Research: Remembering Data for LLMs

Table of Contents

Short-Term Memory - The Context Window *2*

Long Term Memory - Beyond the Context Window *2*

Default Setting **2**

Adding Persistent Memory **3**

Retrieval-Augmented Generation (RAG) 3

Structured Long-Term Storage 3

Vector Embeddings for Memory 3

Extended Context Window 3

Summary 4

Memory in Practice - Commercial LLMs *4*

ChatGPT **4**

Other LLMs **5**

Context Window Limitations and Human Memory **5**

IBM Watsonx and AI Agents - Memory & RAG Implementation *5*

Introduction **5**

IBM's Approach to Short-Term vs. Long-Term Memory **6**

Watsonx Assistant **6**

Retrieval and Knowledge Integration **6**

Models and Agents **7**

Summary **7**

Controlling Memory - Making LLMs Remember or Forget *8*

Short-Term Remembrance - Pinning Information in Context **8**

Long-Term Remembrance - Storage **8**

Short-Term Forgetting **8**

Long-Term Forgetting **9**

Data Privacy Considerations in LLM Memory *9*

OpenAI and ChatGPT **9**

Watsonx and Enterprise AI **9**

Summary of Best Privacy Practices **10**

Conclusion *11*

Sources *12*

Short-Term Memory - The Context Window

As you may know LLMs do not have a “traditional” memory in the way humans do. They rely on the context window, which serves as the model’s short-term/working memory. The context window is essentially the text (prompt + conversation history) that the model can “see” at one time when generating a response. For example, OpenAI’s ChatGPT includes previous messages from the current chat session in its prompt so it can maintain context and continuity.

Now, onto the size of the context window. The size determines how much information the model can hold in its memory. For example, early version GPTs (3.5) had a context window about 4,000 tokens, whereas modern models (4.0) can support up to 32,000 tokens. Older content is dropped (forgotten) from the context window if a conversation exceeds the length of supported token limit. While higher token limit is great for understanding large contexts like books and documents, it comes with the tradeoff of performance and speed. As the context size increases, more tokens need to be processed each time, leading to higher computational costs and slower responses.

To sum it up, at each turn in a conversation, the application (e.g. GPT’s server) will package the user’s latest query together along with selection of previous dialogues according to the context size. If the conversation remains within the token limit, the model can refer to any earlier detail provided by the user. Nevertheless, if the limit is exceeded, older messages will be truncated (or summarized) to make space for new input. In other words, an LLM’s short-term memory is finite and session bound. Once you start a new chat or clear the history, the model has no innate recollection of prior interactions.

Long Term Memory - Beyond the Context Window

Default Setting

By default, LLMs do not retain information across sessions or once the context is cleared. It holds in its memory only whatever occurs in a single chat session. In technical terms, in ordinary conditions, these models perform no “online learning” or weight updates during usage - the model’s knowledge is static after training. Any new fact you tell to GPT is not truly learned by the underlying model to affect responses later. On the other hand, this is a key privacy and safety feature as inputs aren’t immediately put into the model’s global behavior. Shortly, in default, the model has no built-in long-term memory of user-specific data or past conversations.

Adding Persistent Memory

There are ways to overcome this limitation at either system of application level.

Retrieval-Augmented Generation (RAG)

The most popular and the approach we are planning to use it RAG where the system stores information in an external knowledge base (database or vector store) and retrieves relevant pieces to provide to the LLM. This way the model retrieves needed facts to its context window whenever appropriate, without having those facts hard coded in its weights.

RAG is often used to give LLMs up-to-date knowledge by pulling from uploaded documents, but it **also doubles as a form of long-term memory for past interactions**. An agent can index conversation transcripts or prior user data into a vector database and later retrieve the semantically relevant snippets when needed to remind the LLM.

Structured Long-Term Storage

In systems like IBM Watsonx Assistant, conversation state can be saved in session variables or databases. Session variables act as long-term memory during a user's interaction and can persist key information (like username, preferences, account details) across multiple dialog turns. At the end of a session, such variables may be cleared (depending on configuration), or they could be written to a database for persistence between sessions (if an application explicitly does so). This design lets an assistant "remember" things a user said earlier but it's the application's logic storing and re-injecting that data, not the core LLM itself.

Vector Embeddings for Memory

Many advanced AI agents maintain a vector embedding memory (as we are doing in SCORS AI). Each piece of information is embedded into a high-dimensional vector and stored. When the agent needs to recall something, it computes the embedding of the current context or query and finds similar vectors in its vector database. Those relevant items are then added into the LLM's context. This semantic search approach allows recall of facts even from conversations much earlier or beyond the raw token limit, if they were stored.

