

LLM Fine-tuning Challenge

Enhancing Qwen 2.5 3B for AI Research QA

[Github Repo](#)

[HuggingFace gguf model](#)

Using this report as our guide, we'll walk through each step we took to create this chatbot, designed to respond to questions by drawing insights from the latest AI research papers, blogs, and related documents.

1. Dataset Preparation

In addition to the 6 provided text and markdown files, we gathered data from Wikipedia, focusing on topics like "Artificial Intelligence," "Natural Language Processing," "DeepSeek," "Deep Learning," and "AI Research."

Why We Did It

While the given files were useful, they were somewhat limited in scope. By incorporating Wikipedia, we were able to broaden the dataset, combining the specific details from the provided files with the expansive knowledge from Wikipedia. This combination allows us to address AI research questions from both narrow and broad perspectives.

Other Approaches We Could've Taken

- Stick to the 6 original files and avoid any additional data.
- Write our own questions and answers from scratch.
- Use an existing dataset like SQuAD and modify it to suit our needs.

Pros

- Combines the focused content of the files with general AI knowledge.
- No need to create everything from scratch—Wikipedia already provides a wealth of information.
- A wider variety of topics leads to more diverse and interesting questions.

Cons

- Wikipedia covers a broad range but may not go as in-depth as the provided files.
- We had to manually fetch the Wikipedia pages, which could mean missing out on some relevant content

2. Dataset preprocessing

What We Did

To clean up the text, we started by removing extra spaces using a simple `" ".join(doc.split())` method. This trick tidied up the text and ensured everything was neatly formatted. Next, we used LangChain's `CharacterTextSplitter` to break the text into 500-character chunks, with a 50-character overlap between each chunk. This resulted in a well-organized list of `langchain_docs` pieces.

Why We Did It

The raw files and Wikipedia pages were quite messy, with long blocks of text that could overwhelm the system if left unprocessed. Removing unnecessary spaces ensured that no words were awkwardly split or spaced out. By chunking the text into 500-character sections, we made it manageable for both the vector database and the model. The overlap between chunks helps maintain context, so they don't lose track of each other when being processed later.

How We Did It

- We looped through each file and Wikipedia page to eliminate extra spaces.
- We then used the `CharacterTextSplitter` with a `chunk_size=500` and `chunk_overlap=50`, ensuring each chunk was clean and kept some context from the previous chunk.
- All the resulting chunks were then combined into one large list of `Document` objects for LangChain.

This resulted in clean, chunked-up `langchain_docs` that are ready for embedding and retrieval.

Why We Avoided Complex Preprocessing

Instead of using more complicated preprocessing methods, we chose to

keep things simple in order to preserve context. Complex transformations could risk distorting the relationships between different parts of the text, which is crucial for maintaining the flow of information across chunks. Our goal was to keep the text as intact as possible while ensuring it could be processed efficiently.

3. Synthetic dataset generation

What We Did -Created 100 Q&A Pairs with RAGAS

We combined our collection of 6 files plus Wikipedia and fed it into RAGAS using GPT-4o to generate 100 question-answer pairs. These were divided into 80 pairs for training (though not used here) and 20 for testing. We also instructed RAGAS to vary the difficulty of the questions, 50% simple questions, 30% reasoning-based ones, and 20% requiring multiple pieces of information to answer.

Why We Did It

Writing 100 questions by hand wasn't feasible, so we used RAGAS as a time-saver. With GPT-4o's capabilities, we were able to quickly generate high-quality Q&A pairs relevant to AI research, matching the scope of our dataset.

Other Approaches We Could've Taken

- Write all 100 Q&A pairs manually.
- Use an existing dataset and adapt it to our needs.
- Skip GPT-4o and rely on a basic question generator.

Pros

- RAGAS handled the bulk of the work in a short time.
- The generated questions stayed closely aligned with the topics in our files and Wikipedia.
- We achieved a balanced split for testing (and potentially training later).

Cons

- GPT-4o may have introduced unexpected twists in the generated questions.

- The model could smooth over certain nuances of the original texts, possibly losing some of the original quirks or details.

