Personalized LLMs with RAG

DAVID LIN, University of Illinois Urbana-Champaign, USA
RUIYING HE, University of Illinois Urbana-Champaign, USA
JIADONG GUI, University of Illinois Urbana-Champaign, USA
RAY KO, University of Illinois Urbana-Champaign, USA

Existing LLMs perform well in generating high-quality responses based on user queries. However, their memory about the user is often limited and typically restricted to chat history. This results in a lack of access to structured personal information and the LLM response is not sufficiently personalized due to that. To solve this problem, we propose a modified RAG system. Basically, instead of traditional document retrieval, we build a multi-level database to store personal information and use it as contextual input for user queries. The database continuously extracts and stores information during user interactions with the LLM and gradually learns users' characteristics with precision and completeness, serving as an external "memory bank" for the LLM. As a result, our system is capable of more personalized and contextually relevant responses than traditional LLMs.

CCS Concepts: • **Information systems → Personalization**.

Additional Key Words and Phrases: retrieval-augmented generation, information retrieval, large language models

ACM Reference Format:
David Lin, Ruiying He, Jiadong Gui, and Ray Ko. 2018. Personalized LLMs with RAG. *ACM Trans. Graph.* 37, 4, Article 111 (August 2018), 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Large Language Models demonstrated excellent capability in understanding and generating human-like answers based on users' prompt. However, the answer will be quite limited if only based on the information in one query. In 2024, OpenAI first introduced the Memory feature, bringing long-term, cross-chat memory capabilities to ChatGPT. But according to the user feedback, the memory feature has limited capacity,and lacks transparency and reliability. It might overwrite previously stored information without clear user control or explanation of the criteria used. In addition, the system isn't able to effectively distinguish which information is personally significant to the user, raising concerns about the stability and utility of long-term personalization. In addition, privacy concerns related to LLMs have also prompted users to be concerned. To address this challenge, we propose a Personalized LLMs with RAG system. Basically, we propose an external multi-level database for storing structured personal information as a supplementary memory for LLMs. This makes models to incorporate relevant personal data based on semantic similarity with the user query, enabling more personalized responses.

Authors' Contact Information: David Lin, University of Illinois Urbana-Champaign, Champaign, Illinois, USA; Ruiying He, University of Illinois Urbana-Champaign, Champaign, Illinois, USA, larst@affiliation.org; Jiadong Gui, University of Illinois Urbana-Champaign, Champaign, Illinois, USA; Ray Ko, University of Illinois Urbana-Champaign, Champaign, Illinois, USA.

2 Target User

Our target users broadly include all individuals who engage in long-term interactions with LLMs. More specifically, they are individuals with a strong need for personalized interactions. For example, users who rely on LLMs for domain-specific tasks, like medical or legal professionals. Also, content creators who use LLMs as assistive tools also tend to require a high level of personalization.

3 Core functionalities

3.1 Overall Structure

The system consists mainly of two components: 1. User Information Collection and Storage 2. Query Augmentation and Response Generation.
For User Information Collection and Storage, refer to Figure 1, when users sign up, they are prompted to provide relevant personal information (divided into five categories, which will be detailed in a later section). The responses will be converted into vector representations using the model sentence-transformers and stored in the personal_info section of the structured external database, LanceDB. During users' interactions with LLMs, the system continuously extracts new structured information from user queries to update and enrich the database.
For query augmentation and response generation, when users submit a query to the LLM, the system converts the query into embedding vector and apply the HNSW algorithm to retrieve the top 5 most semantically similar personal information entries from the database. In addition, it retrieves relevant prompt history from the session memory. The system combines those components to form an augmented input, which is sent to the LLM for response generation.

