How to Add Memory to RAG Applications and AI Agents

"The knowledge": Sometime in the last 5 months, I built a RAG application, and after building this RAG application, I realised there was a need to add memory to it before moving it to production. I went on YouTube and searched for videos, but I couldn 2019t find anything meaningful. I saw some videos, but these videos weren 2019t about adding persistent memory to a production-ready RAG application. They only talked about adding in-memory storage to a RAG application, which is unsuitable for a full-scale application. It was then that I realized I needed to figure things out myself and write a good article that would guide readers through the thought processes and steps needed to add memory to a RAG application or AI agent. Quick Note: If you are building with Streamlit, you can follow this tutorial to find an easy way to add memory to your Streamlit app.   --- Pre-requisites : 1. Before jumping into the discussion, I want to believe you already know what RAG is and why it is needed. If you 2019re unfamiliar with this concept, you can read more about it [here](https://www.datacamp.com/blog/what-is-retrieval-augmentedgeneration-rag). 2. I also want to believe you already know how to build RAG applications. If you want to learn how to build RAG applications, you can follow my [previous article](https://app.readytensor.ai/publications/how_to_build_rag_apps_with_pinecone_openai_langcha in_and_python_sBFzhbX4GpeQ). 3. For this tutorial, I used MongoDB as my traditional database, Langchain as my LLM framework and OpenAI GPT 3.5turbo as my LLM. But you can use any technologies of your choice once you have understood the workflow. 4. To follow along, `pip install` the libraries below. ``` openai python-dotenv langchain-openai pymongo ``` --- ## Now you are good to go ![15e5944a1.png](1-5e5944a1.png) --- # What Is Memory and Why Do RAG Applications and AI Agents Need Them? Let 2019s use ChatGPT as an example. When you ask ChatGPT a question like ` 201cWho is the current president of America? 201d`, it will tell you 2018` 2018Joe Biden 201c` and then if you go further to ask 201cHow old is he? 201c, ChatGPT will tell you ` 201c88 201d.` Now, here is the question: 201cHow was chatGPT able to relate the second question to the first question and give you the answer you needed without you being so specific in your question? 201d The simple answer to this is the presence of memory. Just like the same way human beings can easily relate to past experiences or questions, ChatGPT has been built to have memory which can help it know when you are asking a question related to the previous question. In my simplest definition, and with regards to RAG and AI agents, memory or adding memory to RAG applications means making the AI agent to be able to make inferences from previous questions and give you new answers based on new questions, previous questions and previous answers. So now that you have known what memory is, the question is: How Can I Add a Memory to My RAG or AI Agent? Here is the concept I came up with. Human beings have memory because they all have a brain that stores information, and they can answer and make decisions based on the information(data) stored in their brains. So to achieve this when building an AI Agent or an RAG application, you need to also give the RAG application a brain by including the following: 1. A database (for storing user 2019s questions, the AI 2019s answer, chat IDs, the user 2019s email etc) 2. A

function that retrieves users 2019 previous questions whenever a new question is asked 3. A function
that uses LLM to check if the current question is related to the previous one. If it is related, it will create a

new stand-alone question using the present question and previous questions. This question will now be embedded and sent to the vector database or AI agent, depending on what you are building. But if the
present question is not related to the past questions, it will send the question as it is. ## Creating a
Database for Storing the User 2019s Questions and AI 2019s Answers Below, I used pymongo to create a
Mongo database so you can have an understanding of the kind of fields you will need.
```python from pymongo import MongoClient from datetime import datetime from bson.objectid import ObjectId  #
Connect to MongoDB (modify the URI to match your setup) client = MongoClient(\"mongodb://localhost:27017/\") db = client[\"your_database_name\"] # The name of your database collection = db[\"my_ai_application\"] # The name of the collection  # Sample document to be inserted document = {     \"_id\": ObjectId(\"66c990f566416e871fdd0b43\"),  # you can omit this to autogenerate     \"question\": \"Who is the President of America?\",     \"email\": \"nnajivictorious@gmail.com\",     \"response\": \"The current president of the United States is Joe Biden.\",     \"chatId\": \"52ded9ebd9ac912c8433b699455eb655\",     \"userId\": \"6682632b88c6b314ce887716\",     \"isActive\": True,     \"isDeleted\": False,     \"createdAt\": datetime(2024, 8, 24, 7, 51, 17, 503000),     \"updatedAt\": datetime(2024, 8, 24, 7, 51, 17, 503000) }  # Insert the document into the collection result = collection.insert_one(document) print(f\"Inserted document with _id: {result.inserted_id}\") ```  In the code above, I created a MongoDB connection using MongoClient and connected to a specified database and collection in MongoDB. I then defined a sample document with fields like `question`, `email`, `response`, `chatId`, and `userId`, along with metadata fields such as` isActive`, `isDeleted`, `createdAt`, and `updatedAt` to track each entry's status and timestamps.  The _id field is assigned using ObjectId, which you can omit to let MongoDB auto-generate it. When insert_one(document) is called, the document is inserted into the `my_ai_application` collection, and MongoDB returns a unique _id for the document, which is printed to confirm the insertion.  Make sure you change your connection credentials and other specific information.  Now that you have created the database and have understood the kind of fields you need in the database, let 2019s now see how to use the database to create a memory.  ## Creating a Function That Retrieves Users 2019 Previous Questions Whenever a New Question Is Asked  Below, we are going to define a function that retrieves the user 2019s last 3 questions from the database using the user 2019s email and the chat_id.  ```python from typing import List  client = MongoClient(\"mongodb://localhost:27017/\") db = client.your_database_name collection = db.my_ai_application # no need to initialize this connection if you had already done it  def get_last_three_questions(email: str, chat_id: str) -> List[str]:     \"\"\"     Retrieves the last three questions asked by a user in a specific chat session.     Args:         email (str): The user's email address used to filter results.         chat_id (str): The unique identifier for the chat session.     Returns:         List[str]: A list containing the last three questions asked by the user, ordered from most recent to oldest.     \"\"\"     query = {\"email\": email, \"chatId\": chat_id}     results = collection.find(query).sort(\"createdAt\", -1).limit(3)     questions = [result[\"question\"] for result in results]     return questions   # Call the function past_questions = get_last_three_questions(\"nnajivictorious@gmail.com\",
```

