DEPI GENERATIVE AI: TEXT GENERATION MODEL

Group Members:

Shahd Mohamed

Fady Boktor

Abdalla Ibrahim

UNDER SUPERVISION OF: ENG MOHAMMED AGOOR

PROJECT DESCRIPTION

We have built a generative AI text generating model which supports users with quick accurate responses using the right prompts. The model has been trained and tested utilizing cleaned and processed using cleaned and pre-processed wiki text data. The code below has been written on google Collab with python.

PROJECT IMPACT

The model is meant to serve the community in reaching faster to the accurate required information saving time. Quick and accurate information savings has much more impact beyond time savings (e.g. Economical and Financial Impact).

PROJECT BREAKDOWN:

The section below outlines the steps involved in fine-tuning a pre-trained GPT-2 language model on the WikiText-2 dataset and defines the architecture for a basic text-generating using PyTorch.

PRODUCT OVERVIEW

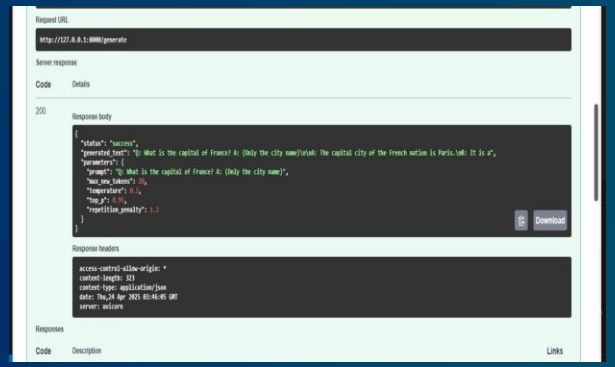
ENVIRONMENT SETUP AND LIBRARIES INSTALLATION

LOADING AND PREPROCESSING DATASET

SPLITTING DATA AND FINE TUNING GPT2 MODEL

(TRAINING AND TESTING)

DEPLOYMENT AND RESULTS



ENVIRONMENT SETUP AND LIBRARIES INSTALLATION:

This section of code is responsible for installing and setting up the required libraries for the project. It uses pip, the package installer for Python, and python -m, which runs a library module as a script.

LOADING AND PREPROCESSING DATA:

This section takes a raw text dataset, cleans it, performs NLP preprocessing (tokenization and lemmatization), and then tokenizes it specifically for the GPT-2 model. The final output, tokenized dataset, is ready to be used for training or fine-tuning a GPT-2 model.

SPLITTING DATA AND FINE TUNING GPT- 2 MODEL:

SPLITTING DATA:

splitting the pre-processed dataset into two parts: (80%) one for training the model (train_data) and (20%) one for validating its performance during training (val_data).

FINE TUNING:

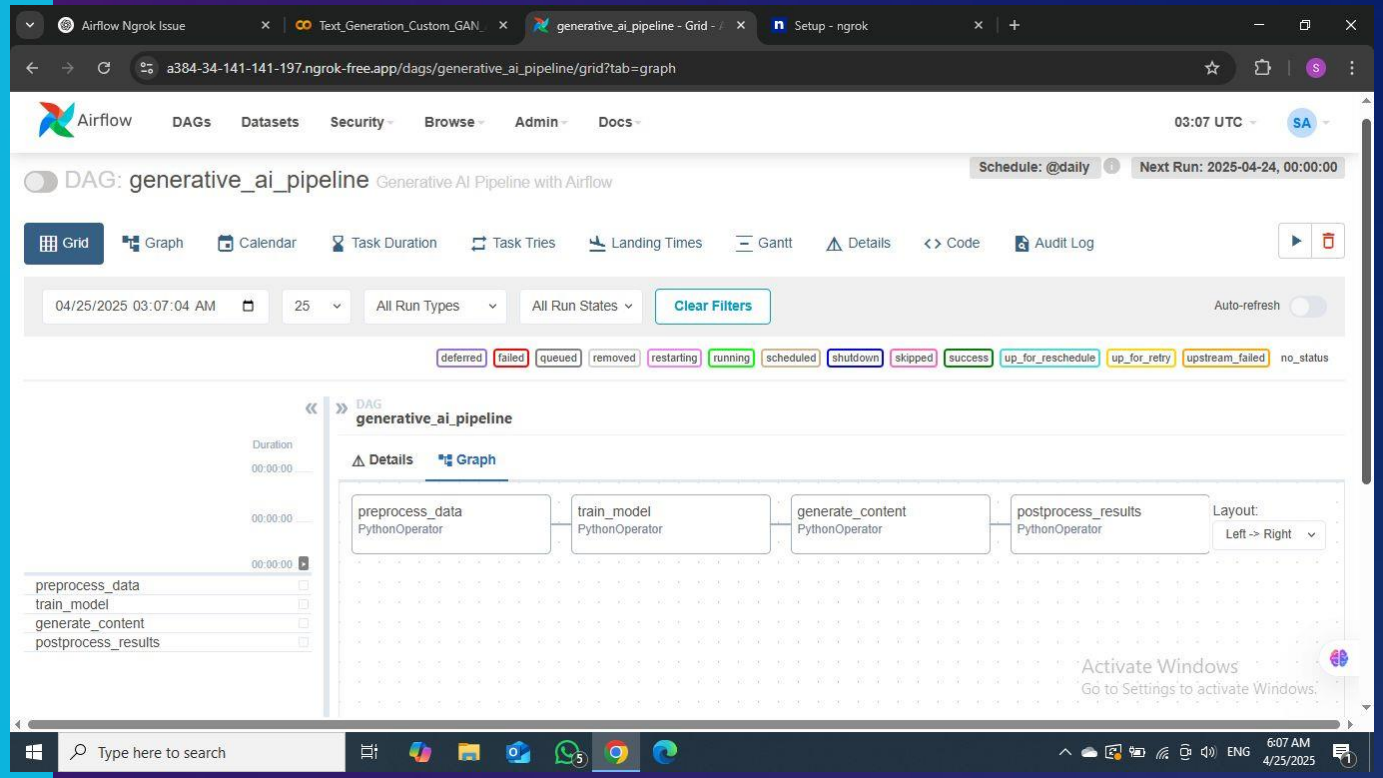
- Libraries and Setup
- Data Preparation
- Training Arguments
- Creating and Training the trainer
- Saving the Fine-Tuned Model
- Checking Training Results

