DEPI Generative AI: Text Generation Model (Technical Documentation)

Group Members:

- Shahd Mohamed
- Fady Boktor
- Abdalla Ibrahim

Document Content:

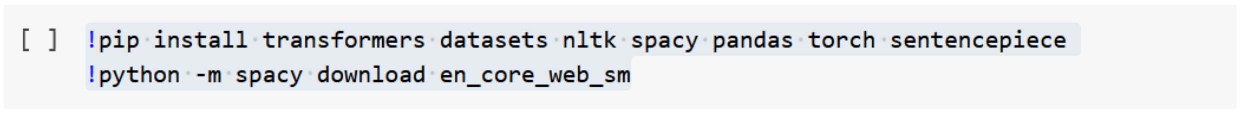
Project Description:
We have built a generative AI text generating model which supports users with quick accurate responses using the right prompts. The model has been trained and tested utilizing cleaned and processed using cleaned and preprocessed wikitext data. The code below has been written on google colab 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.

Environment Set-up and Installing necessary libraries:

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.

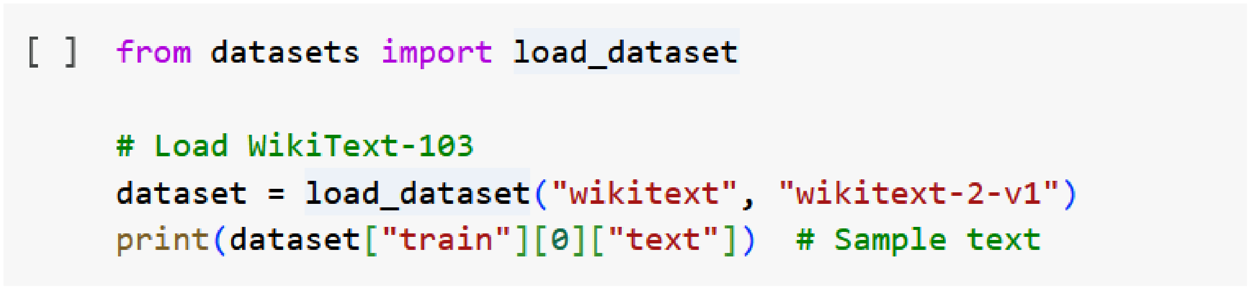


```
[ ]  !pip install transformers datasets nltk spacy pandas torch sentencepiece
     !python -m spacy download en_core_web_sm
```

- **Transformers:** Provides pre-trained models (like GPT-2) and utilities for Natural Language Processing.
- **Datasets:** Enables easy loading and processing of various datasets, including WikiText.
- **nltk:** Natural Language Toolkit, used here potentially for basic text processing tasks (though not explicitly used in the provided snippet beyond installation).
- **Spacy:** A library for advanced Natural Language Processing, used for tokenization and lemmatization.
- **Pandas:** A data manipulation and analysis library (not explicitly used in the core logic but might be used for data exploration).
- **torch:** The core PyTorch library for tensor computations and building neural networks.
- **sentencepiece:** A subword tokenization library (often used with some transformer models, though GPT-2 uses its own tokenizer).
- **en_core_web_sm:** A small English language model for spaCy, downloaded in the second line. This model is used for lemmatization and stop word removal.

Loading Dataset:

This code snippet is responsible for loading the WikiText-2 dataset, which is a collection of text commonly used for language modeling tasks.



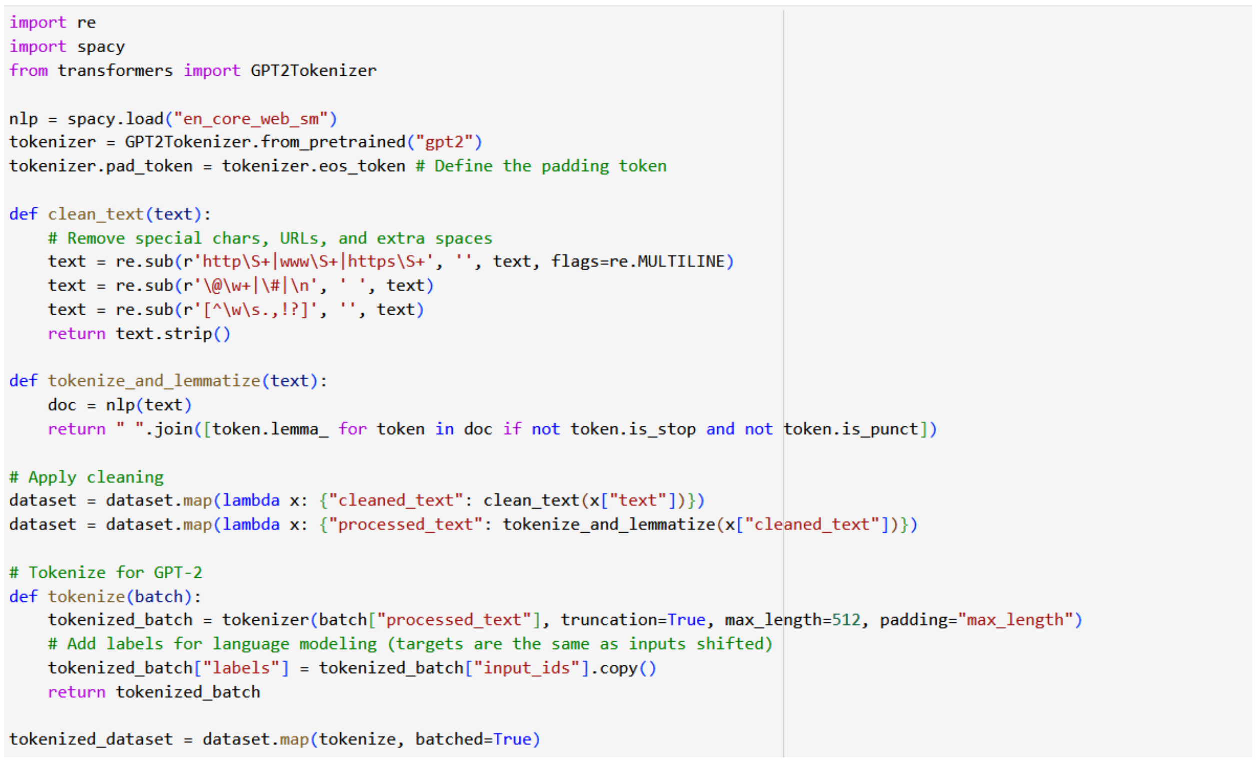
```
[ ]  from datasets import load_dataset

     # Load WikiText-103
     dataset = load_dataset("wikitext", "wikitext-2-v1")
     print(dataset["train"][0]["text"])  # Sample text
```

