Final Report

Title : Domain Specific Intelligent Chatbot for Student Support System.

Name : Ritesh Bhalerao

Application ID : ENGS2547

Name of Guide : Dr. Siba Kumar Udgata

Institution : University of Hyderabad.

Table of Content

Abstract

This report aims to provide information regarding the undertaken project as well as inform regarding the potential and the future scope. This project aims to develop a completely open source chatbot utilizing the power of powerful LLM by Meta - LLama 2. This project inculcates the following important aspects for the development of the chatbot -

1. Retrieval Aware Fine Tuning (RAFT) of LLama2.
2. QnA data generation via LLMs for fine tuning purposes
3. Implementation of Retrieval Augmented Generation (RAG) pipeline.
4. Fine tuning the retriever (Embedding model) for enhanced performance.

Problem definition

The primary problem addressed by this project is the need for an intelligent, domain-specialist chatbot that can provide comprehensive and accurate support to students at the University of Hyderabad. Current student support systems are often limited by a lack of scalability, inability to provide personalized responses, and reliance on human resources, which can lead to delays and inconsistencies in the information provided. The objective is to develop a chatbot leveraging advanced natural language processing (NLP) techniques, specifically using Meta's LLama 2 model, to create an open-source solution that can efficiently handle a wide range of student queries.

Introduction

This project focuses on the development of a sophisticated, domain-specific chatbot designed to assist students by utilizing state-of-the-art NLP technology. The chatbot is built upon Meta's LLama 2, a powerful language model, and involves several key stages. A crucial aspect of this project is the use of Retrieval Aware Fine Tuning (RAFT), a novel and advanced method specifically designed to fine-tune models for QnA tasks when context is provided. This method significantly enhances the chatbot's capability to function as a QnA agent within the Retrieval Augmented Generation (RAG) framework.

In addition to RAFT, the project involves fine-tuning the embedding model to further improve the performance of the retriever component. This ensures that the most relevant information is fetched, allowing the chatbot to provide precise and contextually appropriate responses.

Furthermore, a custom QnA dataset is generated from a comprehensive corpus of subject-specific information, ensuring that the chatbot is well-trained on the relevant academic and administrative content. This dataset generation is pivotal for fine-tuning the model to cater specifically to the needs of the students.

Significant progress has been made to create a base for this project, and the focus for future work shifts towards finding solutions to the identified problems during the course of this internship. The expected outcome is a scalable, efficient, and user-friendly chatbot that can significantly improve student support services at required institutions.

Literature review

The advancement of artificial intelligence (AI) and conversational agents, such as chatbots, has provided new opportunities for assisting student learning and support. Chatbots have been leveraged in campus environments to help students with various aspects of their university life, including information and services.

Existing research has explored the design and implementation of chatbots as virtual assistants to provide information to students. These studies have highlighted the potential of chatbots to serve as intelligent tutors by answering student questions, creating an environment for advanced learning, and supporting student well-being in an accessible, interactive, and privacy-enhanced way.

Furthermore, the use of chatbots in an educational context has been studied, with findings indicating that chatbots can be beneficial for providing answers to student queries, assisting with academic advising, and offering guidance on research and development related to academic pursuits.

A key aspect of this project is the use of Retrieval Aware Fine Tuning (RAFT), a novel technique for fine-tuning language models for QnA tasks when context is provided. This method has been shown to enhance the chatbot's capability to function as a QnA agent within the Retrieval Augmented Generation framework.

In addition, the project involves fine-tuning the embedding model to further improve the performance of the retriever component. This ensures that the most relevant information is fetched, allowing the chatbot to provide precise and contextually appropriate responses.

The literature review also highlights the importance of generating a custom QnA dataset from a comprehensive corpus of subject-specific information.

Overall, the literature suggests that the development of a domain-specific chatbot utilizing advanced NLP techniques, such as RAFT and Retrieval Augmented Generation, can significantly improve the effectiveness of student support services in academic institutions.

Methodology

Retrieval Augmented Generation

Ever since the rise of Large Language Models (LLMs), their applications have increased exponentially, transforming various industries by enabling sophisticated natural language processing capabilities. Today, most chatbots are built on top of these advanced LLMs due to their ability to understand and generate human-like text. However, despite their impressive capabilities, these generalized LLMs encounter two significant problems that impact their effectiveness.

Firstly, LLM training data tends to be hopelessly out-of-date. The training process involves feeding the model vast amounts of text data, but this data is static and represents a snapshot of knowledge at a specific point in time. Consequently, LLMs lack access to the most current information, which can be particularly problematic in fast-evolving fields where recent developments and real-time data are crucial.

Secondly, LLMs are prone to extrapolate when facts aren’t available, leading to the generation of false but plausible-sounding statements—a phenomenon known as "hallucination." When LLMs encounter gaps in their knowledge, they attempt to fill in the blanks based on patterns in the data they were trained on, often resulting in confident yet incorrect responses. This can undermine the reliability of chatbots and other applications that rely on accurate and up-to-date information.