Epoch	Training Loss	Validation Loss
1	0.350200	0.333600
2	0.340700	0.324433
3	0.324100	0.322543

APACHE AIRFLOW CONTENT GENERATION:

Apache Airflow automates and manages workflows in the content generation pipeline, ensuring tasks run in the right order and on time.

- Automates Task Scheduling: Runs tasks like data scraping, preprocessing, and model training at scheduled intervals.
- Workflow Orchestration: Defines task dependencies as Directed Acyclic Graphs (DAGs) to ensure smooth execution
- Monitoring: Tracks task status and provides real-time error handling.
- Scalable and Flexible: Easily scales to handle larger datasets and more tasks.
- Seamless ML Integration: Automates model retraining and content generation.



DEPLOYMENT AND RESULTS

•We have deployed a text generation model using FastAPI, enabling us to generate text from a prompt through an API.

generatee.py

```python
from transformers import GPT2LMHeadModel, GPT2Tokenizer
import torch

# Load tokenizer and model once
model_path = "./saved_model/final model"
tokenizer = GPT2Tokenizer.from_pretrained(model_path, padding_side="left")
tokenizer.pad_token = tokenizer.eos_token  # Set padding token to eos
model = GPT2LMHeadModel.from_pretrained(model_path)
model.eval()

def generate_text(
    prompt,
    max_new_tokens=100,  # Control how much new text to generate
    temperature=0.9,
    top_k=50,
    top_p=0.95,
    repetition_penalty=1.5,
    do_sample=True,
    num_beams=1
):
    # Tokenize the input prompt, allowing truncation and padding to the left
    inputs = tokenizer(prompt.strip(), return_tensors="pt", padding=True, truncation=True, max_length=512)
```

EXPLORER

TEXT GENERATION PROJECT

- app
  - __pycache__
  - __init__.py
  - generate.py
  - main.py
  - model.py
- notebook
- saved_model
- static
- training
  - gpt2_finetune.py
  - wandb_config.py
- venv
- requirements.txt

