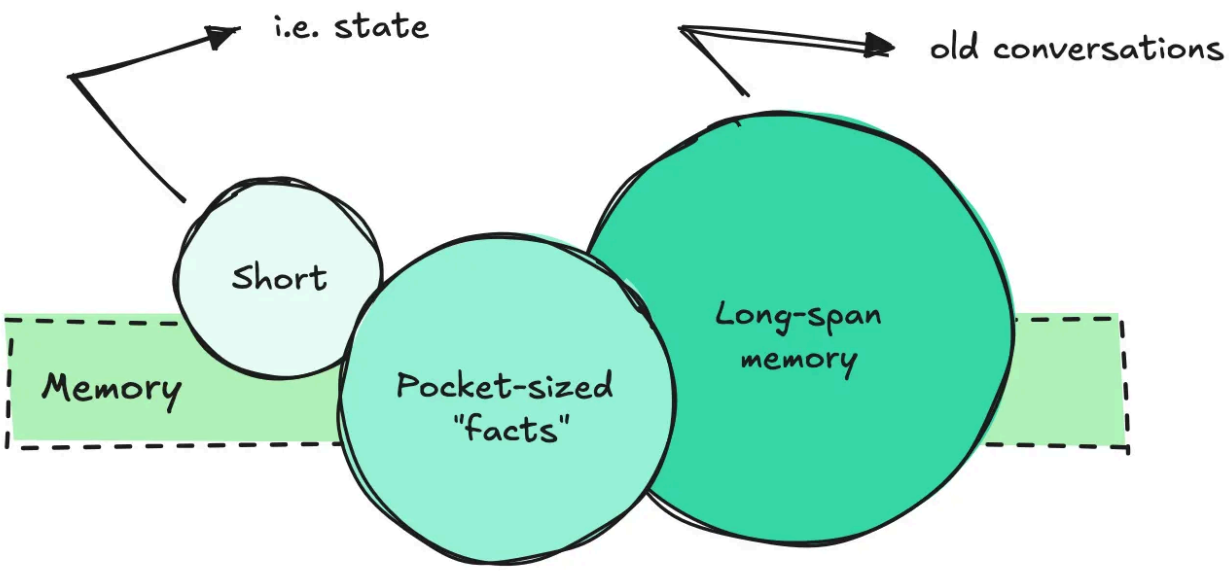
Data Science Collective

BUILDING LLM APPLICATIONS

Agentic AI: Building Long-Term Memory

The problem and current solutions

Ida Silfverskiöld Follow 10 min read · Sep 30, 2025



If you're not a member but want to read this article, see this friend link here.

If you've ever worked with LLMs, you know they are stateless. If you haven't, think of them as having no short-term memory.

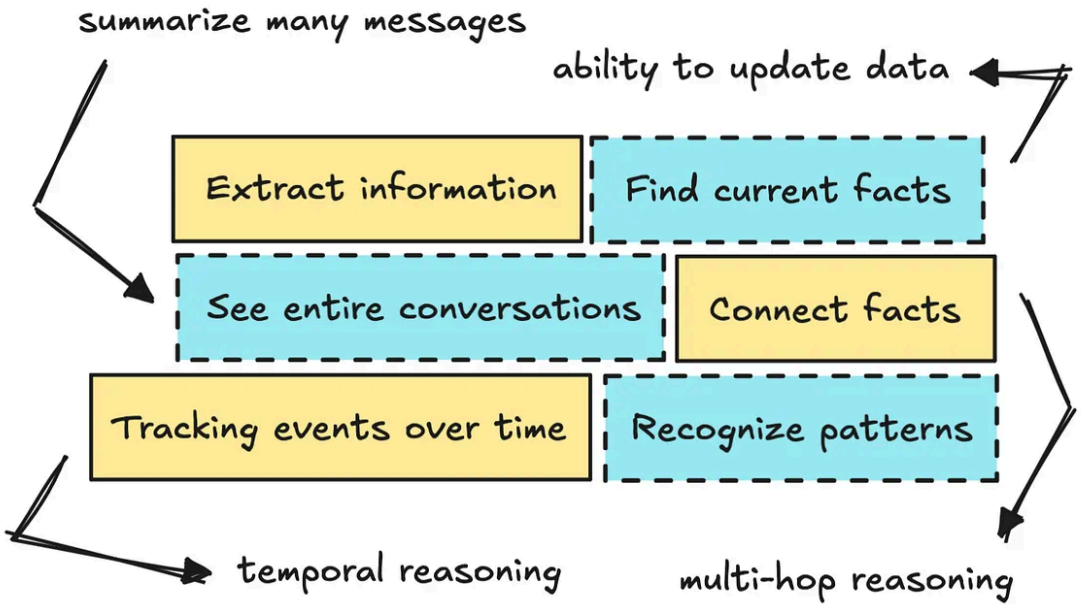
An example of this is the movie *Memento*, where the protagonist constantly needs to be reminded of what has happened, using post-it notes with facts to piece together what he should do next.