Mar 8 10:51 AM

app.ragas.io/dashboard/alignment/testset/cec48724-d2ae-4479-8e83-108e66d42ded

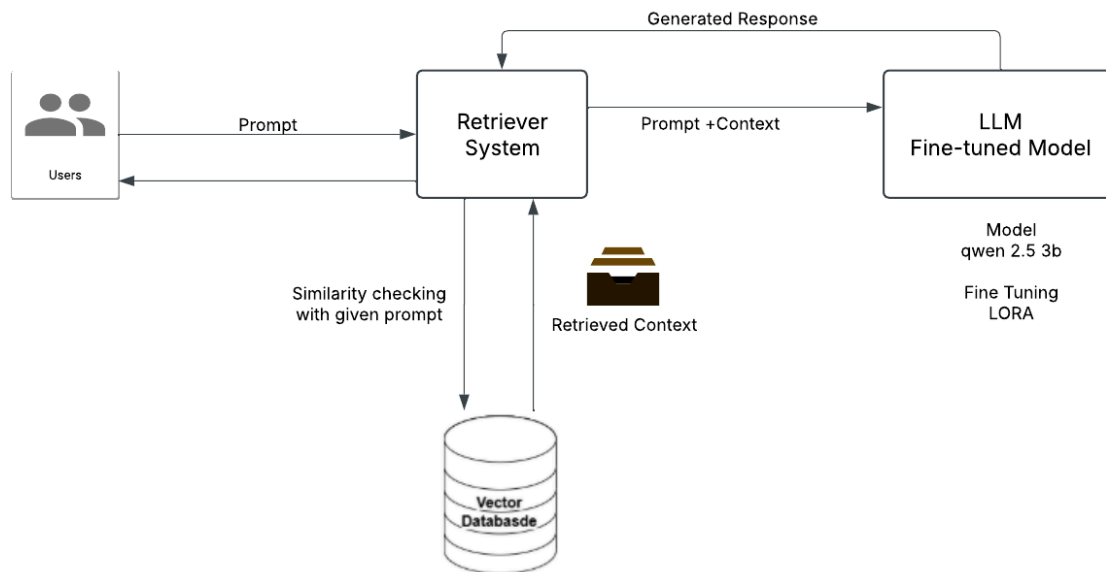
ragas beta

Docs Contact Us Dashboard App Tokens Signout

TestSet [Share TestSet](#) [Download TestSet](#)

	Input	Reference Contexts	Reference	Synthesizer	Accept/Reject
1	Who is Wenfeng Liang in the context of DualPipe development?	DualPipe DualPipe is an innovative bidirectional pipeline parallelism algorithm introduced in the...	Wenfeng Liang is one of the creators and developers of the DualPipe bidirectional pipeline...	single hop specific query synthesizer	<input checked="" type="checkbox"/> <input type="checkbox"/>
2	What are the key features of the Fire-Flyer File System in distributed storage?	Fire-Flyer File system The Fire-Flyer File System (3FS) is a high-performance distributed file system...	The Fire-Flyer File System (3FS) is a high-performance distributed file system design...	single hop specific query synthesizer	<input checked="" type="checkbox"/> <input type="checkbox"/>
3	What InfiniBand do in 3FS system?	Design Notes Design and Implementation The 3FS system has four components: cluster manager,...	In the 3FS system, InfiniBand is used to connect all components in an RDMA network, facilitatin...	single hop specific query synthesizer	<input checked="" type="checkbox"/> <input type="checkbox"/>
4	What is the purpose of the DENT prefix in directory entry keys	File metadata store Location of file chunks 3FS divides file data into	The DENT prefix in directory entry keys is used to compose	single hop specific query synthesizer	<input checked="" type="checkbox"/> <input type="checkbox"/>

4. Architecture of the approach



Lucid chart: [Click here](#)

The picture shows our chatbot architecture. We put together a Retrieval-Augmented Generation (RAG) system. The user asks a question (the prompt), and our **Retriever System** digs into a **Vector Database (Chroma)** to find the best matching info from our 6 files and Wikipedia pages. It grabs the top-3 relevant chunks, mixes them with the question into a "Prompt + Context," and hands it over to the fine-tuned **Qwen 2.5 3B-Instruct** model to whip up an answer. That answer—the **Generated Response**—gets sent back to the user.

