Highlights

Question

Deep-pod

An enhanced podcasting experience

Background

Deep-pod is an app that allows users to interact with their favorite podcasts in new ways through chat summarization and topic extraction. Listeners usually consume podcasts to lighten the dullness associated with a time-consuming activity such as driving, working-out, cleaning the dishes, etc... Being preoccupied with such mechanical tasks, listeners find it hard to take notes, or bookmark an interaction. Users can like an episode or add an episode to their favorites, but these are undependable modes of interaction, since they do not pinpoint the user's intention; for instance, whether the user really liked an episode or just wanted to save it for later consumption, nor what they liked about the episode.

Deep-pod provides a better way to interact with a podcast through chat, summarization, and topic extraction. Users provide a URL to a specific podcast episode and the app will transcribe it and ingest it into a RAG pipeline that powers embeds the text into a vector database that can be searched against user queries, and invokes an LLM to generate a response based on the most relevant topics from the episode. Also, users can get a quick summary of the episode or enquire about the topics discussed in it, such as persons, orgainzations, artists, books, etc...

Data

The tool relies on user input to retrieve the relevant data. The user provides a URL to a specific episode (or just search by podcast name, however, that mode of interaction will only allow the users to interact with the most recent episode). Once the user provides the input, the tool downloads the episode, transcribes it, encodes the text using a text vectorizer, then embeds these vectors into vector database that enables semantic search.

Model

Transcription: The tool uses different flavors of OpenAI's open-sourced Whisper model for transcription. Users can either use the incredibly fast whisperer model that is run on Replicate, but they will need an API key to do that, or they can run the fast whisper model locally, however, that mode is best suited for systems with GPUs.

Embeddings: The tool provides two vectorization options, TF-IDF, and vector embeddings. TF-IDF relies on word counts to determine word relevance, while embeddings are language models that are trained with the specific goal of creating text vectors that capture the semantic meaning of the text. These are usually encoder-decoder models. The tool uses two embedding models: T5 and OpenAI's embeddings 3. T5 is open-source, and is the preferable option since its context size is smaller than OpenAI's embeddings 3. The embeddings 3 model was trained to provide flexible embeddings, which means that the outcome vector can be curtailed while retaining the same performance, at least in theory. Also access to OpenAI's embeddings 3 requires a paid API key.

Indexing: Once the text embeddings are created they are stored in a vector database that will servce as the data source from which the tool will generate responses to user queries. The tool provides two vector database options, Elasticsearch and ChromaDB. Elasticsearch indexes are designed to scale horizontally, and each index is broken down into smaller chunks called shards, that are distributed across the nodes for better performance and scalability. It uses inverted indexes for efficient term matching across documents. ChromaDB indexes on the other hand are designed to scale vertically, taking advantage of multiple CPU cores and large RAM sizes. It uses Hierarchical Navigable Small World graphs to navigate high-dimensional vector spaces swiftly, providing lightning fast vector searches. Elasticsearch can be run locally using Docker or on the cloud, where access is granted with both an API key and a cloud ID. As for ChromDB, it is open source and it's simple API does not require an API key. PS: There is a third option that doesn't use vector databases, that is Minsearch. Minsearch uses TF-IDF vectorization and stores the vectors in simply python dictionaries. This is not a scalable solution, but is kept for demonstration purposes.

Results

The app's design and pricing is user-determined, which means that the user chooses the settings and price (including a free option) based on their use case.

Transcription options: Both transcription modes utilize the same whisper model, so they produce more or less the same result. The main difference lies in performance and access to Replicate's services. If a GPU is available to the user, whether on their own mahcine or on the cloud, then local transcription is preferred and will be free. However, local transcription is slow if run on a CPU, taking aproximately half the episode time to finish (ex. the transcript of a 10 minute episode will be produced in 5 minutes). Replicate runs their models on their GPUs, so the user doesn't need to worry about performance, but they will need to have an API key to access Replicate's services.

Embeddings options: T5 is faster, due to its smaller dimension, and produces better results. OpenAI's embeddings 3 model's large dimension size delays the embedding process, and its performance is impacted when the dimensions are curtailed.

Indexing: Two metrics were used to compare the different indexing options: Hit-Rate and MRR. Elasticsearch produced the hightest MRR, while Minsearch produced the highest Hit-rate. As for performance, Minsearch and ChromaDB are much faster than Elasticsearch.

Interpretation