Addressing these issues requires innovative approaches, such as incorporating retrieval mechanisms to access real-time data and fine-tuning models with domain-specific knowledge to minimize hallucinations and enhance accuracy.

Retrieval augmented generation (RAG) is a strategy that helps address both of these issues, pairing information retrieval with a set of carefully designed system prompts to anchor LLMs on precise, up-to-date, and pertinent information retrieved from an external knowledge store. Prompting LLMs with this contextual knowledge makes it possible to create domain-specific applications that require a deep and evolving understanding of facts, despite LLM training data remaining static.

Retrieval-Augmented Generation (RAG) is an advanced AI framework designed to enhance large language models (LLMs) by incorporating external knowledge bases, enabling the generation of accurate and up-to-date information while providing users with transparency into the generative process. This integration mitigates the inconsistencies and fabrications often seen in traditional LLMs by grounding responses in verifiable facts.

RAG offers multiple benefits, including improved accuracy and reliability, as it allows LLMs to access the most current and dependable information, thereby reducing the likelihood of false claims or hallucinations. Users can cross-reference the model's responses with the original sources, enhancing trustworthiness. Additionally, RAG is cost-effective since it relies on external data rather than requiring frequent and costly retraining of LLMs, making it particularly advantageous for enterprises needing to maintain updated models. Data security is another benefit, as grounding responses in external sources minimizes the risk of sensitive information leaks from the LLM's parameters.

RAG operates through two main phases: retrieval and generation. In the retrieval phase, algorithms search for relevant information snippets from external sources based on the user’s query, which are then appended to the user’s prompt. In the generative phase, the LLM synthesizes a response using both the augmented prompt and its internal knowledge, akin to an open-book exam where the model can refer to external content.

RAG can be implemented in both open-domain and closed-domain settings. In open-domain scenarios, such as consumer applications, the model retrieves information from the internet, whereas in closed-domain settings, like enterprises, retrieval is confined to specific trusted documents to ensure enhanced security and reliability.

One notable challenge with RAG is training models to recognize when they lack sufficient information to answer a query, as LLMs often attempt to provide an answer regardless of their knowledge gaps. Continuous fine-tuning and exposure to a variety of queries can help LLMs learn to identify unanswerable questions and seek additional details when necessary.

Future directions for RAG development focus on improving both retrieval and generation processes, enhancing the efficiency of retrieving relevant information, and structuring it effectively for LLMs. Integrating vector databases to index and retrieve data more efficiently is a promising avenue for boosting RAG's performance.

Overall, RAG represents a significant advancement in AI by addressing some of the intrinsic limitations of LLMs, such as accuracy, reliability, and cost-effectiveness, through grounding models in external, verifiable knowledge. As research continues, further innovations in RAG are expected to refine its capabilities and expand its applications across various domains, making it a pivotal technology in the future of AI.

Now let us consider the simplest RAG pipeline and expand upon each component in brief.

