HS PF

Proof of Concept: ChatBot Creation with Fireworks.ai

Project Documentation Team 4

Julius Trögele
Donata Zwick
Rico Reinbacher

Inhalt

Project Documentation: AI-Based Chatbot for University Study program

Aim and Scope of our Proof of Concept:

Showcase whether it is possible for our team to create a Chatbot based on a pre-trained, open-source large language Model, that will be enhanced via Retrieval Augmented Generation to comply with needs of students of the Wirtschaftsinformatik-Study program. The implementation takes place within the scope of this four ECTS IS-Project.

Introduction

In the beginning, our team consisting of Julius, Donata und Rico, was completely new to the procedure of creating AI tools. We loved the idea to create our own version of a LLM and to try to create a Chatbot that would not only help the three of us but also our whole study program during their studies. We imagined how helpful it would be to have a tool like ChatGPT at our side which would know about all the specifics about each subject in our study program and would know all about our exercises and exams.

In March, we identified several goals and requirements of potential student users

- The tool should offer convenience in studying
- The chatbot should be an expert for the course of Information systems which can answer questions related to the topic
- Chatbot is aware of the scope of our lecture content based on input data i.e. the embedded lecture slides. E.g. what topics the students have to know and how deep has the knowledge to be
- Chatbot should be able to create tasks with solutions based on the lecture content
- Optional: PDF scraper & text converter for lecture slides (pdf) for ease of use

As the reader might recall, three months ago, the newest version of ChatGPT was the Version GPT-3.5. It was already very advanced but missed a special something, for example the information it was able to access was partially outdated. Complex queries, for example relational Algebra and SQL were not processed accurately and exercises regarding topics relevant for us were not created well. We saw the need for a tool that consists of a huge knowledge base regarding our study program and when asked a question the bot would first access the provided input about our study program and prioritize output from that knowledge base. Only when the bot finds nothing in this preferential store of knowledge it should consult the Internet
However, during this semester, another development took place: the ChatGPT Version 4.0 was released to the market and became available also for students.

ChatGPT 4.0 exhibits a better understanding of context, maintaining coherence over longer conversations and has an updated knowledge base, which includes more recent information and advancements as well about the modern topics of our study program.

With 4.0, Students started to create Chats according to their own individual needs. With this Version it is possible for everyone to feed ChatGPT with an own set of information and let it answer follow up question based on the information they fed it in the first place. For example, for students it is common now to load a bunch of .pdf documents into a chat with ChatGPT and use a prompt to tell it that in the future it should prioritize to answer their questions first based on the information provided in the documents and to create exercises with matching solutions based on their individual resources.

At first, that tendency had a demotivating effect on our spirit because that new ability of ChatGPT was making the program that we planned on creating obsolete. However, we decided to continue with our poof of concept and were encouraged to try it ourselves to see whether we are capable of creating a program with the same purpose ourselves.

In the beginning of our endeavour the only resource we possessed was some rudimentary knowledge of programming in Java. However, we soon found out, that it is not usual to program AI related programs with Java but with python.

At the start of the project we started with gaining an overview on the topic of neuronal networks in general by borrowing books in the library related to the topic of natural language processing (NLP), machine learning (ML) and Tokenization. (xx buch quellen)

We accumulated much information in general to explore the field in general. That included collecting information about the various platforms that appeared on the internet in the last couple of years.

1. Decision for Platform: Fireworks.ai vs. Other Platforms

When deciding on a hosting platform for our AI-based chatbot, we evaluated several options, including Fireworks.ai, FreePlay, and OpenAI.

The decision to use Fireworks.ai as the hosting platform for our AI-based chatbot was based on several key factors:

1. Integration Capabilities with various AI Models
2. Team Accessibility
3. Cost-Effectiveness.
4. Familiarity and Expertise

Fireworks.ai provided a highly optimized environment for deploying large language models, such as Llama 3 70b, with seamless integration and scalability. Unlike FreePlay, which has limitations in terms of model size and scalability, Fireworks.ai supports larger models with better performance metrics. Additionally, Fireworks.ai offers robust tools for monitoring and managing deployed models, which are essential for maintaining the performance and reliability of the chatbot.