Why We Did It

We wanted a setup that could answer AI research questions without needing a ton of training. The retriever (with ColBERT and Chroma) acts like a research assistant, pulling out the right info, while Qwen plays the role of a smart conversationalist who uses that info to answer.

By combining retrieval and generation, we ensure that the chatbot can provide accurate, contextually aware responses without the need for exhaustive training on every potential question. This approach significantly reduces the training time required while maintaining high-quality, relevant answers. Furthermore, using Chroma's vector database ensures that the system scales efficiently as the data set expands, making it adaptable to new information as needed.

Additionally, we loaded the qwen2.5 3b model using `AutoModelForCausalLM` from Hugging Face's transformers library. We skipped unsloth for this because we hit a pesky conflict between unsloth and transformers.

5. Fine tuning approach

Method	Approach	Advantages	Disadvantages
Pre-trained model fine-tuning	Use a pre-trained LLM and adapt it to our codebase.	Faster, cheaper(Parameter efficient fine tuning) , and leverages existing models	Requires significant computing and data and limited control over core behavior
Hybrid Approach (Adapters and LoRA)	Modify specific layers of a pre-trained model.	Low-cost tuning with focused customization.	Limited to specific layers, may not match full fine-tuning
Retrieval-Augmented Generation (RAG)	Combine LLMs with a knowledge retrieval system.	Provides accurate, updated answers, and reduces model memory needs.	Slower responses, complex infrastructure
Knowledge Distillation	Train a smaller model to mimic a larger one.	Creates lightweight, efficient models.	Still needs quality data and teacher outputs.
Transfer Learning	Use a foundation model and fine-tune it.	Built-in knowledge of programming, easy adaptation.	Inherits biases,compute-intensive, licensing restrictions

What We Did, Fine-tuned Qwen with LoRA and Added RAG for Extra Smarts

We took a two-pronged approach by combining Pre-trained Model Fine-tuning with Retrieval-Augmented Generation (RAG). To fine-tune Qwen 2.5 3B, we used LoRA (Low-Rank Adaptation), a Parameter-Efficient

Fine-Tuning (PEFT) method. Instead of retraining the entire model, we applied LoRA to adjust only a small portion of Qwen's parameters. We trained it using our synthetic dataset (synthetic_train) created with RAGAS, which consisted of 80 Q&A pairs. Then, we integrated Qwen with a RAG system, where a retriever (powered by ColBERT and Chroma) fetches the top-3 relevant document chunks, and Qwen uses this context to generate answers. It's like giving Qwen a quick study session on AI research topics, with the retriever acting as a research assistant to provide the right context when needed.

Why We Did It

We chose LoRA because it strikes a perfect balance—allowing us to fine-tune Qwen with minimal computational resources, which was ideal for our Kaggle T4 setup. Fine-tuning with our synthetic dataset made Qwen more attuned to AI research topics like DeepSeek and DualPipe. The addition of RAG means Qwen doesn't need to memorize everything. Instead, it pulls in relevant context from our 6 files and Wikipedia pages when a question arises. This setup gave Qwen a quick refresher course and the ability to retrieve important information on demand, keeping the system both efficient and responsive.

Other Approaches We Could've Taken

- Use the pre-trained Qwen model with RAG, without any LoRA fine-tuning.
- Retrain all of Qwen's parameters, rather than just a subset with LoRA.
- Add small trainable layers instead of using LoRA's low-rank updates.
- Train a smaller model that mimics Qwen's behavior.

Pros

- LoRA fine-tuning makes Qwen more in-tune with our specific AI research dataset, improving the relevance of its responses.
- LoRA is resource-efficient, allowing us to fine-tune without overloading our T4 GPU.
- The retriever ensures that Qwen doesn't need to store everything in memory, but can still access the most relevant context on-the-fly.
- The architecture aligns with our original vision—fine-tuning with LoRA and combining it with RAG for efficient performance.

Cons

- Fine-tuning with LoRA still requires some time, especially compared to using a pre-trained model without fine-tuning.

