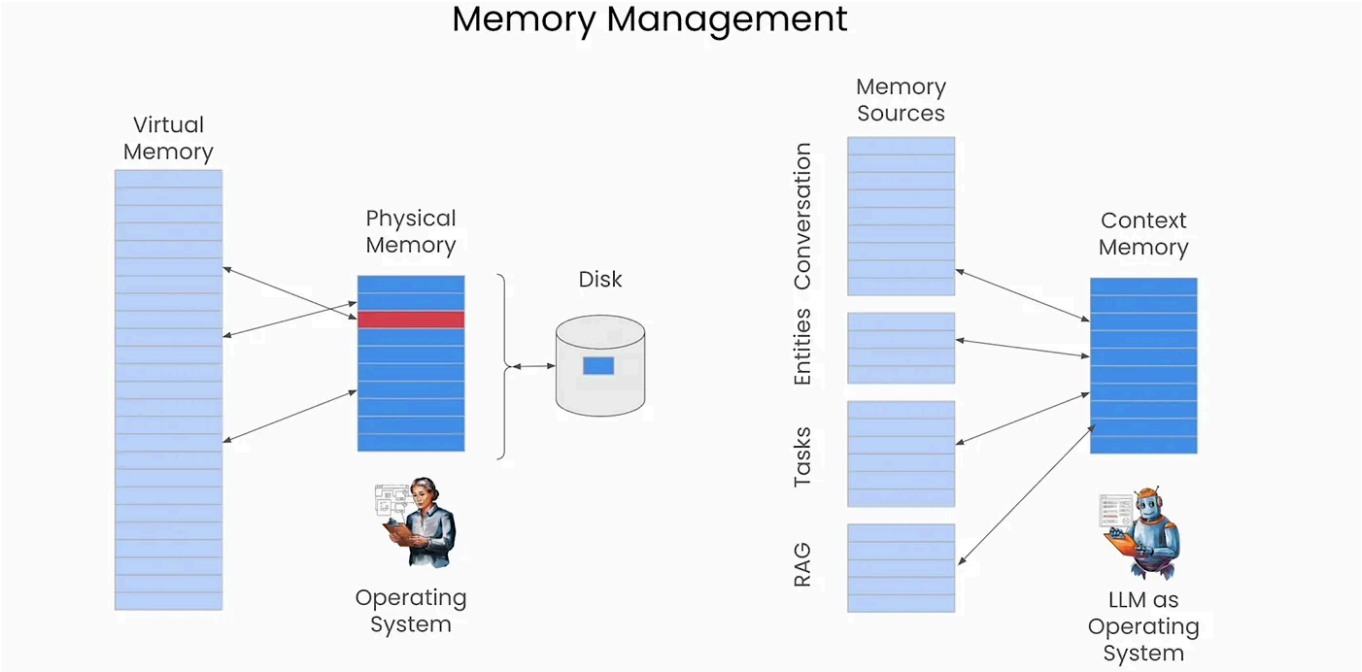
Extending LLM Context Through OS-Inspired Virtual Memory and Hierarchical Storage

Neeraj Kumar Follow 6 min read · Nov 9, 2024



MemGPT manages different types of memory resources, similar to how an operating system (OS) handles virtual and physical memory, along with storage on disk.

OS Memory Management

- **Virtual Memory**: In a traditional OS, virtual memory is an abstraction layer that allows programs to use more memory than what's physically available by swapping data to and from disk. It provides each program with an isolated memory space, improving efficiency and multitasking.
- **Physical Memory**: Physical memory (RAM) is where data is stored temporarily for quick access. In OS memory management, physical memory handles active tasks and data.
- **Disk**: When physical memory is full, data can be transferred to disk storage, a more permanent but slower storage solution. The OS manages this transfer to ensure that high-priority tasks remain in physical memory while less active data is offloaded.

In an OS, the **operating system** coordinates between virtual memory, physical memory, and disk, deciding what data stays in RAM, what goes to disk, and when to swap it back.