To converse with LLMs, we need to constantly remind them of the conversation each time we interact.

Implementing what we call "short-term memory" or state is easy. We just grab a few previous question-answer pairs and include them in each call.

Long-term memory, on the other hand, is an entirely different beast.

To make sure the LLM can pull up the right facts, understand previous conversations, and connect information, we need to build some fairly complex systems.



This article will walk through the problem rather than push a lot of jargon, explore what's needed to build an efficient system, go through the different architectural choices, and look at the open-source and cloud providers that can help us out.

Thinking through a solution

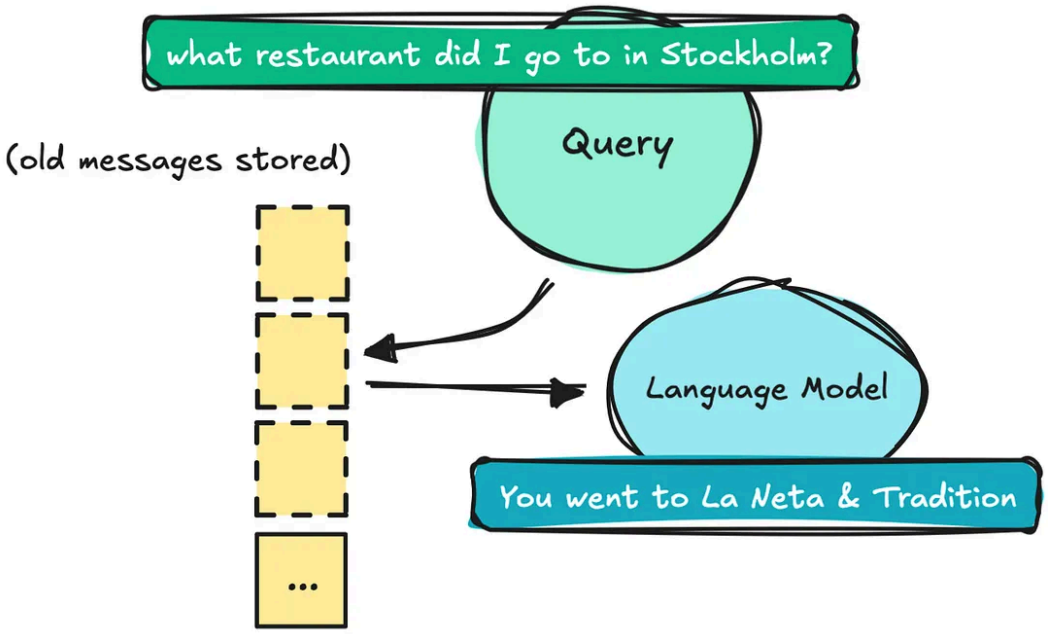
Let's first walk through the thought process of building memory for LLMs, and what we will need for it to be efficient.

The first thing we need is for the LLM to be able to pull up old messages to tell us what has been said. So we can ask it, "What was the name of that restaurant you told me to go to in Stockholm?" This would be **basic information extraction.**

If you're entirely new to building LLM systems, your first thought may be to just dump the entire conversations into the context window and let the LLM make sense of it.

This strategy though makes it hard for the LLM to figure out what's important and what's not, which can lead it to hallucinate answers. Once those conversations grow in size, you would also have to cap them.

Your second thought may be to store every message, along with summaries, and use semantic search (i.e. retrieval in RAG) to fetch information when a query comes in.



This would be similar to how you build standard naive retrieval systems.

The issue with this is that once it starts scaling (without doing anything beyond storing messages), you'll run into memory bloat, outdated or contradicting facts, and a growing vector database that constantly needs pruning.

You might also need to understand when things happen, so that you can ask, "When did you tell me about the first restaurant?" This means you'd need

some level of **temporal reasoning.**

This may drive you to implement better metadata with timestamps, and possibly a **self-editing system that updates and summarizes inputs.**