- **from datasets import load_dataset***:* This line imports the load_dataset function from the datasets library. This function is used to easily download and load various datasets, including WikiText-2.

- **dataset = load_dataset("wikitext", "wikitext-2-v1"):** This is the core part where the dataset is loaded.
    - **load_dataset:** This function fetches the specified dataset.
    - **"wikitext":** This argument indicates the name of the dataset collection (WikiText).
    - **"wikitext-2-v1":** This specifies the particular version or configuration of the WikiText dataset to use (WikiText-2, version 1). The loaded dataset is stored in the dataset variable

Data Preprocessing:

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.



```python
import re
import spacy
from transformers import GPT2Tokenizer

nlp = spacy.load("en_core_web_sm")
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
tokenizer.pad_token = tokenizer.eos_token # Define the padding token

def clean_text(text):
    # Remove special chars, URLs, and extra spaces
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    text = re.sub(r'\@\w+|\#|\n', ' ', text)
    text = re.sub(r'[^\w\s.,!?]', '', text)
    return text.strip()

def tokenize_and_lemmatize(text):
    doc = nlp(text)
    return " ".join([token.lemma_ for token in doc if not token.is_stop and not token.is_punct])

# Apply cleaning
dataset = dataset.map(lambda x: {"cleaned_text": clean_text(x["text"])})
dataset = dataset.map(lambda x: {"processed_text": tokenize_and_lemmatize(x["cleaned_text"])})

# Tokenize for GPT-2
def tokenize(batch):
    tokenized_batch = tokenizer(batch["processed_text"], truncation=True, max_length=512, padding="max_length")
    # Add labels for language modeling (targets are the same as inputs shifted)
    tokenized_batch["labels"] = tokenized_batch["input_ids"].copy()
    return tokenized_batch

tokenized_dataset = dataset.map(tokenize, batched=True)
```

Importing Libraries

- **import re:** This line imports the re module, which is used for working with regular expressions in Python. Regular expressions are powerful tools for pattern matching and text manipulation.
- **import spacy:** This imports the spacy library, a popular natural language processing (NLP) library in Python. It's used for tasks like tokenization, lemmatization, and part-of-speech tagging.
- **from transformers import GPT2 Tokenizer:** This line imports the GPT2 Tokenizer class from the transformers library. The transformers library is developed by Hugging Face and provides pre-trained models and tokenizers for various NLP tasks. GPT2Tokenizer is used to prepare text data for the GPT-2 language model.

Initializing NLP Tools:

- **nlp = spacy.load("en_core_web_sm"):** This line loads a small English language model from spacy, called "en_core_web_sm". This model will be used for NLP tasks like lemmatization.
- **tokenizer = GPT2Tokenizer.from_pretrained("gpt2"):** This line initializes a tokenizer specifically for the GPT-2 model. It loads the tokenizer from a pre-trained GPT-2 model.
- **tokenizer.pad_token = tokenizer.eos_token:** This line sets the padding token for the tokenizer to be the same as the end-of-sentence (EOS) token. This is often done in language modeling to ensure that all sequences have the same length.

Text Cleaning and Preprocessing:

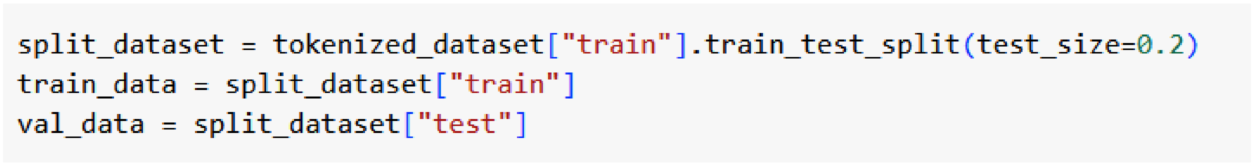
- **clean_text(text):** This function is designed to clean the input text.
  - It uses regular expressions (re.sub) to remove URLs, special characters, Twitter handles (@ mentions), hashtags (#), and extra spaces.
  - It ensures the text is stripped of any leading or trailing whitespace.
- **tokenize_and_lemmatize(text):** This function tokenizes the text and lemmatizes the words.
  - It uses spacy (nlp(text)) to process the text.
  - It then iterates through the tokens and only keeps those that are not stop words (common words like "the", "a", "is") and are not punctuation.
  - It lemmatizes the remaining tokens (converting words to their base form, e.g., "running" to "run") and joins them back into a string.