MemGPT- Memory Management

- **Memory Sources** that contain different types of information.
- **Context Memory**, Similar to physical memory in an OS, MemGPT's context memory aggregates data from multiple sources (conversation, entities, tasks, RAG) that is immediately relevant to the current interaction. This allows MemGPT to access pertinent information without needing to retrieve from deeper storage every time.
- An **LLM acting as the OS**, the language model functions as an "operating system," coordinating between memory sources, managing retrieval and storage, and autonomously deciding what to keep in context memory. Just as an OS decides what data stays in RAM or gets moved to disk, MemGPT decides which parts of memory (e.g., recent conversation history, task states) should remain in its active context for quick access.

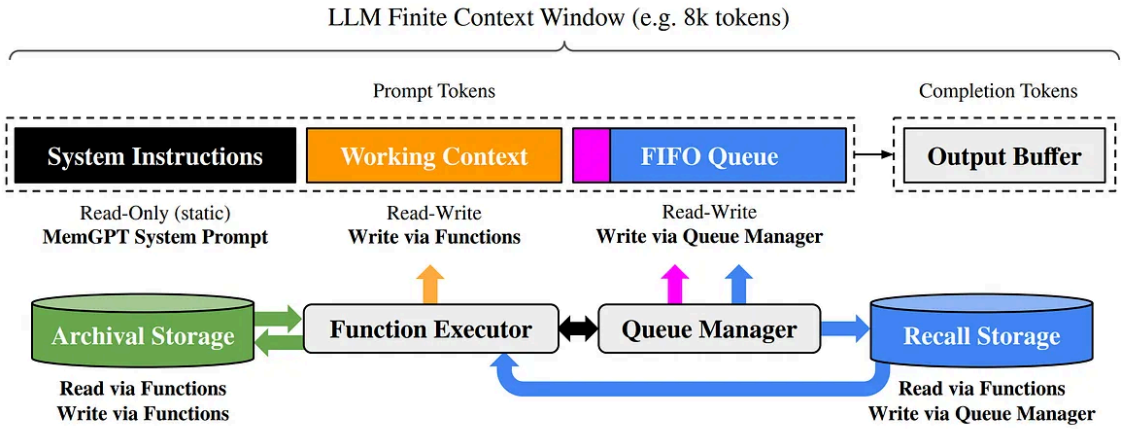


Figure 3. In MemGPT, a fixed-context LLM processor is augmented with a hierarchical memory system and functions that let it manage its own memory. The LLM's prompt tokens (inputs), or *main context*, consist of the system instructions, working context, and a FIFO queue. The LLM completion tokens (outputs) are interpreted as function calls by the function executor. MemGPT uses functions to move data between main context and *external context* (the archival and recall storage databases). The LLM can request immediate follow-up LLM inference to chain function calls together by generating a special keyword argument (`request_heartbeat=true`) in its output; function chaining is what allows MemGPT to perform multi-step retrieval to answer user queries.

Architecture:

1. **Main Context (LLM Finite Context Window)**: This is the main, limited context area accessible to the LLM. It consists of three segments:

— **System Instructions**: These static, read-only instructions guide the LLM on handling memory, specifying tasks, and usage guidelines for MemGPT functions. These instructions help the LLM to decide when and how to manage context.

— **Working Context**: This read-write section holds critical information, like key facts about the user or the agent's persona, stored in unstructured text. It can be updated by the LLM through function calls.

— **FIFO Queue**: A rolling queue that stores recent messages, interactions, and system messages (e.g., memory alerts). As the queue reaches capacity, older messages are replaced with a recursive summary, allowing the LLM to retain a high-level overview.

2. **Queue Manager**: This component manages the FIFO queue and recall storage, handling message overflow and organizing which information is retained within context.

— When a new message arrives, it's appended to the FIFO queue and saved to recall storage. If the queue exceeds a certain length, the queue manager initiates a memory pressure warning, prompting the LLM to store relevant information in working context or external storage.

— If the queue surpasses the maximum token count, it flushes older messages, generating a recursive summary that compresses past interactions into a more concise representation.

3. **Recall Storage**: This is an out-of-context storage system that holds data outside the LLM's context window. The LLM can retrieve this data by explicitly calling functions. It serves as an intermediate memory tier, storing a larger set of historical interactions that can be referenced when needed.

4. **Archival Storage**: This component is a long-term memory, where more distant or seldom-used information is stored. Unlike recall storage, archival storage is intended for less frequently accessed data, but it can still be searched and brought back into the context if relevant.

