Personal Context Repository for Large Language Models (LLMs)

By: Daniel Rosehill. 14-Dec-24

The purpose of this repository is to provide a high-level conceptual outline of a system for enabling users to create, edit and modify a personal contextual repository.

The purpose of the personal contextual repository is to provide an organized data source to improve and contextualize the inference of large language models.

The contention underlying this system is that in a future in which consumers increasingly use a plurality of large language models for their daily work, coupling one's contextual data store to any one model is going to be increasingly untenable. Furthermore, it is argued that the very nature of personal contextual data argues strongly that consumers should remain the controllers of such a data repository and not any one external company.

The question of how consumers can go about building and maintaining their own contextual data store is very much an unaddressed one. This document provides some thoughts about why this might be useful and suggests some initial paths towards developing a proof-of-concept for this purpose.

Comments, pull requests and invitations to collaborate on experiments slash prototypes slash POCs are more than welcome (my email is in the sidebar).

Overview

LLMs are incredibly powerful and useful tools, but whether they derive their knowledge from data they were exposed to during their training period or through more modern and sophisticated techniques like RAG or API calls, their knowledge is primarily general in nature. While a small amount of context can be built up within the confines of a continuing conversation, without express attempts to rearchitect, this contextual data source is both small and ephemeral. Outputs may be agreeable and sometimes informative, but overall will feel very generalist, robotic, and flat. Unlike human interlocutors, the models are being asked to engage in a simulated conversation without any background knowledge about the user

Enriching LLMs through personal context about the user greatly increases their utility.

Simple examples:

- LLMs which have knowledge of their user's movie preferences can instantly provide more guided, targeted and relevant movie recommendations.
- An LLM with simple contextual data about the user's professional life, such as their resume and career aspirations, can provide targeted recommendations for professional development opportunities, new jobs, etc (without needing to be explicitly instructed to do so).
- An LLM which understands the user's computer hardware can instantly provide compatible recommendations for upgrades, etc.

The fairly scant attention that has been paid to personal contextual data storage to date is a great pity. From a technical standpoint, contextual data can be exceedingly light, captured initially in simple JSON files and then vectorized. But this data, though "weighing" almost nothing in technical terms, can yield outsized results in dramatically improving the quality of inference.

Coupling the power of LLMs for reasoning with their increasingly multifaceted general knowledge and then layering onto that an understanding of the specificities of the user yields enormous value. When this works effectively (targeted prompting generating an output which draws upon both relevant data and personal context considered expertly by the LLM), LLMs become qualitatively different technologies than the search engines which preceded them as the favored means for information retrieval.

Contextual data has another important role to play in this respect. When the narrow confines of a user's individual circumstances are widened by an LLM considering a much wider pool of knowledge than your average human has access to or the capability to retain, What may seem like highly unique problems can be placed into their wider context, helping the user to understand that their problems (and successes) are part of the wider tapestry of human experience

Where ChatGPT's implementation of memory failed and succeeded

In March 2024, OpenAI added a memory feature to ChatGPT.

It has been met with mixed results from both expert commentators and ordinary users.

The main deficiency of the memory module is that it is only capable of storing up to 1,400 words of memories in total.

While the memory module has shown many how useful contextual data can be when used in LLM interactions, its utility, aAt the time of writing, is severely hampered by its very small size.

Contextual memory data should be a personal information store

Large language use is experiencing an explosion in popularity.

The frenzied pace of development activity in models, coupled with dramatic changes in pricing such as those recently seen with OpenAI’s shifts, have led many to adopt a slightly more vendor agnostic posture, trying out different models, seeing what works, and just as commonly using different models for different applications (the "pick n mix" approach!).

By trying to tailor their use of models to their different strengths, consumers can better test the waters of this fast moving scene without feeling overly caught up in any particular stream within it. Among enthusiasts, the use of open source and self-hostable LLMs is becoming increasingly mainstream, popular and feasible, providing alternatives to using cloud large language models altogether.

Fragmentation shows that memory context and user interface need to be decoupled.

The increasingly fragmented state of the LLM market, however, poses an unsolved challenge for the question of user memory and context.

It directly challenges the very premise that a memory or context module should necessarily be coupled with any one specific UI, or platform or model.

Today, a user subscribing to both ChatGPT and Anthropic Claude may find that ChatGPT has learned about their occupation through ongoing interactions, but that Anthropic is naive when it comes to this fact. Other attempts have been that specific LLM front ends have sought to centralized the user context data repository in this way. This too is an imperfect solution because it creates a dependency for that front end. One solution could be abstracting the context store one degree further by committing and versioning it in a self standing system.