Applying Preprocessing and tokenization:

- **dataset = dataset.map(...):** These lines apply the cleaning and preprocessing functions to the dataset. The map function is used to apply a function to each element of the dataset.
- The first map call applies clean_text to the "text" column of the dataset and stores the cleaned text in a new column called "cleaned_text".
- The second map call applies tokenize_and_lemmatize to the "cleaned_text" column and stores the result in a new column called "processed_text".
- **tokenize(batch):** This function is used to tokenize the text using the GPT-2 tokenizer.
- It takes a batch of data as input.
- It applies the tokenizer to the "processed_text" column, truncating sequences to a maximum length of 512 tokens and padding shorter sequences to that length.
- It also adds "labels" to the batch, which are used for language modeling (the labels are typically the same as the input sequence, shifted by one position).
- **tokenized_dataset = dataset.map(tokenize, batched=True):** This line applies the tokenize function to the dataset, processing the data in batches to improve efficiency. The resulting tokenized dataset is stored in the tokenized_dataset variable.

Data Splitting:

This section of code is responsible for splitting the preprocessed dataset into two parts: (80%) one for training the model (train_data) and (20%) one for validating its performance during training (val_data).



```
split_dataset = tokenized_dataset["train"].train_test_split(test_size=0.2)
train_data = split_dataset["train"]
val_data = split_dataset["test"]
```

split_dataset = tokenized_dataset["train"].train_test_split(test_size=0.2):

- **tokenized_dataset["train"]:** This accesses the portion of the dataset that was designated for training. Remember that the tokenized_dataset likely contains different parts (e.g., "train", "test", "validation").
- **.train_test_split():** This is a function (likely from a library like scikit-learn) that splits a dataset into two subsets.
- **test_size=0.2:** This argument specifies that 20% of the data should be allocated to the testing/validation set, leaving the remaining 80% for training.
- The result of this line is stored in the split_dataset variable. This variable now contains two subsets of the data.

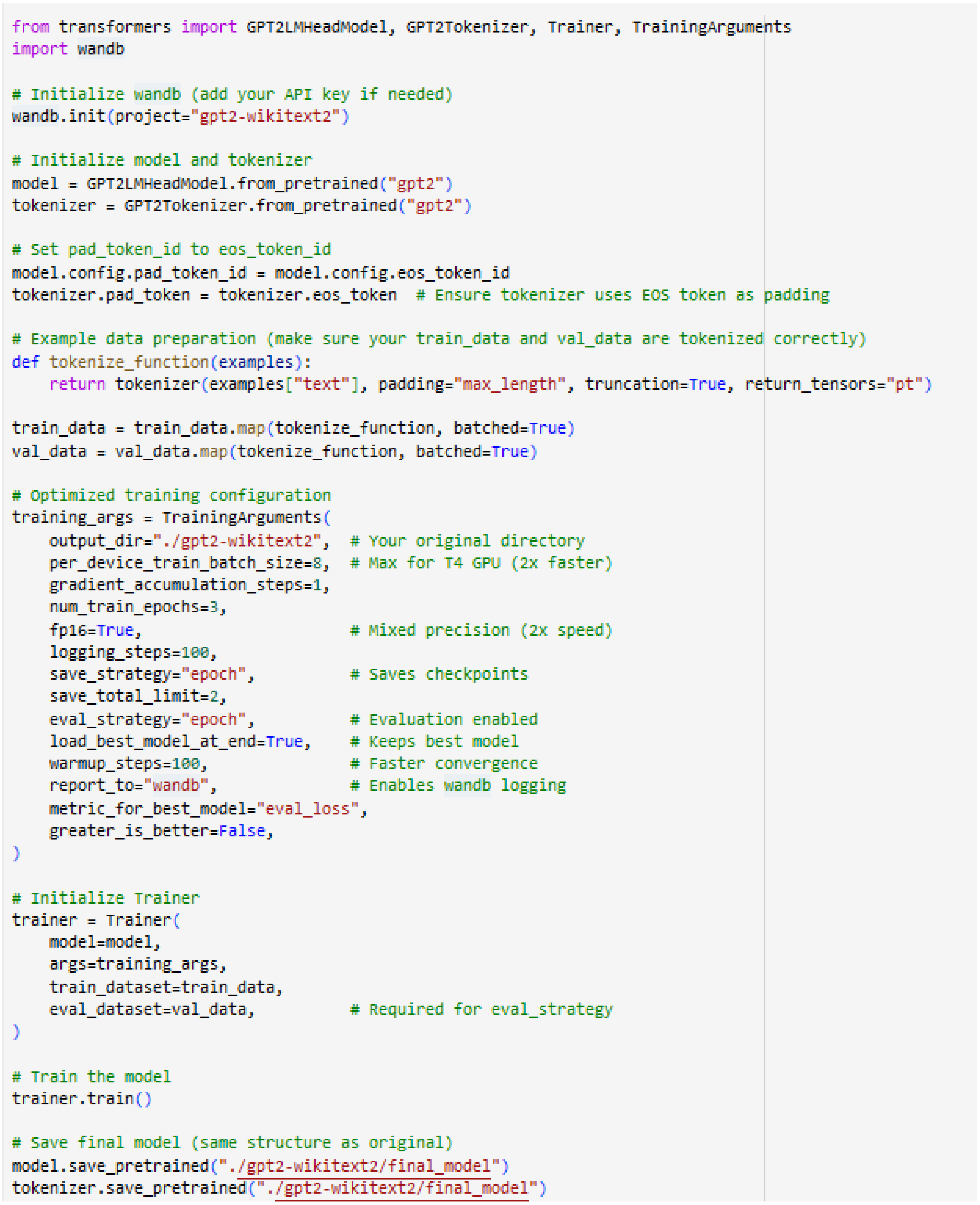
train_data = split_dataset["train"]:

- This line extracts the training subset from split_dataset and assigns it to the variable train_data.

val_data = split_dataset["test"]:

- Similar to the previous line, this line extracts the validation/testing subset from split_dataset and assigns it to the variable val_data

