Retrieval Techniques for Building a Princeton Student Chatbot

Leo Stepanewk
Adviser: Ravi Netravali

Abstract

Large language models (LLMs) are changing the way people find and interact with information. Many emerging applications use LLMs in conjunction with an external database of documents to tailor chatbot responses to a specific domain. This framework is broadly known as retrieval-augmented generation (RAG)[18] and is increasingly a topic of academic research. However, not much research has been written about whether these techniques are practical within production environments. This project aims to evaluate RAG techniques within a new production use case: a chatbot for Princeton students. The system accurately answers queries about academics, extracurriculars, health, and careers by referring to a database of web documents crawled on the Princeton domain. The performance of the chatbot is evaluated over a synthetic question-answer dataset while implementing several proposed retrieval optimizations, such as HyDE[17], reranking, and hybrid search. We achieve near-perfect scores for correctness, faithfulness, and relevancy of our answers for a production-ready setup. We provide our recommendations regarding which retrieval techniques and chatbot settings were most effective given realistic performance tradeoffs.

1. Introduction

Upon entering college, students are overwhelmed with everything they have to learn about campus, academics, and all other aspects of their life for the next few years. Even heading toward graduation, many still have not realized the full extent of the programming made available by the university, in part due to slow information diffusion and simply being unaware of what is out there. This is an unfortunate fact given that, on average, $20,400/student/year is spent on support and engagement at private nonprofit 4-year institutions, comprising 40% of total university expenses[4].

At Princeton, student opportunities are listed across thousands of web pages on the *princeton.edu* domain. The information is not palatable or communicable, and only so many resources can be properly advertised to students via email. The rest of the knowledge is hard to come across, often only through word-of-mouth or with very specific and intentional searching. Synthesizing across multiple pages is also a challenge. Oftentimes, students have to weigh options, compare and contract academic requirements, or manually compile exhaustive lists of resources that are relevant to their individual pursuits. These barriers cause students to not reach their full educational potential.

Meanwhile, the emergence of LLMs is broadly shifting the collective understanding of information retrieval and synthesis. After the publication of the seminal paper on the transformer architecture[25], the field of AI has rapidly evolved products like ChatGPT and crossed the chasm into the consumer spotlight. In January 2023, just two months post-release, ChatGPT exceeded 100 million monthly active users, breaking the record for the fastest-growing consumer app in history[20]. ChatGPT and OpenAI had uncovered an untapped need for people to query information in a conversational way and get synthesized responses instead of performing that mental work themselves. This paradigm shift enabled new entrants to challenge the traditional "search" market primarily dominated by Google. Perplexity.ai is a startup that is building an "answer engine" that functions similar to Google search but gives the user a natural language answer to every query by reading and synthesizing search results with an LLM[16]. As of August 2023, Perplexity uses a custom index of the web, similar to Google's PageRank[15], to find relevant and trustworthy pages to feed into a fine-tuned version of OpenAI's GPT-3.5 model to generate answers[24].

The advances in LLMs, broad adoption of chatbots among consumers, and personal experiences with college inspired the creation of this project. The objective was to apply language modeling to close the knowledge gap that students have and make the education experience at Princeton more fruitful. While general purpose LLMs might have some limited understanding of campus, they will provide incomplete and inaccurate answers unless specific document chunks are given to ground the response in up-to-date and factual data. The paper focuses on figuring out what document retrieval methodologies to use, as that is the primary factor that determines response relevancy and quality.

Since the aim is to release the chatbot to the student body, the evaluation of the retrieval pipeline had to include factors such as latency, generation speed, and cost instead of only theoretical accuracy. These considerations set our research apart from other academic work done in RAG, as there are real tradeoffs and failure modes that are not captured in studies that are purely focused on retrieval performance. There are also domain specific challenges that impact the design of the chatbot and its document retrieval system when intended for student use. We succeeded in creating a chatbot that performs well under these conditions, and from that experience, we offer specific recommendations for RAG techniques that make sense for similar practical applications.