\"52ded9ebd9ac912c8433b699455eb655\") ` ` `  You can change this to retrieve the last five or even ten questions from the user 2019s database by setting `.limit(5)` or `.limit(10).` But note: These questions, together with the new question will still be passed into a system prompt later. So, you need to make sure you aren 2019t exceeding the input token size of your LLM. Now that you have defined a function that retrieves the past questions from the database, you need to create a new function that compares the current question with the previous questions and creates a stand-alone question if needed. But if the new question has nothing to do with the previous questions, it will push the user 2019s question just as it is. Creating a function that creates a standalone question by comparing the new question with the previous questions Below we are going to create a system prompt called new_question_modifier and now use this system prompt within the function we will define. It is this system prompt that does the comparing for us. Check the code below to understand how it works. ` ` ` python from langchain_openai import OpenAI from dotenv import load_dotenv # Load your OpenAI API key from .env file load_dotenv() CHAT_LLM = OpenAI() new_question_modifier = \"\"\" Your primary task is to determine if the latest question requires context from the chat history to be understood. IMPORTANT: If the latest question is standalone and can be fully understood without any context from the chat history or is not related to the chat history, you MUST return it completely unchanged. Do not modify standalone questions in any way. Only if the latest question clearly references or depends on the chat history should you reformulate it as a complete, standalone legal question. When reformulating: \"\"\" def modify_question_with_memory(new_question: str, past_questions: List[str]) -> str: \"\"\"   Modifies a new question by incorporating past questions as context.   This function takes a new question and a list of past questions, combining them   into a single prompt for the language model (LLM) to generate a standalone question   with sufficient context. If there are no past questions, the new question is returned as-is.   Args: new_question (str): The latest question asked.   past_questions (List[str]): A list of past questions for context.   Returns: str: A standalone question that includes necessary context from past questions.   \"\"\"   if past_questions: past_questions_text = \" \".join(past_questions)   # Combine the system prompt with the past questions and the new question   system_prompt = f\"{new_question_modifier}\ Chat history: {past_questions_text}\ Latest question: {new_question}\"   # Get the standalone question using the LLM   standalone_question = CHAT_LLM.invoke(system_prompt)   else: standalone_question = new_question   return standalone_question modified_question = modify_question_with_memory(new_question=\"your new question here\", past_questions=past_questions) ` ` ` The code above creates a stand-alone question using the previous questions, the new question, and the new_question_modifier which is passed into an LLM (OpenAI) - But what do I really mean by a standalone question? A stand-alone question is a question that can be understood by the LLM without prior knowledge of the past conversation. Let me explain with an example 2026 2026 Let 2019s assume your first question is, ` 201cWho is the president of America? 201d` and the LLM answers ` 201cJoe Biden 201d` and then you ask ` 201cHow old is he? 201d` The question, `how old is he?` is not a standalone question because no one can answer the question without knowing whom you are talking about. So what the function above does is: It will look at your new question ` 201cHow old is he? 201c` and compare it with the former question ` 201cWho is the president of America? 201c` Then the LLM will ask itself, 201cIs the recent question related to the past questions? 201c If the answer is yes, it will now modify this new question to something like ` 201cHow old is the current president of America? 201c` or ` 201cHow old is Joe Biden? 201c` and then return this new question so that it will now be embedded and sent to the vector database for similarity search. But if the answer is no, it will pass your question just as it is. This modified question is called a `stand-alone question` because anyone can understand it even without knowing the previous conversation. I hope this is clear d83d de01 270c fe0f Finally, after the function has given you the standalone question, you can now send it to your embedding model and from there to your vector store for similarity search Note: All these steps must be in a single pipeline so that the output of one becomes the input of the next until the user gets his answers. I believe you understand what I 2019m saying d83e dd17 Also, don 2019t forget to try out different system prompts and know what works best for your use case. The system prompt I used here is just an example for you to build on. - IN CONCLUSION I developed this approach after thorough brainstorming, and while it works effectively for the most part, I 2019d genuinely appreciate any feedback you have. I'd also be grateful if you could share any alternative approaches you've tried that might improve upon it. See you in the comment section and thank you so much for reading HAPPY RAGING d83e dd17 d83d de80 You can always reach me on [X: 3rdSon__](https://x.com/3rdSon__) [LinkedIn: Victory Nnaji](https://www.linkedin.com/in/3rdson/) [GitHub: 3rd-Son](https://github.com/3rd-Son)--DIVIDER--"