

Technical Report: *Ask a Philosopher*

1. Abstract

I built and deployed a chatbot called *Ask a Philosopher* for chatting with one of two philosophers: *Aristotle* or *Confucius*. The chatbot uses Retrieval-Augmented Generation (RAG) to answer user queries. For *retrieval*, we use Octen-Embedding-0.6B, a text embedding model for dense embeddings, for finding the closest philosophical texts to the queries to use as reference. For *generation*, we use Llama-3.2-3B-Instruct, an open-source, instruction-tuned LLM from Meta. We only use the retrieved references when the cosine similarity beats a specified threshold for each philosopher. The app is deployed through Render, using Streamlit for the infrastructure. Both retrieval and generation steps are run through a serverless RunPod instance for GPU usage. It also incorporates one of Google's English neural Text-to-Speech systems for the philosopher responses. The chatbot is deployed at this url: <https://ask-a-philosopher.onrender.com/> A password is required.

2. Introduction

In this report I outline the technical design of our *Ask a Philosopher* chatbot. This chatbot was designed to mimic what it would be like to chat with an ancient philosopher. We decided to develop the system for two philosophers, one from western tradition--Aristotle--and one from eastern tradition--Confucius. We wanted the system to accurately represent the philosophers in two specific ways: 1) we wanted the chatbot to use a similar language style to the philosopher, and 2) we wanted the chatbot to accurately reflect the actual philosophical views of the philosopher. We also wanted the system to have an aesthetically-pleasing design, and incorporate Text-to-Speech (TTS) for reading aloud the philosopher responses. In the sections below, I outline the technical choices we made to achieve these goals.

3. Background & Related Work

This project was inspired by *Ask Jesus*, an AI project where users can submit questions to an AI system representing Jesus Christ (https://www.twitch.tv/ask_jesus). In a live Twitch stream, users can submit questions, and the AI system produces a response that is read aloud with TTS software. There is also an avatar representing Jesus, which includes minimal face and hand movements when the response is being read aloud. The *Ask Jesus* project is simultaneously intriguing, hilarious, and disturbing. In this work, we hope to include the intriguing and maybe hilarious characteristics, but hope to not be too disturbing. One choice we made was to *not* include a moving avatar for the philosophers, as that aspect of *Ask Jesus* may contribute to its uncanniness.

Modern chatbots typically use Large Language Models (LLMs) to generate responses to user queries. LLMs are incredible tools, but have noted drawbacks, particularly due to their tendency to *hallucinate*: produce incorrect or made up information. This issue occurs because LLMs are just complex statistical models of language: all they are trained to do is predict next tokens, given previously seen ones. While this generation step can be incredibly powerful when a model has seen enough data, there are no barriers in place to prevent the model from generating non-factual sentences. One method to reduce hallucination in LLM-based chatbots is to be make use of relevant reference documents for user queries. This method is named Retrieval-Augmented Generation (RAG). RAG consists of two steps: *retrieval*, in which we use Information Retrieval to search through a database of documents to find relevant reference documents for the query and *generation*, in which we give an LLM a prompt including both the query itself and the retrieved reference document, and ask it to generate a response to the query. We use a RAG-based system for *Ask a Philosopher*, in the hopes that it will improve the accuracy of the chatbot responses and reduce hallucination occurrence.

4. System Architecture

4.1 Overview

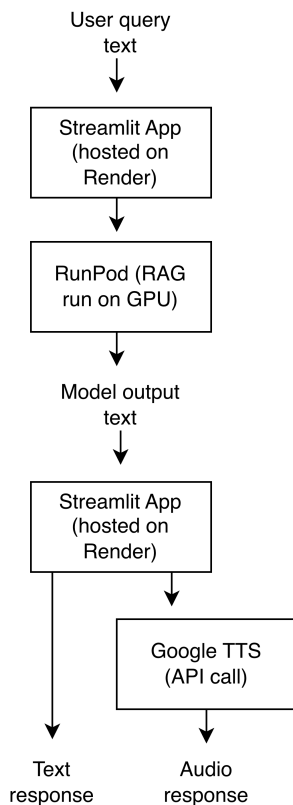
For our application architecture, we use Streamlit, a Python package for building applications (<https://streamlit.io/>). We use Render to host the Streamlit app (<https://render.com/>). Our RAG-based model is run on GPUs through Runpod (<https://docs.runpod.io/overview>). The model's returned text response is post-processed by our Streamlit app, including a Google Text-to-Speech API call to produce audio of the text (<https://docs.cloud.google.com/text-to-speech/docs/list-voices-and-types>).