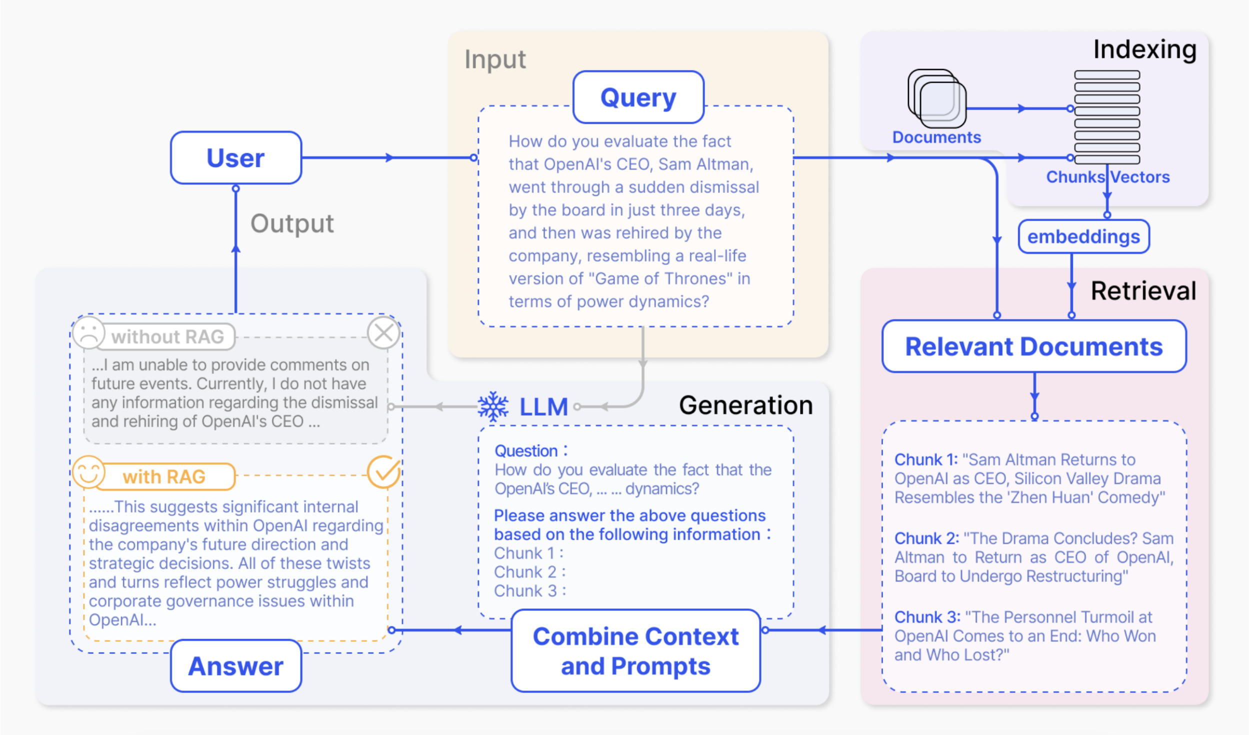
2. Background

2.1. Vector Search

Finding text documents relevant to a search query has been a problem traditionally solved by keyword matching and inverted indexes. However, these techniques often fail to capture the real meaning and intent of the query. In recent years, researchers have trained neural networks to embed text into high-dimensional vectors that capture true semantic meaning. Word2vec[23] is a paper that proposes two new model architectures for word embeddings and demonstrates that the vectors of related words are located near each other in the embedding space. Since then, embedding models like OpenAI's `text-embedding-ada-002`[19] have been developed to vectorize sentences and document chunks for large scale similarity search. Search across millions or billions of high dimensional vectors is a separate area of research that has resulted in the development of approximate nearest neighbor search algorithms[21]. These innovations enable vector databases like Pinecone[11] to provide efficient search and storage for AI applications.

2.2. Retrieval-Augmented Generation (RAG)

RAG is a method for guiding LLM response generation using an external dataset of documents. In its most basic form, RAG involves 1) having a dataset of document chunks associated with their embedding vectors 2) embedding every incoming query 3) using vector search to find the most relevant document chunks and 4) inserting the document chunks into the LLM prompt such that the model refers to them while answering the query[18].



How do you evaluate the fact that OpenAI's CEO, Sam Altman, went through a sudden dismissal by the board in just three days, and then was rehired by the company, resembling a real-life version of "Game of Thrones" in terms of power dynamics?

...I am unable to provide comments on future events. Currently, I do not have any information regarding the dismissal and rehiring of OpenAI's CEO ...

......This suggests significant internal disagreements within OpenAI regarding the company's future direction and strategic decisions. All of these twists and turns reflect power struggles and corporate governance issues within OpenAI...

Question：
How do you evaluate the fact that the OpenAI's CEO, ... dynamics?
Please answer the above questions based on the following information：
Chunk 1：
Chunk 2：
Chunk 3：

Chunk 1: "Sam Altman Returns to OpenAI as CEO, Silicon Valley Drama Resembles the 'Zhen Huan' Comedy"

Chunk 2: "The Drama Concludes? Sam Altman to Return as CEO of OpenAI, Board to Undergo Restructuring"

Chunk 3: "The Personnel Turmoil at OpenAI Comes to an End: Who Won and Who Lost?"

Figure 1: RAG applied to question-answering versus default LLM response[18].

A key benefit of RAG is that it does not require any model training. Any general-purpose LLM can inexpensively adapted to question-answering about a custom dataset. The task of building a custom chatbot is largely reduced to ensuring that the document database contains relevant documents and that these documents are being accurately retrieved for each query. By updating the database with new documents, the chatbot will immediately have access to recent information without finetuning. This is critical for applications like a student chatbot where new data will be added regularly. Research has demonstrated that RAG mitigates hallucinations, a common phenomenon where an LLM generates a factually-incorrect or disconnected response to a prompt[26].

2.3. Synthetic Datasets & LLM-as-a-Judge

The availability of large, well-labeled datasets has always been an issue in machine learning. However, LLMs opened up a new frontier in synthetic data and automated evaluations. Instead of recruiting human labor to curate a dataset, practitioners use LLMs to generate both human-like text inputs and reference answers. Ragas[6] is a popular, open-source project that includes a pipeline to generate question-answer datasets from arbitrary document collections. It uses a series of LLM prompts to iteratively "evolve" questions whose answers depend on retrieving multiple text chunks. Furthermore, LLM-as-a-judge is an idea where the language model itself can be used as a response evaluator. By formulating the prompt such that the LLM outputs a quality score and explanation for its decision, Zheng et al. found that "GPT-4 can match both controlled and crowdsourced human preferences well, achieving over 80% agreement, the same level of agreement between humans" [27]. Hence, LLMs can be used to create a dataset and evaluate another model's performance on it in a scalable, end-to-end manner.

2.4. Hypothetical Document Embedding (HyDE)

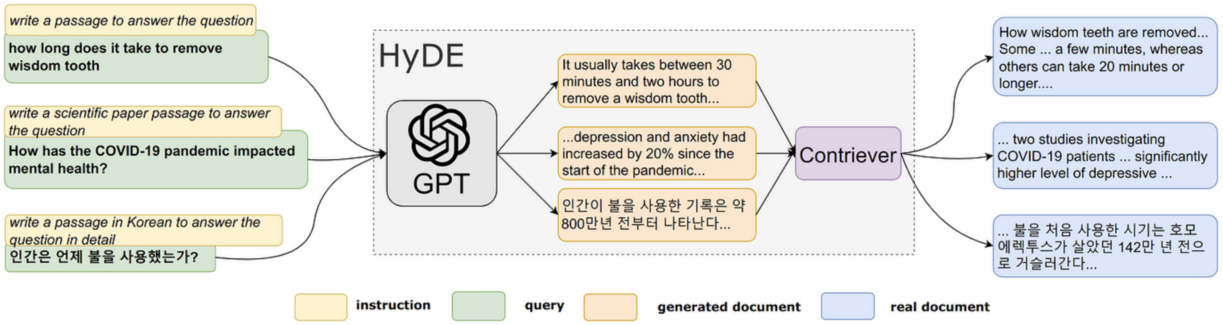


Figure 1: An illustration of the HyDE model. Documents snippets are shown. HyDE serves all types of queries without changing the underlying GPT-3 and Contriever/mContriever models.

Figure 2: Illustration of the HyDE pipeline[17].

One retrieval optimization proposed by Gao et al.[17] involves generating a hypothetical document for each incoming query and embedding it instead of the query itself. This method purposely induces the LLM to hallucinate a text chunk that might look similar to an actual document in the dataset containing the answer to the query. The intuition behind HyDE is that text embedding models are generally trained to minimize the distance between the vectors of two related documents, rather than a document and a query string. The hallucinated text chunk appears much more similar to the relevant document chunks in the database than the query itself, resulting in better retrieval.