5. **Function Executor**: This interprets the LLM's output, allowing MemGPT to manage its memory based on function calls embedded within the generated

text. These functions help transfer data between main and external contexts, enabling MemGPT to handle complex, multi-step memory retrievals.

— The LLM uses functions to save, edit, or retrieve memory, which the function executor manages based on the system's current state. These functions are self-directed and allow MemGPT to move data independently to optimize context space.

6. **Control Flow and Function Chaining**: Control flow in MemGPT allows the system to manage sequences of tasks by chaining function calls. If the LLM requires multiple memory operations, it can flag certain outputs for immediate follow-up processing, which allows for multi-step reasoning or iterative data retrievals without interrupting the user experience.

Queue Manager

The Queue Manager in MemGPT plays a central role in managing the limited memory resources of the main context (LLM's context window). It efficiently organizes incoming messages and system alerts, optimizes memory usage by deciding what information to retain or evict, and allows MemGPT to handle long, complex conversations without overloading the finite context window. Here's an in-depth explanation of each function within the Queue Manager:

1. **Message Management:**

— **Appending New Messages**: When a new message (from either the user or the system) arrives, the Queue Manager appends it to the end of the FIFO queue. This queue acts as a temporary storage that retains the most recent interactions and allows the LLM to generate relevant responses by referencing recent conversation history.

— **LLM Inference Trigger**: After appending a new message, the Queue Manager triggers the LLM inference by sending the current state of the prompt tokens (combined system instructions, working context, and FIFO queue) to the LLM. The LLM then generates a response based on this full prompt context.

— **Storing LLM Output**: Once the LLM generates its response (completion tokens), the Queue Manager saves this response alongside the incoming message in Recall Storage (external memory).

2. **Handling Context Overflow and Eviction Policy**
The Queue Manager operates with a set of thresholds that guide how it handles memory pressure due to limited context space:

— **Warning Token Count**: When the number of prompt tokens in the FIFO queue approaches a critical level (e.g., 70% of the total context window capacity), the Queue Manager inserts a memory pressure warning. This alert signals the LLM to prioritize saving essential information to either the Working Context (within the main context) or to Archival Storage (a longer-term memory outside the main context) before the FIFO queue overflows.

— **Flush Token Count**: Once the queue reaches the Flush Token Count (e.g., 100% of context capacity), the Queue Manager automatically "flushes" the queue to free up space. It evicts the oldest messages (typically around 50% of the context window) and replaces them with a recursive summary.

— **Recursive Summary Creation**: The Queue Manager combines information from the evicted messages with the previous recursive summary, creating a new, updated summary that captures essential information. This summary compresses the conversation history, preserving critical points while freeing up space for new messages.

3. **Function Call Management for Recall and Archival Storage**
When relevant information needs to be accessed from storage:

— **Recall Retrieval**: If the LLM calls a function to retrieve historical messages stored in Recall Storage, the Queue Manager pulls the required messages and appends them to the FIFO queue. This allows the LLM to reference previous conversations or details that might be relevant for the current task.

— **Data Transfer to Archival Storage**: During memory pressure warnings, the Queue Manager also facilitates the movement of less immediately relevant information to Archival Storage. This enables MemGPT to store information that doesn't need to remain in the active context but should still be accessible if recalled later.

4. **Managing Recency and Relevance of Messages**
To ensure that the context remains up-to-date with the most relevant data, the Queue Manager employs strategies that favor:

— **Recent Interactions**: The FIFO queue inherently prioritizes recent messages by removing older ones first.

— **Recursive Summarization**: This keeps track of important past events without requiring full retention of every message, striking a balance between recency and historical reference.

References:

1. MemGPT: Towards LLMs as Operating Systems — https://arxiv.org/pdf/2310.08560

Generative Ai Tools | Agentic Ai | Llm Agent | Memory Management

Written by Neeraj Kumar

63 followers · 7 following

Follow

Senior ML Engineer @ Adobe , PHD @ IIT Delhi, B-Tech @ IIT KGP Connect on Topmate for educational consulting, mock interviews - https://topmate.io/neeraj_kumar

Responses (1)

Write a response

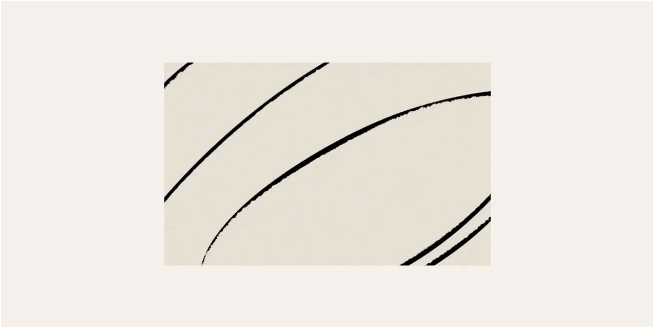
What are your thoughts?