For our model architecture, we use a modified Retrieval-Augmented Generation (RAG) approach. We collected primary texts from each philosopher as our reference documents. For each query, we first find the closest reference document to that query according to cosine similarity. If the cosine similarity value beats a specified threshold, the document is passed to the generation step. If not, we drop the retrieved document, and use a purely generative approach. We generate a response to the user query using an LLM. The prompt for the LLM includes instructions, the query, and optionally a retrieved reference document.

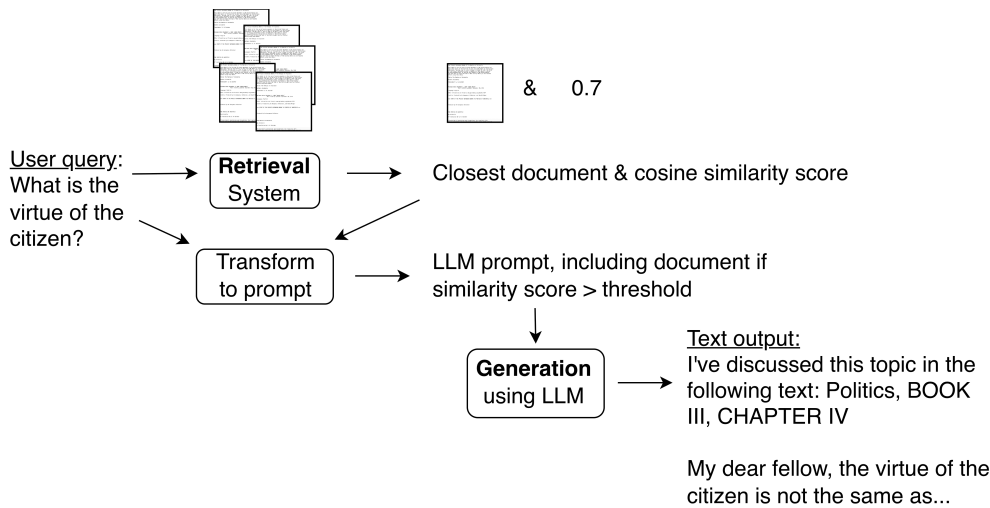
More details on the application and model architecture are provided in the sections below.

4.2 Application and Model Architecture Diagrams

An image of the application architecture is below.



An image of the model architecture is below.



More detail on both of these architectures are provided in the sections below.

