Semantic Search and Gen AI for Internal Data

Introduction

ChatGPT provides a much better experience for finding information compared to traditional search engines. A key strength of ChatGPT is its ability to generate high quality answers to questions rather than provide a list of links to potentially relevant sources. Specifically, ChatGPT is able to generate answers based on the information present in multiple sources and extract just the information relevant to the question. This makes the ChatGPT experience particularly appealing compared to search engines. However ChatCPT is limited in its "knowledge" to the publicly available data.

We discuss how to bring a ChatGPT-like experience to search over internal company data.

Background

Making sense of internal data is one of the top applications of Gen AI[1]. OpenAI has collaborations with large corporations[2] to organise their knowledge bases. There are dozens of ways of "chatting with your own data" including toolkits from Microsoft, Google, and ML companies like Cohere. These toolkits require developers to set things up (from ingesting the data to dealing with integrations / UX). Only the largest customers may get a solution with everything set up according to their needs rather than a toolkit. Furthermore, what happens under the hood behind the APIs is mostly a black box from the point of view of the customer.
There is a large number of constantly evolving open-source and proprietary models and technologies for semantic search, for generative AI, and for glueing the various components together. Two popular HuggingFace leaderboards present open source models for semantic search and for large language models.

This whitepaper describes how SerenityGPT brings a ChatGPT-like experience to internal data. It is a configurable product that leverages state-of-the-art models and consists of our own framework for evaluating and configuring the product for the specifics of customer data.

[1] See e.g., McKinsey's report The economic potential of generative AI: The next productivity frontier
[2] Morgan Stanley wealth management deploys GPT-4 to organize its vast knowledge base

Problem

Most companies have multiple sources of internal data such as customer tickets (e.g., Zendesk, Jira), knowledge bases (e.g., Confluence, Document360), internal comms (e.g., Slack, email), guides and training materials (e.g., Word docs on SharePoint), prior internal research (e.g., gdocs on Google Drive), documentation for internal and proprietary products. An employee (e.g., a support rep, a developer, an analyst, a manager, a marketing associate) needs to find some information in internal data sources (e.g., to respond to a customer query, to see what is known about X and who knows the most about it, to understand if a similar technical issue was seen before, to check if there were prior engagements with Y, to find information in technical documentation). The status quo way of doing this is via keyword-based search in each of the platforms: e.g., open Jira and use Jira Query Language to search over tickets, open Confluence and search over KBs, open Gmail and search over emails, open Slack and search over Slack... The process takes a lot of time even when a user knows exactly what they are looking for and is trying to find it again. When the user finally finds the right answer, there is no way to make it show up at the top of search results next time around.

