CALIFORNIA STATE UNIVERSITY
FULLERTON™

Department of Computer Science

This project has been satisfactorily demonstrated and is of suitable form.

This project report is acceptable in partial completion of the requirements for the Master of Science degree in Computer Science.

Generative AI for Creative Expression

Project Title (type)

Yerrabathini Venkat Jawahar Reddy

Student Name (type)

Abdul Motin Howlader

Advisor's Name (type)

Advisor's signature	Date

Abdul Motin Howlader

Reviewer's name

Reviewer's signature	Date

Generative AI for Creative Expression

Submitted by:
Yerrabathini Venkat Jawahar Reddy
CWID : 885187195
CPSC-597
Abdul Motin Howlader

Abstract

This project develops a Generative AI platform that enhances creative expression by generating synchronized lyrics outputs. Utilizing advanced AI and deep learning techniques, the platform addresses the challenge of creating cohesive multi-modal content that resonates with human creativity. The system integrates a web-based interface implemented with Angular and a backend powered by FastAPI, interfacing with MongoDB for data management. The machine learning model, built on transformer architectures, dynamically generates content based on user inputs, demonstrating the potential to revolutionize creative industries by automating and personalizing content creation.

Introduction

The intersection of artificial intelligence and the lyrics opens innovative pathways in creative fields, particularly in music production. This project focuses on the development of a generative AI model tailored to automate and enhance the process of song lyrics creation. By leveraging advanced machine learning techniques, the platform aids lyricists, musicians, and casual creators by providing a tool that stimulates creative output and streamlines the songwriting process.

Description of the Problem

Songwriting, especially lyrics creation, involves deep artistic insight and emotional expressiveness. The challenge in automating this process lies in designing an AI system that can emulate the intricate human capabilities of creativity and empathy. Current solutions often lack the ability to produce lyrics that truly resonate with the thematic and emotional context desired by users. This project aims to bridge this gap by providing a generative model that understands and manipulates language in a way that aligns closely with human artistic expression.

Project Objectives

The core objective of this project is to construct a generative AI model that outputs song lyrics that are not only original and relevant but also carry emotional depth and artistic integrity. Specific goals include:

- Developing a user-friendly interface that allows intuitive interaction for users to specify input parameters such as mood, theme, and genre.
- Crafting a backend architecture that effectively supports the AI model's operations and ensures responsive interactions with the user interface.
- Training the AI model on a diverse corpus to ensure the generative content is unbiased and representative of a broad spectrum of styles and cultural backgrounds.

Development Environment

The development of the AI platform was undertaken using a suite of modern software tools and robust hardware resources:

Software: Python was chosen for its extensive libraries and frameworks that facilitate machine learning and web development. TensorFlow and PyTorch provided the necessary machine learning capabilities for model training, whereas FastAPI was utilized to construct a performant and scalable backend. Angular was used to develop the front-end to ensure a reactive and engaging user experience.

Hardware: The model was trained and developed using GPUs capable of handling extensive computational tasks, crucial for processing large datasets and performing complex neural network calculations.

Literature Review

This section reviews key literature and foundational technologies that have shaped the development of the generative AI model for song lyric creation. The discussion includes seminal works in the field of natural language processing, specifically the transformer architecture, and extends to modern web technologies such as FastAPI and Angular, which facilitate robust and scalable application development.

Transformer Architecture: "Attention Is All You Need"

The groundbreaking paper by Vaswani et al. (2017), titled "Attention Is All You Need," introduces the transformer model, which has revolutionized the way natural language processing tasks are approached. This model discards the conventional recurrence mechanism in favor of attention mechanisms, providing a more efficient method of handling sequences. The transformer's ability to process data in parallel and its scalability has made it highly effective for tasks such as translation and, pertinent to this project, text generation. The model's architecture, comprising multi-headed self-attention and position-wise fully connected layers, offers a significant departure from earlier sequence-to-sequence models and forms the backbone of the AI component in this project.

Application in Lyrics Generation

Building on the transformer's capabilities, subsequent studies and models have tailored this architecture for creative text generation. Papers such as Radford et al.'s on GPT (Generative Pre-trained Transformer) models adapt the transformer architecture for open-ended text generation by training on diverse internet text. The adaptation of these models for generating song lyrics involves training on domain-specific datasets, allowing the model to capture the stylistic and thematic nuances of songwriting.

FastAPI and OpenAPI for Scalable Backends

FastAPI, a modern web framework for building APIs with Python, is grounded in standard Python type hints. The framework, as discussed by its creator Sebastián Ramírez, is designed to be fast, easy to use, and robust, leveraging asynchronous programming patterns and providing automatic interactive API documentation using Swagger UI. It's built on Starlette for the web parts and Pydantic for the data parts, offering a seamless integration with the OpenAPI standard. This allows for the creation of highly interactive, live API documentation and schema generation, which are critical for debugging and testing during development.

Angular for Responsive Front-End Development

Angular, a platform and framework for building single-page client applications using HTML and TypeScript, is developed by Google. The framework's capabilities in building dynamic, progressive web applications are well documented and have been critical in developing the front-end of this project. Angular's architecture, built around components and services, enables a modular approach to web development that enhances testability and maintainability. Papers and articles on best practices in Angular development highlight its dependency injection, declarative templates, and end-to-end tooling, which improve the productivity and quality of development.

Design and Architecture

Data Management

Effective data management is foundational to the success of machine learning models, especially those involved in natural language processing like the lyric generation system described in this project. This section delves into the meticulous process of data handling, from initial gathering to storage, ensuring the integrity and usability of data for training the AI model.

Data Gathering

The initial phase involved aggregating a diverse dataset suitable for training a model capable of generating song lyrics. The dataset comprised 5 GB of various song lyrics collected from multiple sources to ensure diversity in genre, style, and language. This extensive collection aimed to train a model that could appreciate and replicate a wide range of lyrical styles.

Data Cleaning

Once gathered, the data underwent a rigorous cleaning process. This step was critical to remove any corrupt, irrelevant, or redundant information that could skew the model's learning. Common tasks during this phase included stripping out extraneous metadata, correcting misspellings, and removing non-lyrical content such as author names, dates, or extraneous comments embedded within the lyrics.

Data Formatting

Post-cleaning, the data needed to be formatted consistently to ensure uniformity before it could be effectively used for training. This involved standardizing the structure of the dataset, ensuring each entry was uniformly formatted with consistent use of line breaks, punctuation, and encoding. This standardization is crucial for the tokenizer to perform effectively during the model training phase.

Data Storage

After processing, the data was stored in a format ready for efficient retrieval during the training phase. Given the extensive size of the dataset, it was segmented into 20 parts, as local computational resources were insufficient to process it en masse. Each segment was stored in a manner that allowed quick access and processing, leveraging both local storage and cloud solutions when necessary to balance load and performance. MongoDB was used to manage the data efficiently, allowing for scalable and flexible data handling, including quick updates and retrievals necessary for dynamic model training and lyric generation.