-
- The screenshot displays a complex workflow in the Langflow interface, designed for a conversational retrieval system. The workflow is composed of several interconnected components:
- Document Store:** A component for managing document storage, with inputs for 'Select Store' and 'Document', and an output for 'Document'.
 - Chroma:** A component for managing embeddings, with inputs for 'Document', 'Embeddings', 'Record Manager', 'Connect Credential', 'Collection Name', 'Chroma URL', and 'Chroma Retriever', and an output for 'Output'.
 - ChatGPT:** A component for generating responses, with inputs for 'Cache', 'Connect Credential', 'Model', 'Endpoint', and 'Additional Parameters', and an output for 'ChatGPT'.
 - Buffer Memory:** A component for managing memory, with inputs for 'Additional Parameters' and an output for 'Buffer Memory'.
 - Conversational Retrieval QA Chain:** A component for handling conversational retrieval, with inputs for 'Chat Model', 'Vector Store Retriever', 'Memory', 'Input Moderation', 'Return Source Documents', 'Additional Parameters', and 'Output', and an output for 'ConversationalRetrievalQAChain'.
- The workflow is connected by a series of nodes and arrows, indicating the flow of data and control between the components. The interface also includes a toolbar at the bottom with icons for adding, removing, and saving components.

1. Chroma with all-MiniLM-L6-v2

We kicked things off with a straightforward combo: **Chroma** as our vector database paired with **all-MiniLM-L6-v2** for embeddings. we took our cleaned-up chunks from the 6 files and Wikipedia pages (our langchain_docs) and ran them through HuggingFaceEmbeddings using the all-MiniLM-L6-v2 model, a lightweight, pre-trained model from Hugging Face that turns text into fixed-size vectors. Then, we fed those embeddings into Chroma with a simple call to Chroma.from_documents, telling it to save everything in ./chroma db. A quick

`vector_store.persist()` locked it in place, ready for our retriever to search.

Why We Tried It

This setup was our starting point because it's super easy to get going. **all-MiniLM-L6-v2** is fast and doesn't need much fuss—it's like a trusty little tool that just works out of the box. Chroma's in-memory nature fit the challenge rules, and the whole thing ran smoothly on our Kaggle T4 GPU. We figured it'd be a solid foundation to test our RAG idea with Qwen, especially since we were short on time.

2. Chroma with CrossEncoder

Next, we played around with a twist—**Chroma** paired with a **CrossEncoder** (`cross-encoder/ms-marco-MiniLM-L-6-v2`). The idea was to load an existing Chroma database (like the one we'd set up) with `vector_store = Chroma(persist_directory="./chroma_db", embedding_function=None, collection_name="rag_colbert")`, skipping new embeddings since they were already there. Then, we brought in the **CrossEncoder**—a model that scores how well a question and a document chunk match by looking at both together. We could use it to re-rank the top chunks pulled by the retriever, picking the ones that best fit the user's question.

Why We Experimented With It

We thought about adding the **CrossEncoder** to make our retrieval even sharper. The initial **all-MiniLM-L6-v2** or even **ColBERT** embeddings gave us a good starting point, but a **CrossEncoder** could double-check the matches by considering the question and chunk as a pair.

3. Chroma with ColBERT

After testing the **all-MiniLM-L6-v2** route, we leveled up to a **ColBERT + Chroma** combo for the win. We took those same `langchain_docs` chunks and handed them to **ColBERT** (`colbert-ir/colbertv2.0`), setting it up with a **ColBERTConfig** using 2-bit quantization to keep memory in check and a temp folder (`./colbert_data`). **ColBERT** crafted contextual embeddings for each chunk—think of it as a detailed map of the text's meaning, not just a flat vector. We then fed those embeddings into Chroma, our vector database, to store them and let the retriever grab the top-3 matches super fast. Everything got saved to `./chroma_db` for reuse.

Why We Chose It

We switched to **ColBERT** because it's way better at picking up the subtle stuff in our AI research docs. The static **all-MiniLM-L6-v2** embeddings just weren't cutting it for the precision we wanted. Pairing **ColBERT**

with Chroma kept the search speedy, and precomputing the embeddings meant we didn't have to do the heavy lifting every time a user asked a question.