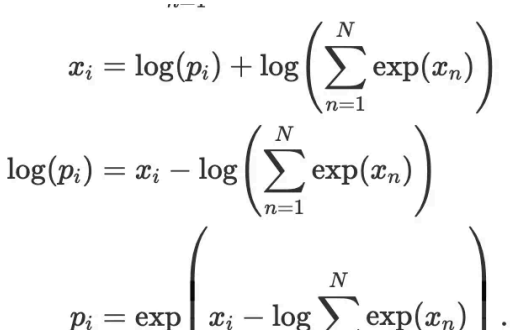
AI & Data Engineering
Dec 12, 2024

Great summary Neeraj!

Reply

More from Neeraj Kumar





$$x_i = \log(p_i) + \log\left(\sum_{n=1}^{N} \exp(x_n)\right)$$

$$\log(p_i) = x_i - \log\left(\sum_{n=1}^{N} \exp(x_n)\right)$$

$$p_i = \exp\left(x_i - \log \sum^{N} \exp(x_n)\right).$$

Neeraj Kumar

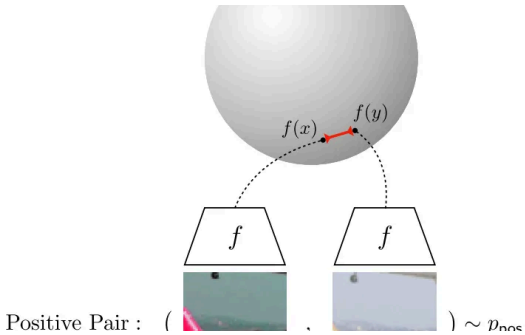
GEPA: Reflective Prompt Evolution —How LLMs Can Learn Without...

The Problem: Why Reinforcement Learning Isn’t Always Enough

Neeraj Kumar

Logsumexp trick and Flash attention- Part 1

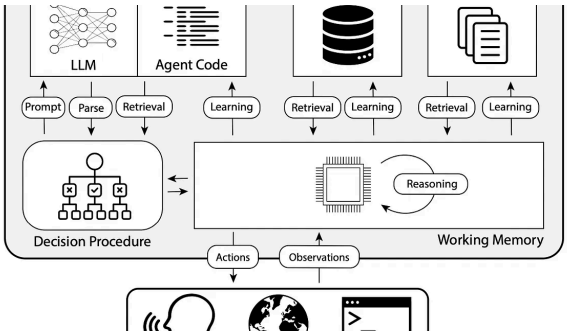
Softmax Concept:



Neeraj Kumar

Why Contrastive loss for unsupervised learning.

Unsupervised learning holds out the possibility that we can acquire transferable...



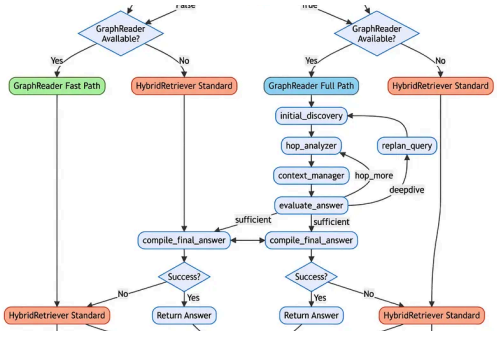
Neeraj Kumar

Memory Service for Agentic Applications

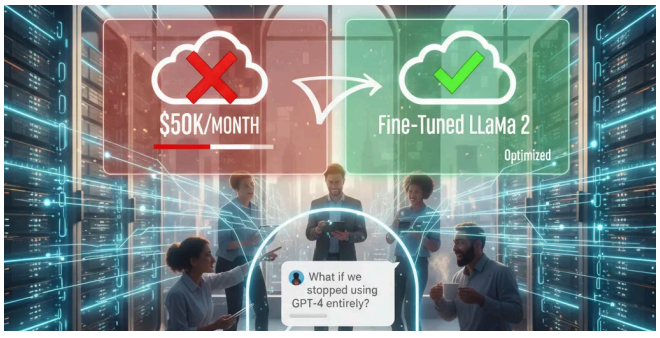
Language models are stateless: they do not persist information across calls. In contrast,...