Transformer Model Architecture

The transformer model represents a significant departure from earlier sequence processing models that relied on recurrent layers. Its design facilitates parallel processing of sequences and has proven particularly effective in tasks involving large text corpora, making it ideal for the lyric generation application. Here's a breakdown of the key architectural components of the transformer model:

Multi-Head Attention

This component is crucial for the model's ability to focus on different parts of the input sequence simultaneously. Unlike single attention mechanisms that aggregate information from the entire sequence into a single context, multi-head attention allows the model to attend to information at different positions. This is achieved by splitting the attention mechanism into multiple heads, each performing an independent attention operation. This design enables the model to capture various aspects of semantic and syntactic relevance across different subspaces, enhancing its ability to understand complex dependencies in text.

Feed-Forward Networks

Each layer in the transformer contains a position-wise feed-forward network, which applies to each position separately and identically. This consists of two linear transformations with a ReLU activation in between. The feed-forward networks are vital for transforming the representation from the multi-head attention layer, allowing the model to better integrate information across different positions. These networks add depth to the model architecture and are critical for processing the non-linearities in the data.

Add & Norm

Also known as residual connections followed by layer normalization, the Add & Norm step is integral to the transformer architecture. Each sub-layer (including multi-head attention and feed-forward networks) in the transformer has a corresponding residual connection around it, followed by layer normalization. This structure helps in mitigating the vanishing gradient problem by allowing gradients to flow through the network directly. The normalization step stabilizes the learning process by ensuring that the distributions of layer inputs remain more consistent across the network.

Input Embedding

The input tokens, which are initially represented by their unique indices, are transformed into vectors through input embeddings. These embeddings serve as trainable parameters in the transformer model, converting discrete tokens into continuous vectors. This representation allows the model to project tokens into a high-dimensional space where relationships between words (like semantic similarity) can be captured more effectively.

Positional Encoding

Since the transformer does not inherently process sequences in a specific order (due to the lack of recurrence and convolution), positional encodings are added to the input embeddings at the bottom of the model architecture. These encodings provide some information about the relative or absolute position of the tokens in the sequence. Positional encodings can either be learned or fixed and are added to the embedding layer outputs to ensure the model maintains awareness of the order of tokens in the sequence, which is crucial for understanding language.

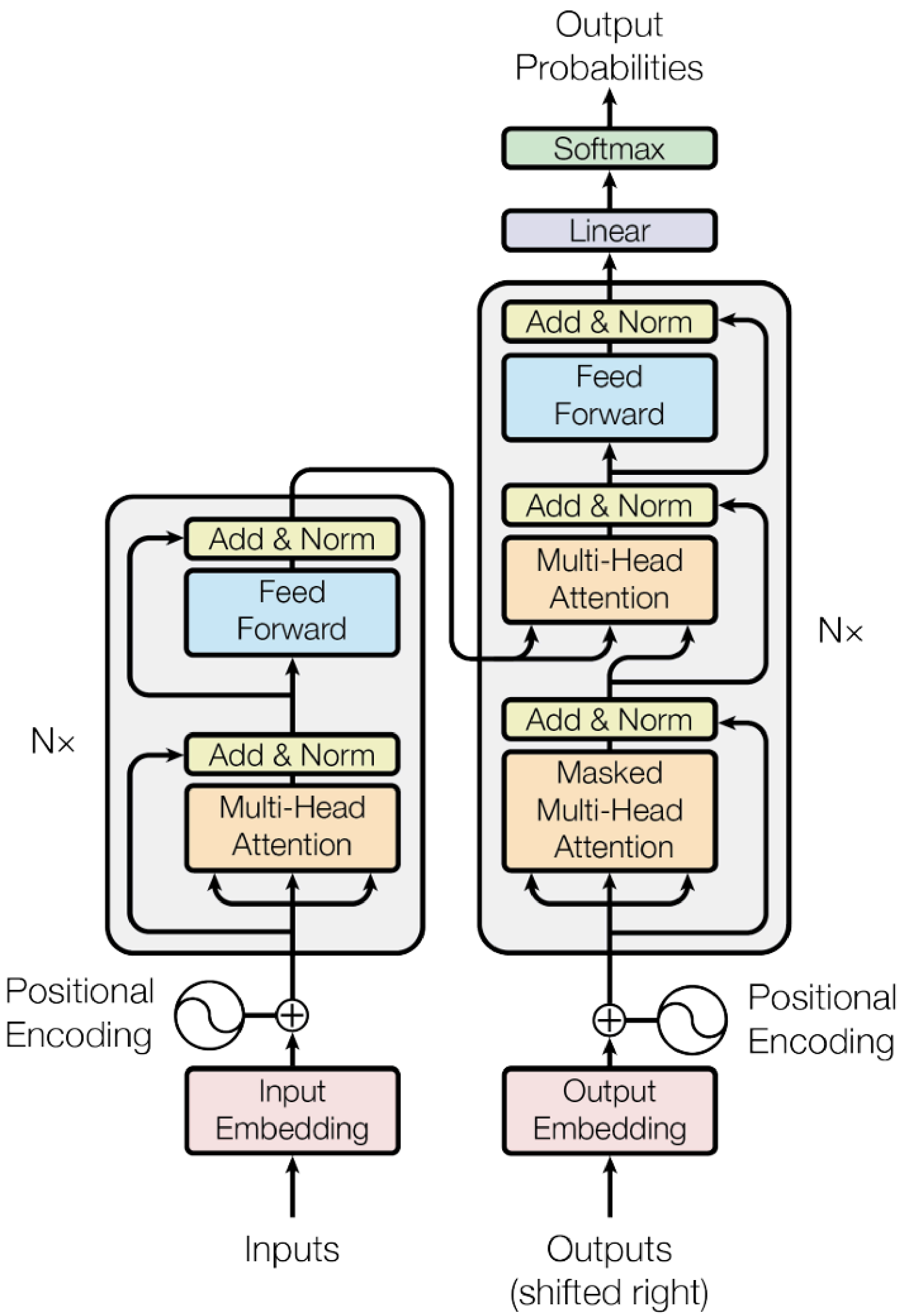


Figure 1: The Transformer - model architecture.

Integration with OpenAI's GPT API

The integration of OpenAI's GPT (Generative Pre-trained Transformer) models via the API provides a robust enhancement to the primary transformer model developed for lyric generation. This integration allows the system to leverage state-of-the-art language models that have been trained on diverse datasets, providing a significant boost in generating creative and contextually rich lyrics.

Here's an exploration of how OpenAI's GPT models are utilized:

OpenAI GPT Overview

OpenAI's GPT models are a series of language prediction models that use deep learning to produce human-like text. They are trained on a mixture of licensed data, data created by human trainers, and publicly available data. These models can generate coherent and contextually relevant text based on the input they receive, making them exceptionally well-suited for tasks requiring a high degree of linguistic understanding and creativity.