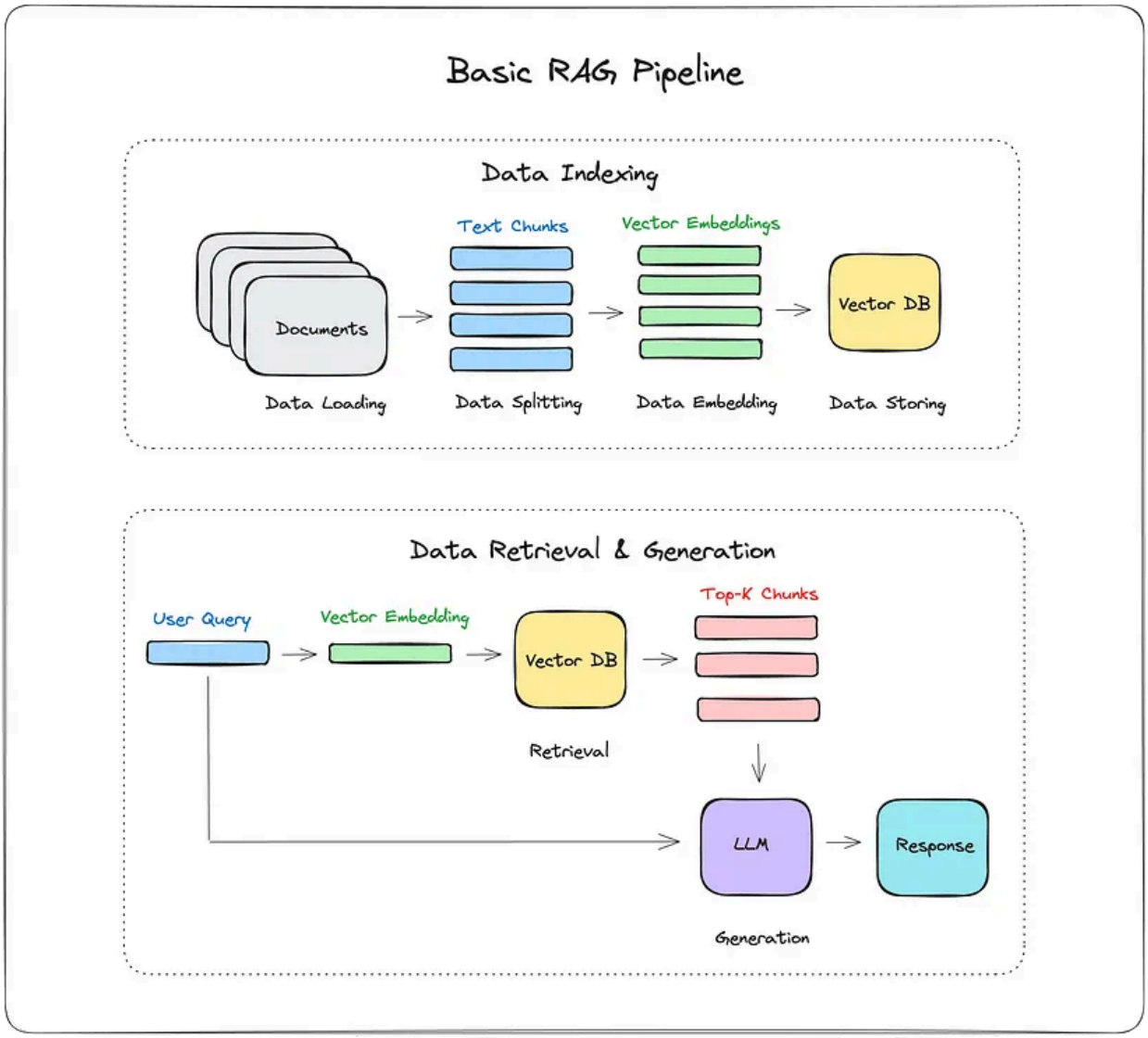


Fig 1

The Basic Retrieval-Augmented Generation (RAG) Pipeline operates through two main phases:

1. Data Indexing

2. Data Retrieval & Generation

Data Indexing

The data indexing can be further divided into two major sections/processes that are Chunking & Embedding

Chunking

It is a crucial step in Retrieval-Augmented Generation (RAG) as it breaks down long documents into smaller, more manageable units. These units, called chunks or passages, are then used for efficient retrieval and provide more focused context for the LLM during response generation.

Advantages of Chunking :

- Improved Retrieval Efficiency: Smaller units allow for faster and more focused similarity searches in the vector database.
- Enhanced LLM Context: Chunks provide the LLM with more specific information for generating relevant and informative responses.
- Flexibility in Retrieval: Different chunks can be retrieved depending on the specific query, leading to more nuanced responses.

Disadvantages of Chunking :

- Information Loss: Chunking can lead to some information loss at boundaries.
- Increased Computational Cost: Chunking adds an additional processing step that may impact overall system speed.
- Tuning for Optimal Performance: Choosing the right chunking technique and parameters requires careful experimentation.

Chunking Techniques

1. Fixed-Size Chunking:

The text is chopped up into chunks based on a predetermined number of characters (e.g., 512 tokens). Each chunk becomes a unit for processing within the RAG model.

Advantages:

Simplicity: Fixed-size chunking is very easy to implement. You just define the desired chunk size and split the text accordingly.
Efficiency: This method is computationally efficient because it doesn't require complex analysis of the text structure.

Disadvantages:

Loss of Context: Since it relies solely on character count, fixed-size chunking can break sentences or paragraphs in half, potentially disrupting the meaning and flow of the text.

Ignores Structure: This approach disregards the inherent structure of the text, such as sentences or paragraphs. This can be problematic for tasks where capturing semantic relationships is crucial.

2. Sentence-Based Chunking:
The text is divided into individual sentences using full stop (.), question mark (?) and exclamation mark (!) as delimiters. Each sentence becomes a separate chunk for processing within the RAG model. It is usually used where understanding the meaning within each sentence is important.

Advantages:

Preserves Meaning: Sentence-based chunking helps retain the semantic sense of the text by keeping sentences intact. This ensures the model receives contextually relevant units for information retrieval and generation.

Improved Efficiency: Compared to fixed-size chunking, it's still a relatively simple method to implement, making it computationally efficient.

Disadvantages:

Limited Scope: While it maintains sentence-level meaning, sentence-based chunking might not capture broader relationships between sentences that span multiple chunks.

Potential Redundancy: Depending on the nature of the text, there could be redundancy across consecutive sentences within a chunk.

3. Sliding Window Chunking:

Sliding window chunking is a technique used to segment data streams or text into overlapping chunks. It combines elements of fixed-size chunking and sentence-based chunking, offering some advantages over both. Unlike fixed-size chunking with clean cuts, the window in sliding window chunking overlaps with the previous window by a certain amount. This overlap ensures that context is preserved across chunks.

Advantages:

Preserve Context: By incorporating overlap, sliding window chunking avoids the issue of breaking sentences or important information at chunk boundaries, which can happen in fixed-size chunking.