main.py

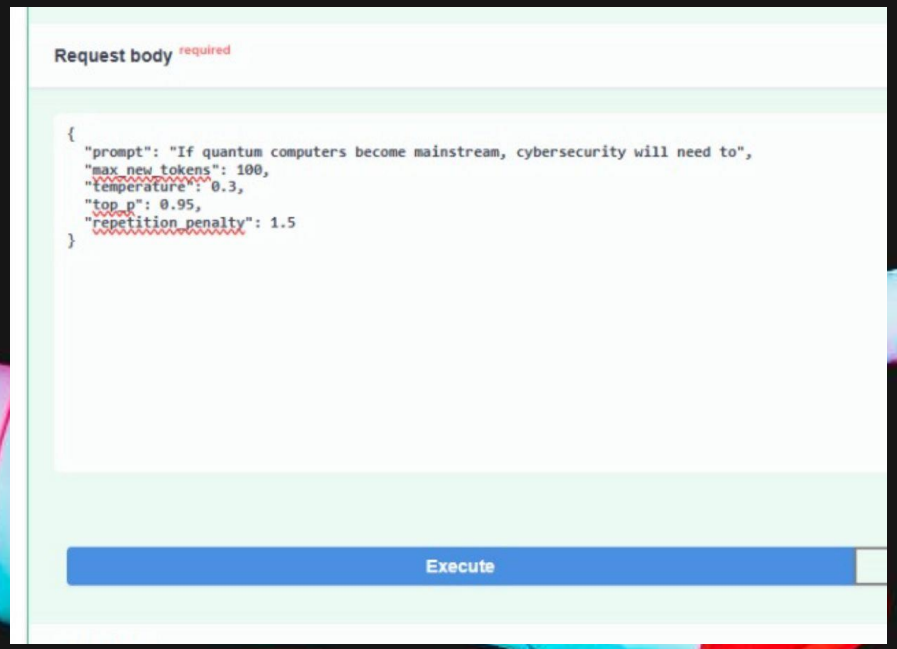
app > main.py

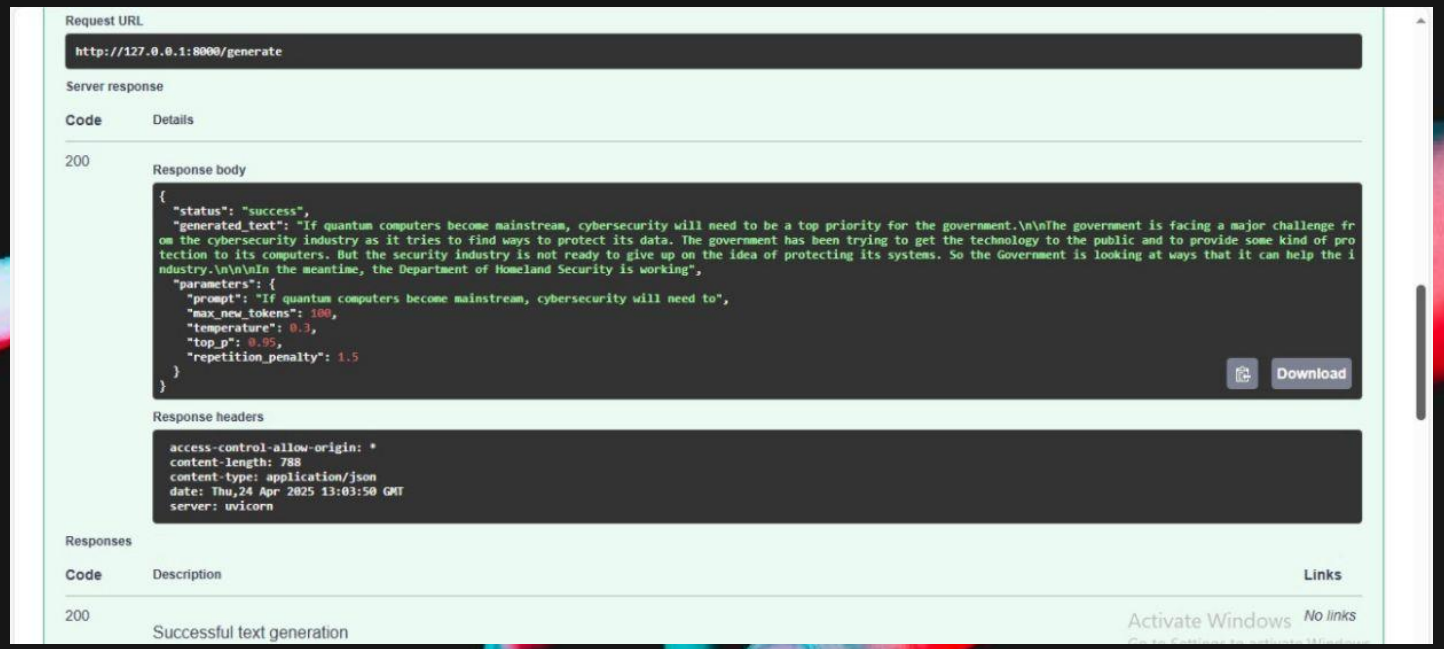
```python
from fastapi import FastAPI, HTTPException, Body, Request
from pydantic import BaseModel, Field
from app.generate import generate_text
import torch
from fastapi.staticfiles import StaticFiles
from fastapi.middleware.cors import CORSMiddleware
import logging

# Setup logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

app = FastAPI(
    title="Text Generation System",
    version="1.0.0",
    description="API for generating text using a fine-tuned GPT-2 model",
    openapi_tags=[{
        "name": "Generation",
        "description": "Text generation endpoints"
    }]
)
```

DEPLOYMENT AND RESULTS





THANK YOU