IBM's concept of "semantic caching" in Agentic RAG refers to this: caching previous queries, contexts, and results so the agent can reuse them later as needed. In essence, the agent builds up its own knowledge base of the interaction history or findings and keeps consulting it throughout the session (and potentially across sessions, if desired).

Extended Context Window

As noted, another way to achieve longer memory is simply to use models with huge context windows (like tens or hundreds of thousands of tokens). IBM's Granite models with 128k-

token windows are an example. Theoretically, you can paste an entire long conversation or large documents directly into the prompt without needing an external database for memory.

However, even 128k tokens, while very large, is still finite. Extremely long or multi-session interactions might still need truncation or external memory if they exceed this limit. Moreover, feeding extremely large prompts can be inefficient. For these reasons, many implementations combine strategies.

Summary

In summary, long-term memory for LLM-based agents is typically an engineered solution on top of the base model. The underlying model doesn't learn new user-supplied facts during chat (no weight updates occur on the fly). Instead, any cross-session memory or personalization comes from storing data externally and reloading it into the prompt when needed. If you want an AI to "remember" something permanently, you either fine-tune the model on that data or maintain it in a knowledge store that the AI can query.

Memory in Practice - Commercial LLMs

ChatGPT

In the public ChatGPT interface, the model remembers what the user has said earlier in the conversation by maintaining an ongoing chat history that is sent to the model with each request. The OpenAI server handles this by keeping a log of the conversation and prepends the last N messages (within token limits) to your prompt each time.

However, if you exceed those limits, ChatGPT will start to lose the earliest messages. Users sometimes notice this as the model suddenly "forgetting" a detail from a long conversation. It's usually because that detail was from too far back and the system dropped it to stay within token capacity. Some applications implement conversation summarization to mitigate this: when the chat gets too long, they summarize older parts and include the summary going forward instead of raw transcripts. For example, open-source frameworks like LangChain provide a ConversationSummaryMemory module that does exactly this.

It's important to note that ChatGPT does not carry over memory between separate chats. If you click "New chat" or the session times out, the next conversation starts fresh. The only way ChatGPT's underlying model "learns" new information is when OpenAI retrains or fine-tunes it on new data, but this training happens offline, not during your live interaction.

Other LLMs

Other commercial LLM chatbots, such as Google Bard, Microsoft's Bing Chat powered by GPT-4, have similar behaviors. They maintain context within a conversation thread but typically do not carry memory across distinct sessions or browser windows. Bing Chat, for instance, has a limit on the number of turns. Anthropic's Claude, known for a very large context, can intake something like 100k tokens of context which allows it to effectively "remember" an entire novel or a day-long chat in one go. Nevertheless, even Claude will not remember anything once a new conversation is started or if that 100k window is exceeded.

Context Window Limitations and Human Memory

The reliance on context windows means LLMs can sometimes act like they have amnesia if the conversation is long or if you switch context. They have no innate understanding that "I mentioned X yesterday, so I should recall that today" unless yesterday's information is provided again. This is a fundamental difference from human memory. Researchers are actively exploring architectural changes (like longer-context support or new memory mechanisms) to give LLMs more persistent memory, but current mainstream models treat each session independently by design.

One workaround now available in ChatGPT is the "Custom Instructions" feature (for ChatGPT Plus users). This lets you save some background information about yourself or your preferences that will be prepended to every new chat by the system. For instance, you could tell ChatGPT "I am a software engineer specializing in X and my preferred tone is formal" and save that as a custom instruction. Then, every conversation (until you change it) the system will include that in the prompt so the model acts as if it remembers those details about you.

IBM Watsonx and AI Agents - Memory & RAG Implementation

Introduction

IBM's Watsonx platform is designed with enterprise AI needs in mind, and this includes robust handling of memory and context in a controlled, privacy-preserving way. In Watsonx, you can build AI **agents** that incorporate both short-term and long-term memory to carry out complex tasks over multiple steps or sessions. IBM's literature often uses the term "Agentic RAG" when referring to combining Retrieval-Augmented Generation with agent-

like behaviors. AI agent in Watsonx can remember what it has already looked up or answered and avoid repeating work or contradicting itself as it moves through a workflow.