Flexibility: The window size and overlap size can be adjusted based on the specific needs of the task. For instance, a larger window with more overlap might be useful for capturing long-range dependencies in the data, while a smaller window with less overlap might be better for tasks requiring more granular analysis.

Disadvantages:

Increased Complexity: Compared to fixed-size chunking or sentence-based chunking, implementing a sliding window requires handling overlaps and potential redundancies across chunks, making it slightly more complex.

Computational Overhead: The overlapping nature can lead to some redundancy in the processed data, potentially increasing computational demands. Deciding the right chuck size and overlap is purely based on experimentation. Please refer to this wonderful blog post to see how you can achieve suitable chuck parameters for your use case.

4. Recursive Character Text Splitter:

As the name suggests, it recursively tries to split the data. The splitter has a set of pre-defined characters it uses to attempt to split the text. These characters typically include things like newline characters (\n), double newline characters (\n\n), spaces ( ), and tabs (\t).

It prioritizes larger chunks. It starts by trying to split the text using the first character on the list (e.g., double newline). If the resulting chunks are still larger than a specified

chunk size, it moves on to the next character type (e.g., newline) and attempts to split again. This continues until the chunks are all smaller than the chunk size or it runs out of character types to try.

Advantages:

Preserves Meaning: By prioritizing splits at natural text boundaries like paragraphs and sentences (using newlines and spaces), the Recursive Character Text Splitter aims to keep semantically related pieces of text together. This helps maintain the overall flow and meaning of the text.

Flexibility: It can be customized with different character sets to prioritize specific splitting behaviours. For instance, you could add punctuation marks to the list if you want to ensure sentences aren't broken up.

Disadvantages:

Computational Cost: Recursion can be computationally expensive for very large texts.
Potential Oversplitting: Depending on the character set and chunk size, there's a possibility of oversplitting the text, creating very small chunks that might not be ideal for all applications.

5. Embedding based chunking
The first step involves dividing the text into smaller segments using various chunking techniques which we have learned so far.Once the text is chunked, each chunk is fed into an embedding model. This model transforms the chunk of text into a dense vector representation. Calculate the semantic similarity (cosine similarity) of the adjacent chucks. If the similarity is above a specific threshold. Merge the 2 chunks.

Advantages:

Enhanced Text Understanding: By considering meaning, semantic chunking can capture complex relationships within the text that might be missed by simpler chunking methods. This deeper understanding can be valuable for various NLP tasks.

Improved Task Performance: In tasks like question answering, text summarization, or sentiment analysis, semantic chunking can help the model focus on the relevant parts of the text and improve the overall performance.

Disadvantages:

Computational Complexity: The reliance on advanced NLP techniques can make semantic chunking computationally expensive, especially for large amounts of text.

Data Dependency: The effectiveness of semantic chunking can be highly dependent on the quality of the training data used for the NLP models involved.

Embeddings

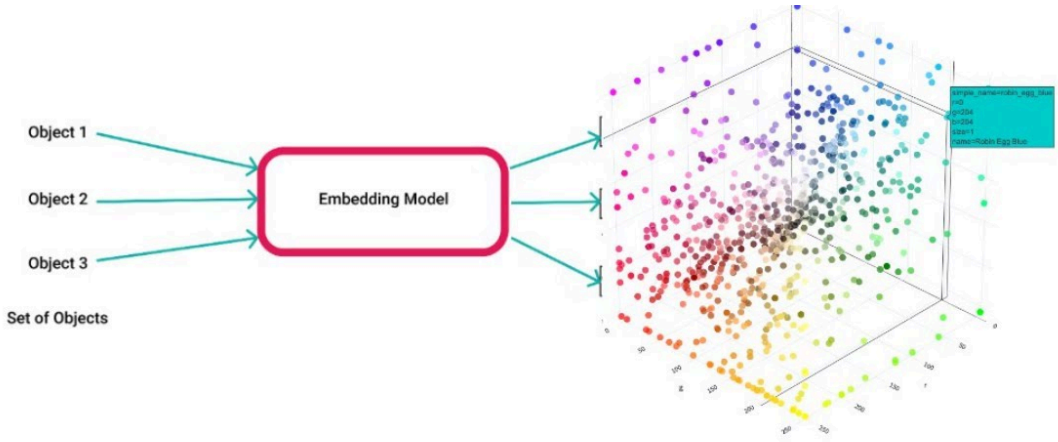


Fig 2

Embeddings are numerical representations of real-world objects like text, images, and audio, created by machine learning models. They translate these objects into a mathematical form that captures their semantic meaning and relationships. This process allows models to perform similarity searches, finding and comparing similar objects based on their vector representations.

Vectors are arrays of numbers representing points in a multi-dimensional space. Each dimension captures specific characteristics of the object, enabling the model to understand complex relationships. For example, a city can be represented by dimensions like latitude, longitude, and population.