Furthermore, Fireworks.ai provides robust collaboration features, allowing multiple team members to access and manage the project simultaneously. This was crucial for our team-oriented workflow.

Compared to OpenAI, Fireworks.ai provides more customizable options for integrating external data sources, which was crucial for our use case involving Retrieval-Augmented Generation (RAG). As students, we are mindful of budget constraints and fireworks.ai also offers competitive pricing models, making it a more cost-effective solution for long-term deployment. These factors collectively influenced our decision to choose Fireworks.ai over FreePlay and OpenAI.

Another significant factor influencing our choice is the familiarity of Fireworks.ai among our professors and supervisors. Leveraging a tool that resonates with the expertise of our mentors fosters smoother collaboration and communication. Their familiarity with Fireworks.ai ensures that we receive relevant guidance and support throughout the development process, enhancing the overall efficiency of our project.

2. Decision of Model: Llama 3 vs. Other Models

Selecting the appropriate AI model was a critical decision. We compared Llama 3, Mistral, and GPT, focusing on

a. performance
b. cost
c. suitability for our specific use case.

Llama 3 70B was selected as the base model for our chatbot due to its superior performance in natural language understanding and generation. Llama 3 offers state-of-the-art language capabilities, outperforming other models like GPT-4 in various benchmarks relevant to our use case. Its large parameter count allows for more nuanced and contextually aware responses, which is critical for an academic setting.

Llama 3 70B's architecture and pre-training allowed it to adapt well to the specific domain of Information Systems. It would fit in better for our subject-specific requirements.

Also, regarding cost efficiency, Llama 3 70B offered a balance of high performance and cost-effectiveness. In contrast, GPT was significantly more expensive, which was a crucial consideration given our budget constraints.

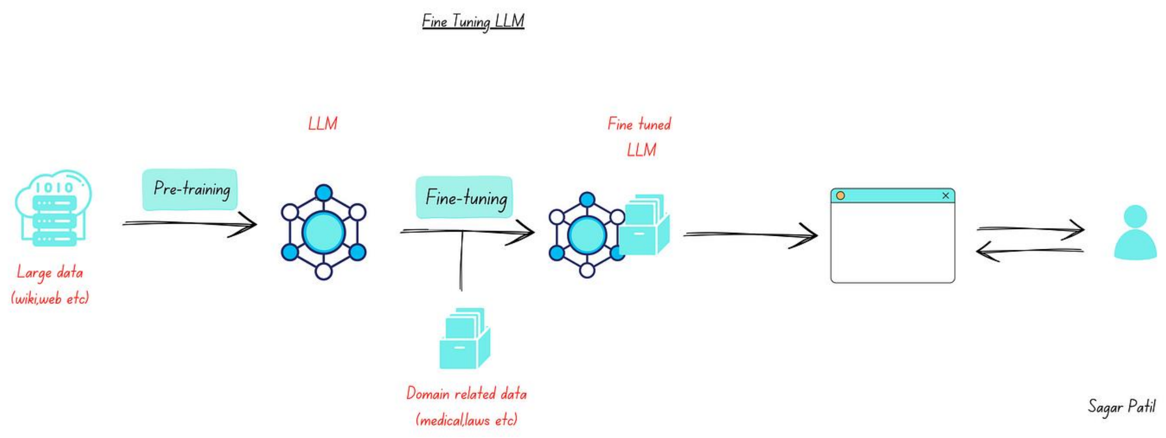
Mistral would have been adaptable but required more effort in fine-tuning compared to Llama 3. The additional workload did not align with our project's timeline and resource availability.

