Here, I will walk you through how FerroGPT operates and explain its frontend-backend connection.

FerroGPT takes the GPT-3 model, davinci, and trains it on research papers relating to ferroelectrics. It is connected to a frontend on the Khan Lab website. Users can input a question, and it returns the answer. If it does not know the answer, it returns “I don’t know” with some related context. Try it out here: https://electrons.ece.gatech.edu/ferrogpt/

Let’s walk through how to run the FerroGPT backend simply on your machine.

These are all the files in the Github repository:

- trainingDataPdfs folder
- trainingDataDocs folder

- createCSV.py
- answer.py
- question.py

- trainginData.csv
- trainingData.pkl
- embeddings.json

- runtime.txt
- requirements.txt

- app.py

- __pycache__
- cert.pem
- key.pem
- Procfile

- websiteJSHTML.txt

Setup:

1. Github
    a. Clone the repository to your machine
    b. Make sure the directory is named “flaskferrogpt”
2. OpenAI
    a. Create an OpenAI account with the khanlabgt@outlook.com email and the GT credit card
    b. Go to https://platform.openai.com/account/api-keys and create an API key
    c. Write down this key for future use
3. AWS
    a. Create an AWS account with the khanlabgt@outlook.com and the GT credit card
    b. Navigate to https://s3.console.aws.amazon.com/s3/buckets?region=us-east-2
    c. Create a bucket named “ferroallembeddings”
    d. In this bucket, upload trainingData.pkl and embeddings.json
    e. Under your account name in the top right, go to “Security credentials”
    f. Under “Access keys”, find and write down your Access Key ID and your Secret Access Key for future use.
4. In the directory on your machine, navigate to createCSV.py
    a. Replace “openai.api_key = "yourKey"” with your OpenAI API key
5. In the directory on your machine, navigate to answer.py
    a. Replace “openai.api_key = "yourKey"” with your OpenAI API key
    b. Replace “aws_access_key_id='your_aws_key_here'” and “aws_secret_access_key='your_aws_key_here'” with your AWS keys
6. Heroku
    a. Create a Heroku account with the khanlabgt@outlook.com email and the GT credit card
    b. Create an application
        i. Click on the "New" button on the top-right corner of the dashboard and select "Create new app".
        ii. Name the app “flaskferrogpt”
    c. In the the “Settings” tab of the Heroku application’s webpage:
        i. Navigate to “Config vars” and create these key-value pairs
            1. AWS_ACCESS_KEY_ID - your key
            2. AWS_SECRET_ACCESS_KEY - your key

            3. S3_BUCKET - ferroallembeddings
        ii. Navigate to “Buildpacks” and select “python”

   d. Deploy the application through Heroku
      i. Install the Heroku CLI: https://devcenter.heroku.com/articles/heroku-cli
      ii. In your terminal, navigate to the flaskferrogpt directory, and run “heroku create”
      iii. Temporarily remove embeddings.json, trainingData.pkl, trainingData.csv, websiteJSHTML.txt, trainingDataPdfs folder, and trainingDataDocs folder from the directory, as these files are too big to be stored in the Heroku application.
      iv. Push your code to the Heroku app by running “git push heroku main”
         1. Any changes that you make on your local machine can be updated in the heroku application by running:
            a. git add .
            b. git commit -m “message”
            c. git push heroku main

Let’s examine how these files are interconnected:

Methods in createCSV.py:

- is_pdf
- is_doc_or_docx
- preprocess
- pdfToText
- extractData
- count_tokens
- reduce_long
- get_embedding
- compute_doc_embeddings
- create

All of the research papers used to train FerroGPT were loaded into the trainingDataPdfs folder, regardless of file type. This means this folder contains files of type .pdf, .doc., .docx, etc.

- preprocess takes the files in trainingDataPdfs and checks if they are a pdf or doc/docx file (is_pdf, is_doc_or_docx).
    - If it is a doc/docx file, it is copied into the trainingDataDocs folder.
    - If it is a pdf file, it is converted to a docx file (pdfToText) and stored in the trainingDataDocs folder.

- extractData takes all the files in the trainingDataDocs folder and separates each one into (title, keyword, content, tokenCount) tuples.
    - It does this by converting .docx to text, then separating the text by keyword.
    - Keywords are titles like “abstract” or “introduction” that can be used to separate the paper into parts.
    - Text is separated into arrays, [[keyword1, text from keyword1 to keyword 2], [keyword2, text from keyword2 to keyword 3],....[lastKeyword, last word in text]].
    - Text from keyword to keyword (labeled “content”) is measured in tokens (count_tokens), which are units of measurement for language models. For example, in ChatGPT, you can only input text up to 2000 tokens.
    - If the content is too long (in this case longer than 1500 tokens), it separates the content into sentence groups with size of 1500 tokens each(reduce_long). This means There will be multiple tuples, such as (title, keyword, content, tokenCount), (title, keyword, content2, tokenCount), etc.

- preprocess is called first to create all the docx. files
- create calls extractData for each docx and adds their (title, keyword, content, tokenCount) tuples to an array.
    - From this array, it creates a dataframe object.
    - Then, it saves this dataframe in trainginData.csv. You can visualize the data here.
    - It converts the dataframe to a pickle object, and stores it in trainingData.pkl.

- compute_doc_embeddings is then called. This method takes in the dataframe, and for each line of (title, keyword, content, tokenCount), it calls get_embedding.

    - This creates embeddings, which is how language models measure the content of text and its relation to other texts.
    - It returns embeddings for the entire dataframe
- All the embeddings are stored in embeddings.json.

Try this yourself:

- In your environment, make sure you are using the python version in the runtime.txt file.
- Also make sure you have all of the requirements in requirements.txt installed.
- Add files into trainingDataPdfs.
- In your terminal, navigate to the repo folder and run “python3 createCSV.py”.
- You will have overwritten trainginData.csv, trainingData.pkl, and embeddings.json with data relating to your .docx files.

Ask a question:

- question.py takes in a question and calls the getAnswer method in answer.py.
- getAnswer first pulls the trainingData.pkl, and embeddings.json files from AWS
- It then takes the question and compares it to the embeddings in embeddings.json to find context in the dataframe that most answers the question. It then returns an answer and the context.

Try this:

- Upload trainingData.pkl, and embeddings.json to AWS
- Open question.py and type a question. Then open your terminal and run “python3 question.py”.The answer will be outputted in the terminal.

How this works on the Khan Lab website:

The app is run through Flask and deployed through Heroku.

- app.py imports getAnswer from answer.py
- Then it starts the Flask application.

- It takes the input from the input box on the website and runs this through getAnswer
- There is code in the website to have a countdown for loading and call the API to fetch the response/answer. I have copied this into wbsiteJSHTML.txt for reference.

Recommendations for future work:

- Upgrade the OpenAI account to extend API access past 90 days
- The (title, keyword, content, tokenCount) tuples sometimes are inaccurate. For example, sometimes the keyword is “conclusion”, but the content is from “references”. This is probably due to confusion when the pdf is transformed to docx then text. To improve the accuracy of the results, try doing pdf to image to text. This may provide more accurate keyword/content pairings. Here is an API for that: https://cloud.google.com/document-ai
- The app works well with straightforward questions. However, it does not know synonyms or related topics of your search. To make it more holistic, try implementing semantic search. Here is a tutorial: https://www.youtube.com/watch?v=ocxq84ocYi0