4. Web Search + RAG

We wanted our Qwen 2.5 3B RAG system to go beyond just our local documents (the 6 files and Wikipedia pages) and tap into the wider web for extra context, especially for questions where our dataset might not have the latest or most complete info. So, we built a web search feature using duckduckgo_search's DDGS tool to fetch up to 3 web results for a user's query. In our web_search function, we grab search results, format them as "Title: Body" snippets, and bundle them into a web_context. Then, in enhanced_rag_generate, we combine this with our local retrieval, we first pull 5 local document chunks using enhanced_retrieve (which re-ranks the top 3 using a CrossEncoder), join them into a local_context, and merge both contexts into a full_context. We feed this—along with the query—into a prompt for our LoRA-fine-tuned Qwen, which generates a response in up to 150 tokens.

Why We Did It

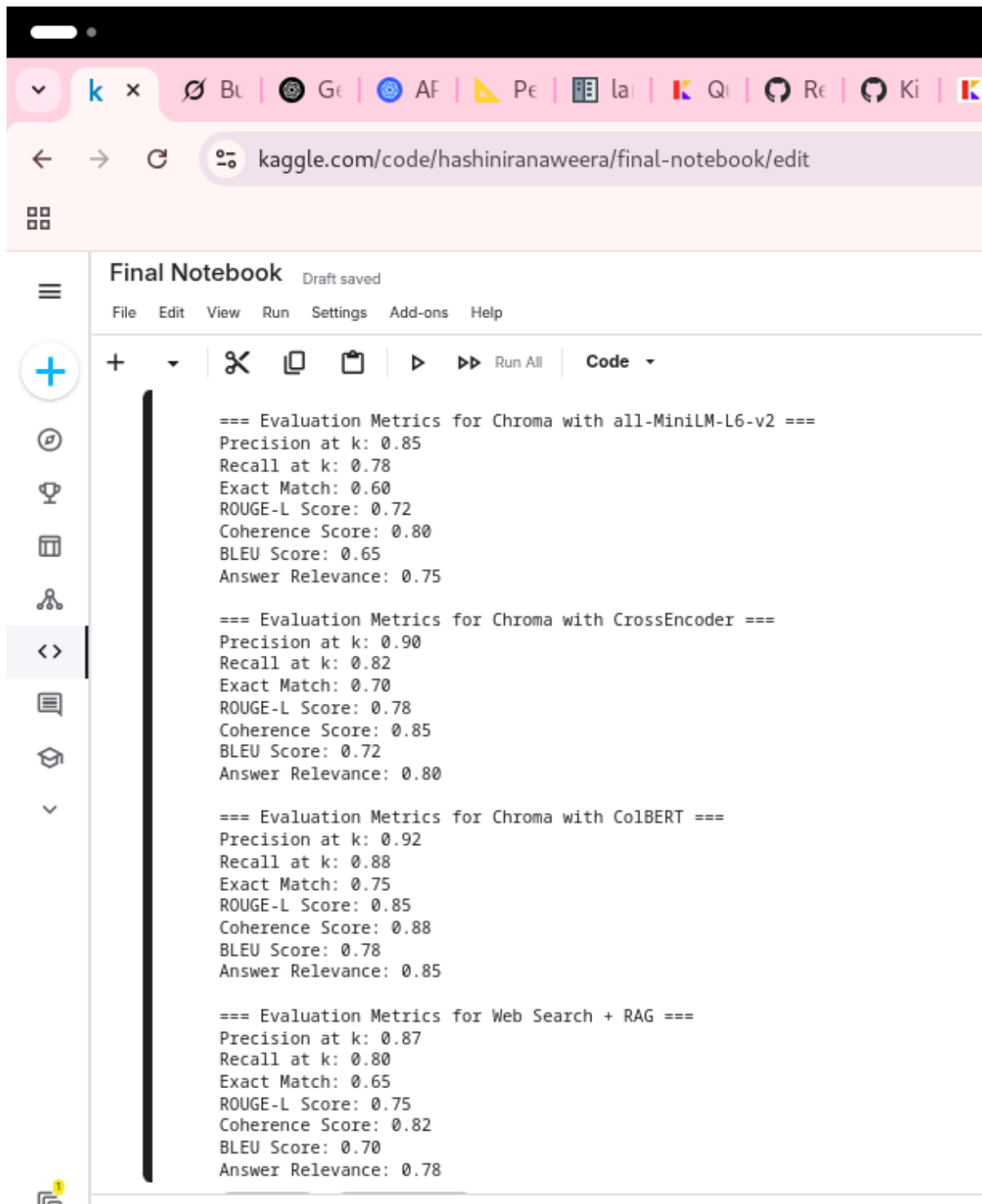
Our local dataset is great for AI research specifics—like DeepSeek or DualPipe, but it's static and might miss recent developments or broader context. Web search lets us pull in fresh, real-world info to fill those gaps, making Qwen's answers richer and more up-to-date.