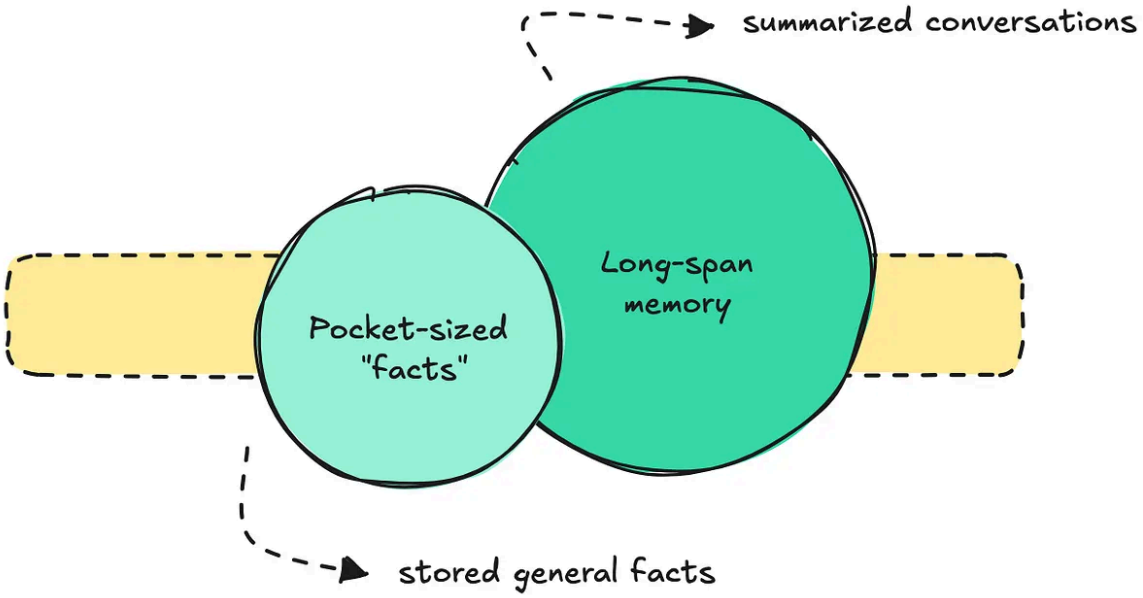
Although more complex, a self-editing system could update facts and invalidate them when needed.

If you keep thinking through the problem, you may also want the LLM to connect different facts (perform multi-hop reasoning) and recognize patterns.

So you can ask it questions like, “How many concerts have I been to this year?” or “What do you think my music taste is based on this?” which may lead you to experiment with knowledge graphs.

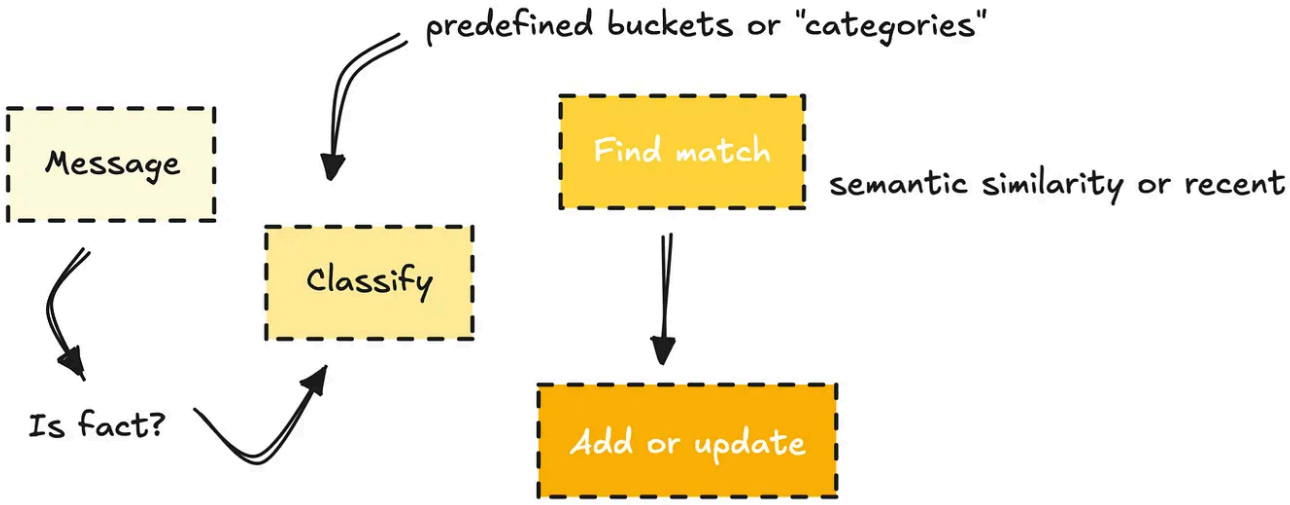
Organizing the solution

The fact that this has become such a large problem is pushing people to organize it better. Most teams seem to organize long-term memory as two parts: **pocket-sized facts** and **long-span memory** of previous conversations.



For the first part, pocket-sized facts, we can look at ChatGPT's memory system as an example.

To build this type of memory, they likely use a classifier to decide if a message contains a fact that should be stored.



Then they classify the fact into a predefined bucket (such as profile, preferences, or projects) and either update an existing memory if it's similar or create a new one if it's not.

The other part, long-span memory, means storing all messages and summarizing entire conversations so they can be referred to later. This also exists in ChatGPT, but just like with pocket-sized memory, you have to enable it.

Here, if you build this on your own, you need to decide how much detail to keep, while being mindful of memory bloat and the growing database we talked about earlier.

Standard architectural solutions

There are two main architecture choices you can go for here if we look at what others are doing: vectors and knowledge graphs.

I walked through a retrieval-based approach at first. It's usually what people jump at when getting started. Retrieval uses a vector store (and often sparse search), which just means it supports both semantic and keyword searches.