API Integration

The system interacts with OpenAI's GPT via a well-defined API that allows sending prompts to the GPT model and receiving generated text in return. This interaction is facilitated through HTTP requests, where the backend server sends a request containing the input lyrics or prompts, and the API responds with generated text. The API uses secure, scalable, and efficient methods to handle requests, ensuring quick responses suitable for real-time lyric generation.

Utilization in Lyric Generation

In the context of this project, OpenAI's GPT API is used to provide additional creative input when generating lyrics. The primary transformer model generates a base lyric output which can then be enhanced or augmented by GPT-generated content. This is particularly useful in cases where the base model struggles with certain lyrical contexts or themes, allowing the GPT model to fill in gaps or suggest alternative phrasings.

Full Stack Integration

The lyric generation system is a complex application that spans several technologies each selected for its particular strengths in handling specific aspects of the application. Here's a breakdown of each component:

Frontend - Angular

- **Framework Choice**: Angular is chosen for the frontend to leverage its robust platform and developer tools that enable building sophisticated single-page applications in a maintainable manner.
- **Data Binding and Reactive Programming**: Angular's powerful data-binding capabilities facilitate an interactive user experience. Reactive programming patterns, supported by Angular's RxJS libraries, allow for efficient handling of data streams and propagation of changes, making the UI highly responsive.
- **Modular Architecture**: Angular's modular component structure enhances code reusability and separation of concerns, which simplifies both development and testing. Components for different aspects of the application, such as user authentication, lyric display, and history management, are developed separately and then integrated seamlessly.
- **Routing and Navigation**: The Angular Router enables navigation between different views and states of the application without reloading the page. This feature is crucial for a single-page application, ensuring smooth transitions and a cohesive user experience.

Backend - FastAPI

- **API Development**: FastAPI is utilized to create a high-performance, easy-to-use backend, providing API endpoints needed for the frontend. FastAPI's ability to handle asynchronous requests makes it well-suited for operations that require calling external services, such as the OpenAI API.
- **Performance and Scalability**: FastAPI's speed and scalability are beneficial for applications needing to handle complex operations and large volumes of requests effectively. The framework supports concurrent handling of requests, which is essential for the lyric generation process that may involve significant processing.
- **Interactive API Documentation**: FastAPI automatically generates interactive API documentation (using Swagger UI and ReDoc), which simplifies development and facilitates testing and integration by allowing developers to quickly test API endpoints.

Database - MongoDB

- **Schema-less NoSQL Database**: MongoDB, a NoSQL database, is used due to its flexibility with unstructured data, which is ideal for storing various data forms generated and used by the application, such as user information, lyrics, and system logs.
- **Scalability and Flexibility**: MongoDB provides high scalability under load and allows for flexible database schemas, making it easy to modify data storage and structure as the application evolves.
- **Integration with Backend**: The Python driver for MongoDB (PyMongo) is used within FastAPI to facilitate data transactions, enabling efficient data storage, retrieval, updates, and deletion operations.

Implementation and Testing

Data Management

The implementation of the lyric generation system required extensive data preparation and management due to the voluminous nature of the dataset. The project initially began with approximately 5 GB of textual data containing song lyrics, which posed significant challenges in terms of processing and computational resources.

Here is a detailed look at the data management strategies employed:

Data Splitting and Reduction

- **Initial Dataset Challenges**: The original 5 GB dataset, when considered for training, estimated a processing time of about 90 days on a local machine—a timeframe that was impractical for the project scope.
- **Data Partitioning**: To manage this, the dataset was divided into 20 smaller chunks. This segmentation allowed the model to be trained on manageable portions of the data, with each part undergoing 3 epochs of training. However, due to time constraints and computational limits, training was completed only on the first four parts.
- **Reduction to 1 GB**: Further data reduction strategies were implemented by filtering the dataset down to 1 GB, which correspondingly reduced the data to manageable token counts of 34,757 for training and 7,077 for testing.

Data Preprocessing and Tokenization

- **Preprocessing Steps**: The data was preprocessed to include start of sequence [SOS] and end of sequence [EOS] tokens. This standardizes the input for the neural network, ensuring that each input sequence is treated uniformly.

- **Enhanced Text Structuring**: Each text entry was structured to incorporate the song's metadata directly into the input data, enriching the context provided to the model. The text column was specifically crafted by concatenating attributes like title, tag, and artist with the lyrics.
- This structuring was crucial for providing the model with a richer set of input features that include both the content and metadata of the lyrics, facilitating a deeper understanding and generation context.
- **Tokenization**: Utilizing the GPT-2 tokenizer from the TikToken library, the text data was converted into a series of tokens. This process is crucial as it transforms raw text into a format that can be fed into the neural network. The tokenizer encodes each piece of text into token IDs, which are then used to train the model.
- **Memory Mapping for Efficiency**: The tokenized data was stored using memory-mapped files. This method enhances the efficiency of data handling, allowing large arrays of data to be manipulated without loading the entire dataset into memory.

Data Storage and Accessibility

- **Efficient Data Handling**: To facilitate efficient access and manipulation of the data during training, the tokenized outputs were saved in binary format using NumPy's memmap functionality. This approach is particularly effective for handling large datasets that exceed the system's memory capacity.
- **Data Integrity and Splitting**: The data was carefully managed to ensure its integrity throughout the training process. Proper train-test splits were maintained to evaluate the model's performance accurately.

The above implementations highlight the extensive data management efforts necessary for handling large datasets in machine learning projects. The strategies of data splitting, preprocessing, and efficient storage not only made the training feasible but also ensured that the system could be iteratively tested and refined.

By implementing these data handling techniques, the project could proceed with model training and further development phases, ensuring that the foundational data was prepared and managed effectively for optimal results in the subsequent stages of model training and system evaluation.

Transformer Model Implementation

The implementation of the Transformer model plays a crucial role in the project, as it forms the core mechanism for generating new song lyrics. Based on the groundbreaking architecture introduced in the "Attention Is All You Need" paper, this project utilizes a

Transformer model that specifically leverages decoder components to generate text. Here we delve into the key components of the model and their roles in processing and generating text.

Input Embeddings and Positional Encoding

- **InputEmbeddings**: The model starts with embedding the input tokens, converting discrete token indices into dense vectors of a specified size (d_model = 384). This embedding layer is critical for interpreting the raw input in subsequent layers.
- **LearnedPositionalEncoding**: Unlike the original Transformer that used fixed positional encodings, this implementation employs learned positional embeddings, adding learned positional information to the input embeddings to maintain the sequence context.