IBM's Approach to Short-Term vs. Long-Term Memory

IBM aligns with the general idea that short-term memory is the immediate context, whereas long-term memory is information preserved across sessions or for future reuse. In an IBM Redbook on watsonx.ai, they describe short-term memory as focusing on "the agent's immediate actions, thoughts, and observations during ongoing interactions", including data retrieved from tools or APIs in the moment. Long-term memory, by contrast, is a store of information accumulated over time: "summarized logs of prior interactions, user preferences, and other contextual info that influence the agent's behavior". For example, a Watsonx AI agent could retain a summary of what a user asked last week and the solution it provided, **so if the user comes back later, the agent can say "as I advised you last week…" – giving a personalized, continuous experience.**

Watsonx Assistant

The Watsonx Assistant provides a concrete mechanism for this. It has session variables (long-term memory within a session) and even the option to store conversation logs for a period. So, that the returning users can have continuity. A developer can choose to persist certain variables to a database keyed by user ID, effectively giving a memory across sessions (e.g., remembering a returning customer's preferences). IBM also allows **masking** of sensitive data in these variables for privacy – marking them as private so they don't appear in logs or are obfuscated with asterisks.

Retrieval and Knowledge Integration

In Watsonx's ecosystem, RAG is a central pattern for giving LLMs knowledge without training. The Watsonx documentation describes a typical RAG pipeline: a vector database holds your domain data, an embedding model fetches relevant chunks based on the query, and the LLM (like an IBM Foundation Model or an open-source model you deploy on Watsonx) then receives the query + retrieved data to generate an answer.

This means any enterprise data you want the model to use can be provided at query time rather than taught to the model weights. For memory purposes, you could also store embeddings of conversation history or past QA pairs in a vector store and have the agent retrieve those when relevant. This is essentially how "semantic caching" of past queries works. The **Agentic RAG** approach IBM touts go a step further by having autonomous agents that can decide when to query the knowledge base, when to use a tool, or when to

recall prior results. The agent's planning logic, aided by memory, helps it handle more complex workflows than a static single-turn RAG system.

Models and Agents

IBM Watsonx's **Granite models** themselves have been **optimized for long contexts** as mentioned. The fact that IBM has open-sourced 3B and 8B parameter models with 128k token windows indicates they see value in models being able to directly ingest large amounts of context (which can simplify memory handling for certain tasks). However, IBM also emphasizes efficiency: large context windows are great but can be wasteful if you stuff them with irrelevant text.

Thus, techniques like agentic chunking **and intelligent retrieval are recommended.** For example, break documents into chunks, embed them with metadata, and fetch only the most relevant pieces for the context. This way, the LLM isn't bogged down by extraneous details and latency stays low even as knowledge scales.

IBM's "What Is AI Agent Memory?" article explicitly states that advanced AI agents incorporate modules for memory and categorizes types such as **episodic memory** (remembering specific events or sessions) and **semantic memory** (stored facts, akin to a knowledge base). Implementing these in Watsonx often involves using Watsonx.data (a data lake-house) or other databases to store logs, and Watsonx.ai's tooling to fetch those logs when needed. IBM provides governance tools as well (Watsonx.governance) that can log all inputs/outputs for audit (optional). By default, IBM does not log your prompts unless you set up governance, protecting privacy.

Summary

In summary, IBM Watsonx gives developers the building blocks to create both short-term conversational memory and long-term knowledge retention but leaves it in your control. You might use context windows and prompt engineering for short-term state, and Watsonx's data stores or external databases for long-term memory (e.g., user profiles, conversation history, documents). The key difference is that in Watsonx (especially the enterprise context), all this data remains under your ownership and control, not used to improve IBM's models behind the scenes. This is critical for privacy.

Controlling Memory - Making LLMs Remember or Forget

Short-Term Remembrance - Pinning Information in Context

To have the model remember a fact during a session, ensure that fact stays within the context window for each prompt. You can achieve this by repeating the fact as needed or using a system-level instruction. This uses up some token budget, but guarantees the model sees it at each turn. In custom implementations, developers sometimes maintain a **"rolling buffer"** of recent messages (short-term memory) and a **persistent prefix** of key facts (like username or other constants) that is always prepended. This way, those key facts never fall out of scope.

Long-Term Remembrance - Storage

If you want the AI to remember something beyond the current session or beyond the context size, you need to save it somewhere and re-inject it later. This could be as simple as writing to a file or database after a chat session such as "store important details: user's account info, unresolved queries". Next time, retrieve and feed them to the model.

In a more sophisticated way, you can store embeddings of the conversation and use similarity search to pull up relevant past points when the user returns. In frameworks like LangChain (IBM Watsonx supports for agent development), you have ConversationMemory components, such as ConversationBufferMemory, ConversationSummaryMemory, VectorStoreRetrieverMemory, etc., that handle this automatically. They abstract the process of saving past interactions and loading them into prompts when needed.