Retrieval is simple to start with: you embed your documents and fetch based on the user question.

But doing it naively, as we talked about earlier, means that every input is immutable. This means that the texts will still be there even if the facts have changed.

Problems that may come up here include fetching multiple conflicting facts, which can confuse the agent. At worst, the relevant facts might be buried

somewhere in the piles of retrieved texts.

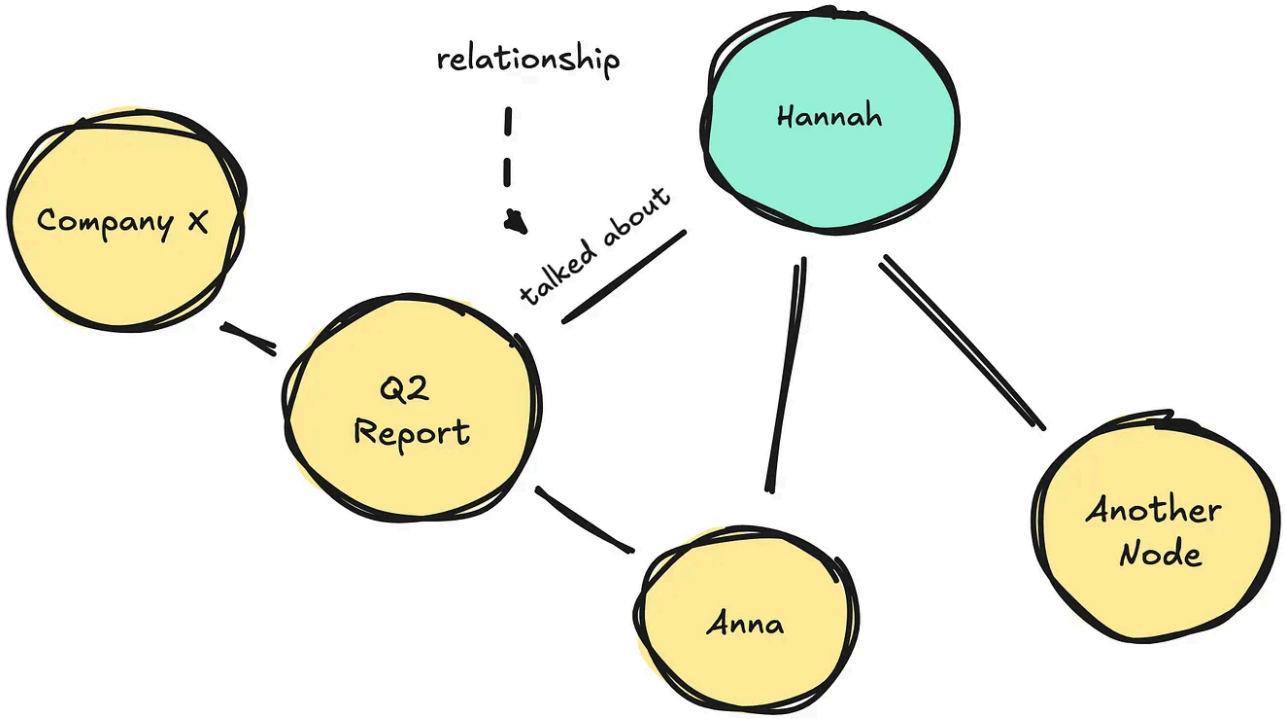
The agent also won't know when something was said or whether it was referring to the past or the future.

As we talked about previously, there are ways around this.

You can search old memories and update them, add timestamps to metadata, and periodically summarize conversations to help the LLM understand the context around fetched details.

But with vectors, you also face the problem of a growing database. Eventually, you'll need to prune old data or compress it, which may force you to drop useful details.

If we look at Knowledge Graphs (KGs), they represent information as a network of entities (nodes) and the relationships between them (edges), rather than as unstructured text like you get with vectors.



Instead of overwriting data, KGs can assign an `invalid_at` date to an old fact, so you can still trace its history. They use graph traversals to fetch information, which lets you follow relationships across multiple hops.

But as we mentioned, we could do something similar with vectors, by fetching old facts and updating them.

But because KGs can jump between connected nodes and keep facts updated in a more structured way, they tend to be better at multi-hop reasoning.

KGs do come with their own challenges though. As they grow, infrastructure becomes more complex, and you may start to notice higher latency during deep traversals when the system has to look far to find the right information.

It can also be pretty expensive to maintain.

Nevertheless, to conclude, whether the solution is vector- or KG-based, people usually update memories rather than just keep adding new ones, add in the ability to set specific buckets that we saw for the “pocket-sized” facts.

They also frequently use LLMs to summarize and extract information from the messages before ingesting them.

If we go back to the original goal (having both pocket-sized memories and long-span memory) you can mix RAG and KG approaches to get what you want.

Current vendor solutions (plug’n play)

I’ll go through a few different independent solutions that help you set up memory, looking at how they work, which architecture they use, and how mature their frameworks are.