Decoder and Self-Attention

- **MultiHeadAttention**: This is the heart of the Transformer model, enabling it to focus on different parts of the input sequence for each output token. The implementation splits the input into multiple heads (h = 6), allowing the model to jointly attend to information from different representation subspaces at different positions.
- **ResidualConnection and Layer Normalization**: Each attention output is fed through a residual connection followed by layer normalization. This helps in stabilizing the learning and avoids the vanishing gradient problem by allowing gradients to flow through the network directly.

Feed Forward Networks

- Positioned after the attention mechanism in each decoder layer, this consists of two linear transformations with a GELU non-linearity. This sub-layer is essential for transforming the representation after aggregating the attended information from the self-attention mechanism.

Output Projection

- **ProjectionLayer**: The final layer of the Transformer, which projects the output of the last decoder layer back to the vocabulary space, where each token's likelihood is computed.

Training Process

The model is trained using batches of song lyrics, where each batch consists of tokenized input sequences and their corresponding output sequences shifted by one token to the right, serving as targets for training. This setup is essential for the model to learn contextual relationships between words in the lyrics.

- **Loss Calculation**: The training involves calculating the cross-entropy loss between the predicted probabilities of the next word and the actual next word in the sequence. This loss guides the model's weights adjustments to minimize prediction errors.
- **Optimization**: Adam optimizer is used for adjusting the weights of the model, which adapts the learning rate for each parameter, helping in faster convergence.

Model Specifications and Training Configuration

- **Batch Size**: 32
- **Learning Rate**: 1e-3
- **Sequence Length**: 128 tokens
- **Model Dimension** (d_model): 384
- **Tokenizer**: Utilizing tiktoken tokenizer with a vocabulary size of 50,304.
- **Total Parameters**: Approximately 49,305,600 parameters

Implementation Insights

The detailed setup includes

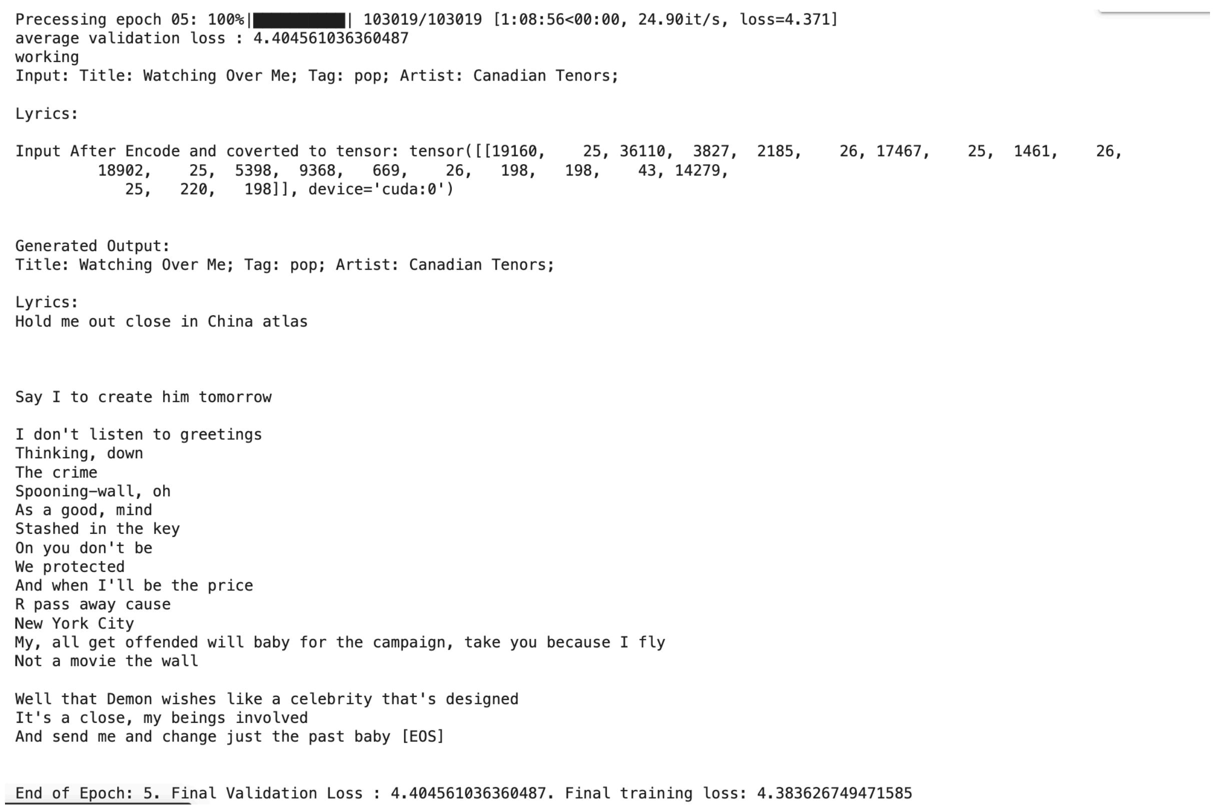
- **Model Configuration**: Parameters such as the size of the model (d_model), the number of layers, heads, and the dropout rate are crucial. These were iteratively adjusted to balance between model complexity and computational feasibility.
- **Tokenization and Data Preparation**: The preprocessing script tokenizes the raw text and structures it into an appropriate format for model training, including special tokens for the start and end of sequences.

Testing and Evaluation

The model's performance is periodically evaluated on a validation set to monitor its ability to generate coherent and contextually appropriate lyrics. This involves generating lyrics from a starting sequence and comparing them against actual lyrics to assess the quality and relevance.

Challenges and Observations

- Handling Large Datasets: Training on a significantly large dataset of 5 GB was computationally expensive and time-consuming. Techniques such as data splitting and token reduction were essential for making the training manageable.
- Model Tuning: Tuning the model's parameters was critical in achieving a balance between overfitting on the training data and underfitting, which would result in less coherent text generation.
- Here below i am sharing the runtime results of one epoch which took more than an hour while running it in GPU.



```
Precessing epoch 05: 100%|██████████| 103019/103019 [1:08:56<00:00, 24.90it/s, loss=4.371]
average validation loss : 4.404561036360487
working
Input: Title: Watching Over Me; Tag: pop; Artist: Canadian Tenors;

Lyrics:

Input After Encode and coverted to tensor: tensor([[19160,    25, 36110,  3827,  2185,    26, 17467,    25,  1461,    26,
         18902,    25,  5398,  9368,   669,    26,   198,   198,    43, 14279,
            25,   220,   198]], device='cuda:0')


Generated Output:
Title: Watching Over Me; Tag: pop; Artist: Canadian Tenors;

Lyrics:
Hold me out close in China atlas



Say I to create him tomorrow

I don't listen to greetings
Thinking, down
The crime
Spooning-wall, oh
As a good, mind
Stashed in the key
On you don't be
We protected
And when I'll be the price
R pass away cause
New York City
My, all get offended will baby for the campaign, take you because I fly
Not a movie the wall

Well that Demon wishes like a celebrity that's designed
It's a close, my beings involved
And send me and change just the past baby [EOS]


End of Epoch: 5. Final Validation Loss : 4.404561036360487. Final training loss: 4.383626749471585
```

Figure 2: My Transformer Model Training Results

Full Stack Integration

The architecture of this project is a seamless integration of a frontend built with Angular, a backend utilizing FastAPI, and MongoDB as a NoSQL database solution, highlighting the comprehensive stack utilized for a modern web application.