3. Why We Chose Retrieval-Augmented Generation (RAG) over Fine-Tuning

After having gathered much information about the topic, we faced a crucial decision: whether to fine-tune our model on the specific dataset or to employ Retrieval-Augmented Generation (RAG). After careful consideration of the requirements, constraints, and benefits of each approach, we chose RAG for several compelling reasons:

3.1 Architecture[1]

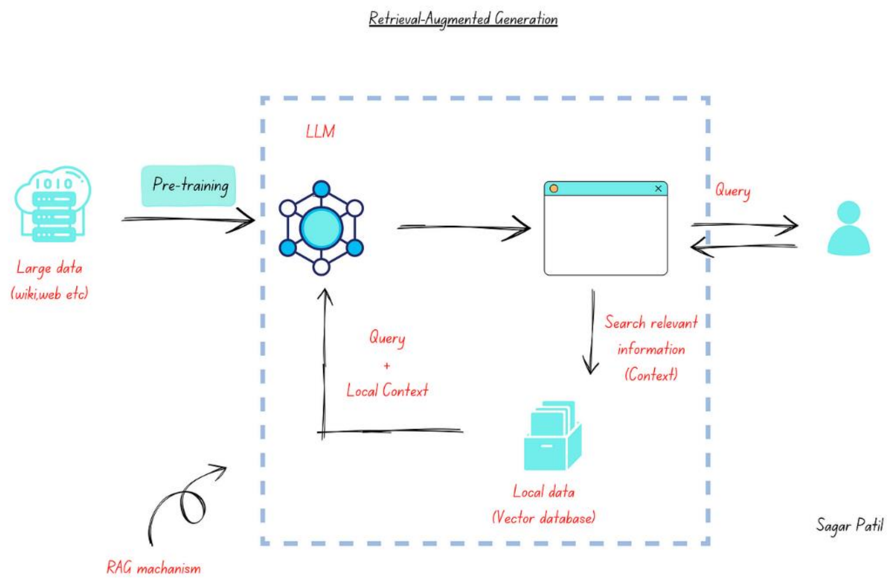
Fine-tuning LLMs:



Fine-tuning LLMs usually start with a pre-trained model (like GPT-3) and fine-tune it by training on task-specific data. The architecture remains largely unchanged, with adjustments made to the model's parameters to optimize performance for the specific task

[1] Picture and Information source: https://ai.plainenglish.io/fine-tuning-vs-rag-in-generative-ai-64d592eca407

RAG:


Retrieval-Augmented Generation
LLM
Pre-training
Large data (wiki,web etc)
Query
Search relevant information (Context)
Query + Local Context
Local data (Vector database)
RAG machanism
Sagar Patil

RAG models have a hybrid architecture that combines a transformer-based LLM (like GPT) with an external memory module that allows for efficient retrieval from a knowledge source, such as a database or a set of documents.

3.2 Efficiency

Training Time and Computational Resources: Fine-tuning a large-scale language model like Llama 3 70B requires significant computational resources and time. It involves training the model on a large dataset to adjust its weights, which can be computationally expensive and time-consuming. In contrast, RAG leverages a pre-trained model and retrieves relevant information from an external knowledge base, reducing the need for extensive retraining.

Resource Allocation: By using RAG, we could allocate our resources more efficiently. Instead of investing in high-end GPUs and extensive training sessions, we focused on optimizing our retrieval system and ensuring that our external knowledge base was comprehensive and up-to-date. (Although it would have been great fun for sure to try to convince the hochschule board and the Rektor to invest in not only big GPUs but also build a new building for them and place new power lines)

3.3 Flexibility

Dynamic Information Retrieval: RAG allows the model to dynamically fetch information from an external knowledge source at inference time. This means the chatbot can access the most current and relevant data without needing to be retrained whenever the underlying information changes. This flexibility is especially important in the academic environment of Information Systems and our study program in general, where information can rapidly evolve.

Adaptability to Diverse Queries: With RAG, the model can handle a wider range of queries by retrieving specific information from the knowledge base, making it more adaptable to various user inquiries. Fine-tuning, on the other hand, would limit the model to the scope of the training data, potentially missing out on nuances and newer information.