Embeddings are generated using deep learning models, often through neural networks. These networks have multiple layers, including an embedding layer that converts input data into vectors. The embedding layer learns to represent data in a high-dimensional space, allowing for effective similarity searches.

In LLMs, embeddings are used to represent the context of words, sentences, and entire documents. This enables the models to understand and generate human-like text by considering semantic relationships. Embeddings make it possible to store and retrieve contextual information efficiently, enhancing the model's performance.

In the Retrieval-Augmented Generation (RAG) pipeline, embeddings play a crucial role in the retrieval phase. When a user query is received, the system searches the vector database for relevant information. The vector database, populated with embeddings of documents, enables quick and accurate retrieval of top-K chunks of relevant data. These chunks are then used to augment the user's prompt, helping the LLM generate more accurate and contextually grounded responses.

Some Advantages are as follows

- Efficiency: Embeddings enable quick similarity searches, improving retrieval speed.
- Accuracy: Embeddings capture semantic relationships, enhancing the relevance of retrieved information.
- Scalability: Embeddings can represent complex, multi-dimensional data, making them suitable for diverse applications.
- Dimensionality: Managing high-dimensional embeddings can be computationally intensive.
- Fine-tuning: Continuous fine-tuning of embeddings is necessary to maintain accuracy and relevance.

Overall, embeddings are fundamental in the RAG pipeline, enhancing the model's ability to retrieve and generate accurate, contextually relevant information by grounding responses in external, verifiable knowledge.
After Chunking & Embedding a vector Index is created upon which the retriever runs queries to fetch the most appropriate vectors corresponding to their respective chunks and thereby retrieving the potentially most relevant context for further processing.

Data Retrieval & Generation

Retrieval

Retrieval is the process of finding documents or passages from a large collection that are most relevant to a given query. In RAG, the retrieved documents become the source material for the LLM to understand the context and generate its response.

Efficiency: Retrieval methods need to be efficient, especially for large document collections. Techniques like indexing and caching can significantly improve retrieval speed.
Relevance Tuning: Retrieval models can be tuned based on specific tasks and user needs. For instance, you might prioritize documents with factual information for some queries or creative content for others.

Types of Search Methods:

Keyword Search: Basic method where documents are matched based on specific keywords present in the query. Simple and efficient but can struggle with synonyms and complex queries.
Vector Search: Represents documents and queries as vectors in a high-dimensional space, where semantic similarities are captured. Efficient for large datasets and better at handling semantic relationships.
Hybrid Search: Combines multiple methods (e.g., keyword search for initial retrieval and vector search for reranking) to leverage their respective strengths.

Reranking:

Importance: After initial retrieval, reranking is crucial because the first set of retrieved documents may not be perfectly relevant or diverse enough.

Purposes of Reranking:

- Enhancing relevance and diversity in retrieved documents.
- Improving the quality of information fed into the LLM.
- Reducing noise by filtering out irrelevant or redundant documents.

Approaches to Reranking:

Learning to Rank (LTR): Uses machine learning models to refine the initial retrieval results. Models learn from labeled data to prioritize documents most relevant to the query. Pointwise, Pairwise, and Listwise Methods: Different approaches within LTR for ranking documents based on relevance scores.

Cross-Encoders: Neural network models that encode both the query and documents into a shared vector space to measure semantic similarity.

Maximum Marginal Relevance (MMR): Balances relevance and diversity by iteratively selecting documents based on their relevance to the query and their dissimilarity to already selected documents.

BM25: A scoring method that considers factors like term frequency and inverse document frequency to rank documents based on relevance to the query.

LLM ReRanker:

Concept: Uses the LLM itself to rerank documents, leveraging its understanding of language and context to assess relevance.

Implementation: Can involve generative ranking (LLM generates scores for documents) or extractive ranking (LLM identifies key information from documents to guide reranking).

Later on this retrieved context along with the user's query is structured into a suitable prompt for the LLM in use and the response is generated by the LLM via next token prediction.

Workflow

For this particular ChatBot development Retrieval Aware FineTuned (RAFT) Llama 2 7B model along with a fine-tuned Embedding model BAAI/BGE-v1.5-en was utilized into the RAG pipeline. A synthetic data set was generated for fine tuning purposes. Following is detailed overview of entire workflow and the steps involved.

Data Generation

In RAFT, we create training data that includes a question (Q), some documents (Dk), and a corresponding chain-of-thought (CoT) answer (A*) that’s based on information from

one of the documents (D*). We distinguish between two types of documents: the 'oracle' documents (D*) that have the information needed for the answer, and 'distractor' documents (Di) that don't help with the answer. Some of the training involves having the right document along with distractions, while other times, we only include distractor documents to encourage the model to rely on its memory rather than just the documents provided.

