AI Chatbot Technical Report

In March, Ruben began investigating the potential of implementing a chatbot to answer questions about the contents of Hopkins Press books. Overall, this process has shown the creation and deployment of the chatbot to be possible, as well as the GPT API being accurate at answering questions about the content of the book.

Procedure/Flow

Setup

- **Generate embeddings**: This requires an EPUB format to automate with code. The book content is sent in chunks to the Embeddings API, which sends back the embeddings (vectors). The more similar two given vectors are, the more similar the text content they represent.
    - Ruben has built a python application that will be able to be run on local machines. The application uses a GUI to allow users to select chapters of an EPUB to get embeddings for and store the output in a desired format (for now, the only format available is JSON).
- **Host content and vectors**: the content and vectors need to live in a database somewhere. In this use case, a "vector database" is not needed: content and its vector representation can be stored side-by-side in a document or SQL database.

Usage (When a Customer Uses the Chatbot)

- **Embed query**: when a user submits a query, that query is sent to the Embeddings API to get a vector representation using the ***text-embedding-3-small*** model.
- **Fetch data**: the query vector representation is sent to the database to retrieve semantically similar sections from the book, along with a section's metadata.
- **Submit data to GPT**: the returned book content, along with conversation history and system messages are sent to the GPT to generate a human-readable response.
- **Display message on screen**: finally, the human-readable response is shown on screen to the user.

3 Separate Products

When Ruben began this investigation, he thought of this as a single application: the chatbot. During the process, he discovered that there are multiple moving pieces.

- **Chatbot Interface**: the most obvious part. This refers to the actual UI, the server-side connections to other pieces, its integration into Drupal, etc. This could be used for a Sales Assistant, but also for a post-purchase Text Analysis Assistant, as Stacey suggested.
- **OpenAI Embeddings**: mathematically representing text from books (as vectors). This can be used to find semantically relevant results for a customer's query, as well as for generating metadata, finding similar resources across the catalogue, extracting all content on a particular subject, or any other area where we want to evaluate content holistically and (near) instantaneously.
- **Prompt Engineering and Custom GPTs**: how the natural language is processed. Prompt engineering is vital to the other two; however, Embeddings can be used independently, either for business or open-source goals.

- art of this project. These prompts (and associated functions, API calls, and other tools in development) can be assigned to a custom Assistant. This reduces the amount of information sent with each API request and *potentially* enables the use of conditional logic within system prompts. This feature is still in beta testing from OpenAI.

The chatbot interface requires the other two; however, Embeddings can be used independently, either for business or open-source goals.

Costs and Limitations

The associated embedding costs for this are minimal. To embed a whole book, the text-embedding-3-small model uses between .5 and 1.5 cents. The expected cost would thus be stable and estimable based on the number of books to be embedded.

The associated GPT costs should also be minimal, but they may be more variable. While each request is limited by number of tokens, there is currently no usage restriction on length of conversation. (In other words, someone could abuse this by conducting an indefinitely long conversation with the chatbot.)

Additionally, there are some tasks the chatbot cannot currently complete:

- **Answering questions about a specific chapter**: when the user query is embedded and compared to text chunks in the database, there is currently no way of ensuring that the returned text chunks come from the right chapter. This is because there is nothing semantically linking the chapter number to any content in the chapter. **To implement this functionality**, we would need to build our own AI classification model to first sort user queries into categories and run a different database query depending on the sorted category. (There are existing tools that claim to do this, but Ruben has not yet tested them to satisfaction.)
- **Answering questions about the author, price, shipping, etc**: this information is not stored in the database, but even if it were, there would still be trouble answering a user query. This is for the same reason as the above point: there is no way to categorize what question a user is asking, so the code would never "know" when to request the information from the database. This *could* be achieved currently by sending *all* the book information with each request to the GPT API, but this would increase costs and could lead to the GPT hallucinating. Otherwise, this would require the AI classification model as well.
- **Searching for other books**: if the returned information about the book leads to the user indicating that this book isn't right for them, the chat will end with something like "I'm sorry I couldn't help you." Currently, this is not possible because the full range of books is not in the database. Even when the other books are in, this will still be limited by the lack of categorizing user queries. While expanding the vector search to include any/all books is easy, without the query categories, the code would never know "when" to query the database for different books

Additionally, the current chatbot demo was built using React.js, which lends itself to interfaces like this. In Drupal, there could be more restrictions or pain points that did not exist when developing the demo.

Conclusions

Thus far, the chatbot has worked remarkably well. Creating the embeddings, building the interface, and developing the prompts have required writing, testing, and tweaking code, but it has resulted in a chatbot that can answer user queries about the contents of a book. The limitations that the chatbot currently has can be solved by pre-sorting the user question before querying the database or GPT API, though this would require a (separate) custom-built AI tool. With user testing, the chatbot could be optimized and further refined to answer questions better and better.