3.2 Vectorization and Storage of User Information

Each piece of personal information is stored in the format "summarized text description + vector embedding of summarized text + tags". For the vector embedding, the format in our database is a L2-normalized embedding. To convert from human text to embedding, the system applies the sentence-transformers/all-MiniLM-L6-v2 model, followed by L2 normalization to ensure uniform scaling across all vectors. The reason that we choose to convert user information into vector representations is that it allows semantic information in natural language to be mapped into a high-dimensional space, making semantically similar sentences to be closer together, facilitating more effective similarity-based retrieval. In addition, we adopt

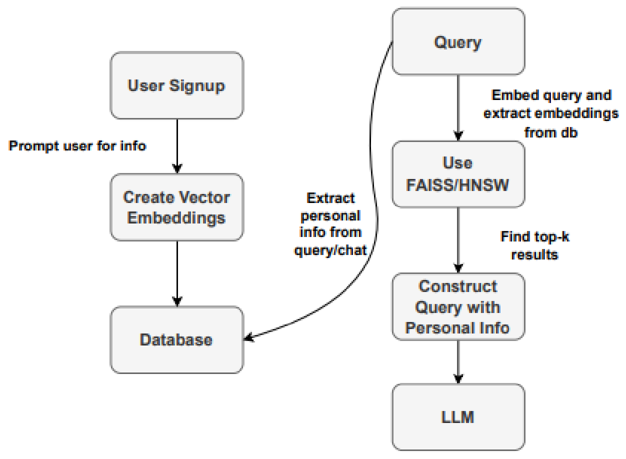
User Signup
Prompt user for info
Create Vector Embeddings
Database
Extract personal info from query/chat
Query
Embed query and extract embeddings from db
Use FAISS/HNSW
Find top-k results
Construct Query with Personal Info
LLM

Fig. 1. Overall System Structure

the all-MiniLM-L6-v2 model. This is because compared to larger models like BERT-Large, it is lightweight and offers significantly faster inference speed while maintaining comparable performance. Considering the scale of our project, it will be more suitable.

3.3 Dynamic Extraction and Update of User Information

In order to continuously refine user information based on interactions between the user and the LLM, our system also supports the extraction of structured personal information from the chat memory. Every time a user submits a new query, the system extracts the following 5 key semantic dimensions from the input: relevant topics, entities, query type, user intent, and preferences. The system performs the extraction through a prompt-based approach, where the system calls Google's Gemini API. The response from Gemini is then cleaned and parsed into structured information.

3.4 Data Expiry and Relevance Management

To prevent our database from growing indefinitely and to avoid redundant or outdated information, which increases retrieval time, we implement a modified FIFO-based data maintenance mechanism. The mechanism has a maximum capacity limit and performs regular cleanup. Specifically, we keep track of the number of times of each piece of personal information was used and when that information was last added. When the number of personal information is higher than the defined threshold, we move into the stage of deletion. During the deletion stage, we find all pieces of data that are less than 1 standard deviation of the average and sort those results by timestamp from oldest to newest. Then we remove the max_entries x 0.2 oldest results. If the amount of values less than 1 standard deviation of the average is less than max_entries x 0.2, then we randomly remove max_entries x 0.2 results.
Compared to simply deleting data in time order, our mechanism takes into account how frequently each piece of information is used. This allows us to prioritize removing of older and less frequently accessed data, which reflects the more important information to users in practical use.

3.5 Retrieval and Prompt Construction

After each user submits a query, the system will perform semantic similarity-based vector retrieval to fetch structured personal information as well as previous chat memory, then construct an augmented prompt that is sent to the backend large language model. For personal information retrieval, we store user's personal information as short text description, vector embedding of the text description and tags. Tags represent the core semantic labels of the user's personal information. In our design, we require tags to store five key types of information: relevant topics, main entities, query type, user intent, and user preferences. During the query stage, we follow the algorithm as Figure 2.
First, we use large language models to summarize the query in a concise informative sentence along with tags about the description. The information provided to the LLM includes not only the query itself but also the previous chat history. Then we use dense embedding retrieval (we will introduce the method later) to select the top N documents (Default N is 20). And from the top N documents, we embed the tags into vectors and assign each retrieved document with a rank score based on retrieval (e.g. the top result is assigned 20, the 2nd top result is assigned 19, 3rd top is 18, etc). After that, we compare all retrieved documents with the tags of the current query with cosine similarity. For each tag, if the retrieved document is above a certain cosine similarity score then we increment the rank score by 1. Based on the final rank score we select the top k documents (Default k is 5), but we also ensure each document has a rank score higher or equal to 18. The algorithm, compared to relying solely on LanceDB's default embedding-based retrieval ranking, improve personalization accuracy and result relevance. The LanceDB's default ranking method retrieves documents based on dense vector similarity to the query embedding, which might cause the potential semantic drift of embedding-based similarity. To solve the problem, we introduce additional semantic signals by leveraging structured tags, and the tags encode higher-level information such as relevant topics, entities, query type, user intent, and preferences. Combining dense embedding retrieval ranking and a secondary re-ranking based on tag cosine similarity, the layered ranking makes the final k retrieved documents not only globally similar to the query in vector space, but also semantically aligned with the user's actual intent, interests and preferences. In addition, we set the reserved thresholds for both the tag similarity score and the final overall rank score. This could help filter out low-quality tag matches and make sure that tag-level matching reflects true semantic relevance. It also guarantees that the final personalized documents which are supplemented into the query are of high semantic quality, so that the final prompt incorporates only meaningful user information.