For the creation of this kind of dataset, OpenAI's GPT 3.5 was utilized. The data was generated from corpus gathered from various books namely - Galvin, Tanenbaum & Stevens. A total of 4k samples were generated, which are not even close to sufficient, but good enough to demonstrate the potential of this project considering the time & resources available.

```
Question: The Oberoi family is part of a hotel company that has a head office in what
city?

context: [The Oberoi family is an Indian family that is famous for its involvement in
hotels, namely through The Oberoi Group]...[It is located in city center of Jakarta,
near Mega Kuningan, adjacent to the sister JW Marriott Hotel. It is operated by The
Ritz-Carlton Hotel Company. The complex has two towers that comprises a hotel and the
Airlangga Apartment respectively]...[The Oberoi Group is a hotel company with its head
office in Delhi.]

Instruction: Given the question, context and answer above, provide a logical reasoning
for that answer. Please use the format of: ##Reason: {reason} ##Answer: {answer}.

--------------------------------------------------------------------------------

CoT Answer: ##Reason: The document ##begin_quote## The Oberoi family is an Indian
family that is famous for its involvement in hotels, namely through The Oberoi Group.
##end_quote## establishes that the Oberoi family is involved in the Oberoi group, and
the document ##begin_quote## The Oberoi Group is a hotel company with its head office
in Delhi. ##end_quote## establishes the head office of The Oberoi Group. Therefore,
the Oberoi family is part of a hotel company whose head office is in Delhi. ##Answer:
Delhi
```

Fig 3

Figure 3 shows an example data sample that would be generated for RAFT.

Fine Tuning LLM

Once the training data is prepared, the fine-tuning process involves:

Training the model: Using supervised learning, the model is trained to generate chain-of-thought style answers from the provided context, which includes questions and sets of documents.

Objective: The goal is for the model to learn to identify and prioritize relevant information from the oracle documents while disregarding the distractor documents.

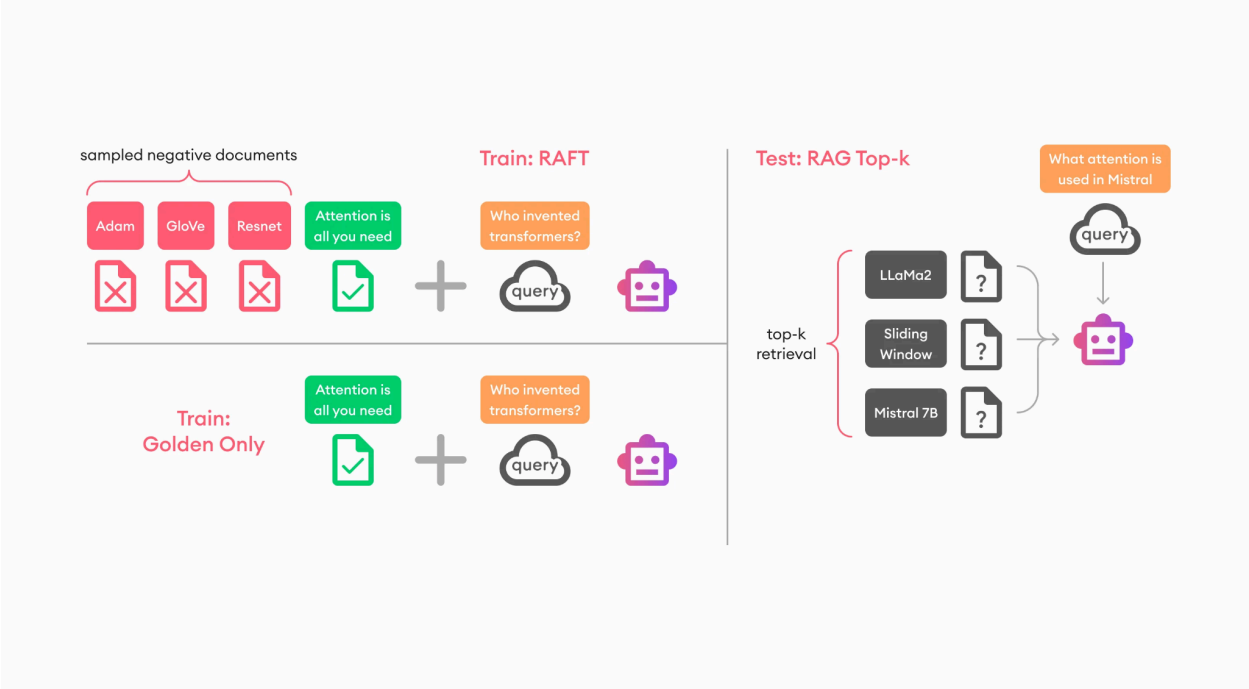


Fig 4

Fine-tuning and retrieval augmented fine-tuning (RAFT) are advanced techniques used to optimize large language models (LLMs) for specific domains or tasks, enhancing their accuracy and applicability.

Fine-tuning involves training an LLM on domain-specific datasets before deployment. Unlike retrieval-augmented generation (RAG), which queries external sources in real-time, fine-tuned models are pre-trained on domain-specific documents. This method reduces "hallucinations" (incorrect or implausible outputs) and enhances domain authority by incorporating specific terminology, nuances, and style relevant to the domain. Fine-tuning is efficient with lower latency compared to RAG since it doesn't require real-time queries but depends heavily on access to large, high-quality training datasets. Periodic retraining is necessary to maintain relevance and accuracy as domain knowledge evolves.