Provider	Community	Founded	GitHub	⭐ Stars	Open Source
Mem0	🚀 Fast-growing	June 2023	followers 805	35.2k	✅ Apache-2.0
Letta	💬 Active dev community	Oct 2023	followers 482	17k	✅ Apache-2.0
Zep	🤝 Moderate community	Aug 2024	followers 313	11.6k	⚠️ Graphiti CE (Apache-2.0)
MemoRAG	🧪 Small research group	Sep 2024	followers 118	1.8k	✅ Apache-2.0
Memary	🧠 Niche community	April 2024	followers 45	2.3k	✅ MIT
Cognee	🔄 Moderate	Aug 2023	followers 122	5.8k	✅ Apache-2.0

Long term mem providers — I always collect resources in this repository | Image by author

Building advanced LLM applications is still very new, so most of these solutions have only been released in the last year or two.

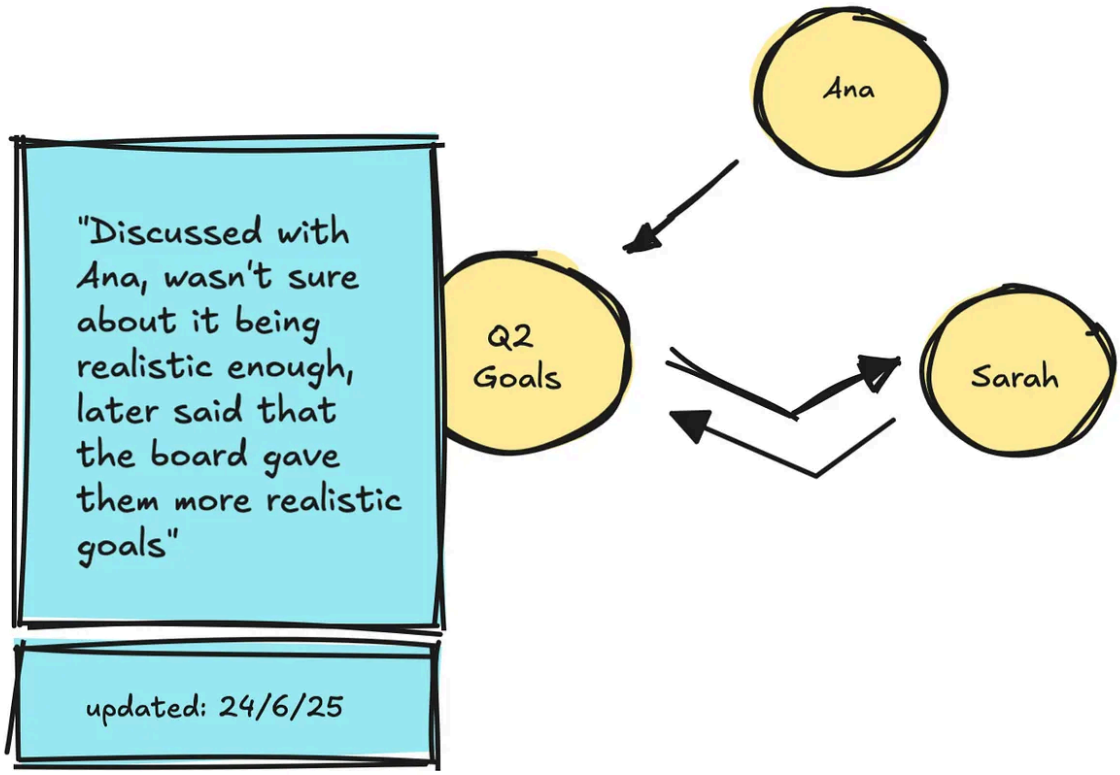
When you're starting out, it can be helpful to check out how these frameworks are built to get a sense of what you might need.

As mentioned earlier, most of them fall into either KG-first or vector-first categories.

Provider	Based	Optional KG	Self-Editing / Agentic	Rolling Summaries	Categories
Mem0	🧱 Vector	✅ Yes	✅ Yes	❌ Not explicit	✅ Yes
Letta	🧱 Vector	⚠️ Partial	✅ Yes	⚠️ Partial (memory blocks)	✅ Yes
Zep	🧠 KG	-	✅ Yes	✅ Auto chat summarization	✅ Yes
MemoRAG	🧱 Vector	❌ No	✅ Yes	❌ Uses long-range model	❌ No
Memary	🧠 KG	-	✅ Yes	⚠️ Plans "rewind" feature	✅ Yes
Cognee	🧠 KG	-	✅ Yes	❌ No auto summaries	✅ Yes

Mem provider features — I always collect resources in this repository | Image by author

If we look at Zep (or Graphiti) first, a KG-based solution, they use LLMs to extract, add, invalidate, and update nodes (entities) and edges (relationships with timestamps).



