Project Documentation

Overview

The **Nanograd Engine** serves as the core backend for our project, implementing key components like **Stable Diffusion**, **LLMs (Large Language Models)**, **tokenization**, and an **Arabic chatbot**. The project is structured with a clear separation between backend, frontend, and database layers, ensuring a clean, modular design. Below is a detailed explanation of the project components, design patterns, and architecture.

Backend: Nanograd Engine

Core Features

1. **Stable Diffusion**: Implements image generation using the stable diffusion technique.
2. **LLMs (Large Language Models)**: Supports advanced language models for text generation and chat-based interactions.
3. **Tokenization**: Efficient tokenization methods that are optimized for both English and Arabic, crucial for handling text data.
4. **Arabic Chatbot**: A chatbot tailored for Arabic language interactions, offering responses in Egyptian Arabic.

API (FastAPI)

The API layer is built using **FastAPI**, serving as the interface between the frontend and the backend Nanograd Engine. This API handles requests for:

- **Stable Diffusion image generation**
- **LLMs-based chat conversations**
- **Tokenization**
- **Arabic chatbot interactions**

API Routes:

- `/generate-image` (POST): Triggers the stable diffusion engine for image generation.
- `/chat` (POST): Handles chat requests using LLMs or the Arabic chatbot.
- `/tokenize` (POST): Performs tokenization on input text.

All of these services are exposed via HTTP on **localhost:8000**.

API Design Pattern: Decorator

In the API code, decorators are used to modify the behavior of API endpoints, especially for handling tasks like request validation and response formatting.

Frontend

The frontend is built using **JavaScript** to interact with the backend via API calls and **HTML/CSS** for the user interface design. It runs on **localhost:5500**. The frontend is responsible for:

- Sending API requests to the backend.
- Displaying the output (such as generated images, chat responses, etc.).
- Managing user interactions.

Creating Project Page

Before the user accesses the engine, they are directed to the **Creating Project Page**, where:

- The user enters the **name of the project**.
- The user selects the **hardware runner** for the project (such as CPU, GPU, etc.).

Once the user has entered this information, they are directed to the Nanograd Engine page, where they can start using the engine's functionalities.

Database and Checkpoints

The dataset and generated results are stored in two formats:

1. **JSON**: Structured data storage for easier manipulation and retrieval.
2. **Checkpoints**: Pre-trained models for stable diffusion and LLMs are saved as checkpoints, which are loaded when needed for inference.

Design Patterns Implemented

1. **Facade (Nanograd Library)**: The entire Nanograd library serves as a facade, abstracting complex operations like stable diffusion, tokenization, and LLMs behind a simple API. This ensures that users interact with a clean and simple interface, hiding the complexity.
2. **Decorator (API Layer)**: Used to enhance API functionality, such as input validation, authorization, or response transformation, without modifying the core logic.
3. **Proxy (Frontend HTML)**: The authentication for the active and non-active users
4. **Prototype (Stable Diffusion)**: The stable diffusion objects can be considered part of a prototype pattern. Each time an image is generated, it involves creating a new instance or variation of a stable diffusion model object to experiment with.

Testing

A comprehensive testing unit is built for the backend:

- **Stable Diffusion**: Image quality, performance, and response times are validated.
- **LLMs and Chatbot**: Language models and chatbot interactions are tested for accuracy and efficiency, particularly for Arabic responses.
- **API**: End-to-end tests are run to ensure API functionality, correctness, and performance

Unit-test before debugging:

```
FFF
======================================================================
FAIL: test_apply_blueprint (__main__.TestApp.test_apply_blueprint)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:\Users\Esmail\Desktop\nanograd\fastapi\test.py", line 31, in
test_apply_blueprint
    self.assertEqual(response.status_code, 200)
AssertionError: 401 != 200

======================================================================
FAIL: test_chatbot_arabic (__main__.TestApp.test_chatbot_arabic)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:\Users\Esmail\Desktop\nanograd\fastapi\test.py", line 46, in
test_chatbot_arabic
    self.assertEqual(response.status_code, 200)
AssertionError: 401 != 200

======================================================================
FAIL: test_tokenize_text (__main__.TestApp.test_tokenize_text)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:\Users\Esmail\Desktop\nanograd\fastapi\test.py", line 39, in
test_tokenize_text
    self.assertEqual(response.status_code, 200)
AssertionError: 401 != 200

----------------------------------------------------------------------
Ran 3 tests in 0.141s

FAILED (failures=3)
PS C:\Users\Esmail\Desktop\nanograd>
```

Unit-test after debugging

```
....
----------------------------------------------------------------------
Ran 4 tests in 533.817s

OK
sys:1: ResourceWarning: unclosed <socket.socket fd=2428, family=2, type=1,
proto=0, laddr=('127.0.0.1', 58740), raddr=('127.0.0.1', 11434)>
```

Local Development

To run the project locally:

Backend

1. Start the FastAPI server for the backend:

   ```
   uvicorn main:app --reload --port 8000
   ```

Frontend

1. Serve the frontend using any local server (e.g., live-server):

   ```
   live-server --port=5500
   ```

Done by: 1- Esmail Atta Ibrahim Gumaan 23160148
2- hussam Al-maswari 22160157