2.5. Query Rewriting

Prior to document retrieval, the query can also be rewritten by an LLM to optimize performance. Ma et al. introduced a trainable *rewrite-retrieve-read* pipeline where a smaller language model adjusts the input query to optimize retrieval of web documents for the primary, black-box LLM[22]. The same process works to reasonable effectiveness even when a general-purpose LLM is used as a query rewriter. One important use case of query rewriting is ensuring relevant retrieval during extended chatbot conversations, where the user asks queries relating to previous context. Queries such as "tell me more about this" will not yield the correct documents when fed into a vector search pipeline unless an LLM rewrites the query to include the necessary context based on past messages.

2.6. Reranking

To further improve RAG performance, Zhuang et al. explored the use of an LLM as a precise reranker of documents following a coarse vector retrieval step[28]. Upon receiving a query, an application first uses vector search to retrieve a large set of documents, ensuring high recall. Then, an LLM reads these documents, numerically evaluates their relevancy, and selects a subset of the best chunks to use for actual answer generation. By separating the retrieval process into two stages, Zhuang et al. gains the benefits of retrieving a large number of chunks while not sacrificing in precision and response generation speed (which would be slow and expensive if all chunks selected by the coarse retrieval were directly fed to the LLM).

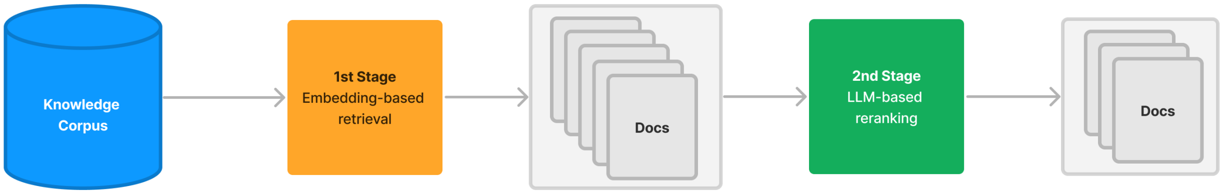


Figure 3: Two-stage retrieval with top-k vector search and LLM reranking[14].

2.7. LlamaIndex

LlamaIndex is an open-source Python framework specifically designed for the development of applications that integrate LLMs[7]. It provides a scaffold for creating chatbots and several useful data ingestion utilities for RAG. It also includes out-of-the-box implementations for HyDE and LLM response evaluators. We used LlamaIndex for this project to quickly iterate on chatbot parameters and retriever pipeline implementations.

3. Procedure

3.1. Dataset Creation

We started by collecting a set of 725 URLs located on the Princeton domain. We initially compiled a list of root URLs by hand to encompass a diversity of topics, and then used an automated crawler to download the text off each URL as well as any URLs the root pages linked to. The website contents were saved as separate plain text files in a folder, which we then injested using LlamaIndex's `SimpleDirectoryReader`. This class reads each file and chunks the text into separate `TextNode` objects, totaling 1969 nodes for the entire dataset. The length of each chunk is 1024 tokens by default, and each node automatically generates the `text-embedding-ada-002` vector embedding and file metadata.

We randomly selected 50 URLs to form a representative question-answer dataset. We chose this sample size to ensure that our tests run quickly and inexpensively (since each response generation and LLM-based eval makes a billable API call). The full text of every website was inserted into a `gpt-4-turbo-preview` prompt that asks the model to generate a realistic question-answer pair based on the content. When testing the chatbot, we would pass in the question as a query, produce a response, and then compare the response against the reference answer.

3.2. Response Evaluation

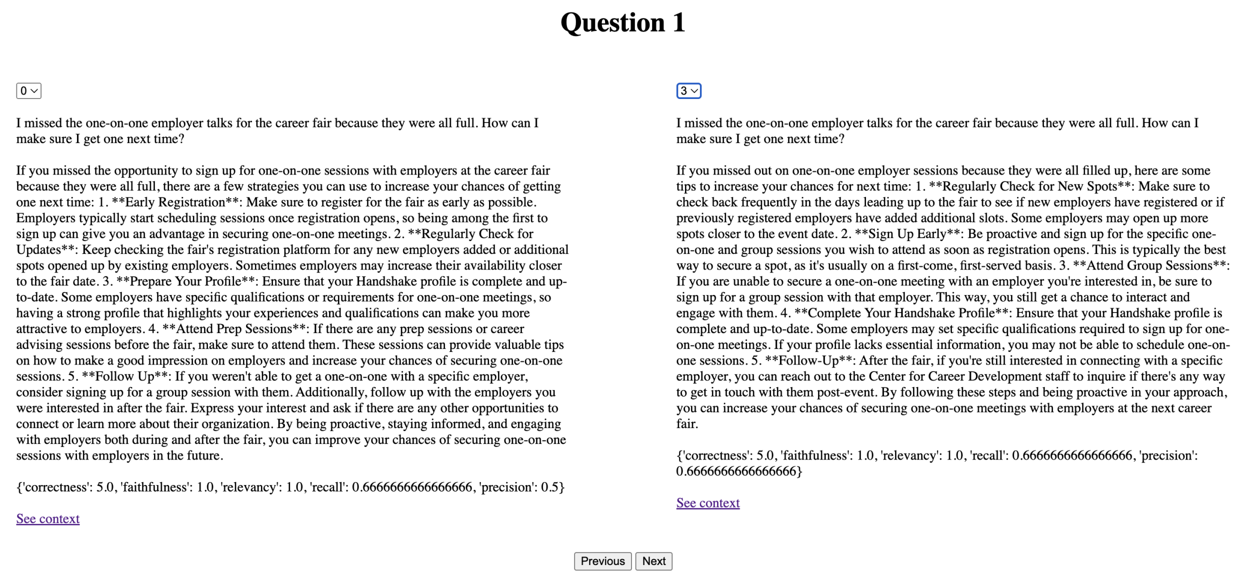
The judging of the chatbot responses consisted of three LLM-based evaluators:

`CorrectnessEvaluator` assesses the correctness and inclusion of specific details within each response as compared to the reference answer[2]. The underlying implementation is a `gpt-4-turbo-preview` prompt with instructions to rate the output on a scale from 0.0 to 5.0 in increments of 0.5. A high-scoring response includes all or most of the details from the reference answer and does not have factual errors.

`FaithfulnessEvaluator` tracks whether the response contains hallucinations compared to the input context chunks[3]. The LLM response outputs either 0.0 or 1.0 if a hallucination is or is not present. These hallucinations include factual errors, contradictions, and irrelevant outputs.