Fine-tuned models may lack the ability to access the most current information, as they are not dynamically querying external sources like RAG models. They can also be less

interpretable due to their complex training processes, and the need for substantial initial training data can pose challenges in some contexts.

RAFT combines the strengths of RAG and fine-tuning by integrating real-time retrieval with domain-specific training. Proposed as a hybrid approach, RAFT aims to prime LLMs with domain-specific knowledge through pre-training on relevant documents while leveraging real-time retrieval for enhanced accuracy and adaptability. This method ensures that the model has access to up-to-date information during generation, akin to an "open-book exam" scenario where it has studied relevant documents beforehand.

Benefits of RAFT:

- Improved Accuracy: RAFT enhances model accuracy and reduces hallucinations compared to both RAG and traditional fine-tuning by combining domain-specific training with real-time information retrieval.
- Efficiency: It improves efficiency and speed by reducing the need for large-scale fine-tuning datasets and leveraging real-time retrieval.
- Scalability: RAFT models are easier to scale and adapt to new domains since they expand retriever data rather than requiring retraining of the entire model.
- Versatility: They have demonstrated competence across various applications such as product recommendations, sales strategy, FAQ automation, and market trend analysis.

Researchers have successfully applied RAFT in diverse domains, enhancing LLM capabilities in niche applications. By integrating domain-specific training with real-time retrieval, RAFT provides a robust solution for improving model performance in complex tasks that require both deep domain knowledge and real-time data access.

In conclusion, while traditional fine-tuning optimizes LLMs for specific domains through pre-training on relevant datasets, RAFT takes a step further by integrating real-time retrieval with domain-specific training. This hybrid approach not only enhances model accuracy and efficiency but also ensures adaptability across different domains and applications, making it a suitable training paradigm for our use-case.

Fine Tuning Embedding model

Customizing embedding models is crucial for enhancing the effectiveness of retrieval-augmented generation (RAG) applications, especially in domain-specific contexts where standard embeddings may not capture the nuances and specific

knowledge required. Embedding models like BGE-large en v1.5 can be fine-tuned specifically for company or domain-specific data to significantly boost retrieval performance.

In this process, the model is trained on an anchor-positive pairs dataset. Anchors typically consist of questions or queries, while positives are the context or answers derived from relevant documents within the domain. This approach ensures that the embeddings are optimized to retrieve contextually relevant information based on the queries posed, enhancing the quality of retrieval.

Fine-tuning BGE-large en v1.5 embeddings for such purposes involves training the model on domain-specific datasets, which helps it learn the specific nuances, terminology, and contextual relationships within that domain. This customization reduces the risk of retrieving irrelevant or outdated information, improving the overall accuracy and relevance of responses generated by the RAG application.

By integrating domain-specific training into embedding models like BGE-large en v1.5, we can achieve more effective and reliable information retrieval capabilities tailored to our specific needs and contexts. This approach not only enhances the utility of RAG applications but also supports more accurate decision-making and knowledge dissemination within specialized domains.