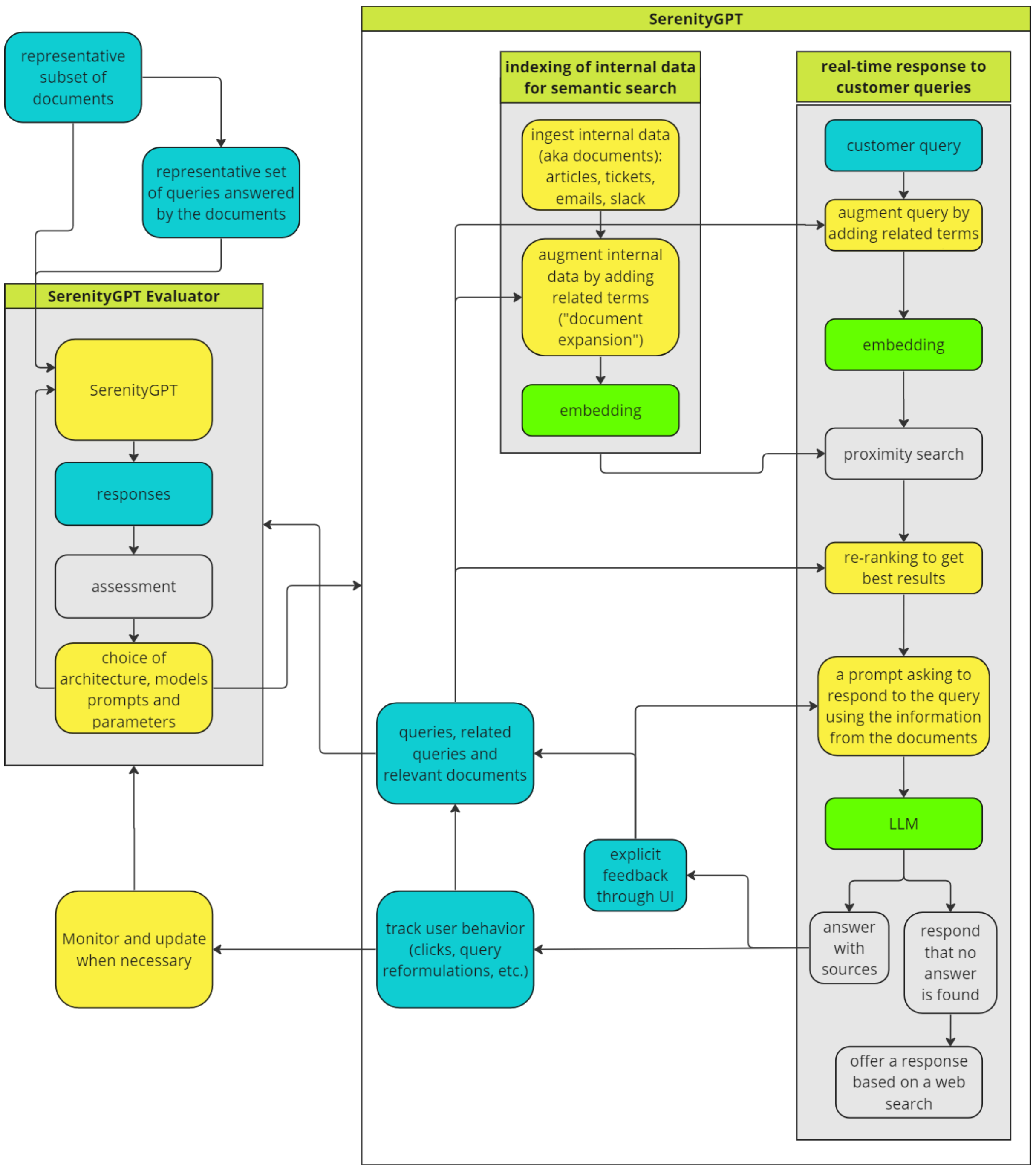
A key requirement is that some or all of the data is confidential and cannot leave customer servers at all or cannot be sent to third parties like OpenAI.

Solution

SerenityGPT solves the problem by bringing state-of-the-art models and utilising best practices to configure a solution for the specifics of customer data and needs. SerenityGPT offers the following functionality:

- a unified search interface
- a semantic search experience (rather than keyword based)
- a clear and concise answer mentioning the relevant sources
- learning based on how the system is used
- a way to specify correct answers to queries
- a whitebox solution that respects access rights and confidentiality of data: SerenityGPT can be configured so that no data leaves customer servers.

In the simplest configuration the UI includes a search box to enter a query and receive the answer together with links to sources. For example, the user might search for "Why am I getting a purple screen?" and the answer will be returned based on the information in the technical knowledge bases and support tickets with references to the corresponding sources. The two main back-end components making this possible are semantic search and a large language model (LLM). Semantic search finds documents that are likely to contain the answer to the query. The content of these documents is then passed to the LLM as part of the prompt asking it to find an answer to the query using only the information in the prompt. The rightmost branch of the diagram below shows the flow of a query through the system to get an answer.


SerenityGPT
indexing of internal data for semantic search
real-time response to customer queries
representative subset of documents
representative set of queries answered by the documents
SerenityGPT Evaluator
SerenityGPT
responses
assessment
choice of architecture, models prompts and parameters
ingest internal data (aka documents): articles, tickets, emails, slack
augment internal data by adding related terms ("document expansion")
embedding
customer query
augment query by adding related terms
embedding
proximity search
re-ranking to get best results
a prompt asking to respond to the query using the information from the documents
LLM
answer with sources
respond that no answer is found
offer a response based on a web search
queries, related queries and relevant documents
explicit feedback through UI
track user behavior (clicks, query reformulations, etc.)
Monitor and update when necessary

The configuration of the back-end of the system is illustrated in the middle and left parts of the diagram. The first step is to obtain a representative sample of the data to run through our evaluator module that compares configuration options such as models, prompts, and system configurations (e.g., whether or not to perform query/document expansion and re-ranking). The choices are also guided by the preferences of the customer regarding compute costs and latency.

Once the configuration choices are made, we ingest customer data. This happens once for all of the existing data and then new data is added automatically either in real-time or on a regular basis. The historical data ingestion involves creating an "index" of the data based on the meaning of the texts (i.e., a database of content including embeddings). Technically, we take paragraph-long chunks of data and represent its semantics in a searchable format. Documents are indexed together with metadata that stores information such as the link to the original document, the title, the author, the creation date, the category, the restrictions on who can access the document. All of this information can then be used to restrict and improve search results and answers.