`RelevancyEvaluator` checks whether the response and the input context chunks are actually relevant to the question[10]. The LLM evaluator outputs either 0.0 or 1.0. A common cause of low scores here is the chatbot being given irrelevant text chunks.

The numerical score for each metric is the average across the 50 questions. Each evaluator also produces feedback in paragraph form, which we logged for each individual test run to use for debugging, alongside the individual scores, chatbot responses, and input text chunks. We made a simple web interface using Python and Flask that let us qualitatively compare performance on questions across different test runs and inspect every step of the process.



Question 1

I missed the one-on-one employer talks for the career fair because they were all full. How can I make sure I get one next time?

If you missed the opportunity to sign up for one-on-one sessions with employers at the career fair because they were all full, there are a few strategies you can use to increase your chances of getting one next time: 1. **Early Registration**: Make sure to register for the fair as early as possible. Employers typically start scheduling sessions once registration opens, so being among the first to sign up can give you an advantage in securing one-on-one meetings. 2. **Regularly Check for Updates**: Keep checking the fair's registration platform for any new employers added or additional spots opened up by existing employers. Sometimes employers may increase their availability closer to the fair date. 3. **Prepare Your Profile**: Ensure that your Handshake profile is complete and up-to-date. Some employers have specific qualifications or requirements for one-on-one meetings, so having a strong profile that highlights your experiences and qualifications can make you more attractive to employers. 4. **Attend Prep Sessions**: If there are any prep sessions or career advising sessions before the fair, make sure to attend them. These sessions can provide valuable tips on how to make a good impression on employers and increase your chances of securing one-on-one sessions. 5. **Follow Up**: If you weren't able to get a one-on-one with a specific employer, consider signing up for a group session with them. Additionally, follow up with the employers you were interested in after the fair. Express your interest and ask if there are any other opportunities to connect or learn more about their organization. By being proactive, staying informed, and engaging with employers both during and after the fair, you can improve your chances of securing one-on-one sessions with employers in the future.

{'correctness': 5.0, 'faithfulness': 1.0, 'relevancy': 1.0, 'recall': 0.6666666666666666, 'precision': 0.5}

See context

I missed the one-on-one employer talks for the career fair because they were all full. How can I make sure I get one next time?

If you missed out on one-on-one employer sessions because they were all filled up, here are some tips to increase your chances for next time: 1. **Regularly Check for New Spots**: Make sure to check back frequently in the days leading up to the fair to see if new employers have registered or if previously registered employers have added additional slots. Some employers may open up more spots closer to the event date. 2. **Sign Up Early**: Be proactive and sign up for the specific one-on-one and group sessions you wish to attend as soon as registration opens. This is typically the best way to secure a spot, as it's usually on a first-come, first-served basis. 3. **Attend Group Sessions**: If you are unable to secure a one-on-one meeting with an employer you're interested in, be sure to sign up for a group session with that employer. This way, you still get a chance to interact and engage with them. 4. **Complete Your Handshake Profile**: Ensure that your Handshake profile is complete and up-to-date. Some employers may set specific qualifications required to sign up for one-on-one meetings. If your profile lacks essential information, you may not be able to schedule one-on-one sessions. 5. **Follow-Up**: After the fair, if you're still interested in connecting with a specific employer, you can reach out to the Center for Career Development staff to inquire if there's any way to get in touch with them post-event. By following these steps and being proactive in your approach, you can increase your chances of securing one-on-one meetings with employers at the next career fair.

{'correctness': 5.0, 'faithfulness': 1.0, 'relevancy': 1.0, 'recall': 0.6666666666666666, 'precision': 0.6666666666666666}

See context

Figure 4: The web interface comparing responses to a question across two test runs, including the numerical scores and provided context chunks.

3.3. Response Generation

Our initial objective was to iterate over general chatbot hyperparameters to see which ones performed the best. We focused on the testing the following:

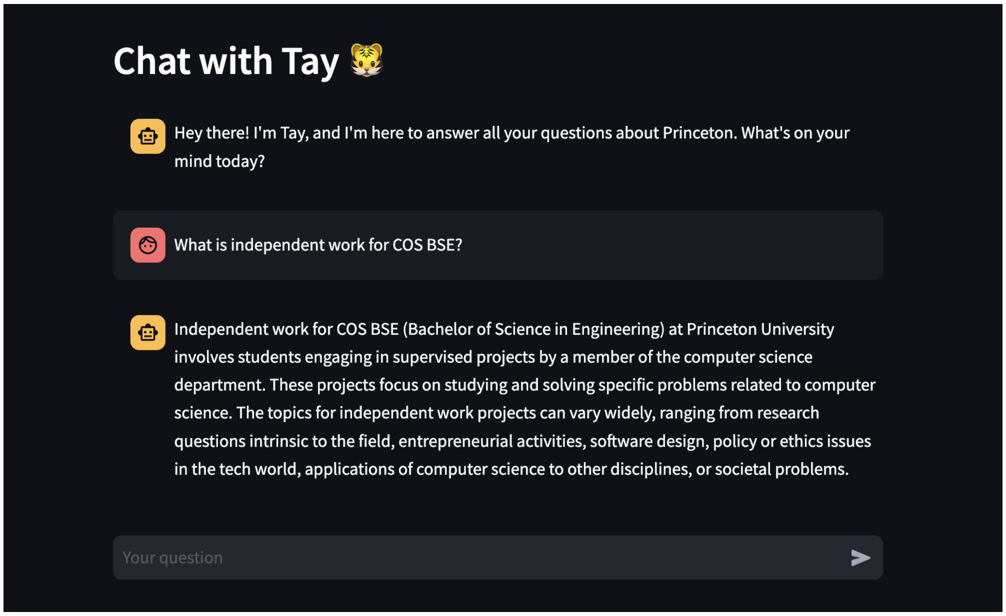
`LLM_MODEL`: either `gpt-3.5-turbo` or `gpt-4-turbo-preview`.

`TOP_K`: the number of text chunks supplied per response from the vector retrieval stage.

`CHAT_MODE`: `simple`, `context`, `condense_question`, or `condense_plus_context`[13]. These modes are provided by LlamaIndex and relate to how the chatbot handles queries with retrieval. `simple` mode queries the LLM directly without RAG. `context` mode trivially retrieves chunks from the vector store based on the user query and inserts them into the system prompt, while `condense_question` and `condense_plus_context` perform query rewriting based on the conversation. `condense_plus_context` rewrites the query using both past conversation messages and retrieved context.