FastAPI Backend

- FastAPI is chosen for the backend due to its high performance and ease of use for creating RESTful APIs. It supports asynchronous request handling, which is crucial for handling I/O-bound tasks efficiently.
- The backend is responsible for handling authentication, data retrieval, and interaction with the MongoDB database. It includes endpoints for user management (registration, authentication), and song prediction processing.

User Registration and Login Route

- These endpoints manage user registration and login functionality. User details and authentication data are stored and retrieved from MongoDB, supporting security measures like OAuth2 and password hashing as described.

Predictions Route

- This endpoint handles the song lyric generation requests. It interacts with the Transformer model and possibly leverages the OpenAI API to generate lyrics based on user inputs or contextual data. The results are then stored in MongoDB under a PredictionModel schema and sent back to the user.

Get-History Route

- This allows users to retrieve historical data on their previous predictions. It fetches data from MongoDB where the predictions are stored.

Transformer Model and OpenAI API

Model Implementation

- The Transformer model, as detailed, focuses on text generation using a decoder-only architecture. It utilizes learned positional encodings and a multi-head self-attention mechanism to process and generate text. The model is trained on batches of song lyrics to learn contextual relationships between words, optimizing its parameters using an Adam optimizer.

OpenAI API Integration

- This might refer to utilizing additional capabilities from OpenAI (like GPT models) to enhance the text generation process. The backend's integration with this API facilitates dynamic and contextually relevant lyric generation.

MongoDB Database

- MongoDB, a NoSQL database, is used to store user data and generated lyrics efficiently. Its schema-less nature allows flexible and rapid development, suitable for applications like this where the data model might evolve.
- The database schema includes:
    - **UserModel** for storing user information, including authentication details and user settings. This model is crucial for supporting user-specific functionalities like tracking past predictions.
    - **PredictionModel** for storing details about each song lyric prediction request, including the metadata and the results from the GPT model. This allows users to retrieve their past predictions and provides data that can be analyzed for insights into usage patterns and model performance.

Security and API Integration

- The system uses environmental variables to manage sensitive information like database credentials and API keys, ensuring that these details are not hard-coded into the application’s source code.
- The security.py module utilizes passlib and bcrypt for secure password hashing. FastAPI’s security utilities are used to handle bearer token authentication, which is essential for protecting routes and ensuring that only authenticated users can access their data.
- The integration with OpenAI's API is managed through the backend, where FastAPI routes handle requests to generate new song lyrics. This modular approach keeps the API interactions clean and maintainable.

Angular Frontend

- Angular provides a robust platform for developing dynamic single-page applications (SPAs). The frontend of this project utilizes Angular for its reactive programming capabilities and efficient state management, which ensures that the user interface is responsive and interactive.
- The implementation involves using Angular components and services to handle user interactions and data display. Angular’s data-binding features are extensively used to ensure that changes in the application state are immediately reflected in the view, enhancing the responsiveness of the application.
- Angular’s Router is configured to manage navigation between different views of the application, such as the login page and the prediction interface, providing a smooth user experience.

Testing and Evaluation

- The application is thoroughly tested using Angular’s testing frameworks alongside FastAPI’s test client to ensure that all components interact correctly.
- The backend's integration with MongoDB and OpenAI's API is tested to ensure that data flows correctly through the system and that the song lyrics generated meet the expected quality.