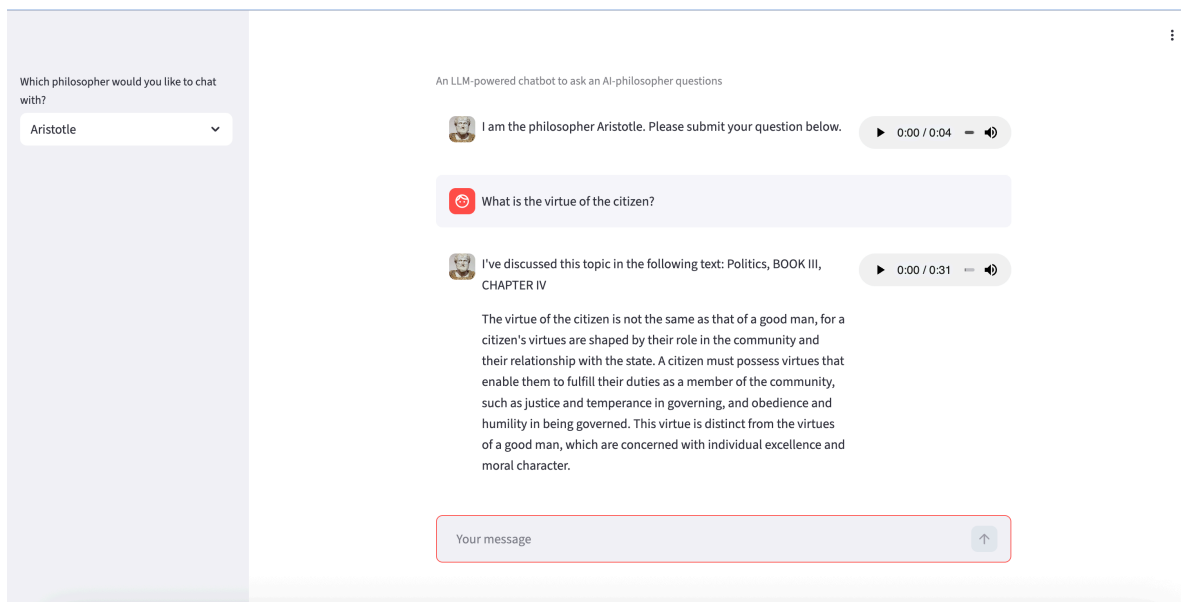
4.3 Application Components

Streamlit App

The user interface is written with Streamlit, a Python package for quickly building data apps. Our entire Streamlit application is written in ~200 lines of code. Streamlit includes built-in functionality for a chat timeline, making it simple to set up. I also incorporated a sidebar allowing the user to decide whether to chat with AI-Aristotle or AI-Confucius. The default philosopher is Aristotle. The application is also password-protected.

The application starts with displaying an introduction from the philosopher, and provides a text box for the user to enter a question. When the query is submitted, it gets added to the chat timeline, and passed to RunPod through an HTTP request, described below. When the response comes back, the app then does some post-processing. The response text gets cleaned up, and also passed to a free neural TTS system for English from Google TTS. Specifically, we used the voice named `en-US-Neural2-J` (more info here: <https://docs.cloud.google.com/text-to-speech/docs/list-voices-and-types>). Once this process is completed, the text response is displayed, alongside a button allowing the user to play the audio of the response if they wish. Then, the user can ask another question which is added to the timeline, or switch to the other AI-philosopher.

An image of the *Ask a Philosopher* application is below.



RunPod GPU Access

Generating text from LLMs is a computational intensive task that requires GPU usage to be efficient. Thus, we set up remote GPU access using RunPod, an inexpensive cloud service built for AI applications. RunPod offers either a "pod" approach, in which an entire GPU (or multiple) are rented and only used by you, or a "serverless" approach in which whenever a run is executed, a GPU from their collection is grabbed for that execution alone. We opted for the serverless approach, since we wanted flexible access.

The RAG-based model (described in detail below) is implemented primarily using the Transformers and PyTorch Python libraries (<https://huggingface.co/docs/transformers/en/index>; <https://pytorch.org/>). I wrote a RunPod handler script that has a handler function defining how the model runs on a given query, for a given philosopher. The RunPod handler script is accessed through an HTTP request sent to the endpoint.

Note that when the application has not been used recently, the RunPod endpoint GPU worker has to "cold start", which can cause delays. However, once the worker is active, it stays active for two minutes, allowing a user to see quick responses to their queries. During the presentation of this work, we set the system to have a constantly running worker, to prevent any cold starts during the demo.

Render Hosting

A final detail is that the Streamlit application is hosted on Render, an easy and inexpensive service for deploying web applications. It was as simple as connecting the Streamlit app GitHub repository to Render's service, and then verifying all secrets and passwords were appropriately added to Render.

4.4 Model Components

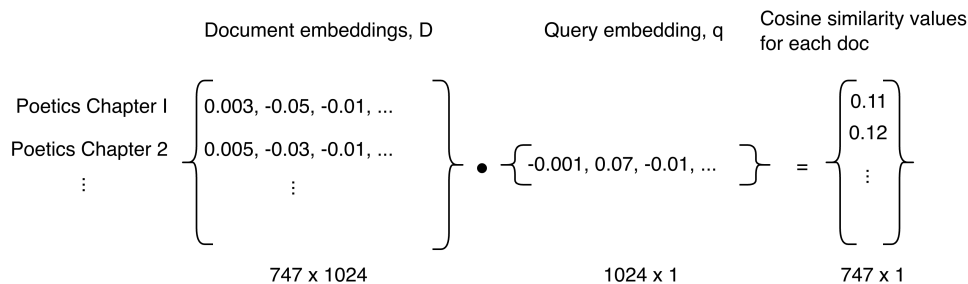
Gathered Reference Documents

For each philosopher, we collected their primary texts, splitting them by book and/or chapter as necessary. After splitting the texts, they were gathered in a single JSON file for each philosopher for ease of usage. Details on which texts were chosen for each philosopher and how we split up the texts are included in the Appendix.

