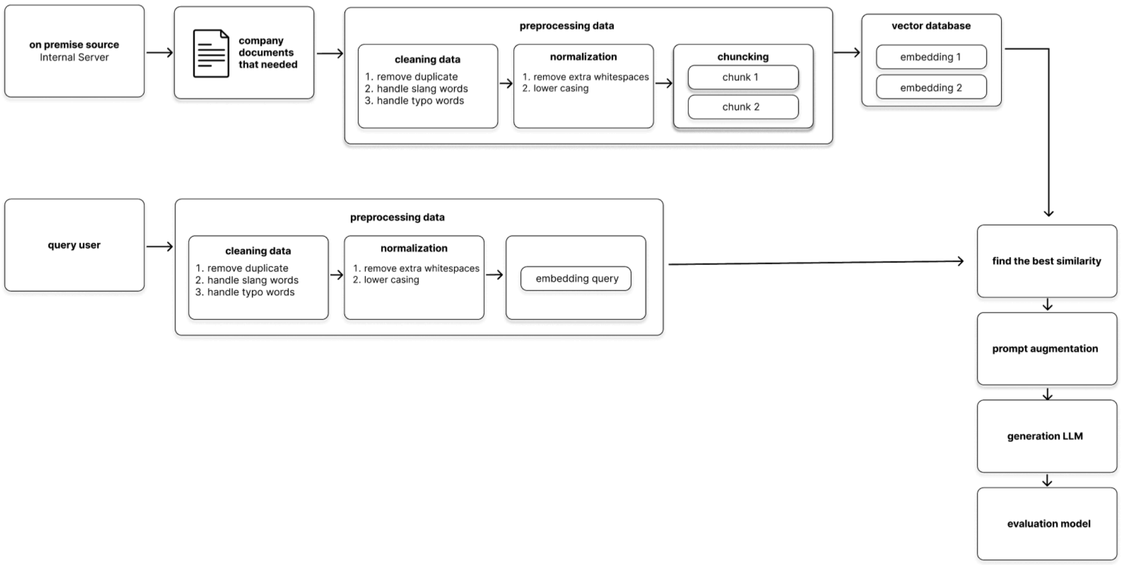
Question:
How would you create a LLM chat (a chatgpt-like solution or RAG applications) that uses only some company internal database? Please give details step by step, including data preparation, model evaluation, etc. It would be **a plus** if you could create strategies for reducing or preventing LLM hallucinations.

Answer:



My step if I will ever create a LLM chat with the company internal database can be seen on the diagram on top of this. The explanation will be explained below.

- From the internal database, I will get the documents needed from on premise source, such data from internal server, SQL, and uploaded documents. The dataset will be conducted in tabular form that consists of all of the texts.

  This table is an example result of the dataset

Documents	**Text**
Document 1	This is text from document 1
Document 2	This is text from document 2

- With the documents that needed to create a LLM, the next step will be preprocessing the data. Preprocessing data in the LLM pipeline needed to be done by cleaning the data, such as removing duplicate words (sometimes, there are some repetitive words in the data), handling slang words, and typo words. Moreover, normalization also needs to convert text into a standard format, such as removing extra whitespaces (/n) and lower casing of the text.

- Following this, the next step of it will be to create chunks. Chunk can be explained as creating smaller segments or groups of a larger text to be processed individually by the model. For example, if a document consists of 10.000 words, it could be splitted into 5 chunks of 2.000 words each. However, there are several other techniques to create chunks, and they need to be adjusted based on the data and business context.

  This table is an example result after creating the chunks

Chunk_id	document_id	text_chunck
1	10	“I work at Mandiri Sekuritas”
2	10	“And I learned a lot from it”

- The next step after creating the chunks is vectorization. Vectorization is a numerical representation (vectors) of texts that can be read and processed by computer. Vectorization is mostly done if the LLM doesn’t use a pre-trained embedding model, however since In this process, I assume the production will use a pre-trained embedding model.

  A pre-trained embedding model is a model that has already been trained on a large dataset to generate numerical representations that can capture the semantic meaning and does not have to be done manually. The chuck will be processed using an embedding model and their dimensions will depend on the specific pre-trained embedding model, since every model has a different embedding size.
- Furthermore, after the vectorization process was done by a pre-trained embedding model, it was stored in the vector database. It needs to be stored into a vector database because it allows for efficient similarity search and retrieval when querying the data from the user.

  This table is an example result after vectorization.

Chunk_id	document_id	text_chunck	vector
1	10	“I work at Mandiri Sekuritas”	[0.12, -0.05, 0.03, …]
2	10	“And I learned a lot from it”	[0.17, -0.02, 0.01, …]

- After that, the process will be moved to the query from the user. The query will also be processed by preprocessing data by cleaning and normalization. However, the query will not be created as a chunk because the query does not have many words rather than documents. Instead, it will be vectorized using the embedding model.
- Following that, once each data from the dataset and the query has been converted into a vector, the process will calculate the similarity between the

query vector and each chunk vector. There are also many techniques to calculate the similarity cosine similarity is one of the most popular because it measures similarity based on the angle between vectors. In this process, the best chunk will be the highest similarity score.

- After finding the similar chunk to the query vector, the next step will be prompt augmentation. This process includes adding relevant context from chunks to the user's query. The purpose of it is to provide the model additional information to generate more accurate answers.
  Example:
  Query: What is explained in Document 1?
  Instruction: Answer the question based on the context.
  Context: Document 1 explains the writer story when worked at Mandiri Sekuritas
- Subsequently, the next process will be generalization by applying LLM (Large Language Model), such as ChatGPT to generate the final answer. The LLM will be fed by the augmented prompt and need to answer based on the user's query and provide context that also needs to answer to the user when it doesn't know the answer. Otherwise, the model can generate an answer where it doesn't appear in the original documents that is known as hallucination. Hallucination often happens when the LLM tries to please or continue the conversation, even if it doesn't have any information about it. To mitigate hallucination, there are several strategies that can be used. First of all, it can use more advanced prompting methods to generate more accurate and relevant responses, such as chain-of-thought prompting that orders the LLM to explain the reason for the answer. Second of all, it can also use human review for every LLM answer. Human validations can verify whether the answer aligns with the documents and correct the errors. Moreover, as far as I know, LLM also can be done by fine tuning to generate the best answers and improving model performance. Lastly, the strategy to handle hallucination of LLM will be using technique such as word level, where the model can compare the generated answer to the source, for example:
  Answer: "I like to sleep at Mandiri Sekuritas"
  Original: "I like to work at Mandiri Sekuritas"
  Since the entities ("sleep" vs "work") did not match, the system can flag the answer to be possible hallucination and need to be fixed. Overall, all these techniques can ensure that the LLM output is aligned with the actual data.
- The last step of the LLM pipeline process will be evaluating the model to determine the correctness and relevance to the user query using metric evaluation. There are a lot of metric evaluations to evaluate LLM, such as F1-score, precision, recall, and MRR (mean Reciprocal Rank) that are usually used in LLM. These metrics can evaluate LLM based on different

aspects, such as precision can measure the proportion of answers that are correct and MRR with measuring how quickly the first relevant document appears. Overall, all the metric evaluations that use also need to be aligned with the purpose of evaluation and business purposes.