When you ask a question, it performs semantic and keyword search to find relevant nodes, then traverses to connected nodes to fetch related facts.

If a new message comes in with contradicting facts, it updates the node while keeping the old fact in place.

This differs from Mem0, a vector-based solution, which adds extracted facts on top of each other and uses a self-editing system to identify and overwrite invalid facts entirely.

Letta works in a similar way but also includes extra features like core memory, where it stores conversation summaries along with blocks (or categories) that define what should be populated.

All solutions have the ability to set categories, where we define what needs to be captured with the system. For instance, if you're building a mindfulness app, one category can be "current mood" of user. These are the same pocket-based buckets we saw earlier in ChatGPT's system.

One thing, that I talked about before, is how the vector-first approaches has issues with temporal and multi-hop reasoning.

For example, if I say I'll move to Berlin in two months, but previously mentioned living in Stockholm and California, will the system understand that I now live in Berlin if I ask months later?

Can it recognize patterns? With knowledge graphs, the information is already structured, making it easier for the LLM to use all available context.

With vectors, as the information grows, the noise may get too strong for the system to connect the dots.

With Letta and Mem0, although more mature in general, these two issues can still occur.

For knowledge graphs, the concern is about infrastructure complexity as they scale, and how they manage growing amounts of information.

Although I haven't tested all of them thoroughly and there are still missing pieces (like latency numbers), I want to mention how they handle enterprise security in case you're looking to use these internally with your company.

Provider	Enterprise Security
Mem0	🔐 Hosted with encryption, org/project roles, GDPR-friendly delete. Uses Graphlit (SOC 2 not stated).
Letta	☁️ Self-hosted or managed server. User auth & ID-partitioned memory. Graphlit-based. No public SSO details.
Zep	✅ SOC 2 Type 2. Encrypted at rest/in transit, access controls, JWT, and deletion API ("Right to be Forgotten").
MemoRAG	🏠 Self-host
Memary	🏠 Self-host
Cognee	🏠 Self-host

Mem cloud security — I always collect resources in this repository | Image by author

The only cloud option I found that is SOC 2 Type 2 certified is Zep. However, many of these can be self-hosted, in which case security depends on your own infra.

These solutions are still very new. You may end up building your own later, but I'd recommend testing them out to see how they handle edge cases.

Economics of using vendors

It's great to be able to add features to your LLM applications, but you need to keep in mind that this also adds costs.

I always include a section on the economics of implementing a technology, and this time is no different. It's the first thing I check when adding something in. I need to understand how it will affect the unit economics of the application down the line.

Most vendor solutions will let you get started for free. But once you go beyond a few thousand messages, the costs can add up quickly.

Provider	1K msgs/mo	10K msgs/mo	100K msgs/mo	1M msgs/mo
Mem0	🆓 Free	🆓 Free–$29	💸 $249	🏢 Enterprise (custom)
Letta	🆓 Free	💸 $20	💸 $750	🏢 Enterprise (custom)
Zep	🆓 Free	🆓 Free	💸 ~ $112.50	💸 ~ $1,237
MemoRAG	💻 GPU Server (~$150–300/mo)	💻 GPU Server (~$150–300/mo)	💻 Multi-GPU ($500+)	🖥️ Cluster ($1K+/mo)
Self-host	🖥️ Small VM (~$15/mo)	🖥️ Small VM (~$15–20/mo)	🖥️ Medium VM ($50–$100/mo)	🖥️ Large VM ($200+/mo)

"Estimate" mem pricing per messages — I always collect resources in this repository | Image by author

Remember if you have a few hundred conversations per day in your organization the pricing will start to add up when you send in every message through these cloud solutions.

Starting with a cloud solution may be ideal, and then switching to self-hosting as you grow.

You can also try a hybrid approach.

For example, implement your own classifier to decide which messages are worth storing as facts to keep costs down, while pushing everything else into your own vector store to be compressed and summarized periodically.

That said, using byte-sized facts in the context window should beat pasting in a 5,000-token history chunk. Giving the LLM relevant facts up front also helps reduce hallucinations and in general lowers LLM generation costs.

Notes

It’s important to note that even with memory systems in place, you shouldn’t expect perfection. These systems still hallucinate or miss answers at times.

It’s better to go in expecting imperfections than to chase 100 % accuracy, you’ll save yourself the frustration.

No current system hits perfect accuracy, at least not yet. Research shows hallucinations are an inherent part of LLMs. Even adding memory layers doesn’t eliminate this issue completely.

I hope this exercise helped you see how to implement memory in LLM systems if you’re new to it.

There are still missing pieces, like how these systems scale, how you evaluate them, security, and how latency behaves in real-world settings.

You’ll have to test this out on your own.

If you want to follow my writing you can connect with me at LinkedIn, or keep a check out for my work here, or via my own website.