It's bad from a data sovereignty standpoint, too

From a data sovereignty standpoint too it can be argued that the very idea of vendors attempting to monopolize users contextual data is inherently problematic. While efforts to regulate how companies manage very small pieces of PII are legion, almost no scrutiny has been afforded to the question of what kind of data governance framework those storing much more far reaching contextual data should be subjected to.

This is especially strange when the peculiar sensitivity of the data is considered. Beyond containing personally identifiable information (PII), it may also contain much more emotional and sensitive data like a user's life aspirations, challenges, etc. A strong answer from the open source community could be precisely what's needed to instill in users that this data should not be freely up for grab.

The idea: a personal context repository (self-managed / self-hosted / SaaS with safeguards)

Here is the central part of this idea and the argument underpinning it:

We're accustomed to having a cloud data storage pool in which we aggregate our information (consider Google Drive as an example).

Many would argue that Google have proven themselves to be *reasonably* responsible custodians of the data that we routinely commit into their servers in this matter. Those who feel less rosy in their assessment can self host document management systems. But there remain a variety of approaches open to users who wish to gather and manage this pool of data. When we need to connect that data to another source, say a CRM, the standard and entrenched approach now is to do that via integrations. In doing so, we assert our expectation and right to manage our own primary data source and ask that those wishing to tie into our ecosystem by providing us with additional services to find convenient ways to do so.

A similar posture could be adopted in the world of our use of large language models.We could have a place where our contextual data is gathered and which we use to enhance the capabilities of whatever AI systems we are interacting with, *(but particularly large language models given their easy accessibility and the fact that they are particularly well-suited for this kind of information).*

Much as we don't need to integrate our Google Drive with every single SAAS tool that we use, a similar case by case method could be used in the growing plurality of AI tools we use. In certain types of generative AI work, a personal context RAG pipeline might not actually be advantageous at all - consider, for example, text to image generation tools. For these tools, user preferences could continue to be provided through the more simple and established mechanisms, like user settings or small snippets of preference text that the get injected to the models by mechanisms like text precedence. For tools like chatbot-centric LLMs however, with a very far and wide ranging set of possible use cases, a much more wide-reaching and robust bank of contextual data about the user could have far more significant benefits.

How a detailed personal context "repo" could be developed and structured

The development of a personal context repository could be achieved through two means.

It's important to point out that these are not mutually exclusive. Additionally, it might be possible for users to choose one or the other path depending on their level of comfort with external providers having access to their personal contextual repository. Some degree of separation within the repository would almost certainly be desirable. For example, a user might choose to have a separate branch of the repository which they provide to the AI tools that they use in the course of executing their professional responsibilities, and another much more wide ranging branch used for personal projects.

A more skeptical user could only provide read access to LLMs, for example, while someone who was more comfortable with the idea could provide both read and write access, allowing the LLMs access to the data (ie, read access) while allowing them to write back modifications or additions to the data storage. If access could be. controlled at a more granular level, as is increasingly becoming the standard in other applications, the write permission could be fine-grained so that only creating new documents was permitted, but editing existing context or deleting it were prohibited.

Depending on how security controls were implemented and the user's preferences, one of the two following methods could be used for developing contextual data at scale and at pace:

The first is through deliberate context setting exercises. This method could be open ended in its implementation. Given the multi modality tools at our disposal today, various ways in which this could be conducted can be imagined. A user could dictate an hour of open ended discussion about their life story into an audio recorder, then run that through Open AI Whisper, and finally have that parsed in the database. Alternatively, more structured approaches could be used to streamline the development of an effective detailed and nuanced context repository.

The user, for example, may interact with a specialised context setting AI tool which “interviews” them and then gathers common points of context into discrete documents. This can be implemented very easily through the simple mechanism of a LLM assistant. A configuration text can be supplied informing the assistant that its purpose is to help the user to develop pieces of context data and that can be instructed to return with those when it feels it is gathered enough information about A discrete topic. The format can even be controlled so that the assistant is instructed that its data summaries should be provided in the form of JSON or whatever file format the user prefers.