3.4. Demo

We created a demo[5] of the chatbot using Streamlit, giving us a convenient interface for manual testing and sharing. Conversations with the chatbot are logged to a MongoDB instance.



Chat with Tay 🐯

Hey there! I'm Tay, and I'm here to answer all your questions about Princeton. What's on your mind today?

What is independent work for COS BSE?

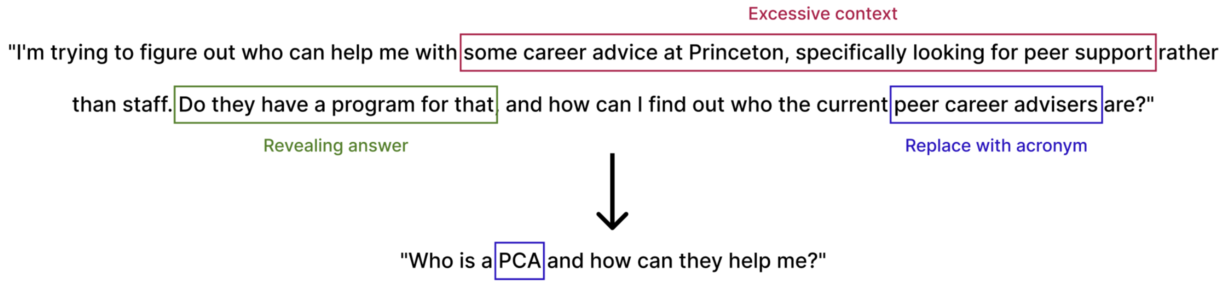
Independent work for COS BSE (Bachelor of Science in Engineering) at Princeton University involves students engaging in supervised projects by a member of the computer science department. These projects focus on studying and solving specific problems related to computer science. The topics for independent work projects can vary widely, ranging from research questions intrinsic to the field, entrepreneurial activities, software design, policy or ethics issues in the tech world, applications of computer science to other disciplines, or societal problems.

Figure 5: The Streamlit user interface with an example conversation.

3.5. Dataset Revision

After performing the first round of testing, we decided to revise the dataset to make it more realistic and challenging while focusing more on retrieval performance. We regenerated the test set with a new sample of 50 web pages where each page produced a question, reference answer, and ids for the text chunks that the chatbot is supposed to retrieve to answer the question. Instead of using a single prompt to generate just the question and reference answer, we created a chain of prompts that also labeled the chunks used for the answer as relevant. In other words, if a website's text consisted of S chunks, then $X \subseteq S$ chunks would be identified as the ground truth for the retriever. On average, each question had 2.06 chunks associated with it in this new dataset.

After the automated question-answer-context generation step, we manually revised each question to be more similar what a student would type into the chatbot. These revisions included changes like replacing full length phrases with acronyms and removing context that a student would assume is implicit. We were not successful in automating this step through prompting, so the manual rewriting was the best option given our student background. Many of these revisions were expected to increase the difficulty of retrieving the correct chunks.



"I'm trying to figure out who can help me with some career advice at Princeton, specifically looking for peer support rather than staff. Do they have a program for that, and how can I find out who the current peer career advisers are?"

"Who is a PCA and how can they help me?"

Figure 6: An example of a manually rewritten query that removes excessive context and spelled-out acronyms to make it more realistic and challenging for the document retriever.

3.6. Retriever Evaluation

We added recall and precision as metrics alongside the correctness, faithfulness, and relevancy evaluators from earlier.

$$recall = \frac{num\_relevant\_chunks}{total\_relevant\_chunks}$$

$$precision = \frac{num\_relevant\_chunks}{num\_retrieved\_chunks}$$

Recall is the percentage of relevant chunks retrieved out of the total number of relevant chunks. Precision is the percentage of chunks provided to the chatbot that are labeled as relevant. These metrics would enable us to assess any advanced retrieval techniques we were planning to implement to handle the increased complexity of the dataset. We evaluate both recall and precision at four retrieved chunks (which is a hyperparameter we selected from the first testing round) for every retrieval optimization to enable fair comparison to the baseline.

3.7. Retriever Testing

For the second round of testing, we implemented five retriever configurations (where $k = 4$):

`baseline`: a simple top-k vector retriever.

`hyde`: top-k vector retrieval based on the embedding of a hypothetical document (using `HyDEQueryTransform`).

`hybrid`: returning chunks merged from top-$k/2$ vector retrieval and top-$k/2$ keyword retrieval (using `SimpleKeywordTableIndex`). If there is overlap, include additional chunks from the vector retriever until k chunks are found.

`reranker`: retrieve $4k$ chunks with vector search and then use `LLMRerank` to narrow to the top-k (using `LLMRerank` batch size of $2k$).

`reranker + hybrid`: retrieve $3k$ chunks with vector search and k chunks with keyword retrieval, then use `LLMRerank` to narrow to the top-k (with batch size of $2k$).

Each retriever tests a hypothesis or tries to mitigate a failure case we discovered while testing the chatbot configuration selected from the first round of testing. We also performed timing tests for how long retrieval takes in each case relative to the full response generation time. The remaining chatbot hyperparameters were selected based on the results from the initial dataset and kept constant.

4. Results

4.1. Initial Dataset Evaluation

Model	**Chat Mode**	**Top K**	**Correctness**	**Faith**	**Relv**
`gpt-3.5-turbo`	`simple`	–	3.20	–	–
`gpt-4-turbo-preview`	`simple`	–	3.82	–	–
`gpt-3.5-turbo`	`condense_question`	2	3.87	1.0	0.96
`gpt-3.5-turbo`	`context`	2	4.47	1.0	0.96
`gpt-3.5-turbo`	`condense_plus_context`	2	4.42	1.0	0.94
`gpt-3.5-turbo`	`condense_plus_context`	4	4.61	1.0	0.98
`gpt-4-turbo-preview`	`condense_plus_context`	4	4.85	0.98	0.96

Table 1: Results from the pre-revision dataset.

The performance on the initial dataset was evaluated across model, chat mode, and top-*k* quantity. The baseline for correctness was set by the two `simple` tests. We observed that GPT-4 set a higher bar for correct responses by a large margin compared to GPT-3.5. This performance was obvious still not satisfactory since most responses did not include information from the document store. Out of the actual RAG modes, we observed that the `condense_question` mode performed the worst on correctness while the `context` mode scored the highest. However, we needed query rewriting to maintain natural extended conversations as explained in section 2.5. Thankfully, the `condense_plus_context` mode performed equally well, and it was able to handle conversations in practice unlike `context` mode. We continued with using `condense_plus_context` for all of our testing moving forward.