❤

Data Science | Programming | Llm | Artificial Intelligence | Agentic Ai

Published in Data Science Collective

871K followers · Last published 1 day ago

Follow

Advice, insights, and ideas from the Medium data science community

Written by Ida Silfverskiöld

5K followers · 36 following

Follow

Building stuff | Connect 🤝 github.com/ilsilfverskiold & linkedin.com/in/ida-silfverskiold | ilsilfverskiold.com | Tech trends 🧡 docs.safron.io

Responses (6)

Bgerby

What are your thoughts?

Anshraj he/him

Oct 2

This is a great overview of the memory challenges in LLMs! This reminds me of when I was building a chatbot for customer service; getting it to remember past interactions was crucial for providing personalized support. The "post-it notes" analogy is... more

19 1 reply Reply

Yvan Callaou he

Oct 1

This is a brilliant and incredibly clear breakdown of the fundamental challenges in building long-term memory for AI. Your analysis of the trade-offs between vector-based RAG and Knowledge Graphs perfectly articulates the "why" behind the hybrid... more

19 1 reply Reply

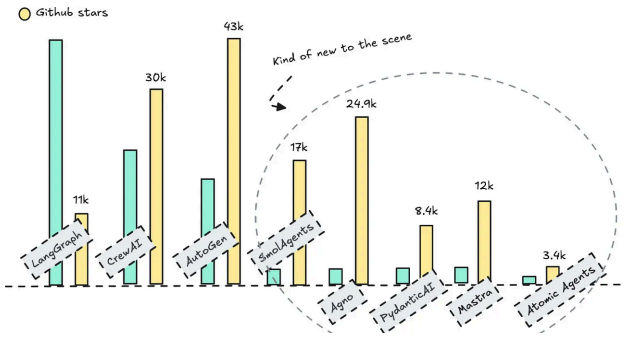
Aleksandra Todorovska
Oct 1

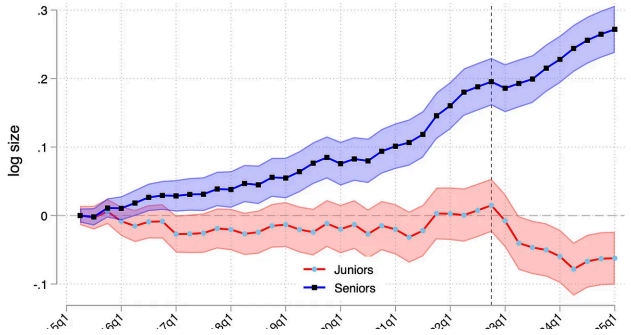
Great insights explained with simple words and easy on the eye drawings for beginners. Well done and keep em coming!

17 1 reply Reply

See all responses

More from Ida Silfverskiöld and Data Science Collective





DSC In Data Science Collective by Ida Silfverskiöld

Agentic AI: Comparing New Open-Source Frameworks

By looking at functionality and learning curves

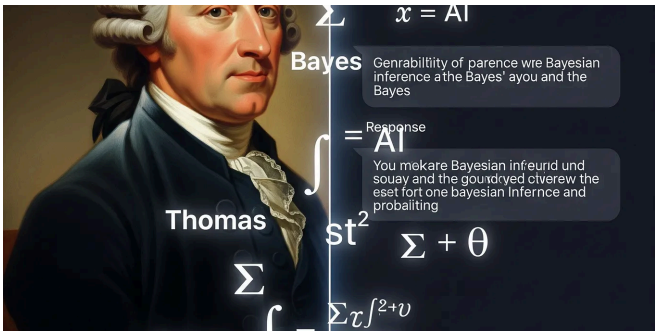
Apr 15 2.1K 67

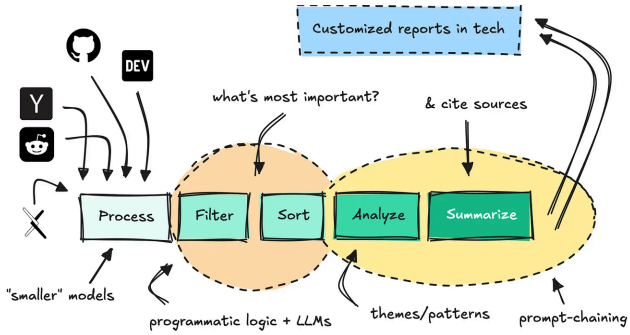
DSC In Data Science Collective by Andres Vourakis

AI and the Data Science Job Market: What the Hell Is Actually...

What aspiring, junior, and senior data scientists should know to stay future-proof

Sep 16 1.5K 57





DSC In Data Science Collective by DrSwarnenduAI

RAG is Just Bayesian Inference: The Mathematical Truth AI Companie...

How Silicon Valley Accidentally Reinvented 18th Century Mathematics and Called It...