Indexing can be vastly improved by making it aware of the domain-specific keywords or jargon. The reason for this is that language models (both embedding models and LLMs) are trained with standard vocabularies that likely do not include domain-specific terms and acronyms. This also helps with terms/acronyms that have different meanings in the specific data set. We incorporate a dictionary that a client can provide and/or automatically create one based on the data set. We do this via *document expansion* and *query expansion*[3].

We further improve the quality of search by

- using LLMs to generate additional document expansions (e.g., generate questions about the document and include them in the document)
- incorporating keyword search for queries containing jargon
- incorporating use-case specific document quality models based on document freshness, popularity, source, author, page rank of the document, etc.
- adding advanced (re-)ranking models that use query and document jointly to improve on embedding-based search

These choices are made in the evaluator module based on the performance metrics.

Search is self-improving based on user interactions. This includes sources clicked by the user, query reformulation and other behavioural metrics. For example, the documents that get viewed more get bumped up in the search results for the corresponding queries.
Search is improved based on direct feedback that can be provided via thumbs up/down buttons for the answer and for the sources.

The best search results are used by an LLM to produce a concise answer to the user question potentially coming from multiple sources. Through careful prompt engineering we make sure that the answers are based exclusively on the client's internal data, so the user can be confident that the information provided is correct. The answer comes with the links to the source documents from which the information is coming. If there is not enough information for a good answer, our prompt makes sure that LLM makes it explicit that not enough information is available. We fallback to Google results in these cases.

We also offer an interface to specify the document that contains the correct answer for a given query. The document will then be used to provide the answer for the given query and semantically similar queries. This allows the user to improve the search for difficult cases when the correct answer is not found by the search. Similarly, the user can mark out-of-date or incorrect documents to be excluded from search results. In addition to taking an immediate effect, the explicit feedback is used to further improve the system by learning the attributes of good and bad answers (for example, learning how quickly different types of documents get outdated and down-weighting the older documents accordingly). Optionally, we provide the ability to specify the correct answer directly for cases when it is not contained

[3] Document Expansion by Query Prediction

in any of the documents. The answer will then be treated as a new document and will be used to provide information for related queries.

Security of customer data

Our standard approach is to keep all confidential data on customer servers without sending it to third parties such as OpenAI. We have two options for indexing of historical data:

- indexing can be done on a customer server
    - we will need access to a GPU server for a few hours
- indexing can be done via encrypted API calls to our server. No data passed via API is retained on our server

We have three options for phrasing the answer via an LLM:

- an open-source LLM such as Llama 2 can run on a customer GPU server. We choose the size of the model based on performance and cost preferences.
- an open-source LLM such as Llama 2 can run on a SerenityGPT server and be accessed via encrypted API calls at query time. No data will be retained on the SerenityGPT server.
- GPT-4 can be accessed via Microsoft: no data will be shared with OpenAI and no data will be used for training models or retained by Microsoft for more than 30 days. Microsoft will retain the data for up to 30 days for abuse monitoring purposes only. See Data, privacy, and security for Azure OpenAI Service.

Links to learn more about semantic search and Gen AI

Andrej Karpathy's RNN implementation in 100 lines of code and a talk Visualizing and Understanding Recurrent Networks

- Play with these 100 lines of code to get a feel for RNNs. This will make it easier to understand transformers. Think about how RNNs are able to learn beyond the patterns from individual windows by keeping "state".

Check out Jay Alammar's step-by-step visual walkthrough of the transformer architecture: The Illustrated Transformer

Andrej Karpathy's lectures Neural Networks: Zero to Hero

- intro to neural networks, language modeling and backpropagation
- building GPT from scratch

Get the appreciation for embeddings

and

- https://huggingface.co/blog/how-to-train-sentence-transformers
  Intro to Sentence Embeddings with Transformers
- Check out the BERT and the E5 papers

See how it all fits together

- Stanford Webinar - GPT-3 & Beyond
- https://www.pinecone.io/learn/series/nlp/question-answering/

Prompts used in RAG

- ChatGPT + Enterprise data with Azure OpenAI and Cognitive Search by Microsoft

A summary of RAG techniques

- https://towardsdatascience.com/10-ways-to-improve-the-performance-of-retrieval-augmented-generation-systems-5fa2cee7cd5c