Fine Tuning GPT-2:



```python
from transformers import GPT2LMHeadModel, GPT2Tokenizer, Trainer, TrainingArguments
import wandb

# Initialize wandb (add your API key if needed)
wandb.init(project="gpt2-wikitext2")

# Initialize model and tokenizer
model = GPT2LMHeadModel.from_pretrained("gpt2")
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

# Set pad_token_id to eos_token_id
model.config.pad_token_id = model.config.eos_token_id
tokenizer.pad_token = tokenizer.eos_token  # Ensure tokenizer uses EOS token as padding

# Example data preparation (make sure your train_data and val_data are tokenized correctly)
def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length", truncation=True, return_tensors="pt")

train_data = train_data.map(tokenize_function, batched=True)
val_data = val_data.map(tokenize_function, batched=True)

# Optimized training configuration
training_args = TrainingArguments(
    output_dir="./gpt2-wikitext2",  # Your original directory
    per_device_train_batch_size=8,  # Max for T4 GPU (2x faster)
    gradient_accumulation_steps=1,
    num_train_epochs=3,
    fp16=True,                      # Mixed precision (2x speed)
    logging_steps=100,
    save_strategy="epoch",          # Saves checkpoints
    save_total_limit=2,
    eval_strategy="epoch",          # Evaluation enabled
    load_best_model_at_end=True,    # Keeps best model
    warmup_steps=100,               # Faster convergence
    report_to="wandb",              # Enables wandb logging
    metric_for_best_model="eval_loss",
    greater_is_better=False,
)

# Initialize Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_data,
    eval_dataset=val_data,          # Required for eval_strategy
)

# Train the model
trainer.train()

# Save final model (same structure as original)
model.save_pretrained("./gpt2-wikitext2/final_model")
tokenizer.save_pretrained("./gpt2-wikitext2/final_model")
```

Importing Libraries and Initializing wandb:

- It starts by importing necessary components from the transformers library, including the GPT-2 model (GPT2LMHeadModel), tokenizer (GPT2Tokenizer), Trainer (Trainer), and training arguments (TrainingArguments).
- It also imports wandb, a library for experiment tracking. wandb.init() initializes a new Weights & Biases run to log the training process.

Loading Pre-trained Model and Tokenizer:

- It loads a pre-trained GPT-2 model and tokenizer using from_pretrained("gpt2"). This provides a starting point for fine-tuning instead of training from scratch.
- It then sets the padding token ID of the model and tokenizer to the end-of-sequence (EOS) token ID. This ensures consistency during training.

Preparing the Data:

- tokenize_function tokenizing the input text using the loaded tokenizer. It applies padding and truncation to ensure consistent input lengths.
- The train_data and val_data are then processed using the map function to apply this tokenization to the entire dataset in batches.

Defining the training arguments:

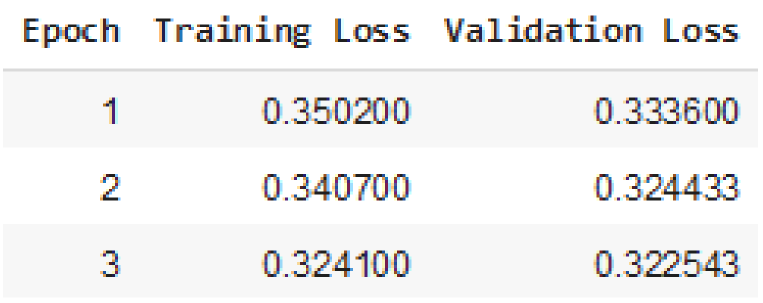
- TrainingArguments is used to configure the training process.
- It sets parameters like output directory, batch size, number of epochs, mixed precision training, logging frequency, checkpoint saving strategy, evaluation strategy, and more.

Creating and Training the Trainer

- A Trainer object is created using the model, training arguments, and the training and validation datasets.
- trainer.train() starts the fine-tuning process.

Saving the Fine Tuned Model:

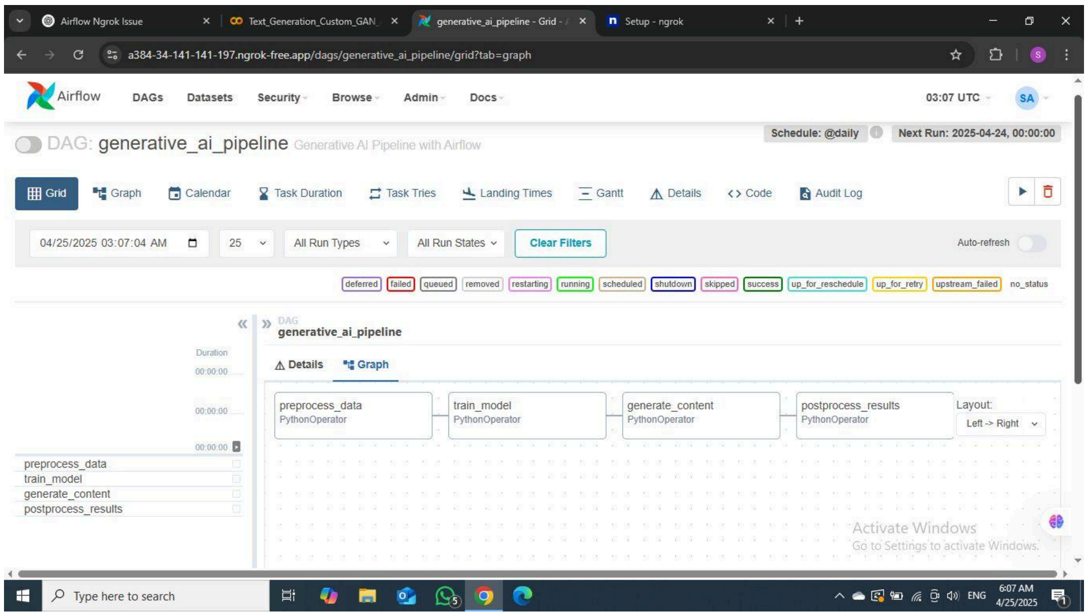
- After training, the fine-tuned model and tokenizer are saved to a specified directory using save_pretrained().
- A message is printed to confirm the completion of training and the location of the saved model.