Retrieval

Retrieval was done using dense vector representations with the text embedding model Octen-Embedding-0.6B, which produces embeddings of size 1024 (<https://huggingface.co/Octen/Octen-Embedding-0.6B>). For each chapter, we obtain a representation of the chapter by averaging text embeddings of the sentences in the chapter. To save time, these representations were pre-computed and stored in a csv, which was stored alongside the other code on RunPod. Then during usage, we just obtain the embedding of the query, and calculate cosine similarity between the query embedding and the chapter embeddings. We obtain the document with the highest cosine similarity value, and return both the document itself and the cosine similarity value. These returned variables are then passed to the generation step.

A representation of the retrieval step for Aristotle is below.



Generation

For generation, we use the open-source, instruction-tuned LLM, Llama-3.2-3B-Instruct (<https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct>). At the generation stage, we have already received the user query, most similar reference document, and the cosine similarity value between the query and the document. To determine whether to use the reference document in our generation prompt, we check the cosine similarity value against a threshold. For Aristotle, we use a threshold of 0.65, meaning that if the cosine similarity value is ≥ 0.65 , we use the reference document, and if it is < 0.65 , we drop the reference document, as it is not considered relevant enough. For Confucius, we do the same, but use a threshold of 0.5. These

thresholds were determined through experiments on questions with known document sources. The prompts used for the queries that used the retrieved document or did not are below.

Prompt A -- Use retrieved document:

System: You are the ancient philosopher, Aristotle. Respond to this question as A
<INSERT text of retrieved document>

Keep your response very short, 2-3 sentences.

User: <INSERT text of user query>

Prompt B -- Drop retrieved document, just use generation:

System: You are the ancient philosopher, Aristotle. Respond to this question as A

User: <INSERT text of user query>

Note somewhere that we do not give previous chat history to the LLM, thus for each new query the bot does not take previous queries into account.

Finally, we send our generated LLM response back to the Streamlit application. If we used the retrieved document, then we prepend information on which document was found to the displayed response, with the following format: "I've discussed this topic in the following text: < INSERT name of found chapter >". Otherwise, we just display the generated text.

4.5 GitHub Repositories

The code is split into two repositories.

Main repository, including RunPod functionality

The main repository is at this URL: https://github.com/alexandrasalem/ask_a_philosopher. This repository contains the code and data files for running the RAG model, as well as the RunPod handler function and Dockerfile for setting up the RunPod system. It also contains other scripts that were used for pre-processing the philosopher texts, testing the RAG model, and evaluating the results.

Streamlit app repository

We separated the Streamlit application into its own GitHub repository, here:

https://github.com/alexandrasalem/ask_a_philosopher_streamlit. This repository is highly lightweight and just includes the necessary scripts and data for setting up the Streamlit application.

5. Alternative Approaches Tried

5.1 Retrieval

In our initial retrieval set-up, we used sparse vector representations using TF-IDF. We wanted to try this simple classical method first, as it is highly efficient. We had some success with retrieving the relevant texts with this system, but TF-IDF can suffer from vocabulary mismatch potential, especially for these ancient texts with archaic vocabulary that differ across different translation. This could lead to issues where a relevant text for a query is not retrieved because the user used slightly different terminology than that present in the philosophical texts. Thus, we switched to a modern dense embedding based on LLMs, which allow flexibility in exact word choice.

There are many dense embedding models available. We chose ours by consulting the leaderboards on HuggingFace for retrieval tasks (<https://huggingface.co/spaces/mteb/leaderboard>).