The dense embedding retrieval method we mentioned above is HNSW method. Hierarchical Navigable Small World (HNSW) is a graph-based algorithm designed for efficient Approximate Nearest Neighbor search, and is the indexing method used by LanceDB. As shown in figure 2, the algorithm constructs a Small-World Graph structure, where vectors serve as nodes and semantic similarity defines the edges. During the retrieval, the search begins at a randomly selected entry point in the highest layer, and performs local greedy

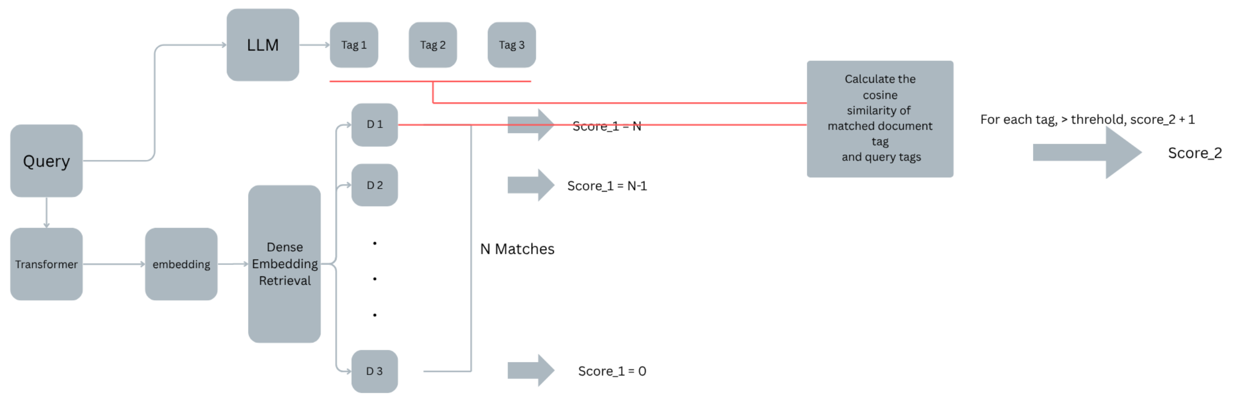
LLM
Tag 1
Tag 2
Tag 3
Query
Transformer
embedding
Dense Embedding Retrieval
D 1
D 2
D 3
Score_1 = N
Score_1 = N-1
Score_1 = 0
N Matches
Calculate the cosine similarity of matched document tag and query tags
For each tag, > threhold, score_2 + 1
Score_2

Fig. 2. Personal Information Retrieval

search at each level. The reason why we choose HNSW method is because our database is of moderate size and does not require complex partitioned indexing. However, our data is frequently updated, and HNSW supports dynamic insertion and expansion.

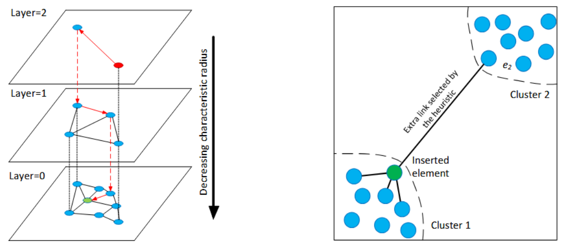
Layer=2
Layer=1
Layer=0
Decreasing characteristic radius
Extra link selected by the heuristic
Cluster 2
Inserted element
Cluster 1
e_2

Fig. 3. HNSW illustration

3.6 Questions designed

As mentioned in the overall structure section, when users sign up, they are prompted to provide relevant personal information. The personal information is used to construct a structured user profile database, enabling personalization in our system.
To achieve it, we design questions across five main categories:
1. user preferred interaction styles and preferred assistant roles
2. user professional background
3. user learning interests and study habits
4. user hobbies and lifestyle
5. short-term and long-term personal goals
The 5 categories are closely related to the user's purpose of interacting with the LLMs, their background knowledge, and their preferred answers style. Collecting this information is a key step for making our systems aligning with users needs.