Short-Term Forgetting

If during a conversation the user says "Please forget what I just told you" – can the model comply? The model cannot erase a piece of text from the prompt it already saw (it has no selective amnesia feature on the model side). However, the application could choose to remove or mask that portion from the conversation history going forward. For instance, if a user mistakenly reveals a password and says forget it, a diligent system could omit that turn from all future prompt assemblies. Some researchers have tried prompt tricks like telling the model "Do not use the information from message X going forward," but the model might or might not obey that reliably (and it still has that info in the prompt unless you strip it out). So, the safe way to forget is **eliminate the data from the context**. This is straightforward in a custom app; in ChatGPT's interface, the equivalent would be to delete the entire chat or turn off history. OpenAI's interface does allow users to delete past conversations and even individual messages now (in ChatGPT Enterprise/Business), which ensures those will not be used in context or for training.

Long-Term Forgetting

By default, forgetting is the norm – new session, no memory. But if you have implemented a long-term memory store, you need a strategy to purge or archive data when it's no longer needed. This is important for privacy compliance. IBM Watsonx provides controls for this: you can configure how long to retain chat history in Watsonx Orchestrate, with a default of 30 days and options up to 365 days. After the retention period, older messages are automatically deleted and become inaccessible, effectively forcing the AI to forget those conversations. If a user account is removed, Watsonx will delete all associated chat history immediately. Similarly, if you're using a vector database for memory, you might implement a policy to delete vectors older than N days or when a user requests deletion. The bottom line: **to make an AI forget, remove the data from any prompts and any persistent storage** that the AI or its retrieval components might draw upon.

Data Privacy Considerations in LLM Memory

When discussing how LLMs handle user-provided data, privacy, and data security are paramount. Especially, in enterprise settings like we have in Selco.

OpenAI and ChatGPT

- **Free ChatGPT/Plus with history on:** Conversations are stored indefinitely (for your account's history) and may be used for model training/improvement.
- **ChatGPT with history off:** Conversations not used for training and deleted from OpenAI systems after ~30 days.
- **ChatGPT Enterprise/Business or API usage:** Data not used for training at all. API data may be retained briefly for abuse monitoring but then deleted. Enterprise allows org-level control and auditing of conversations (admins can see logs) but those logs are isolated to your org.
- **User ownership:** OpenAI's terms say you own the content you input and output (they don't claim IP on your prompts or the answers), but you should still avoid inputting anything confidential unless you trust the service's privacy measures.

Watsonx and Enterprise AI

IBM's approach is to **put privacy first by design**. According to IBM's documentation and experts, "IBM does not use the content that you upload to watsonx or the output generated by a foundation model to train or improve any IBM-developed model." All your prompts, data, and outputs remain your own. **In fact, IBM states it cannot even** access **your prompts or model outputs unless you explicitly save them or choose to share them.**

When you use Watsonx.ai, all your assets (prompts, fine-tuned models, notebooks, etc.) are stored in a dedicated cloud storage bucket that is single-tenant and encrypted. IBM provides the service and infrastructure, but your data is isolated within your account's resources.

Notably, IBM's policies indicate that they **do not log prompt submissions or store unsaved prompts** beyond your session. If you run a prompt in the playground and don't save it, it's not kept. They do track usage metrics (for billing, performance) but this does not include the content of your prompts. No prompt or output is ever used to retrain IBM's foundation models without your permission. In other words, you won't suddenly find your proprietary sentence appearing in some future IBM model's knowledge.

For **Watsonx Assistant (Conversational AI)** specifically, IBM gives you tools to comply with privacy needs: you can mask variables (so sensitive info like names or IDs appear as ***** in any logs), you can control data retention as mentioned, and you can completely disable storing conversation logs if required. If you integrate Watsonx Assistant on your website, you might choose to keep transcripts for analytics or purge them immediately – it's up to you as the owner. IBM isn't snooping on those chats behind the scenes.

This difference in philosophy is crucial for enterprise adoption. Many companies are uneasy using ChatGPT directly with internal data because of the possibility (however small) that the data could leak or be seen by AI trainers. IBM's Watsonx is marketed as a solution to that: an AI platform "built for business" where you have **full control, privacy, and data isolation**. Indeed, an IBM blog emphasizes: "Your data stays private — from input to inference to storage. IBM respects data boundaries and does not reuse any of your AI content.".

Summary of Best Privacy Practices

- Use platforms or services that clearly do not use your data for anything beyond serving you. IBM Watsonx.ai is one, Azure OpenAI is another (Azure explicitly states your data is only yours and not used to improve the base model).
- Implement data retention policies: don't keep conversation logs longer than needed. IBM's default of 30 days can be a good practice, or even shorter for highly sensitive chats.
- Mask or encrypt sensitive fields. For instance, if a user provides a social security number that you need to remember within the conversation, store it in a secure variable and mask it in any logs or outputs.
- Provide a mechanism for users to request deletion of their data (and honor it by wiping their memory records from your storage).

