and slow disk storage, an agent has a tiered memory system. The key is to intelligently manage what information is loaded into the LLM's limited "working memory" (the context window).

Main Context (RAM)

🧠 Fast & Limited

The LLM's direct context window. Holds the system prompt, recent chat history, and critical working memory.

<—–>

External Context (Disk)

📚 Vast & Slower

A vector database or file system where long-term knowledge (like a PDF handbook) is stored.

Concept 2: Self-Editing Memory

The core idea of MemGPT: give the LLM the tools to manage its own memory. Instead of us deciding what's important, the agent learns to save critical information itself.

1. User provides new info: "My name is Bob."

↓

2. LLM Reasons: "This is important. I should save it to my 'human' memory section."

↓

3. LLM Calls a Tool: It generates a function call.

```
core_memory_save(
  section="human",
  memory="The human's name is Bob."
)
```

↓

4. Memory is Updated: Our Python code executes the function, updating the `memory_blocks` dictionary, which is then available for all future turns.

Concept 3: Agentic RAG

This memory system enables a powerful, agent-driven RAG process. The agent actively decides when it needs more information and goes to get it from its external memory.

1. User asks a question: "What is the company's vacation policy?"

↓

2. LLM Reasons: "I don't have this in my working memory. I need to search the external handbook."

↓

3. LLM Calls a Tool: It generates a search query for the vector database.

```
archival_memory_search(
  query="vacation policies"
)
```

↓

4. Memory is Augmented: The tool returns relevant passages from the PDF. This information is temporarily loaded into the LLM's main context for this turn.

↓

5. Generation: The LLM uses the newly retrieved context to generate a fact-based, accurate answer for the user.

Conclusion

By giving LLMs tools to manage a memory hierarchy, we transform them from simple chatbots into more capable, autonomous agents that can learn, reason, and solve complex, multi-step problems. This is the core idea of treating LLMs like an **operating system** for intelligence.