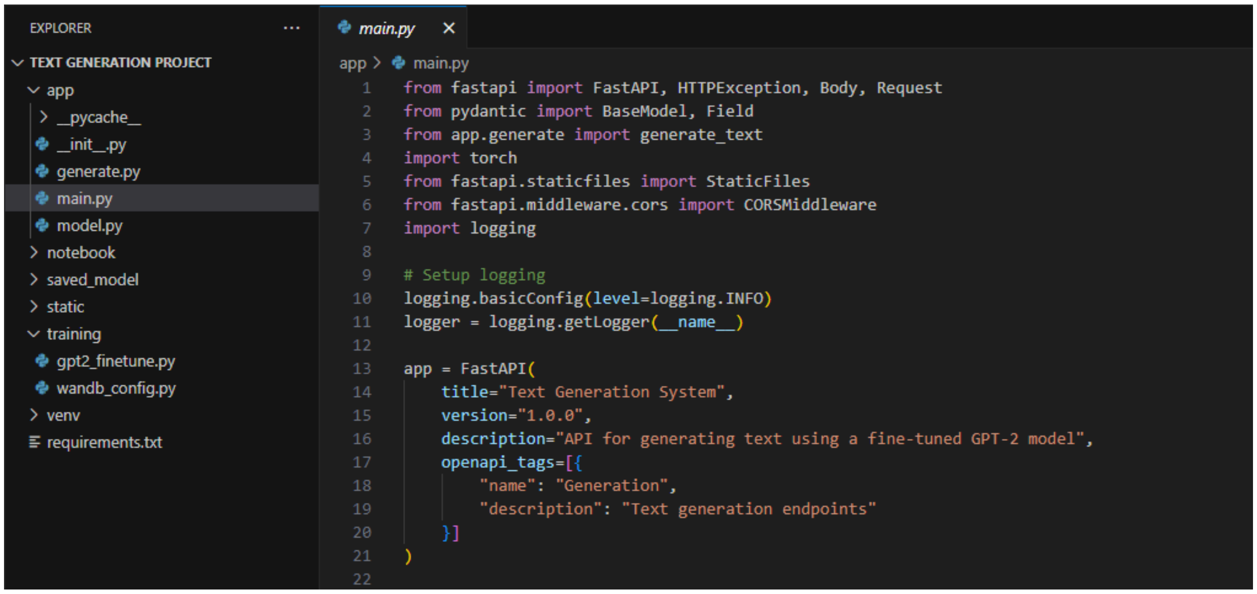


Epoch	Training Loss	Validation Loss
1	0.350200	0.333600
2	0.340700	0.324433
3	0.324100	0.322543

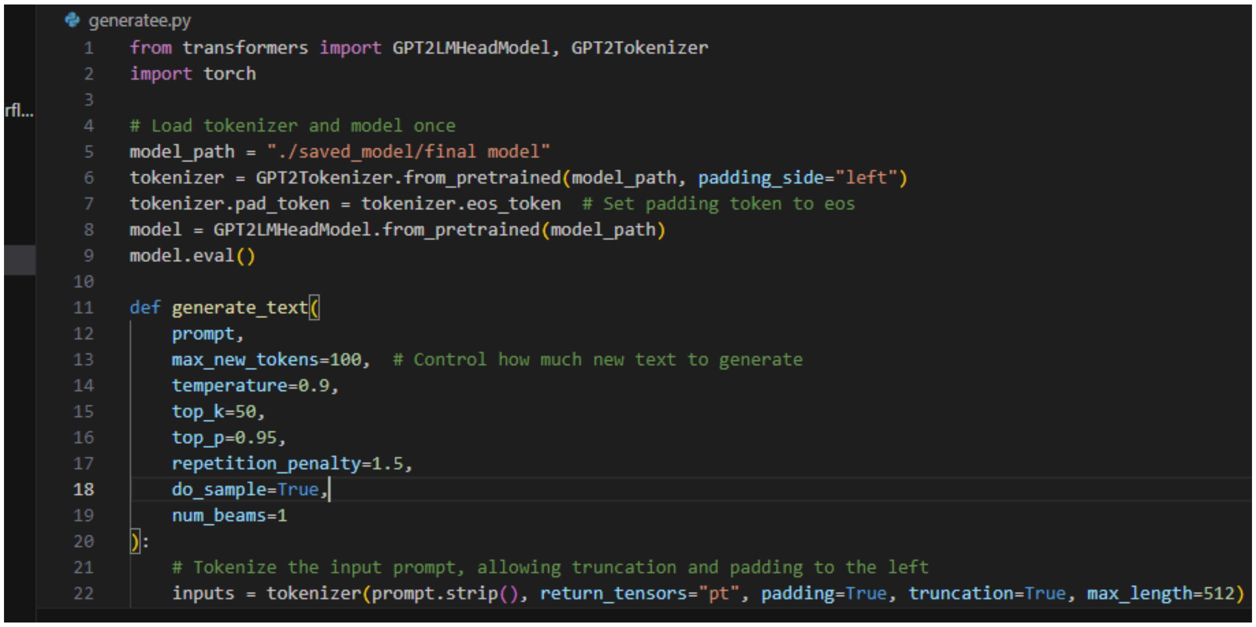
Results and Interface:

We have deployed a text generation model using FastAPI, enabling us to generate text from a prompt through an API. as illustrated in the screenshots below.



