Project Proposal

An AI platform for fine-tuning, deploying and inferencing on LLMs with RAG

Project Lead: Rahul Dave

Authors:

- Ratna Sambhav (sambhav.ratna@gmail.com)
- Prayash Panda (pkpanda234@gmail.com)
- Arjun Benoy (arjunbinoy.vishnu@gmail.com)
- Joyson Chacko George (joysoncgeorge2001@gmail.com)

Background and Motivation:

With the large-scale industry adoption of large language models, there is very high demand to train a large language model on custom/private data and then use that model for applications like chatbot, automated data extractor/formatter, etc. What proprietary LLM APIs fail to achieve is data privacy, and lightweight, secure and cheaper deployment.

Hence, training a custom LLM is impossible without specialized experts and a proper infrastructure, costing non-tech companies a large effort, time and cost to adopt this technology. This gap is what we are trying to bridge by introducing a “no-code platform” which any non-tech professional can easily use to train and deploy their own LLM and use it in their day to day workloads. Each member of our team has experience in one way or another in this particular area, and thus are focussed on bringing this to reality.

Scope and Objectives:

This lack of specialized no-code AI platform for non-tech personnel for training their LLMs, is kind of an issue, since everybody wants their LLM but no one has the knowledge on the technical side of things. Our AI platform with intuitive and easy user interface, will provide all the support that a user needs to fine-tune their model.

With a click of few buttons, they can orchestrate the whole process of fine-tuning and deploying their own LLM, on their own cloud account on their own private data, with only the cost of training on the GPU, and after that based on the method of deployment, cost on each API call or per hour of cloud server used. After deployment, they can either use the API directly to their LLM or use our platform to access their deployed LLM. Following are our project objectives:

1. To create three automated pipelines for three stages: Fine-tuning, Model deployment and inference.
2. A reactjs frontend with user login feature, calling a FastApi backend to execute these pipelines with a submit button on frontend.
3. Use Infrastructure As Cloud service to do the resource intensive tasks of model fine-tuning and deployment on GCP/aws cloud.
4. To showcase working of the whole system by fine-tuning and deploying a 2b and 7b LLM models using this architecture and also implement a RAG pipeline for a set of documents.

Data Sources:

Source of data:

1. *Math-Instruct* and *Arithmo-data* dataset from HuggingFace. These are open source datasets.
2. EDGAR Database: An online repository available for public access. Access to the EDGAR database is available for free on the SEC's official website (www.sec.gov/edgar).

Description of dataset:

1. *Math-Instruct* and *Arithmo-data* are basically instruction based Question-Answer datasets which contain queries and responses in chain-of-thought format.
2. EDGAR Database(Electronic Data Gathering, Analysis, and Retrieval) is a comprehensive online repository maintained by the U.S. Securities and Exchange Commission (SEC) that provides public access to corporate financial filings, such as annual reports and registration statements. It can be accessed by using APIs.

Key attributes:

1. *Math-Instruct* :
   Instruction: MCQ type math related questions with instruction.
   Output: Answer to the question with a chain-of-thought style solution.
2. *Arithmo-data*:
   Question: Algebra based questions.
   Answer: COmplete solution to the question in a chain-of-thought pattern.
3. EDGAR Database:
   Text based large set of documents.

Relevance to the project:

1. *Math-Instruct* and *Arithmo-data* will be used for fine-tuning the model using our architecture to ultimately deploy an LLM with pure mathematical skills.
2. EDGAR Database: This will mainly validate the correct working of our RAG pipeline. Retrieval of the correct document from the database and correct response, all based on a single query will showcase an versatility of our platform and make it useful for real world scenarios.