Visual Overview and Interface

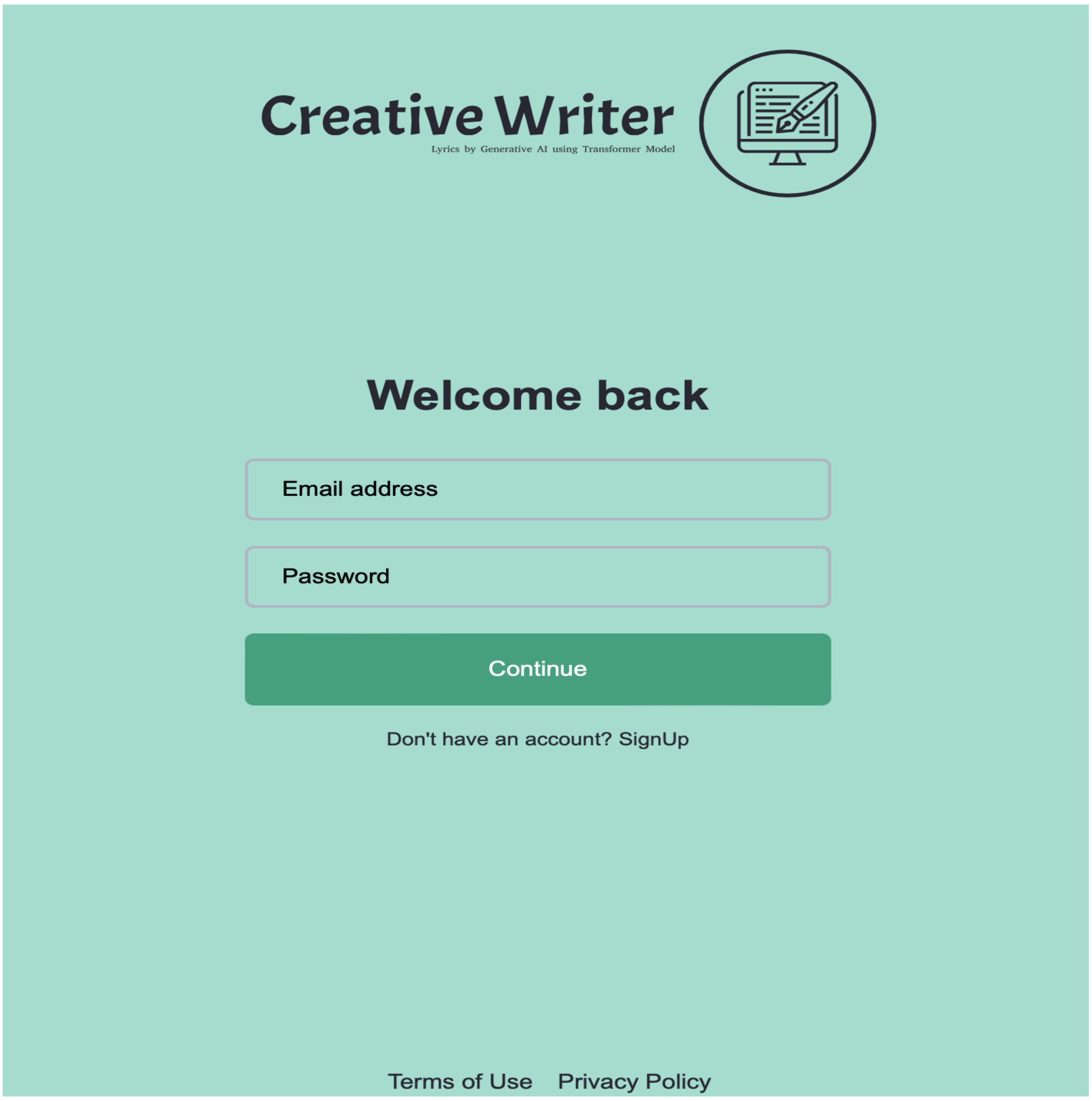
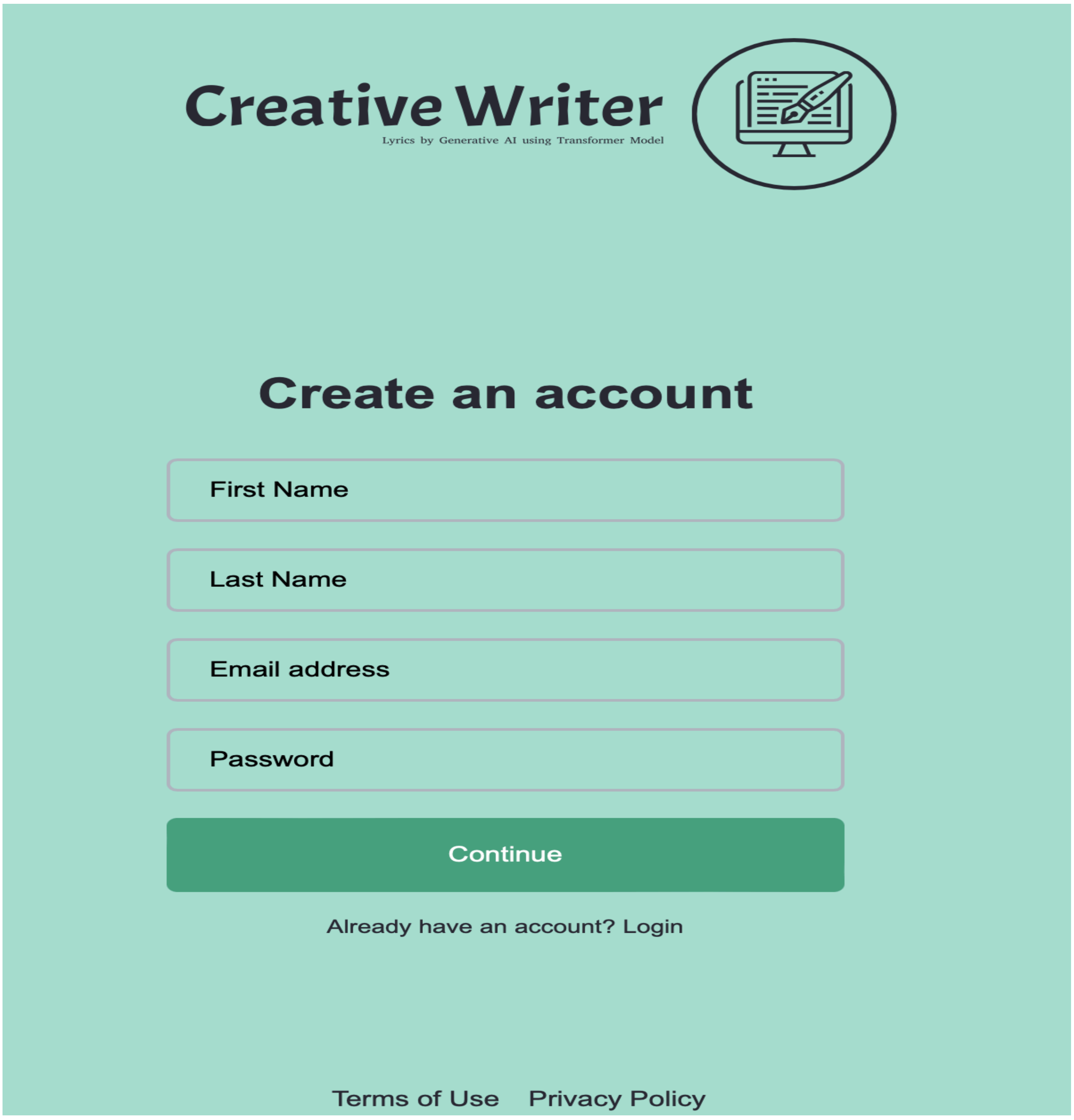