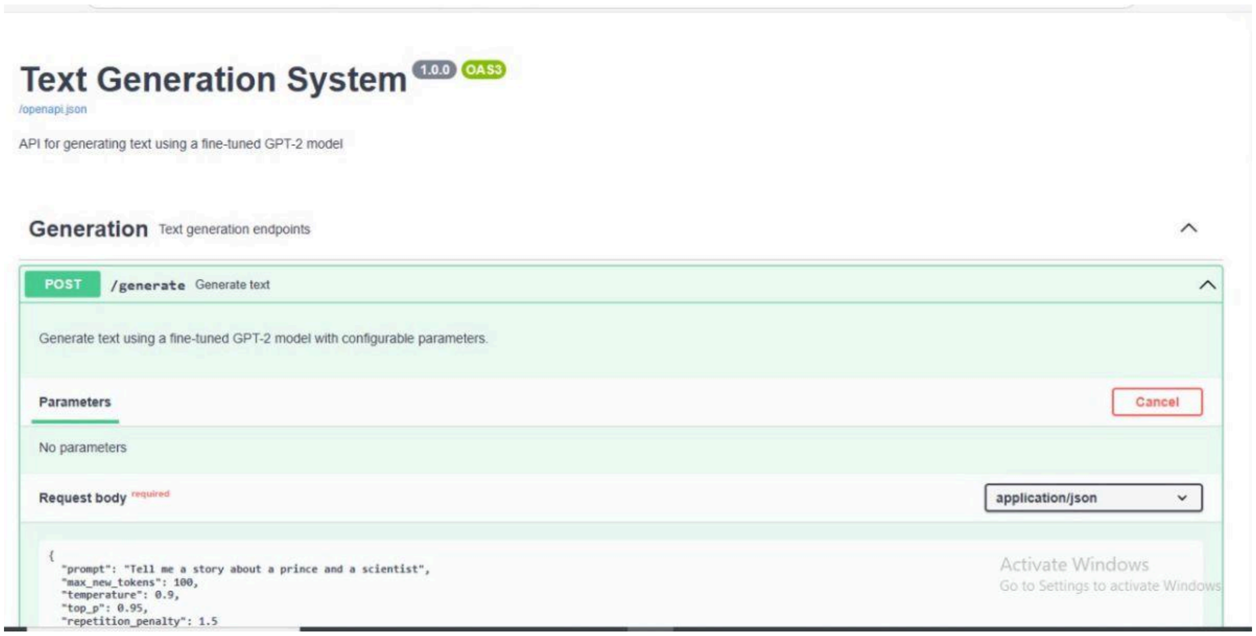


```python
from fastapi import FastAPI, HTTPException, Body, Request
from pydantic import BaseModel, Field
from app.generate import generate_text
import torch
from fastapi.staticfiles import StaticFiles
from fastapi.middleware.cors import CORSMiddleware
import logging

# 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"
    }]
)