- Avoid fine-tuning the model with raw user chat logs unless necessary – and if you do, scrub them first. Often, a RAG approach can achieve the needed specialization without risking hard-coding private info into the model.
- Keep the human in the loop for oversight when an AI is using memory to make sure it's not inadvertently exposing something it shouldn't. For example, when retrieving past info, ensure the info truly belongs to that user and is appropriate to include.

Conclusion

LLMs "remember" user-provided information primarily through **context windows** and clever system design, rather than by internal weight updates. In a single conversation, models like ChatGPT can appear to recall everything you've said – but under the hood, it's because that conversation text is being fed back into the model each time (until it hits a limit). There is no magical long-term memory inside the vanilla model: if you wipe the context, the model forgets instantly.

To extend memory, developers use external storage and retrieval (long-term memory via databases, vector stores, etc.) or rely on ever-larger context windows. IBM's Watsonx platform exemplifies the state-of-the-art in this regard, giving tools for **short-term memory** (session context, rolling buffers) and **long-term memory** (knowledge bases, semantic caching) within a controllable, secure framework. Techniques like Agentic RAG enable AI agents to accumulate knowledge over multiple interactions while still grounding themselves in up-to-date information.

Crucially, the question of how long an AI remembers is not measured in clock time but in tokens and design choices. ChatGPT could "remember" a conversation from a year ago if that conversation is provided again in the prompt today. Conversely, it could forget what you said 2 minutes ago if the context window overflowed or the session was reset. IBM Watsonx allows configurable retention – for example, automatically purging chat records after 30 days for compliance – ensuring that any long-term memory is explicitly managed and not indefinite.

For a consulting firm building AI solutions like Selco, these insights mean you should carefully architect your AI's memory:

- Decide what the AI truly needs to remember to serve the user and for how long.
- Use **retrieval mechanisms** to supply that memory on demand rather than enlarging the base model (which keeps the model general and reduces risk of leakage).
- Implement controls for **forgetting**, both for user requests and for routine cleanup.

- And choose partners or platforms (like Watsonx or a self-hosted model) that align with your privacy requirements, so you can confidently tell your clients that their data won't be siphoned away or used without consent.

In essence, current LLMs are powerful text predictors with no innate long-term memory – but with thoughtful system design, we can impart them a memory: one that we **program** and **govern**. This gives us the best of both worlds: AI that remembers what we want it to, for as long as we want, and forgets the rest. By leveraging techniques like context management, RAG, and IBM's agent frameworks, and by upholding strict privacy practices, we can build AI solutions that are both intelligent and trustworthy with user data.

Sources

- Belcic, Ivan, et al. What Is Agentic RAG? IBM Think Blog, 2023, https://www.ibm.com/think/articles/what-is-agentic-rag.
- Bhushan, Anand. "ChatGPT, Data Privacy & You: What Every Enterprise Employee Should Know." Medium, 2025, https://medium.com/@anandbhushan/chatgpt-data-privacy-you.
- GuardianOwl Digital. "ChatGPT vs IBM Watson: Which Is Your Data Safer With?" GuardianOwl Digital Marketing, 2024, https://guardianowldigital.com/chatgpt-vs-ibm-watson-data-privacy.
- IBM Cloud Docs. "Using Variables to Manage Conversation Information." IBM Watson Assistant Documentation, 2024, https://cloud.ibm.com/docs/watson-assistant?topic=watson-assistant-variables.
- IBM Documentation. "Configuring Data Retention Time for AI Chat." Watsonx Orchestrate, 2024, https://www.ibm.com/docs/en/orchestrate?topic=configure-data-retention-time.
- IBM Research. "What's an LLM Context Window and Why Is It Getting Larger?" IBM Research Blog, 2024, https://research.ibm.com/blog/llm-context-window.
- IBM Think Blog. "What Is AI Agent Memory?" IBM, 2023, https://www.ibm.com/think/articles/ai-agent-memory.
- Riemer, Kai, et al. "AI Doesn't Really 'Learn'—Understanding the Limits of ChatGPT." Medium, 2025, https://medium.com/@kai_riemer/ai-doesnt-really-learn.
- Yenduri, Jyotsna G. "Day 20 — Securing Foundation Models in IBM Watsonx." Medium, 2025, https://medium.com/@jyotsna_yenduri/day-20-securing-foundation-models-in-ibm-watsonx.