Figure 3: Login Page



Creative Writer

Lyrics by Generative AI using Transformer Model

Create an account

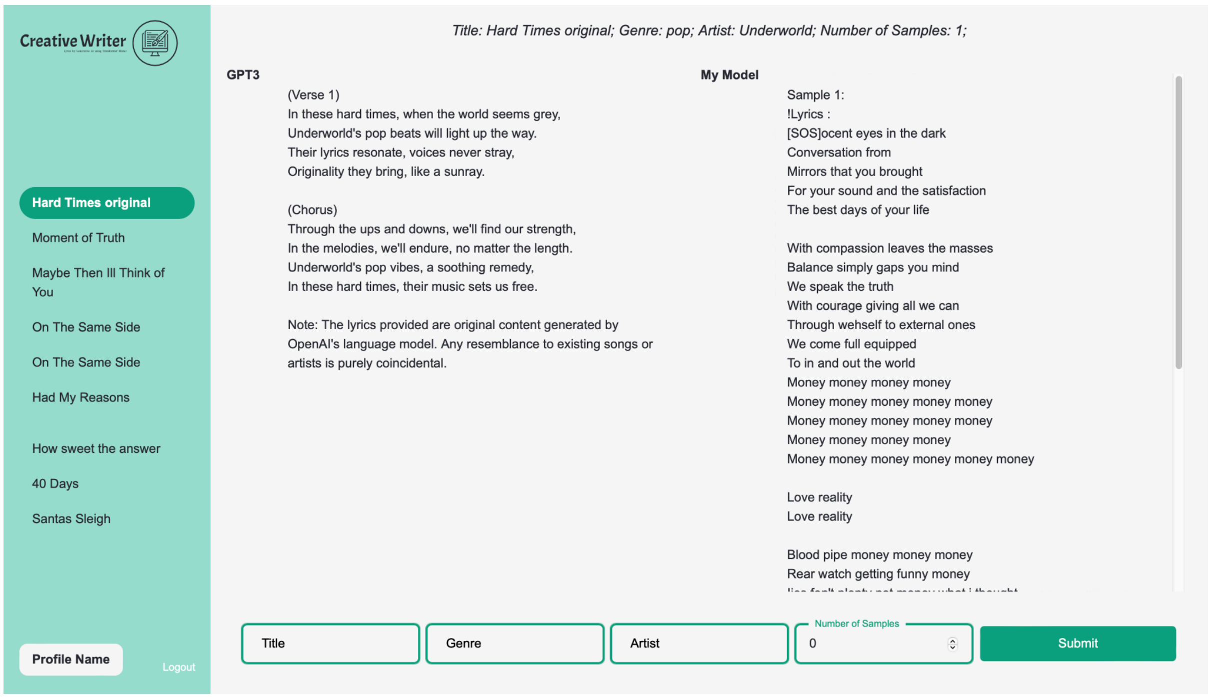
First Name

Last Name

Email address

Password

Figure 4: SignUp Page



Title: Hard Times original; Genre: pop; Artist: Underworld; Number of Samples: 1;

GPT3

(Verse 1)
In these hard times, when the world seems grey,
Underworld's pop beats will light up the way.
Their lyrics resonate, voices never stray,
Originality they bring, like a sunray.

(Chorus)
Through the ups and downs, we'll find our strength,
In the melodies, we'll endure, no matter the length.
Underworld's pop vibes, a soothing remedy,
In these hard times, their music sets us free.

Note: The lyrics provided are original content generated by OpenAI's language model. Any resemblance to existing songs or artists is purely coincidental.

My Model

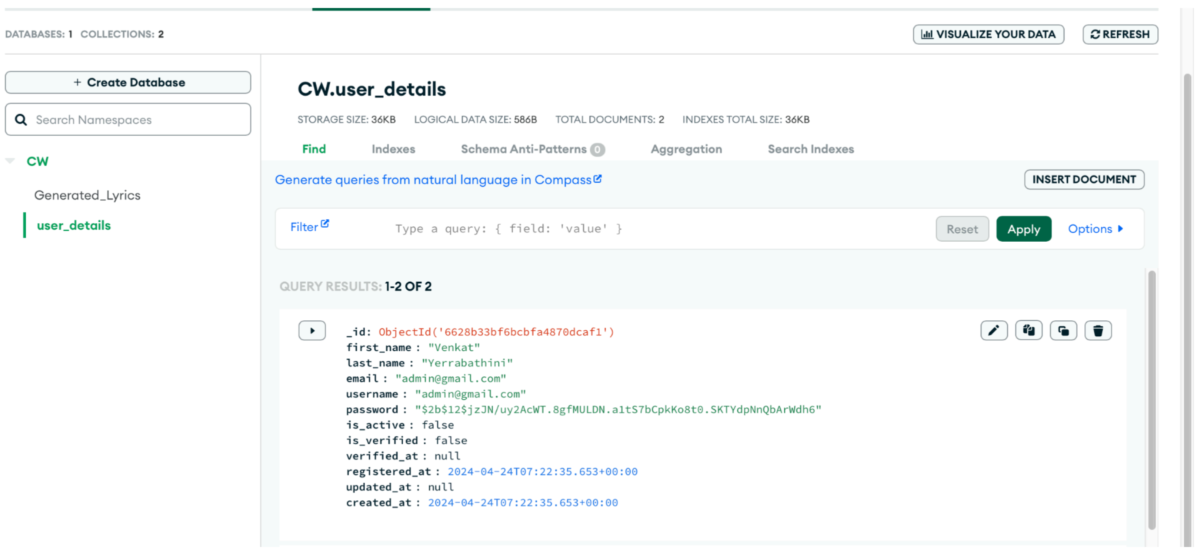
Sample 1:
!Lyrics :
[SOS]ocent eyes in the dark
Conversation from
Mirrors that you brought
For your sound and the satisfaction
The best days of your life

With compassion leaves the masses
Balance simply gaps you mind
We speak the truth
With courage giving all we can
Through wehself to external ones
We come full equipped
To in and out the world
Money money money money
Money money money money money
Money money money money money
Money money money money
Money money money money money money

Love reality
Love reality

Blood pipe money money money
Rear watch getting funny money

Figure 5: Prediction Page



CW.user_details

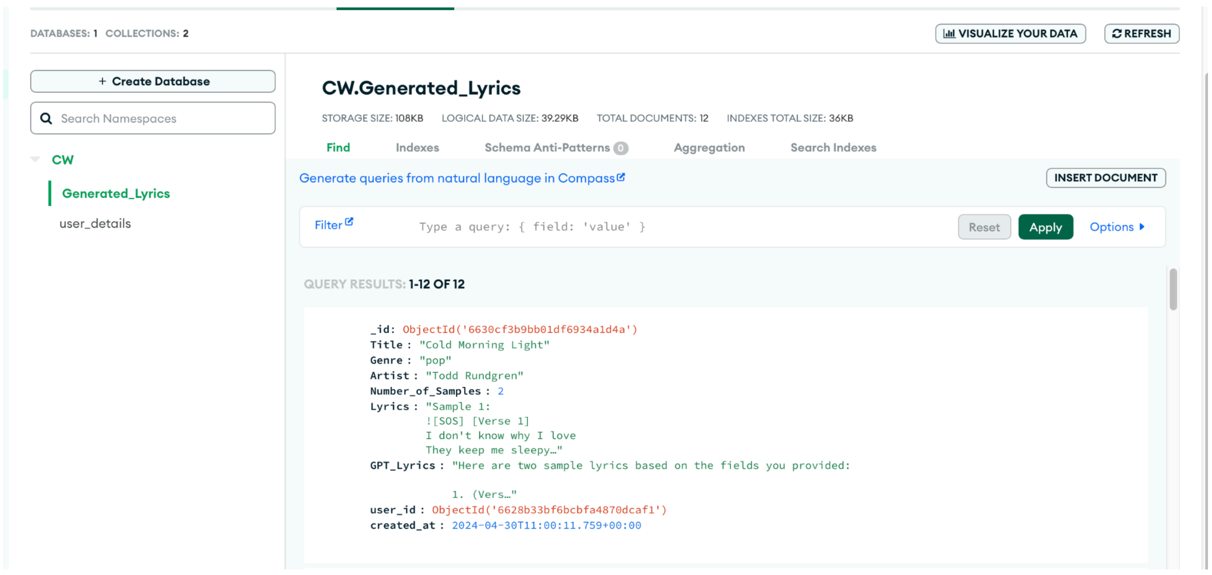
STORAGE SIZE: 36KB LOGICAL DATA SIZE: 586B TOTAL DOCUMENTS: 2 INDEXES TOTAL SIZE: 36KB