Next, we increased top-*k* from two chunks to four, which increased performance by another significant margin. By feeding in more chunk to the LLM, we improve the odds of getting more relevant text to base the answer on at the expense of speed and cost. Beyond $k = 4$, we felt that the user experience with using the chatbot was not satisfactory.

Finally, we tried changing the model from `gpt-3.5-turbo` to `gpt-4-turbo-preview`. This produced another meaningful bump in the correctness score, but using GPT-4 was much slower and about 20x as expensive as GPT-3.5[8]. The average response from GPT-3.5 took 10.47 seconds,

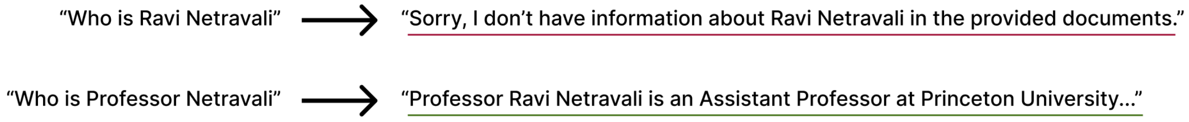
Model	Average time/query (s)
`gpt-3.5-turbo`	10.47
`gpt-4-turbo-preview`	35.31

Table 2: Average time to generate a response start-to-finish (responses are of comparable length).

while the response of similar length from GPT-4 took 35.31 (Table 2). For production purposes, we concluded that using GPT-3.5 is the correct decision given that its performance was only marginally worse than GPT-4. Our final configuration coming out of the first round of testing was using `gpt-3.5-turbo`, `condense_plus_context` chat mode, and top-4 chunk vector retrieval.

4.2. Failure Case

While we were pleasantly surprised by the chatbot's performance on quantitative metrics, we noticed certain cases where it would fail to answer basic factual queries during manual testing. Specifically, when queried about non-obvious proper nouns, the chatbot would answer that it could not find any relevant documents.



"Who is Ravi Netravali" → "Sorry, I don't have information about Ravi Netravali in the provided documents."

"Who is Professor Netravali" → "Professor Ravi Netravali is an Assistant Professor at Princeton University..."

Figure 7: A vector retrieval failure due to a proper noun versus where it has sufficient context.

We realized that this behavior was a failure case of embedding the query for direct use in vector search. While the chatbot would not find relevant documents for a query like "who is Ravi Netravali," it would successfully answer "who is Professor Netravali." In the first query, the text embedding model cannot effectively vectorize the query because there is no surrounding context. The model itself will not know the meaning behind names or proper nouns (unless the person or noun is extremely famous and highly represented in its training corpus). However, in the second query, the word "professor" gives just enough context for the embedding model and retrieval pipeline to find the document containing the descriptions of several professors, among whom is Ravi Netravali.

This success is still largely a coincidence and cannot be relied upon. To fix this, we introduced hybrid retrieval as one of the advanced retrievers being tested on the revised dataset.

4.3. The Need for Revision

When debugging the proper noun issue, we realized that the dataset questions were potentially too easy for the chatbot. As seen in Figure 8, the generated questions include details and hints that a real user would never include in a query. The embedding model then produces too accurate of a vector and gives the chatbot misleadingly high performance on the tests. This prompted us to do manual rewriting of the questions post-generation, as well as including retrieval-specific metrics to have better insight into that aspect of the system.

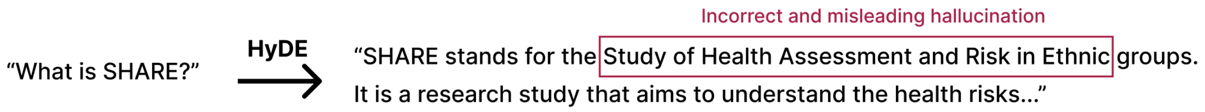
4.4. Revised Dataset & Retriever Evaluation

Retriever	**Correctness**	**Faithfulness**	**Relevancy**	**Recall@4**	**Precision@4**
`baseline`	4.50	1.0	0.94	0.60	0.30
`hyde`	4.35	0.92	0.88	0.53	0.28
`hybrid`	4.38	0.94	0.84	0.57	0.26
`rerank`	4.54	0.98	0.94	0.58	0.29
`rerank + hybrid`	**4.57**	**1.0**	**0.96**	**0.63**	**0.31**

Table 3: Results from the post-revision dataset.

Beginning with the baseline retriever and chatbot (`gpt-3.5-turbo`, `condense_plus_context` chat mode, and top-4 vector retrieval), we saw high performance on correctness, faithfulness, and relevancy. We also measured a recall@4 and precision@4 of 0.60 and 0.30 respectively, which are reasonable numbers. In particular, the maximum average precision@4 would be around 0.5 given that most queries have two relevant chunks. The recall informs us that the baseline retriever finds 60% of the labeled relevant text chunks on average per query.

HyDE underperformed the baseline by a significant margin across metrics. Our analysis of the results showed that the hypothetical document embeddings were often wrong and misleading given the types of queries in the dataset. When faced with queries heavy in acronyms, HyDE hallucinates documents that are unrelated to the question, leading to the retrieval of irrelevant chunks. We concluded that this is not a good technique to use when dealing with our target audience.



"What is SHARE?"

"SHARE stands for the Study of Health Assessment and Risk in Ethnic groups. It is a research study that aims to understand the health risks..."

Figure 8: An example where HyDE generates an incorrect hypothetical document for SHARE, the sexual assault advising office at Princeton.

Our custom hybrid retriever also did poorly. We designed this retriever to output two chunks from vector retrieval and two chunks from keyword retrieval to combat the issue with not finding proper nouns. However, this custom scheme degraded the quality of documents retrieved. Keyword retrieval in particular appeared to be fairly situational, and it often returned chunks that were irrelevant. On average, it appeared to be better to include more vector search results.

4.5. Evaluating Rerank Retrievers

Given the insights from the tests so far, we expected the `rerank` retriever to outperform the baseline. This retriever finds the top $4k$ results from vector search and then uses `LLMRerank` to select the top k chunks from that set ($k = 4$ in our testing). Retrieving a higher number of chunks from the vector store seemed to correlate with better performance, which made it surprising when this retriever failed to beat the baseline.