In summary, our choice of Retrieval-Augmented Generation (RAG) over fine-tuning was driven by considerations of efficiency, flexibility, cost, and performance. RAG's ability to be based on a pre-trained model and dynamically retrieve and integrate relevant information from an external knowledge base matched exactly the idea we hat from the beginning and therefore made it the ideal solution for our AI-based chatbot project.

4. Installation Process: Python, Pip, Python Libraries

To set up our development environment, we followed these steps:

1. **Python Installation**: We installed Python 3.8 from the official Python website.
2. **Pip Installation**: Pip, the package installer for Python, was installed to manage project dependencies.
3. **Library Installation**: We installed necessary libraries, transformers, and others, using pip

5. Learning to Code in Python: Tools and Resources

Our team self-taught Python coding through a combination of online resources and practical application, i.e. learning by doing and trial and error.

Extensive use of official documentation and community-contributed tutorials on GitHub, Stack Overflow and Hugging face. Also, we watched many youtube tutorials.

6. Reading Books for Knowledge

To deepen our understanding of AI and machine learning, we read several key books:

- "Neuronale Netze programmieren mit Python" by Steinwendner, J. und Schwaiger, R. (Rheinwerk Verlag)
- "Deep Learning für KI und Data Science" by Deru, M. und Ndiaye, A. (Rheinwerk Verlag)
- „Eigene KI-Anwendung programmieren“ by Karatas M.

7. First Interaction with AI Through Code: Main Prompt Overridden

[Our finally developed code can be found in the file AI_conversation.py]

To get in contact with our chosen AI model, our first interaction should take place from our own device through an API key. To set that up, we studied the fireworks.ai Documentation [2].

Our initial interaction with the AI model involved overriding the system prompt to customize the chatbot's responses. This process, similar to starting with fine-tuning, allowed us to tailor the model to better suit our specific requirements.

We became acquainted with several specifications like Streaming, temperature and the top_k.

- Streaming:

By default, results are returned to the client once the generation is finished. Another option is to stream the results back, which is useful for chat use cases where the client can incrementally see results as each token is generated. However, we decided not to include streaming, as we decided that the user should receive the output only as a finished paragraph to not lead to confusion.

- Temperature:

Temperature allows you to configure how much randomness you want in the generated text. A higher temperature leads to more “creative” results. On the other hand, setting a temperature of 0 will allow you generate deterministic results which is useful for testing and debugging. We chose a temperature of 1.3 in the beginning, because that leads to a desirable mix of determination and

- Top_k:

Top-k is sampling method where the k most probable tokens are filtered and the probability mass is redistributed among tokens. For trying out and saving tokens, we chose a top_k value of n=1 in the beginning.

While we were busy getting familiar with the specifications of Meta’s Model Llama 2, they released a new Model Llama 3 in April[3]. That put us up with the cumbersome task of changing the already employed model Llama 2 into the newer version. However, it behaved similarly so it was not a major difficulty.

Another challenge we encountered was that in the beginning, we were only able to interact with one prompt and to receive one answer. However, we found out how to maintain a longer conversation by using an array list called “messages” and a while loop to store the previous conversation.

[2] Fireworks Documentation: https://docs.fireworks.ai/getting-started/introduction
[3] Release of llama 3 : https://ai.meta.com/blog/meta-llama-3/

8. Data Preparation

[the respective file is the course_data.txt]

Data preparation involved collecting, cleaning, and organizing relevant data. This step ensured that the data fed into the model was accurate, relevant, and well-structured.

For that step, we received lecture slides from Prof. Dr. Schuler in pdf format. In order to process that, we first extracted the text only from the slides with the help of ChatGPT into one plain text file.

9. Integrating APIs and Libraries: OpenAI, Pinecone, and Fireworks.ai

Our project's architecture relied on the integration of three main APIs/libraries:

- **Fireworks.ai**: Hosted our AI model and facilitated vector integration.
- **OpenAI Library**: Used for generating vector embeddings.
- **Pinecone Library**: Managed the storage and retrieval of vectors.

The integration process involved setting up API connections and ensuring seamless data flow between these components.