We started out by trying a purely LLM-based approach, and a purely RAG-based approach. The purely LLM-based approach worked well, but sometimes the answers were generic and could have been better rooted in the philosophical texts. In the purely RAG-based model, the generation step would *always* use the top retrieved document. However, in this application, it is possible that a user would ask questions that we would not expect Aristotle or Confucius to have ever touched upon. For instance, a user could ask AI-Aristotle, "What do you think of AI?". The retrieval step would still find the most relevant chapter to that question, even though none of them should be relevant. In the end, we landed on a hybrid between LLM-only and RAG, that uses retrieved documents when they are relevant enough, but backs off to LLM-only otherwise.

5.2 Generation

We experimented with a few different LLM prompts as we developed *Ask a Philosopher*. One important observation was that the LLM responses tended to be fairly long. Thus, we added the

instruction to "Keep your response very short, 2-3 sentences.", which seemed to lead to more concise responses.

Unlike ChatGPT, we treated each user query as an entirely new chat. Thus, the previous chat history was not included as memory for the LLM. This choice was motivated by not wanting the AI-philosopher to be influenced too much by user queries or experience cascading errors from previously answered queries. However, including that history would be an interesting alternative.

5.3 Application Hosting Service

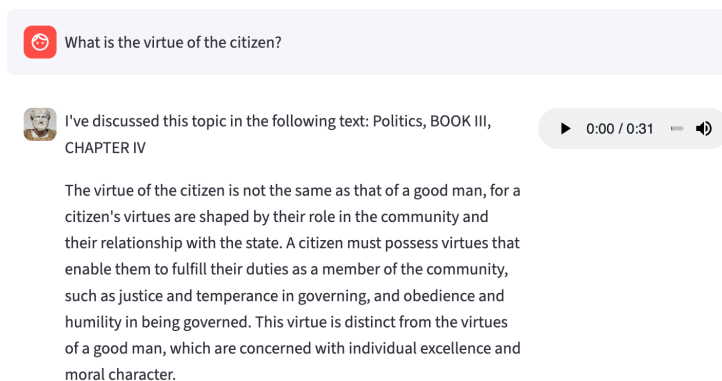
Initially, we hosted the Streamlit application directly through Streamlit Community Cloud (<https://streamlit.io/cloud>). They provide free hosting of your apps. However, the issue is that they do not keep the applications live. When it is not used for some time, it has to be rebuilt by the cloud services. This takes a few minutes, and thus is not ideal for our use-case. So, we switched to hosting on Render, which provides very low-cost website hosting. There are other options, such as Heroku that would likely have been equally suitable (<https://www.heroku.com/>).

6. Examples

A few selected examples of *Ask a Philosopher* in use are below.

6.1 Aristotle

Here is one where the system found a reference text:



The screenshot shows a chat interface with a light blue header containing a question: "What is the virtue of the citizen?". Below the question is a video player. The video player has a small profile picture icon on the left, followed by the text: "I've discussed this topic in the following text: Politics, BOOK III, CHAPTER IV". To the right of the text is a video player control bar showing "0:00 / 0:31" and a play button icon. Below the video player, there is a text block that reads: "The virtue of the citizen is not the same as that of a good man, for a citizen's virtues are shaped by their role in the community and their relationship with the state. A citizen must possess virtues that enable them to fulfill their duties as a member of the community, such as justice and temperance in governing, and obedience and humility in being governed. This virtue is distinct from the virtues of a good man, which are concerned with individual excellence and moral character."

And here is one where the system dropped the found reference text:



What is human happiness?



Human happiness, my dear friend, is not a fleeting state, but rather a virtuous habituation of the soul, wherein one's actions and character align with the mean between excess and deficiency. It is the result of cultivating the virtues, such as wisdom, courage, and justice, which enable us to live a life of moderation and balance. In this way, happiness is not a mere feeling, but a state of being that arises from living a life of eudaimonia.

▶ 0:00 / 0:26 — 🔊

6.2 Confucius

Here is one where the system found a reference text:



What is righteousness?



I've discussed this topic in the following text: Mencius, Chapter 21, ,

"My son, righteousness is like a cup or a bowl, fashioned from the natural goodness of humanity. It is not something that can be forced or created, but rather it is the natural unfolding of one's character, like the growth of a tree or the blossoming of a flower. When we are allowed to flourish, our true nature shines through, and we exhibit the virtues of benevolence, righteousness, and propriety."