When we looked at the RAG pipeline through our web inspector, we noticed that the chatbot responses were still high quality and most provided context chunks were relevant. This led us to hypothesize that our tests underestimate the performance of reranking because of limitations with creating our synthetic dataset. When we made our dataset, we could not feasibly have a global view of all documents when labeling chunks as relevant or irrelevant. Our implicit assumption was that a given web page was the only page that had data helpful to answering the question. However, other chunks in the dataset could provide plausible answers to the same question despite not being identified as such. This issue is exaggerated when using the reranking pipeline since it retrieves 4x the number of chunks from vector search compared to the baseline. The probability of plausible unlabeled chunks being given to the chatbot becomes much higher. We confirmed a couple these cases, and so we believe the recall and precision for both `rerank` and `rerank + hybrid` slighly lower than what is reality.

Nevertheless, `rerank + hybrid` beat the baseline performance across metrics by a slight margin. The key insight was that instead of writing our own policy for merging vector and keyword-retrieved chunks (like in `hybrid`), we can delegate that job to the reranker which can prioritize the keyword chunks in the specific situations where they are more relevant than the vector ones. Qualitatively, this setup had the best responses and avoided the previously-observed failure cases.

4.6. Retriever Timing

Retriever	**Average time/query (s)**
`baseline`	0.22
`hyde`	3.62
`hybrid`	0.23
`rerank`	2.41
`rerank + hybrid`	2.33

Table 4: Average time to retrieve top-4 chunks per query.

Timing the execution of the different retrievers, we find that the ones implementing an LLM prompt (`hyde`, `rerank`, and `rerank + hybrid`) taking significantly more time to retrieve chunks for a query compared to the baseline. This is expected, since making an LLM API call adds network latency and generation time. The time spent retrieving documents is added directly to the time-to-first-token, leading to a less responsive model and worse user experience. Referring back to Table 2, we see that using `rerank + hybrid` retrieval would increase the time to a complete response by 22% compared to the GPT-3.5 baseline. The increased latency introduces another tradeoff to consider when building the RAG pipeline. Although `rerank + hybrid` has the best retrieval performance, it's debatable whether that gain is worth it given higher user wait times.

5. Conclusions

5.1. Difficulties with Datasets

Designing a good dataset to test this RAG system proved to be difficult. We first encountered challenges with making questions representative of real usage, which required manual intervention and rewriting. Then, we noticed that the text chunks in the dataset were not perfectly labeled. Our strategy for generating synthetic question-answer pairs could not label chunks as relevant outside of a single web page. Furthermore, throughout the entire dataset creation process, we needed to consider API costs, generation speed, and intermittent failures. While someone could use a complicated chain of prompts to form a high quality synthetic dataset, this pipeline might consume hundreds of dollars in API credits and take a long time to complete. As it is running, there is a high likelihood of an LLM call failing, whether the API provider had an error or the model did not generate an output in the right format. These barriers make it hard to create a synthetic dataset of nontrivial complexity. The limitations of the dataset used in this project contributed to us eliminating poor retrieval setups more than proving the effectiveness of more promising configurations. For definitively evaluating retrievers like `rerank + hybrid`, we would need to do a better job at labeling all relevant chunks across all pages.

5.2. Good Baseline Performance

We were surprised at how well the baseline top-*k* vector retrieval worked and how difficult it was to beat its performance. Even with advanced retriever methods, the metrics for the baseline remained comparable. It also offers significantly lower latency in retrieval. One hypothesis to explain this performance is that our dataset and use case is not complicated enough to justify using advanced RAG optimizations. The marginal gains in answer quality might not outweigh the slower retrieval pipeline. We need to continue monitoring the chatbot performance as we scale up the dataset size to see when the additional techniques become necessary. One future option might also be to have a lightweight classifier identify whether a query is simple or complex, and only call the slower but higher performing retrieval pipeline if the query is complicated enough.

Our recommendation right now would be to continue using the baseline chatbot setup, but add in additional context chunks from the keyword retriever to achieve the hybrid search effect. To compensate for the slowdown in generation speed, we would consider using a different model other than the ones provided by OpenAI. We saw that we could get away with using a lighter weight model for the chatbot without incurring a heavy loss in performance. There are several new models like Llama3 and Claude that could operate even faster and cheaper than GPT-3.5 and gracefully handle our increased chunk requirements.

5.3. Hybrid Search and Reranking

We should continue looking at reranking as a potential part of the retrieval pipeline. Rerankers give the chatbot an incredibly useful ability to merge retrieval results across sources into one set of top-*k*. In our testing, we made us of this property in `rerank + hybrid` to merge the chunks returned by the two index types without writing custom rules for how to do so. As seen in our tests of the `hybrid` retriever, writing our own policy for combining vector and keyword search results does not lead to good performance because of the situational utility of keyword retrieval. A reranker would have no problem including a relevant chunk retrieved using keywords in cases where it makes sense. As we continue to add data sources to the chatbot beyond a single database of documents, the need for reranking will become more profound. We would require a reranking model with significantly lower latency to make the tradeoff worth it right now.

5.4. Qualitative Feedback

Without qualitative feedback and testing, it would not be possible to evaluate the chatbot. A real user's perception of its performance stretches beyond any quantitative metric. We plan to beta test this chatbot extensively before releasing it to the broader student population at Princeton. We need to see what queries students will actually ask the chatbot, whether the system gives reasonable responses to opinion-based questions, and what additional data sources we should add. Right now, we've noticed that the dataset does not contain too many documents describing social life. This is likely a limitation of sourcing pages from the *princeton.edu* domain, which will not explicitly describe aspects of college life like parties or social groups. While there are possible ways to source and add this data into the chatbot, we do not know if student will even use the system for that purpose. It's possible that our users will only ask it for academic queries with concrete responses, or that they will expect a more subjective and opinionated conversation partner focused on campus life. Either way, we will not know this until we put the chatbot into the hands of students.

6. Next Steps

6.1. Crawling the Full Dataset

To deliver a fully-functional system to the student body, we need to expand the document dataset as much as possible. At a minimum, this means crawling all pages on the *princeton.edu* domain instead of a select few. Starting at some URL, the web crawler would need to recursively extract links to adjacent pages and download the text content. To handle the large number of pages, we would need to implement concurrency and a storage solution like a SQLite database to keep track of which pages have been traversed. Depending on the needs of our users, we may need to crawl other sites where Princeton students discuss and answer each other's questions about the university. For instance, the "real talk princeton" Tumblr page contains over 70,000 student Q&As collected over several years[9]. The introduction of these external data sources will require heavy validation and testing of the chatbot to ensure that it does not reflect any harmful biases in its responses. For now, we believe in keeping the chatbot limited to official university data. The crawling will also need to be rerun at regular intervals to keep the data updated.