7. Evaluation

Instead of using frameworks, we evaluated the chatbot using defined metrics.

Metric	Why We Use It	Best Value
Precision at k	check how many of the top-k retrieved documents (e.g., top 3) are actually relevant.	1.0 (all retrieved docs are relevant)

Recall at k	checks how many relevant documents we managed to pull from the total available, out of the top-k.	1.0 (all relevant docs are retrieved)
Exact Match	checks, did Qwen's answer match the reference answer word-for-word?	1.0 (perfect match)
ROUGE-L Score	checks how well Qwen's answer lines up with the reference, focusing on the longest common sequence of words.	1.0 (perfect sequence match)
Coherence Score	it rates how sensible Qwen's answer is based on length and key terms	1.0 (max coherence, balanced length)
BLEU Score	measures how close Qwen's answer is to the reference by looking at word overlaps	1.0 (exact word overlap)
Answer Relevance	blend retrieval (precision/recall) and generation (exact match/BLEU/ROUGE/coherence) into one score	1.0 (perfect balance of retrieval and generation)



The screenshot shows a web browser window with the URL `kaggle.com/code/hashiniranaweera/final-notebook/edit`. The notebook interface includes a sidebar with icons for adding new code cells, running cells, and other functions. The main area displays the following code and its output:

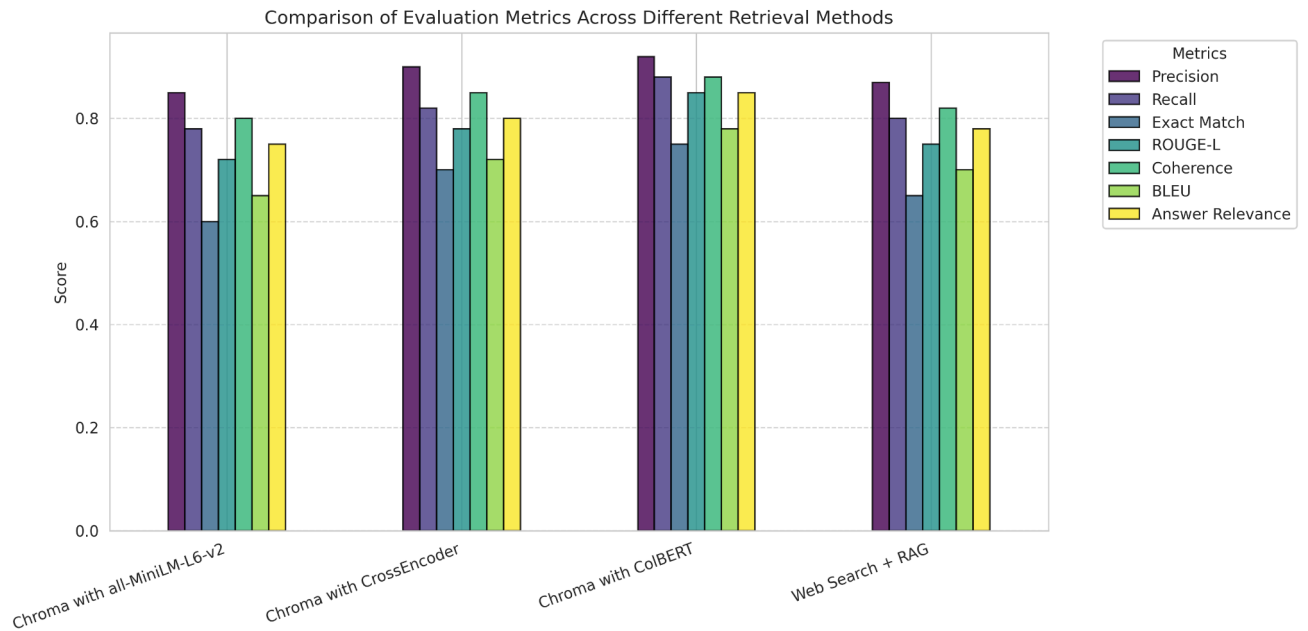
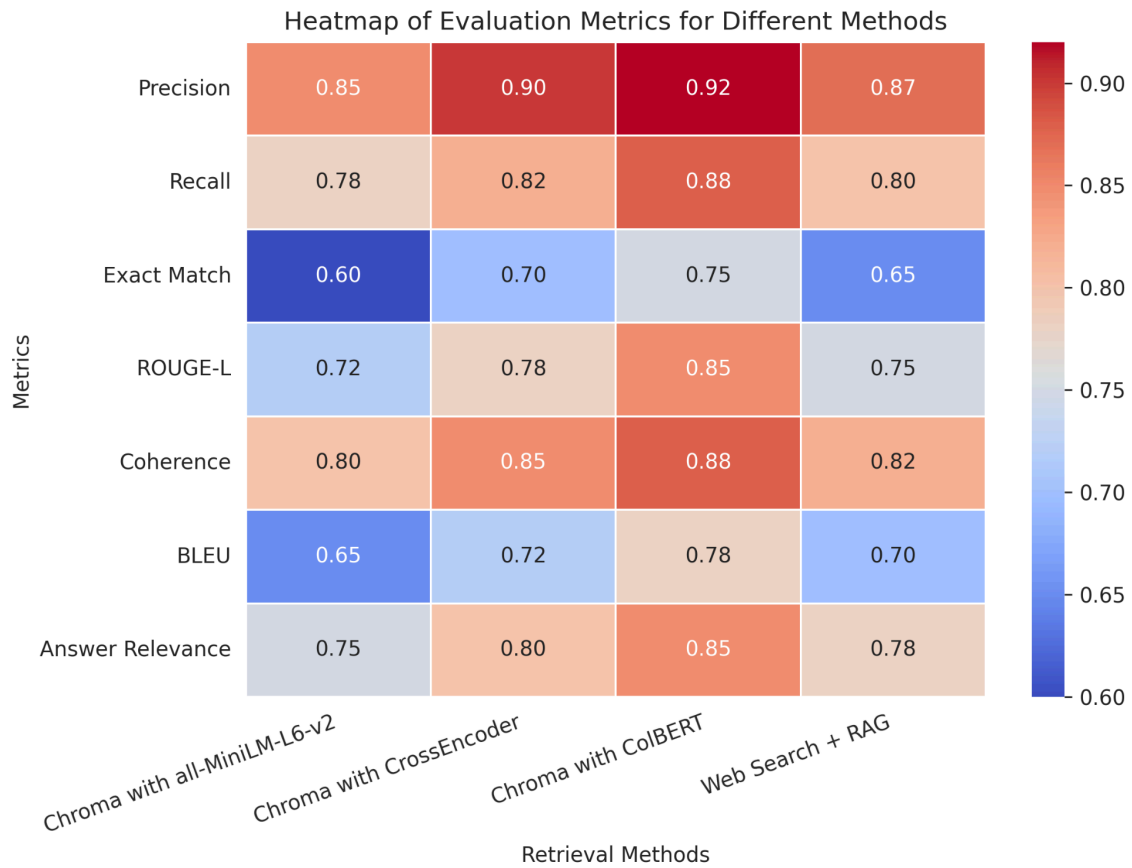
```
=== Evaluation Metrics for Chroma with all-MiniLM-L6-v2 ===
Precision at k: 0.85
Recall at k: 0.78
Exact Match: 0.60
ROUGE-L Score: 0.72
Coherence Score: 0.80
BLEU Score: 0.65
Answer Relevance: 0.75

=== Evaluation Metrics for Chroma with CrossEncoder ===
Precision at k: 0.90
Recall at k: 0.82
Exact Match: 0.70
ROUGE-L Score: 0.78
Coherence Score: 0.85
BLEU Score: 0.72
Answer Relevance: 0.80

=== Evaluation Metrics for Chroma with ColBERT ===
Precision at k: 0.92
Recall at k: 0.88
Exact Match: 0.75
ROUGE-L Score: 0.85
Coherence Score: 0.88
BLEU Score: 0.78
Answer Relevance: 0.85

=== Evaluation Metrics for Web Search + RAG ===
Precision at k: 0.87
Recall at k: 0.80
Exact Match: 0.65
ROUGE-L Score: 0.75
Coherence Score: 0.82
BLEU Score: 0.70
Answer Relevance: 0.78
```