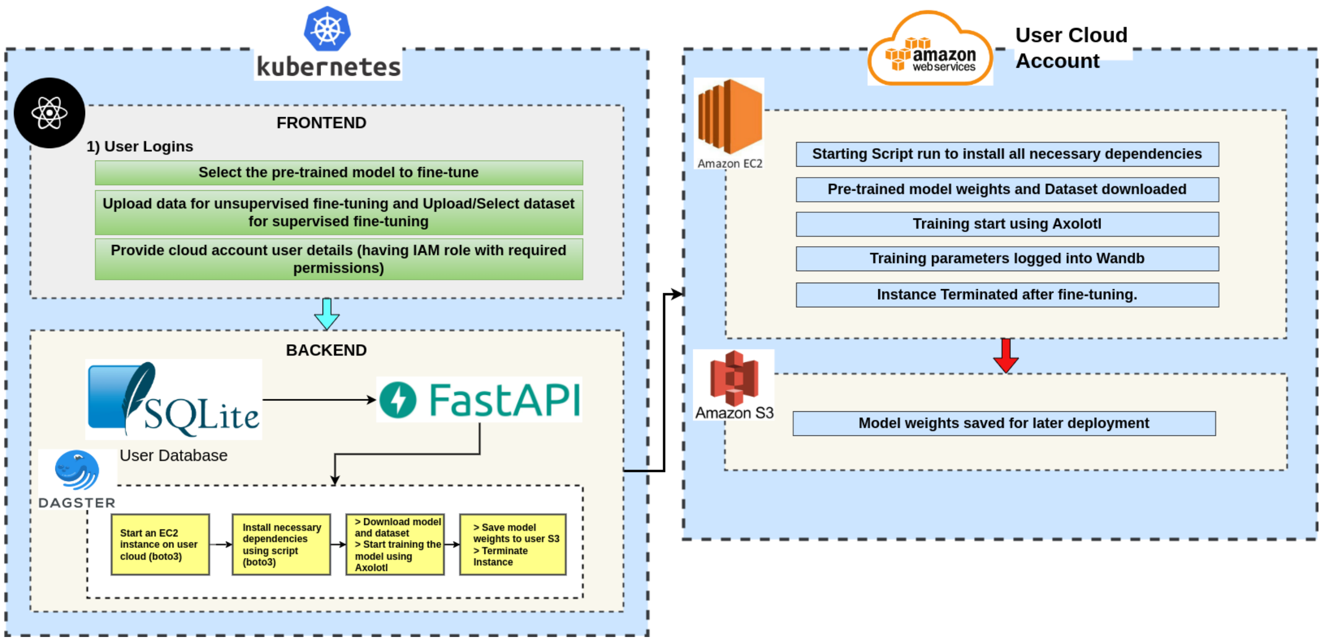
Data Handling after building our website and all the necessary pipelines:
Throughout the three stages of model training users can either choose data from our provided list or provide their own.

1. For Unsupervised: User uploads documents (.pdf, .txt, .doc) or compressed files, text will be automatically extracted from there, processed and used for training.
2. For Supervised: Users can either choose from the provided list (list from huggingface/kaggle) or upload their own dataset.
3. RLHF/RLAIF: Either choose from the defined sets or upload.

Scope and Preliminary Design

- Our AI platform, in order to fulfill this purpose of providing no code platform for training LLMs, has to be intuitive and easy. Thus it starts with a very clean and responsive frontend. Our backend will be highly scalable using container orchestration with an integrated database for keeping user's credentials and details. Whole system will be deployed on cloud (aws preferably).

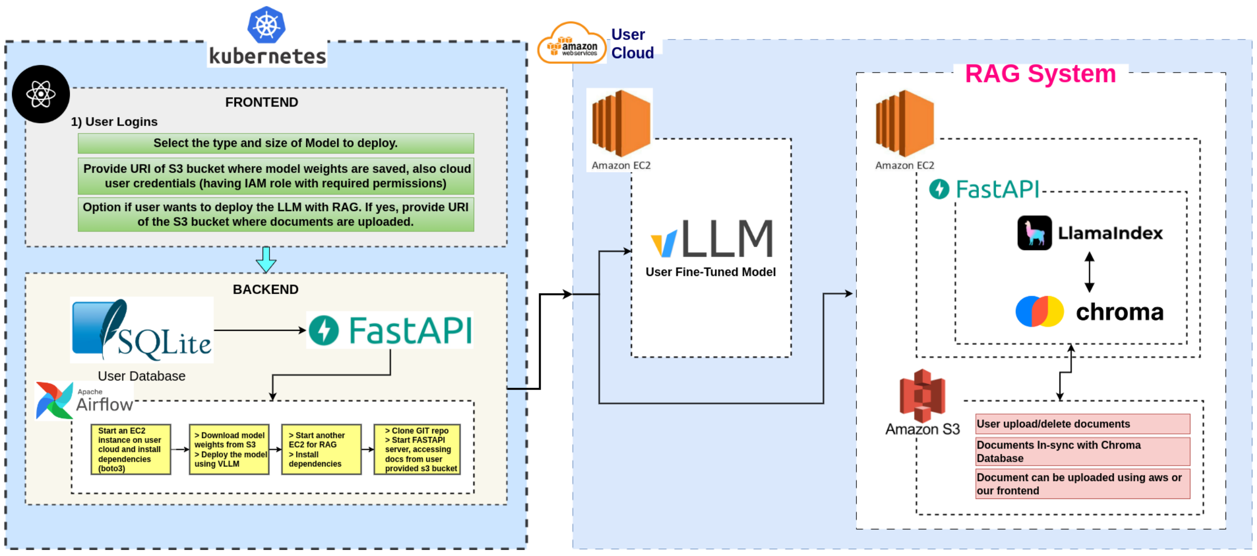
LLM Model Fine-Tuning:


kubernetes
FRONTEND
1) User Logins
Select the pre-trained model to fine-tune
Upload data for unsupervised fine-tuning and Upload/Select dataset for supervised fine-tuning
Provide cloud account user details (having IAM role with required permissions)
BACKEND
SQLite
FastAPI
User Database
DAGSTER
Start an EC2 instance on user cloud (boto3)
Install necessary dependencies using script (boto3)
> Download model and dataset
> Start training the model using Axolotl
> Save model weights to user S3
> Terminate Instance
amazon web services
User Cloud Account
Amazon EC2
Starting Script run to install all necessary dependencies
Pre-trained model weights and Dataset downloaded
Training start using Axolotl
Training parameters logged into Wandb
Instance Terminated after fine-tuning.
Amazon S3
Model weights saved for later deployment

- For LLM fine-tuning, the user chooses the model type that he wants to fine-tune, data on which to fine-tune, quantization method to apply, etc. along with credential details of the user with necessary access to their cloud account (aws/GCP).

- With the provided details, our backend will start a job using Apache Dagster and orchestrate the whole process of starting an instance of necessary capacity, downloading pre-trained model and data from HuggingFace, perform fine-tuning (using Axolotl to support most types of LLM architectures and quantization and other SOTA techniques offered by Axolotl), publish a dashboard of model training using wandb and assessing the model on the defined benchmarks based on task. After fine-tuning, the model will be deployed using VLLM on the EC2 instance. API to run inference on the model will be provided to the user as well as functionality to run inference on the website itself will be available to the user with a basic chat-ui interface.
- Our AI platform will support three stages of model training Unsupervised, Supervised, RLHF/RLAIF. The last stage will not be necessary for most tasks, but only for tasks where LLMs are supposed to generate responses in a particular manner, for example minimizing toxicity.
- Apart from that, users can also use their model on the AI platform with our predefined templates like summarization, key-value extraction, etc.

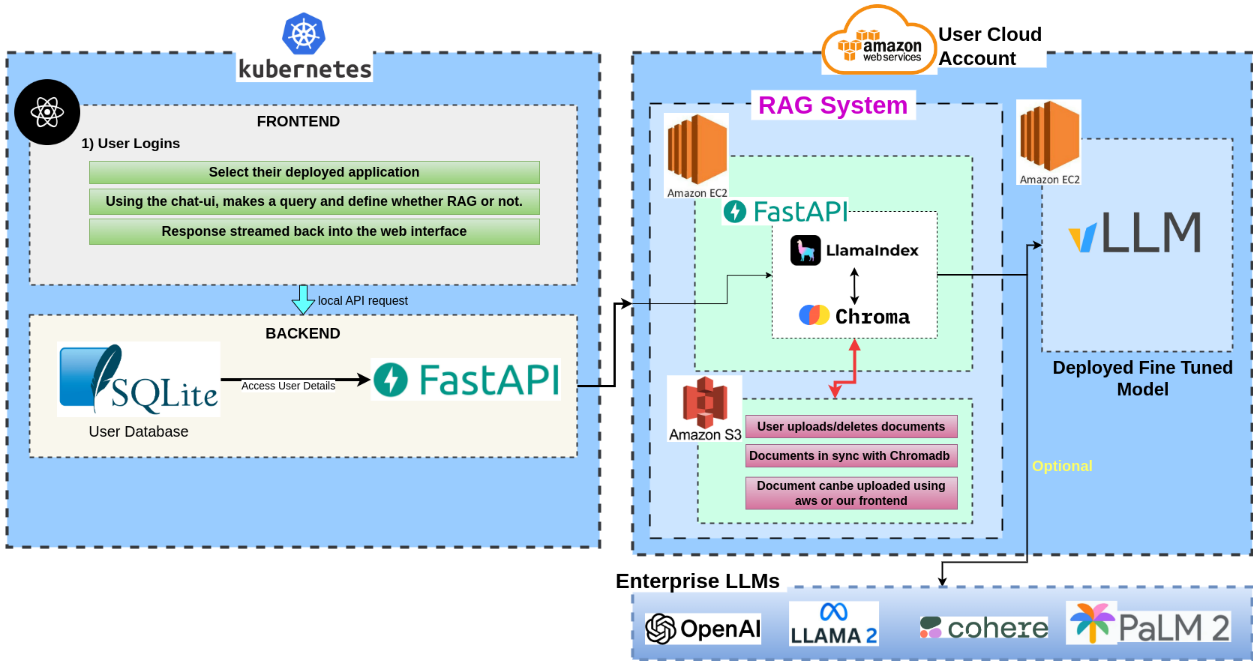
LLM Model Deployment:


kubernetes
FRONTEND
1) User Logins
Select the type and size of Model to deploy.
Provide URI of S3 bucket where model weights are saved, also cloud user credentials (having IAM role with required permissions)
Option if user wants to deploy the LLM with RAG. If yes, provide URI of the S3 bucket where documents are uploaded.
BACKEND
SQLite
User Database
FastAPI
Apache Airflow
Start an EC2 instance on user cloud and install dependencies (boto3)
> Download model weights from S3 > Deploy the model using VLLM
> Start another EC2 for RAG > Install dependencies
> Clone GIT repo > Start FASTAPI server, accessing docs from user provided s3 bucket
amazon webservices
User Cloud
Amazon EC2
vLLM
User Fine-Tuned Model
RAG System
Amazon EC2
FastAPI
LlamaIndex
chroma
Amazon S3
User upload/delete documents
Documents In-sync with Chroma Database
Document can be uploaded using aws or our frontend

- After fine-tuning is finished and model weights saved to S3 bucket, users will have the option to deploy the LLM model using VLLM with RAG or no-RAG system.
- If users specify to deploy with RAG, there will be two instances initiated using Dagster, one where LLM is deployed using VLLM and other where a FASTAPI based system will be deployed automatically taking code from specified GitHub repo to serve RAG based queries.

- With the help of Llamaindex and chroma database, queries are answered using relevant documents.
- Users can upload/delete documents from the S3 bucket which will be in sync with the Chroma database, any upload/delete will trigger the task of updating the chroma database, to keep the vector database continuously in sync.

LLM Model Inferencing:



kubernetes
FRONTEND
1) User Logins
Select their deployed application
Using the chat-ui, makes a query and define whether RAG or not.
Response streamed back into the web interface
local API request
BACKEND
SQLite
User Database
Access User Details
FastAPI
User Cloud Account
RAG System
Amazon EC2
FastAPI
LlamaIndex
Chroma
Amazon S3
User uploads/deletes documents
Documents in sync with Chromadb
Document canbe uploaded using aws or our frontend
Amazon EC2
vLLM
Deployed Fine Tuned Model
Optional
Enterprise LLMs
OpenAI
LLAMA 2
cohere
PaLM 2

- Users login into their account and access a basic chat-ui where they can ask questions from their deployed LLMs.
- We access the particular API of the user's deployed LLM from the user details saved on the user database (local db on the backend). On the user's prompt, the api request will be sent securely to the EC2 instance where the model is deployed using VLLM.
- Each API request, having options for RAG, can either directly generate responses based on the query or go through the RAG system. If the inference request has answer_from_doc parameter set to True, the query will go through llamaindex, which being connected to the chromadb vector database, retrieve relevant documents and will generate a response based on those documents.
- For converting documents to embeddings, bge-en-large or UAE-large models (top on MTEB leaderboard) can be used. Larger documents can be chunked into sizes of 512 tokens and average of the embeddings of all the chunks can be taken for getting a single embedding for the whole document. These embeddings can be stored into the chromadb vector database.

- For adding or deleting documents from the vector database, users can either directly add/delete documents from the s3 bucket (which is in sync with the vector db) or will have the option to do so on the frontend. Any addition/deletion of a document from the s3 bucket will trigger an Dagster job, which will call the API to add/delete that document from the vectord db, keeping the s3 bucket in sync with the vector db.
- Response can be streamed back to the frontend or can be sent all at once, based on the user's preference.

Milestones:

1. Python code for fine-tuning LLMs using Axolotl.
2. Python code to build a RAG system.
3. Python code to deploy an LLM using **VLLM**.
4. Python code for inferencing on deployed LLMs.
5. Automated pipeline for all the tasks, executed on gcp/aws instances.
6. Scalable and fault tolerant backend system executing these pipelines.
7. An intuitive and easy frontend.

Summary:

With the option to fine-tune LLMs, deploy it on secured systems and use it with an option for RAG, multiple use cases can be achieved.
Apart from that, if the user only wants to use the RAG system on their vast collection of documents using public LLM APIs, they can do so by connecting their document stores (s3 or other storage) to our website and providing their temporary credentials of their LLM API. With almost no extra cost, they can talk with their documents, having their data secured in their own account.

References:

1. https://github.com/run-llama/sec-insights
2. https://arxiv.org/abs/2305.18438
3. https://github.com/OpenAccess-AI-Collective/axolotl
4. https://github.com/vllm-project/vllm