This approach is highly flexible and allows for the user to choose what kind of interview they wish to conduct with the LLM. Assuming that the Assistant has access to the context repository as a whole, this could be a very wide ranging conversation, much as two friends might have in a pub. Or the conversation could be much more focused, deliberately dialing down on a couple of niche topics with the hope of developing deep context in these specific areas. Almost any type of information that could improve the accuracy of outputs could be included here. For a user with mental health struggles, who is using a model to prepare notes for therapy sessions. this might be a broad outline of their current problems. While this contextual data might need to be updated somewhat frequently, over the long term it might save the user a lot of time and having to repeat the same data over and over again. Career related data would be especially advantageous for anyone leveraging LLMs for brainstorming in career progression, job performance, continuous professional development, or looking for a new job. As in the previous use case, the advantages remain the same. The user does not have to repeat the same data irrespective of LLM they're interacting with, and the outputs provided by the model can be far more useful and targeted.

The second path to building up a context repository could come from large language models feeding data back into this repository during the course of ordinary interactions with the user. This process would mirror approximately the flow seen in ChatGPT (currently) in which users are not explicitly asked for their permission for data gathered in conversations to be written into the memory, rather, this happens as an autonomous background process during interactions. This provides a more fluid way for the contextual information to be gathered and parsed. The only drawback is the size constraint seen in the chat GPT implementation Another limitation that a user who rarely touches upon topics and then suddenly decides to talk about them will find that the context for that particular subject is essentially totally bare. As is often the case, combining methods might be the most effective. A robust backbone database, developed deliberately, would provide an excellent foundation for Interactions with any conceivable model. While light touch modifications and additions being fed in through ongoing use with the model could make sure that that primary vault of information remains updated and detailed without the user having to do much in the way of ongoing maintenance.

Given the sensitivity of the information contained, granular and rigid write and read access controls would need to be instituted (much as users need to carefully review requests for integrations with their Google Drive).

The important of context being an evolving data pool

It's important to point out that while some pieces of context are basically immutable (consider where a person was born) other pieces of personal context data are in a regular or periodic state of flux (consider, for example, whether a user is currently looking for a job). At some points in their life this will be true and at other points this will be false.

To be both as accurate and useful as possible, therefore, a personal context library would need to be rewritten at intervals. These small adjustments to the context pool could be updated in a continuous and natural matter (see: above). During a conversation, a user might tell the large language model that he or she has just started a new job. If the model had previously noted that the user was job hunting the file could simply be updated with the LLM using logical inference to determine that the job hunt was no longer in progress In this way the context repository could be continuously updated and in a sense be self correcting.

Paths to implementation & POC

In order to prove the concept, an experiment could be devised.

The following components of the stock are envisioned and would be needed:

- For the deliberate context setting method to function. a frontend for managing, editing, adding personal context could be developed. This store could be writing data directly to a vector database on the back end. This front end could consist of a system for navigating the existing context or much as one might use a CMS. Stored contextual data could be categorized into sections for ease of use and reference, and basic search functionalities could allow the user to quickly retrieve a specific context point that they need to edit.

- A RAG pipeline could be developed running between the context repository database and a large language model. For feasibility testing, this could be an open source but capable model such as LLAMA 3.3 70 TB. Any platform which allows the user to set up a RAG pipeline could be suitable for implementing a proof of concept.

- A series of evaluation prompts could be tested to benchmark the extent to which the model is utilizing the context available to it without needing to be explicitly instructed to do so by the user, something which cannot and should not be expected. For example, the user might ask the LLM for advice upon personal training they might undertake. The hoped-for response would be that the LLM guides its response through the vantage point of the context or about the user's career. The failure criterion here would be the LLM, providing generalist career advice or asking them for data that was already in context these responses demonstrating either ignorance of the context repository's existence or failing to know when using it would be appropriate.

If the ultimate idea is that users gain control over this data, and platforms and vendors need to react, then the availability of this feature within those platforms is a external decision that the user does not have control over. The best hope is probably that a critical mass of users could begin using this methodology, putting pressure on the providers to develop integrations for connecting this data pool to their models if that is users' preferred access mechanism.

Use-Case

While much innovation is currently taking place in the realm of RAG for business use cases (for example pipelines providing internal CRM data into internal LLM tools), very little attention has been given to the question of how memory and context could be optimally handled for private users.

Given that private use has been the major driving force of development in LLMs to date, this in time might be seen as a mistake.

For more technically advanced users with the ability to provision their own pipelines, this could be an area in which an open source project with a wide network of contributors could prevent the inevitable redundancy of effort involved when users individually work on these systems, each assuming that nobody else has considered this. Bringing together multiple perspectives on implementation would almost certainly result in a better implementation

The model proposed above necessitates careful attention to data management, but could spark an interest in the question of data sovereignty and federacy in the era of generative AI, which can only be expected to keep growing.