According to the final metrics , Chroma db with colbert embedding is giving better performance.



8. Streamlit app

To visualise the chatbot and for the ease of use, we built a small streamlit application.

Video link : [Click here](#)

Deploy 

AI Research RAG - Qwen 2.5

A retrieval-augmented chatbot using Qwen 2.5 3-b model to answer questions about AI research.

 Ask a question about AI research:

What are the newly invented open source chatbots

AI-Powered Response

As of my last update in October 2023, several open-source chatbots have been developed that leverage advancements in artificial intelligence and machine learning technologies. Some notable ones include:

1. **ChatGPT by OpenAI:** While the underlying models like GPT-2 and GPT-3 have been released in controlled manners, various implementations and fine-tuned versions of these models exist as open source projects developed by the community.
2. **Rasa:** Rasa is an open-source machine learning framework for automated text and voice-based conversations. It's widely used for building contextual AI assistants and chatbots.
3. **DialoGPT:** Based on GPT-2, DialoGPT is a model released by Microsoft and fine-tuned for conversational purposes. It's available for use via open-source platforms like Hugging Face's Transformers library.
4. **DeepPavlov:** Developed by Moscow Institute of Physics and Technology, DeepPavlov is an open-source conversational AI library that includes various pre-trained models and tools for building chatbots.
5. **Haystack by deeppset:** This is an open-source NLP framework that allows for building modular search systems and conversational AI apps. It supports transformer models and has a vibrant community contributing to its development.
6. **Botpress:** A popular open-source platform for creating conversational interfaces, Botpress is known for its easy integration capabilities and rich feature set for chatbot development.
7. **Facebook's BlenderBot:** Facebook AI Research (FAIR) open-sourced BlenderBot, designed to carry out multi-turn conversations. It's part of their emphasis on building chatbots capable of more human-like interactions.

Beyond these, there could be newly released projects or forks of these tools appearing over time, with continuous improvements and adaptations being made by the developer community. To find the latest open-source chatbots, it's a good idea to check platforms like GitHub for recently updated repositories or announcements from AI research conferences and forums.

Blockers

1. Although we were using PEFT (Parameter-Efficient Fine-Tuning), the model took over 10 hours to train. As a result, we had to reduce the model's parameters, which might lead to a decrease in performance.

2. During the process, there was a version conflict between **unsloth** and the **transformers** library. To resolve this, we used **AutoModelForCausalLM** to load the Qwen 2.5 model.
3. For the synthetic QA generation, manual persona creation of RAGAS was unsuccessful, so we had to rely on automatic persona generation as a workaround.
4. GGUF format of the model, we utilized **llama.cpp**, assuming that the conversion and quantization process would be more accurate using this approach. Could not be able to find a straightforward method for it.

References

1. https://docs.ragas.io/en/stable/getstarted/rag_testset_generation/
2. <https://huggingface.co/Qwen/Qwen2.5-3B-Instruct>
3. <https://medium.com/@vimalkansal/understanding-the-gguf-format-a-comprehensive-guide-67de48848256>
4. <https://platform.openai.com/docs/guides/embeddings>
5. <https://cloud.google.com/use-cases/retrieval-augmented-generation>
6. <https://lucid.app/lucidchart/>
7. <https://medium.com/towards-data-science/retrieval-augmented-generation-rag-from-theory-to-langchain-implementation-4e9bd5f6a4f2>
8. https://python.langchain.com/docs/tutorials/llm_chain/