10. Research on Vector Embedding and Vector Database via Pinecone

In order to set up an external knowledge base, we needed to find out how to set up a vector database. Therefore, we needed to prepare Vector embeddings. Our research focused on understanding how to convert textual data into numerical vectors that the AI model could efficiently process. Soon, with the help of a fellow colleague, we found that Pinecone fitted our need best, because it is free and offers a team working setting. We studied the pinecone documentation[4] and decided to use Pinecone for vector database management due to its robust indexing and search capabilities.

11. Creating Vectors

[embeddings_openAI.py file creates vectors based on the OpenAI library using the embeddings model nomic-ai/nomic-embed-text-v1.5]

Creating vectors involved using the OpenAI library to generate embeddings from our textual data. We selected the embeddings model based on the recommendation of Fireworks.ai documentation. As a dimension we chose the range 512 which corresponds to 512 floats per text embedding.

[Create_Vector.py]

[4]Pinecone documentation: https://docs.pinecone.io/home

- Creates vectors for our file path
- Imports embeddings_openAI and pinecone_interaction (explanation see next step)
- Using the create_vector_embeddings function based on the OpenAI library extension
- We split the text into chunks of 500 characters with the chunk_text function

[Pinecone_interaction.py file]

- Imports two libraries: pinecone library by pinecone and the python library uuid library
- A problem was to find out how to filter the vectors.
- That we solved inside the Upsert_vector function which inserts or updates (if already existing) vectors. The function also processes the meta data which in our case is the plain text we prepared earlier.
- With a similarity search a comparison between two vectors takes place returning the vector Id and the score and in our case also the meta data, to find out how similar the already saved database vectors and the queried vectors are. The higher the number of the score the better.
- Fetch_vector
- These vectors are then stored in Pinecone's vector database for efficient retrieval and integration

[Retrieve_Vectros_data.py]

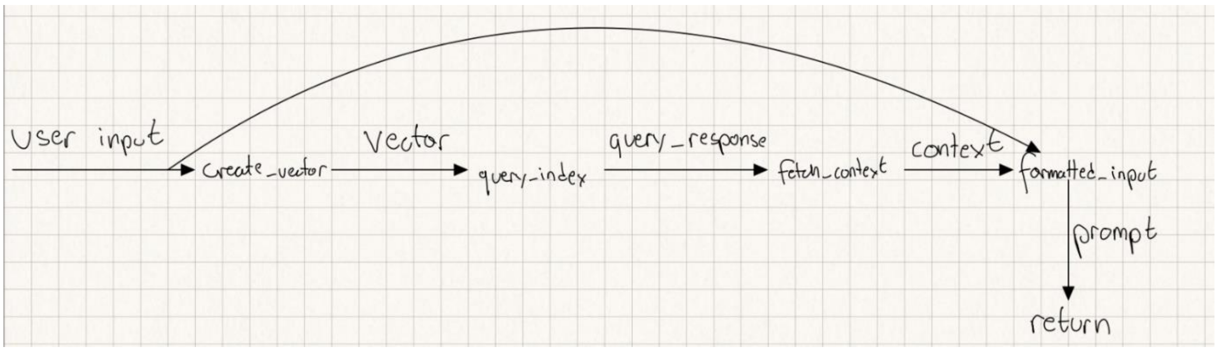
- fetches vectors from pinecone database based on vector ID in order to further work with the vectors

12. Problem: Turning Vectors back to text

At this point we have no idea how we can tell our model to preferably access our Vector database before generating own artificial response.

We handled it with a function called handle_user_input which is located in the AI_conversation.py file.

It takes the user input as well as a top_k value and creates a vector. Using this vector it creates a variable called ”query_response” by querying the index for the previously created query vector. It returns multiple answers in equal number to the top_k value. Then it fetches the relevant context to the query response, formats the fetched context in a context variable. This prompt is then handed over to the Llama Model.



13. Conclusion

In conclusion, we are happy to have proved our concept. However, we spent way more time than the amount of time the 4ECTS suggests. It was very interesting to dive into these currently discussed topics and to have gained practical insights.