Sep 6 724 25

DSC In Data Science Collective by Ida Silfverskiöld

Build Research Agents for Tech Insights

Using a controlled workflow, unique data & prompt chaining

Sep 10 442 4

See all from Ida Silfverskiöld

See all from Data Science Collective

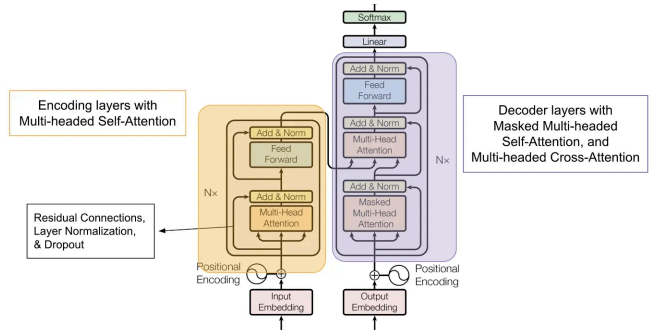
Recommended from Medium

In Data Science Collective by Erdogan T

Build Your Private Language Model: Local and Specialized For...

A complete step-by-step guide from setup to deployment of local language models, makin...

6d ago 644 4

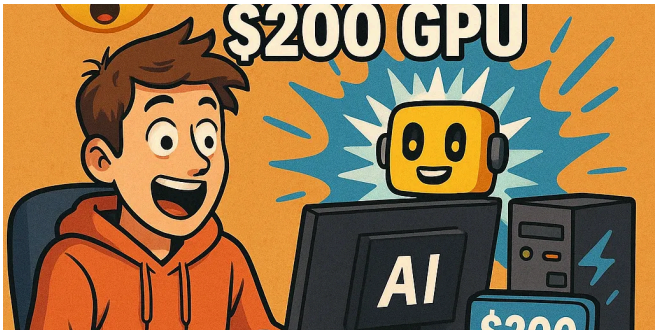


In Towards AI by Ashish Abraham

No Libraries, No Shortcuts: LLM from Scratch with PyTorch

The no BS guide to build, train, and fine-tune a Transformer architecture from scratch

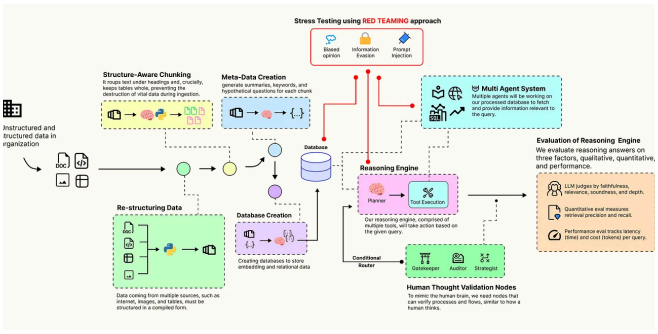
Oct 2 646 7



In Towards Deep Learning by Sumit Pandey

Meet oLLM: The Secret Sauce to Run Huge AI on Tiny Hardware

oLLM slashes LLM memory use: Run 100k context GPTs on 8GB GPUs. A lightweight...

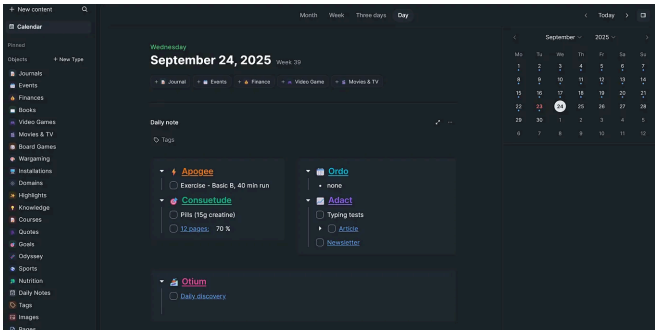


In Level Up Coding by Fareed Khan

Building an Advanced Agentic RAG Pipeline that Mimics a Human...

Ambiguity Checks, Multi-Tool Planning, Self-Correction, Causal Inference and more.

Oct 2 150 9



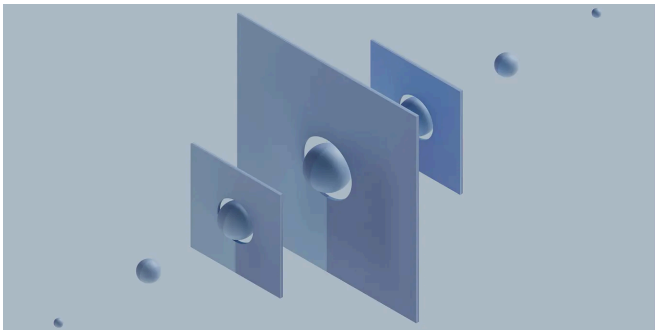
Tosny

7 Websites I Visit Every Day in 2025

If there is one thing I am addicted to, besides coffee, it is the internet.

Sep 23 2.5K 95

Sep 16 1.3K 19



In AI Advances by Kuriko Iwai

Mastering Autoencoders (AEs) for Advanced Unsupervised Learning

Explore the core mechanics of AEs with essential regularization techniques and...

6d ago 329 2

See more recommendations