See all from Neeraj Kumar

Recommended from Medium



Akash Goyal

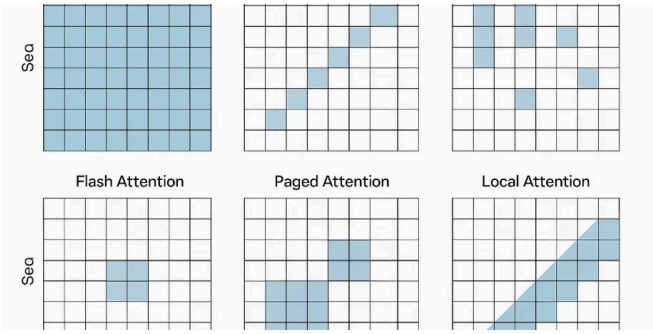


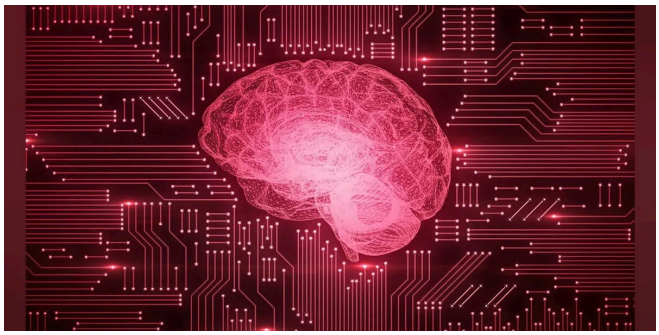
Amjń

GraphMind: A Full-Stack Agentic Architecture for Deep Retrieval a...

The SLM Revolution: Why We Ditched GPT-4 for Fine-Tuned...

The moment we realized bigger doesn't mean better





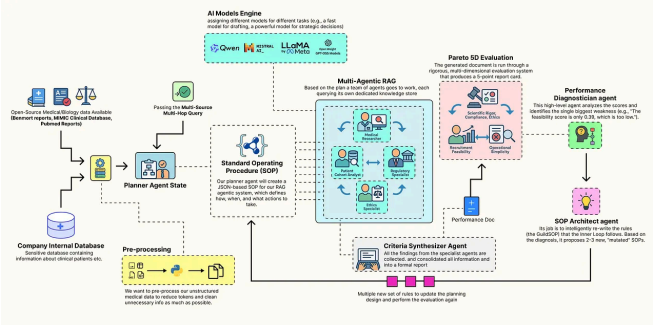
Aadishagrawal

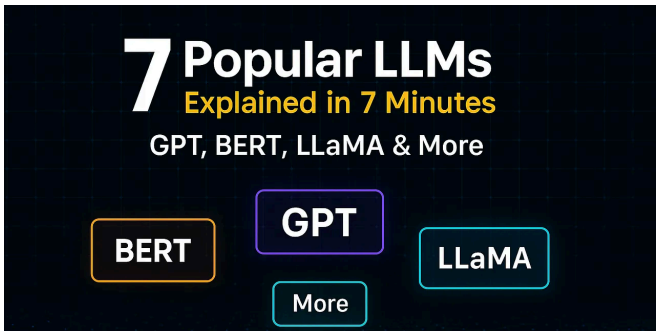
In The Chip Whisperer by Emily Yan

The Evolution of Attention Mechanisms: Scaling Transforme...

AI That Gets Silicon

The past two years didn't just add chatbots to engineering desktops—they changed how...





In Level Up Coding by Fareed Khan

Building a Self-Improving Agentic RAG System

Specialist agents, multi-dimensional eval, Pareto front and more.

Rohan Mistry

🔍 7 Popular LLMs Explained in 7 Minutes: GPT, BERT, LLaMA & More

Curious about what powers ChatGPT, Claude, or Google Gemini? Dive into the architecture...

See more recommendations