```



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)
```



Text Generation System 1.0.0 OAS3

/openapi.json

API for generating text using a fine-tuned GPT-2 model

Generation Text generation endpoints

POST /generate Generate text

Generate text using a fine-tuned GPT-2 model with configurable parameters.

Parameters

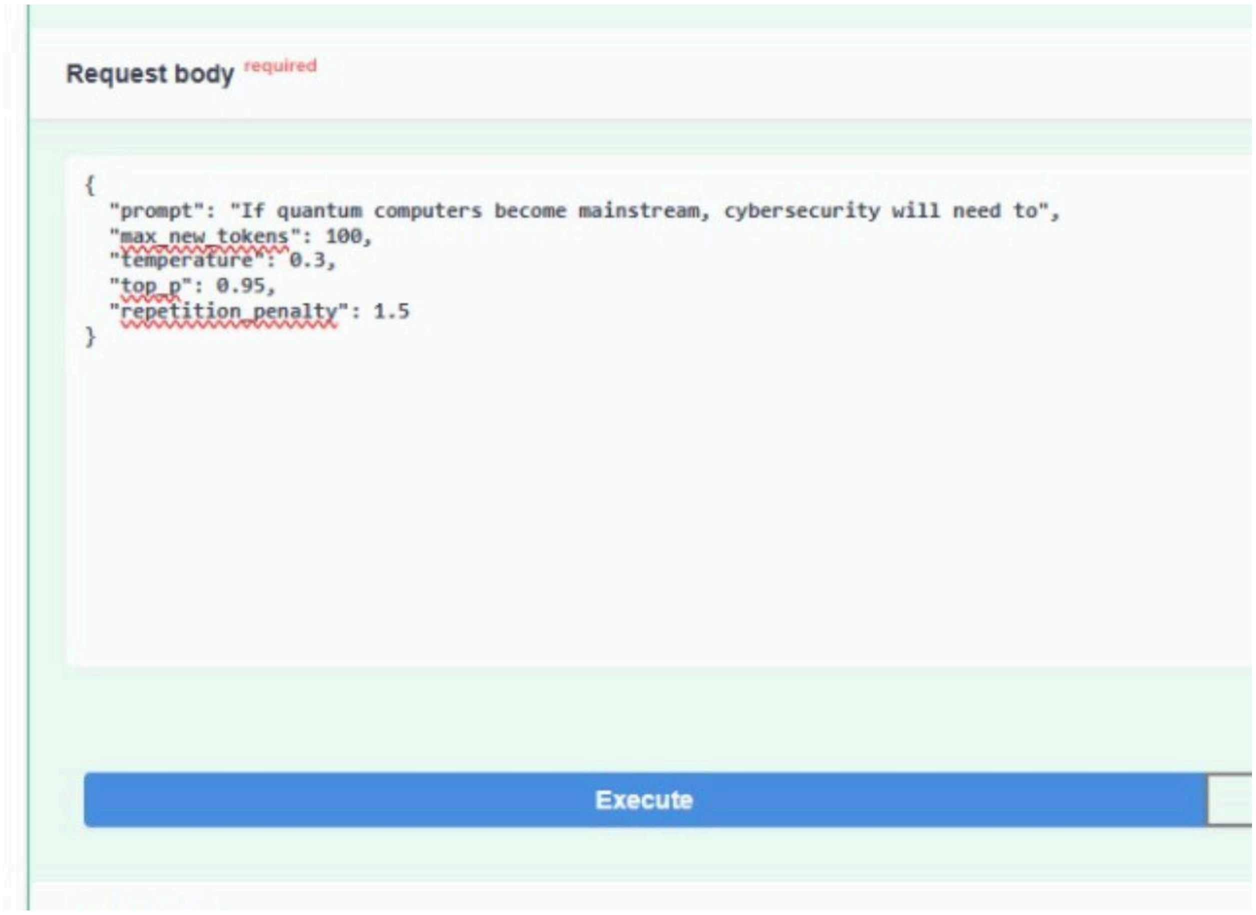
Cancel

No parameters

Request body required

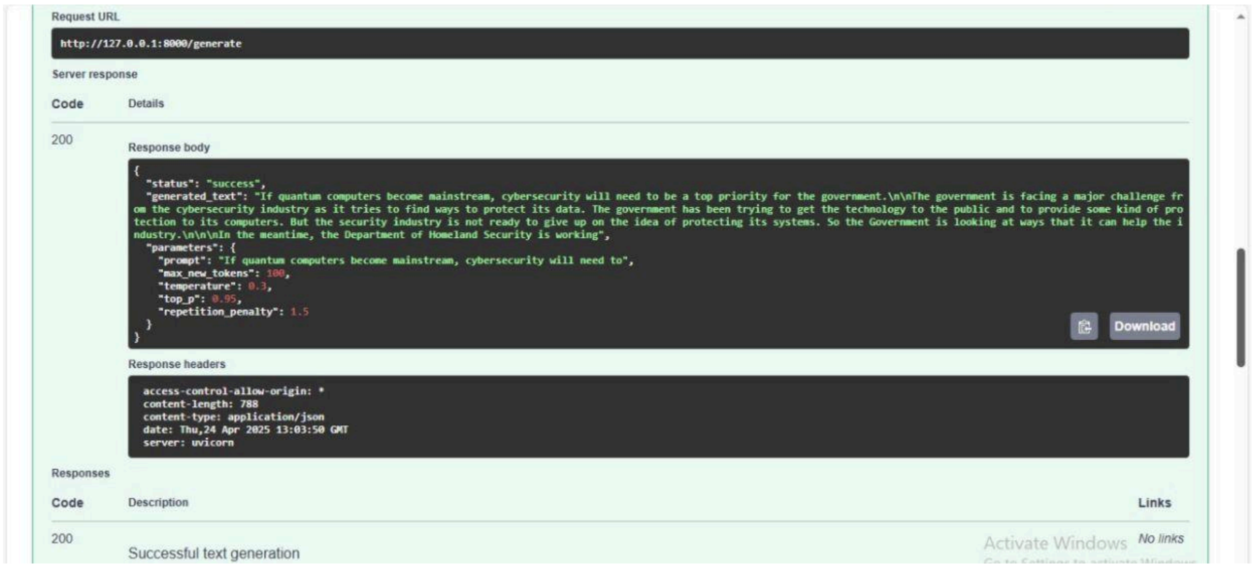
application/json

```
{
  "prompt": "Tell me a story about a prince and a scientist",
  "max_new_tokens": 100,
  "temperature": 0.9,
  "top_p": 0.95,
  "repetition_penalty": 1.5
```



Request body

```
{
  "prompt": "If quantum computers become mainstream, cybersecurity will need to",
  "max_new_tokens": 100,
  "temperature": 0.3,
  "top_p": 0.95,
  "repetition_penalty": 1.5
}
```



Request URL