▶ 0:00 / 0:27 — 🔊

And here is one where the system dropped the found reference text:



What do you think about social media?



Social media is a tool, not an end in itself. Its value lies in how it is used, for it can either bring people together or drive them apart, depending on the intentions of its users.

▶ 0:00 / 0:10 — 🔊

7. Evaluation

Evaluating free-form generated text is a difficult open problem in NLP. Our goal was to have our system that reflected the language style of the philosopher, as well as their philosophical beliefs.

We have not quantitatively evaluated whether the language *style* of the chatbot accurately reflects the philosopher. However, qualitative observation of our responses showed many

expected patterns. For example, in the second Aristotle example above, the response starts "Human happiness, my dear friend, is ...". This friendly and conversational introduction reflects the conversational nature of Aristotle's texts.

To get at whether the system reflected the philosophical beliefs of the philosophers, we evaluated the performance on a set of 16 question-answer pairs. We conducted this evaluation only for Aristotle, since we had an expert on Aristotle on our team. The results are below.

7.1 Performance for Aristotle

An expert on Aristotle on our team developed a testing set of 16 questions and answers associated with specific Aristotle source texts. An example is below:

Question: What is the virtue of the citizen?

Answer: To be capable both of commanding and obeying.

Source: Politics III

We ran each of the questions through our final system. We calculated two performance metrics:

1) Correct document found: whether the retrieval system found the correct source document; and 2) Generated response correct: our expert annotator labeled if the AI-Aristotle responses were "correct". The performance at these two metrics is below:

- Correct document found: 10 Yes, 5 None (doc dropped), 1 No, 62.5% accuracy
- Generated response correct: 12 Yes, 4 No, 75% accuracy

8. Appendix

A. Philosophical Texts

A.1 Aristotle

We used the following texts for Aristotle:

- **The Poetics of Aristotle**, translated by Butcher, S. H. (Samuel Henry), 1850-1910: <https://www.gutenberg.org/ebooks/1974>
- **The Categories**, translated by Edghill, E. M. 1928: <https://www.gutenberg.org/ebooks/2412>
- **Politics: A Treatise on Government**, translated by Ellis, William, 1730-1801: <https://www.gutenberg.org/ebooks/6762>

- **The Nicomachean Ethics of Aristotle**, translator not listed: <https://www.gutenberg.org/ebooks/8438>
- **The Athenian Constitution**, translated by Kenyon, Frederic G. (Frederic George), Sir, 1863-1952: <https://www.gutenberg.org/ebooks/26095>
- **History of Animals**, translated by Cresswell, Richard, 1815-1882: <https://www.gutenberg.org/ebooks/59058>
- **Metaphysics**, translated by W. D. Ross, year not listed: <http://classics.mit.edu//Aristotle/metaphysics.html>
- **Rhetoric**, translated by W. Rhys Roberts, year not listed: <http://classics.mit.edu//Aristotle/rhetoric.html>

A.2 Confucius

We used the following texts for Confucius, which are together considered the *Four Books*:

- **Great Learning**, translated by James Legge, 1893: <https://sacred-texts.com/cfu/conf2.htm>
- **Doctrine of the Mean**, translated by James Legge, 1893: <https://sacred-texts.com/cfu/conf3.htm>
- **Analects**, translated by James Legge, 1893: <https://sacred-texts.com/cfu/conf1.htm>
- **Mencius**, translated by James Legge, 1895: <https://sacred-texts.com/cfu/menc/index.htm>

B. Philosophical Texts Pre-processing

We obtained text files of all the philosophical texts using the links above. For both Aristotle and Confucius, we defined a "document" for retrieval in our system as a chapter of one of the texts, if the text is split that way, or the entire text itself otherwise. Aristotle's texts are quite long and typically split into Books, followed by Chapters. Confucius's texts are short and not split into chapters, with the exception of the work *Mencius*.

We split the text files into these documents using regular expressions. We made sure to exclude text from the preambles, introductions, or closing notes. For each author, we gathered their documents into a JSON file. The JSON file included the following information for each chapter: text_name, book_label, chapter_label, chapter_text, id.

This process resulted in 727 chapters for Aristotle and 49 chapters for Confucius.