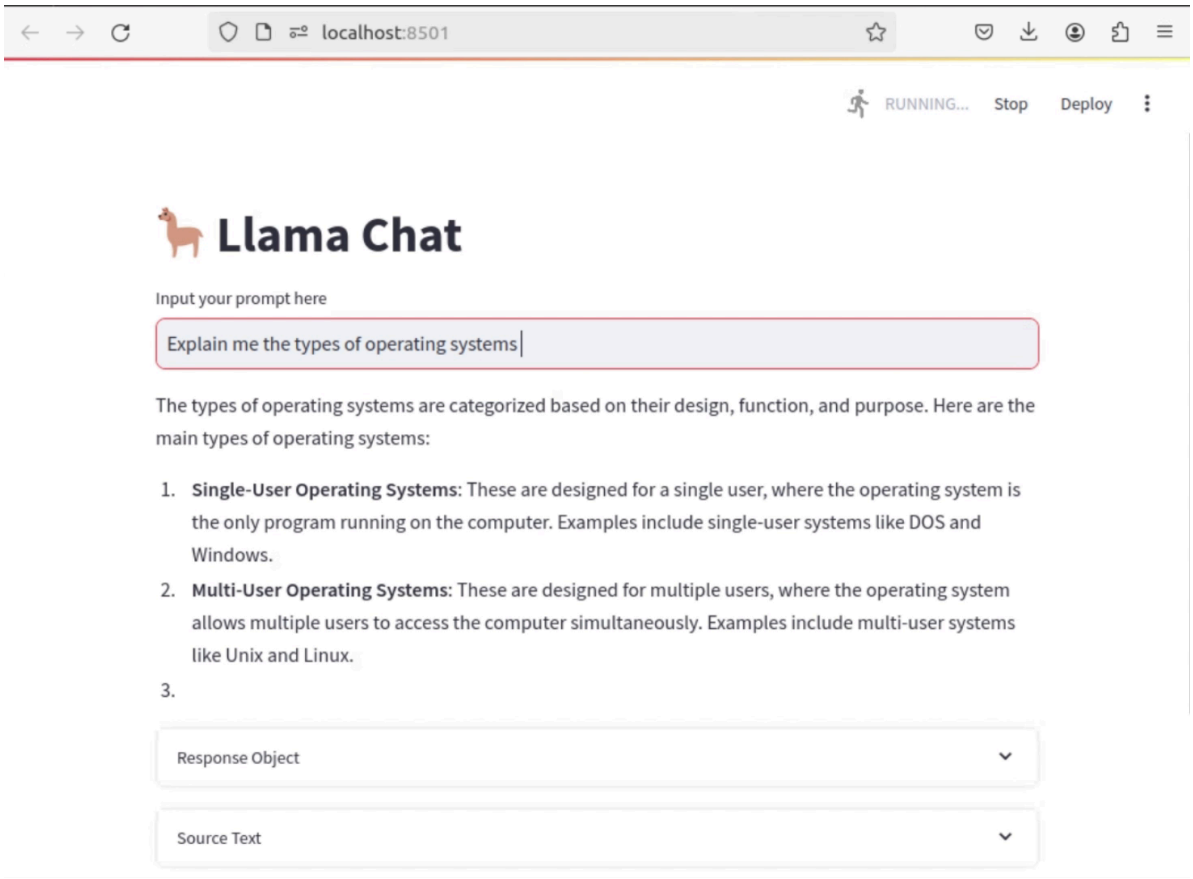
Adjustments made

For our particular use case, considering the resources, time and the aim of the project, some adjustments were done.

- Quantized Llama 2 model has been used instead of full precision to make it feasible to run
- Parameter Efficient Fine Tuning has been adopted instead of full finetuning
- Not using larger embeddings which support bigger context sizes to reduce the latency
- Loading the model in 4bit at inference time as well
- Limited data generation

Results

UI Snapshot



Repository link

https://github.com/Riteshbhalerao11/Llama-OS

Future work

A complete base for future work has been created, many potential problems have been identified. RAG pipeline along with the RAFT pipelines and Embedding finetuning are in place and set up. Some of the problems identified are as follows -

1. Using Llama 2 like models which are humongous in size are not efficient for small scale applications like a subject specific OS when it comes to target users being students or institutions.
2. These large models add to the cost and compute significantly to make it a usable system.
3. Training these models is expensive.

One promising approach is to adopt smaller, bespoke chat models tailored specifically for smaller-scale applications instead of relying on larger pre-trained language models like LLaMA 2. For instance, developing an in-house trained model such as LongFormer (approximately 600 million parameters) designed for operating systems could offer significant advantages. Compared to LLaMA 2, which requires substantially larger resources, LongFormer operates efficiently on smaller GPUs, delivering faster performance while being significantly more compact—approximately 1000 times smaller in size. This strategic shift not only optimizes resource utilization but also enhances responsiveness tailored to specific operational needs.

Conclusion

In conclusion, the development of a domain-specific intelligent chatbot for student support systems at the University of Hyderabad has been a great learning experience, integrating advanced NLP techniques like RAFT and Retrieval-Augmented Generation (RAG). Leveraging Meta's LLama 2 model and fine-tuned embeddings, the project aimed to enhance the accuracy, efficiency, and scalability of student services. Future work will focus on optimizing model efficiency and scalability while exploring alternative, cost-effective AI solutions tailored to smaller-scale applications.

References

1. Zhang, Tianjun, et al. "Raft: Adapting language model to domain specific rag." *arXiv preprint arXiv:2403.10131* (2024).

2. Touvron, Hugo, et al. "Llama 2: Open foundation and fine-tuned chat models." *arXiv preprint arXiv:2307.09288* (2023).

3. FlagOpen. "FlagEmbedding." GitHub, 2024, https://github.com/FlagOpen/FlagEmbedding.

4. Drjulija. (2024). What is Retrieval-Augmented Generation (RAG)? Medium. Retrieved July 10, 2024, from https://medium.com/@drjulija/what-is-retrieval-augmented-generation-rag-938e4f6e03d1

5. Mehul, J. (2024). RAG Part 1: From Naive to Advanced. Medium. Retrieved July 10, 2024, from https://medium.com/@j13mehul/rag-part-1-from-naive-to-advanced-cb40674a7738

6. Mehul, J. (2024). RAG Part 2: Chunking. Medium. Retrieved July 10, 2024, from https://medium.com/@j13mehul/rag-part-2-chunking-8b68006eefc1

7. Mehul, J. (2024). RAG Part 3: Embeddings. Medium. Retrieved July 10, 2024, from https://medium.com/@j13mehul/rag-part-3-embeddings-ff415eb9fed9