```
http://127.0.0.1:8000/generate
```

Server response

Code	Details
200	Response body

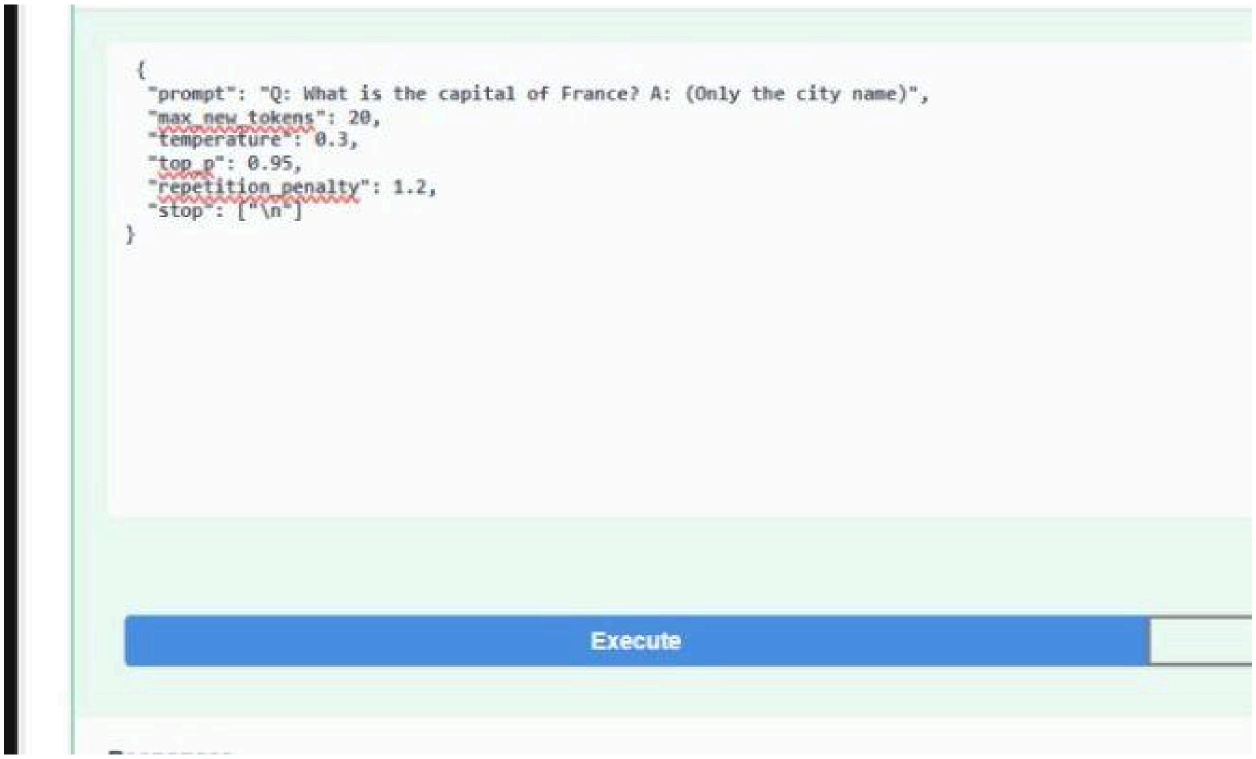
```
{
  "status": "success",
  "generated_text": "If quantum computers become mainstream, cybersecurity will need to be a top priority for the government.\n\nThe government is facing a major challenge fr
om the cybersecurity industry as it tries to find ways to protect its data. The government has been trying to get the technology to the public and to provide some kind of pro
tection to its computers. But the security industry is not ready to give up on the idea of protecting its systems. So the Government is looking at ways that it can help the i
ndustry.\n\n\nIn the meantime, the Department of Homeland Security is working",
  "parameters": {
    "prompt": "If quantum computers become mainstream, cybersecurity will need to",
    "max_new_tokens": 100,
    "temperature": 0.3,
    "top_p": 0.95,
    "repetition_penalty": 1.5
  }
}
```

Response headers

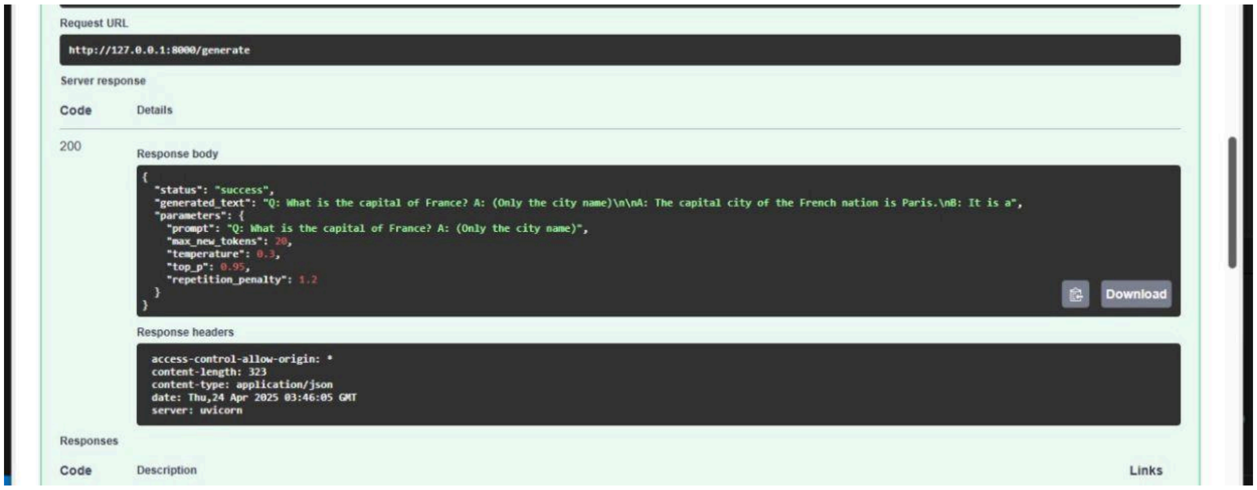
```
access-control-allow-origin: *
content-length: 788
content-type: application/json
date: Thu,24 Apr 2025 13:03:50 GMT
server: uvicorn
```

Responses

Code	Description	Links
200	Successful text generation	*No links*



```
{
 "prompt": "Q: What is the capital of France? A: (Only the city name)",
 "max_new_tokens": 20,
 "temperature": 0.3,
 "top_p": 0.95,
 "repetition_penalty": 1.2,
 "stop": ["\n"]
}
```



Request URL

```
http://127.0.0.1:8000/generate
```

Server response

Code Details

200

Response body

```
{
  "status": "success",
  "generated_text": "Q: What is the capital of France? A: (Only the city name)\n\nA: The capital city of the French nation is Paris.\nB: It is a",
  "parameters": {
    "prompt": "Q: What is the capital of France? A: (Only the city name)",
    "max_new_tokens": 20,
    "temperature": 0.3,
    "top_p": 0.95,
    "repetition_penalty": 1.2
  }
}
```

Response headers

```
access-control-allow-origin: *
content-length: 323
content-type: application/json
date: Thu,24 Apr 2025 03:46:05 GMT
server: uvicorn
```

Responses

Code Description Links