4 Evaluation Results

We conducted a qualification evaluation of our system. Basically, we simulated a user and provided personal information by answering the designed questions. To ensure the experiment was controlled, we only provided three meaningful pieces of personal information and asked three questions to LLM, each related to one of the provided entries. Finally we compared the responses generated by the LLM before and after incorporating personal information. In the experiment, we provided the following three pieces of personal information: 1. I prefer High risk strategy. 2. I prefer using python, Vue and Flask. 3. I learn best by doing hands-on activities or building things.
Table 1 is a brief summary and analysis based on the comparison.
Based on the experimental results, we can see compared to using only an LLM, our system incorporates personal information to provide personalized responses. In addition, we can see the answer to the third question reflects the focus of the second question (on vector databases and LLMs), indicating that our system integration emphasizes previous chat history. In other words, it highlights the memory capabilities of the LLM.

Question	Original Answer without Info	Answer with Info
Assume I have $10,000 in assets. I'd like you to give me an investment recommendation.	with $10,000, prioritize establishing an emergency fund, consider a Roth IRA if eligible, and invest the remainder in low-cost, diversified index funds or ETFs, adjusting your allocation based on your risk tolerance and time horizon. Always consult with a qualified financial advisor for personalized guidance.	With $10,000 and a high risk tolerance: First, set aside $3,000–$5,000 for emergencies; Then, invest in a Roth IRA if you're eligible; Allocate the remaining funds to 100% equities via low-cost ETFs (e.g., VTI, VT, QQQ);Consider dollar-cost averaging for better entry timing; Reinvest dividends, maintain a long-term outlook, and stay disciplined.
I'm a beginner and I want to build a RAG system. Please give me some advice.	Focus on building a minimal viable RAG system. Start small, use a simple dataset, chunk it, embed with OpenAI, store in Weaviate, retrieve and answer using GPT. Try LangChain or LlamaIndex for easier setup.	Start small with a simple dataset; Use fixed-size chunking (e.g., 200–500 characters) with slight overlaps; Embed with OpenAI's text-embedding-ada-002; Store vectors in Weaviate, set up schema and use near_vector queries; Retrieve relevant chunks and use clear prompts with "no knowledge" fallback; Debug by printing retrieved chunks and prompts; Follow tutorials, iterate, and use environment variables for API security.
I want to learn statistics. Please give me some advice.	Lists the foundational knowledge that needs to be mastered. Provides suitable learning paths and resources, including recommended online courses and textbooks, and recommends R as the programming language. Emphasizes hands-on practice with specific suggestions, outlines key statistical concepts, highlights applications in the computer science field, and presents a workflow for statistical analysis using R.	Lists the foundational knowledge that needs to be mastered. Provides suitable learning paths and resources, emphasizing "Skip the overly theoretical treatments early on" and "Focus on Practical Application." Recommends online courses and textbooks, but avoids more theoretical options. Suggests both R and Python as programming languages. Emphasizes hands-on practice with specific suggestions. Outlines key statistical concepts and their applications in computer science, especially in the field of AI, with particular mention of applications in vector databases and large language models (LLMs). Presents a workflow for statistical analysis using Python.

Table 1. C

5 Conclusion

We proposed a Personalized LLM with RAG. We introduced an external structured memory system and effectively enhanced the personalization capabilities of large language models. The system collects user-provided personal information, vectorizes it, and dynamically updates it during user interactions. Then it combines each query with semantically relevant personal context and previous conversation history. Through controlled experiments, we demonstrated that the system provides significantly more personalized responses compared to sole LLM.

Acknowledgments

We provide an open-source implementation of our system. The code and resources are available at: https://github.com/davidlin4171/Personalized-LLM/tree/main.