6.2. Production System Design

This project has been running fully locally apart from the Streamlit demo, which is not a long term deployment solution. We will be architecting a system to serve the current chatbot to the student body while being highly extensible to new features. Our current inclination is toward using Postgres with the pgvector extension[5] to enable vector search. While Postgres does not have vector search capabilities out of the box, this extension lets the database become a versatile storage system for AI applications. Combining both traditional SQL and vector search in the same database query opens up exciting possibilities for retrieval. For instance, if a student asks the chatbot for events in the coming month relating to a specific topic, the system could first filter for events in the database based on date and then perform vector search only on the filtered results. A regular vector search pipeline would fail to utilize structured data like dates in this way. Keeping all data within Postgres, as opposed to using a dedicated vector store like Pinecone with a separate structured database, lets us perform these kinds of retrieval operations in a single step and reduces system complexity.

6.3. Agentic Capabilities

The long term vision is to develop a full-fledged agent to assist students throughout their college journey. The current RAG system is only the first step in this vision. Agents interact with tools, have personalized, long-term memory, and can plan effectively to execute complex tasks. These features can be reasonably achieved with the Princeton ecosystem. Student apps already have access to various APIs to retrieve data[12], and we can build additional infrastructure for logging all sorts of campus information. The agent can directly call these APIs and storage systems and execute autonomous actions to achieve the goals of the user. As the student interacts with the agent, it can learn the user's general preferences, their background, major, interests, and aspirations, tailoring all future conversations and giving more relevant responses. The Princeton CAS authentication system[1] is widely used to set up user accounts that integrate with the university login, and it can be used as a starting point for storing a personal chatbot memory for every student. The agent will ultimately tie together all the disparate data sources and APIs on campus into one unified interface.

6.4. Acknowledgements

I would like to thank my advisor, Professor Ravi Netravali, for his exceptional guidance during our weekly meetings, as well as the Princeton IW program for facilitating this research project.

References

[1] "Central Authentication Service | Department of Computer Science Computing Guide — csguide.cs.princeton.edu," https://csguide.cs.princeton.edu/publishing/cas, [Accessed 29-04-2024].

[2] "Correctness Evaluator - LlamaIndex — docs.llamaindex.ai," https://docs.llamaindex.ai/en/stable/examples/evaluation/correctness_eval/, [Accessed 29-04-2024].

[3] "Faithfulness Evaluator - LlamaIndex — docs.llamaindex.ai," https://docs.llamaindex.ai/en/stable/examples/evaluation/faithfulness_eval/, [Accessed 29-04-2024].

[4] "Fast Facts: Expenditures (75) — nces.ed.gov," https://nces.ed.gov/fastfacts/display.asp?id=75, [Accessed 29-04-2024].

[5] "GitHub - pgvector/pgvector: Open-source vector similarity search for Postgres — github.com," https://github.com/pgvector/pgvector, [Accessed 29-04-2024].

[6] "Introduction | Ragas — docs.ragas.io," https://docs.ragas.io/en/stable/index.html, [Accessed 29-04-2024].

[7] "LlamaIndex, Data Framework for LLM Applications — llamaindex.ai," https://www.llamaindex.ai/, [Accessed 29-04-2024].

[8] "Pricing — openai.com," https://openai.com/pricing, [Accessed 29-04-2024].

[9] "real talk princeton — realtalk-princeton.tumblr.com," https://realtalk-princeton.tumblr.com/, [Accessed 29-04-2024].

[10] "Relevancy Evaluator - LlamaIndex — docs.llamaindex.ai," https://docs.llamaindex.ai/en/stable/examples/evaluation/relevancy_eval/, [Accessed 29-04-2024].

[11] "The vector database to build knowledgeable AI | Pinecone — pinecone.io," https://www.pinecone.io/, [Accessed 29-04-2024].

[12] "tigerapps.org," https://tigerapps.org/resources.html, [Accessed 29-04-2024].

[13] "Usage Pattern - LlamaIndex — docs.llamaindex.ai," https://docs.llamaindex.ai/en/stable/module_guides/deploying/chat_engines/usage_pattern/, [Accessed 29-04-2024].

[14] "Using LLM's for Retrieval and Reranking — LlamaIndex, Data Framework for LLM Applications — llamaindex.ai," https://www.llamaindex.ai/blog/using-llms-for-retrieval-and-reranking-23cf2d3a14b6, [Accessed 29-04-2024].

[15] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[16] J. Chen, "How Perplexity.ai Is Pioneering The Future Of Search — forbes.com," https://www.forbes.com/sites/joannechen/2023/09/06/how-perplexityai-is-pioneering-the-future-of-search/?sh=2120165dad91, 2023, [Accessed 29-04-2024].

[17] L. Gao *et al.*, "Precise zero-shot dense retrieval without relevance labels," *arXiv preprint arXiv:2212.10496*, 2022.

[18] Y. Gao *et al.*, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2023.

[19] W. N. Greene, Sanders, "New and improved embedding model — openai.com," https://openai.com/blog/new-and-improved-embedding-model, 2022, [Accessed 29-04-2024].

[20] Hu, "Chatgpt sets record for fastest-growing user base - analyst note," https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/, 2023, [Accessed 28-04-2024].

[21] S. Jayaram Subramanya *et al.*, "Diskann: Fast accurate billion-point nearest neighbor search on a single node," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[22] X. Ma *et al.*, "Query rewriting for retrieval-augmented large language models," *arXiv preprint arXiv:2305.14283*, 2023.

[23] T. Mikolov *et al.*, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[24] A. Team, "Perplexity copilot enhancements," https://www.perplexity.ai/hub/blog/copilot-on-perplexity, 2023, [Accessed 28-04-2024].

[25] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[26] Y. Zhang *et al.*, "Siren's song in the ai ocean: a survey on hallucination in large language models," *arXiv preprint arXiv:2309.01219*, 2023.

[27] L. Zheng *et al.*, "Judging llm-as-a-judge with mt-bench and chatbot arena," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[28] S. Zhuang *et al.*, "Open-source large language models are strong zero-shot query likelihood models for document ranking," *arXiv preprint arXiv:2310.13243*, 2023.