Type a query: { field: 'value' }

QUERY RESULTS: 1-2 OF 2

```
_id: ObjectId('6628b33bf6bcbfa4870dcaf1')
first_name : "Venkat"
last_name : "Yerrabathini"
email : "admin@gmail.com"
username : "admin@gmail.com"
password : "$2b$12$jzJN/uy2AcWT.8gfMULDN.a1tS7bCpkKo8t0.SKTYdpNnQbArWdh6"
is_active : false
is_verified : false
verified_at : null
registered_at : 2024-04-24T07:22:35.653+00:00
updated_at : null
created_at : 2024-04-24T07:22:35.653+00:00
```

Figure 6: Mongo DB User Details Collection


DATABASES: 1 COLLECTIONS: 2

VISUALIZE YOUR DATA | REFRESH

+ Create Database

Search Namespaces

CW

Generated_Lyrics

user_details

CW.Generated_Lyrics

STORAGE SIZE: 108KB LOGICAL DATA SIZE: 39.29KB TOTAL DOCUMENTS: 12 INDEXES TOTAL SIZE: 36KB

Find | Indexes | Schema Anti-Patterns 0 | Aggregation | Search Indexes

Generate queries from natural language in Compass

INSERT DOCUMENT

Filter `Type a query: { field: 'value' }` Reset Apply Options

QUERY RESULTS: 1-12 OF 12

```
_id: ObjectId('6630cf3b9bb01df6934a1d4a')
Title : "Cold Morning Light"
Genre : "pop"
Artist : "Todd Rundgren"
Number_of_Samples : 2
Lyrics : "Sample 1:
        ![SOS] [Verse 1]
        I don't know why I love
        They keep me sleepy…"
GPT_Lyrics : "Here are two sample lyrics based on the fields you provided:

            1. (Vers…"
user_id : ObjectId('6628b33bf6bcbfa4870dcaf1')
created_at : 2024-04-30T11:00:11.759+00:00
```

Figure 7: Mongo DB Generated Lyrics Collection

High-Level Architecture Of A Web Application

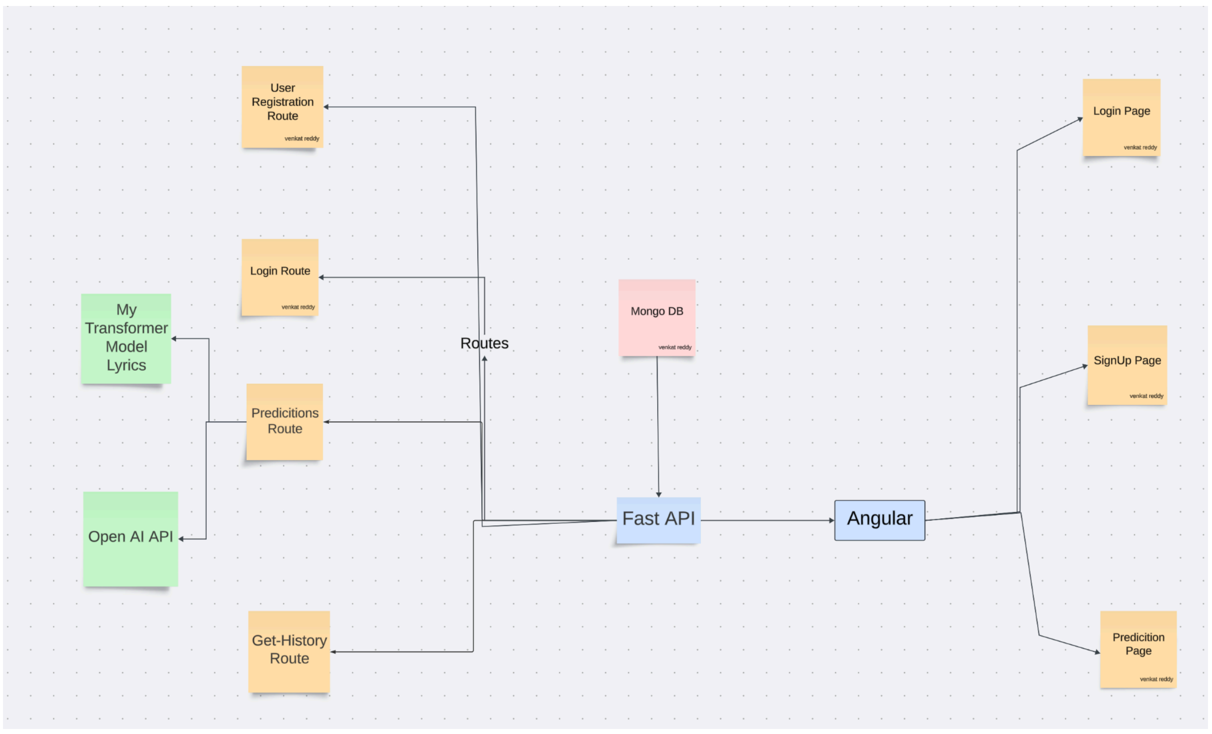


Figure 8: API Interaction with Transformer Model and Client(Angular)

The diagram is divided into three main areas:

1. **Routes (Backend)**
    - FastAPI: Acts as the web framework managing routes for user registration, login, predictions, and fetching historical data.
    - MongoDB: The NoSQL database used to store user data and prediction results.
2. **Model (AI and Lyrics Generation)**
    - My Transformer Model Lyrics: The AI model based on the Transformer architecture that generates song lyrics.
    - Open AI API: Potentially another model or external API used for additional features or enhancement in lyric generation.
3. **Frontend (Web Application)**
    - Angular: The frontend framework responsible for dynamic content rendering, interacting with the backend via API calls, and providing a user interface.
    - Pages (Login, SignUp, Prediction): Various user interfaces for interacting with the application.

Step-by-Step Guide to Start the Project

Create a Local Virtual Environment

For Windows:

- Open Command Prompt and navigate to your project directory.
- Create the virtual environment by running '**python -m venv myenv**'.
- Activate the virtual environment with '**myenv\Scripts\activate**'.

For macOS and Linux:

- Open Terminal and navigate to your project directory.
- Create the virtual environment with '**python3 -m venv myenv**'.
- Activate the virtual environment with '**source myenv/bin/activate'**.

Install Required Libraries

Install all dependencies listed in your requirements.txt file by executing **pip install -r requirements.txt.**

Prepare the Dataset

Assuming you have your raw data, use the data_format.py script to process and split the data into **train.bin** and **test.bin** files:

- Run the data formatting script with python **data_format.py**.

Train the Model

Modify the **config.py** file according to your requirements to adjust hyperparameters. Then, initiate the training process by running python **train_with.py**. Monitor the output to ensure the model is training and the losses are decreasing as expected.

Set Up Backend Configuration

Ensure your **MongoDB** credentials, **OpenAI API key**, and **SECRET_KEY** are correctly set in **BackEnd/Core/security.py** and **BackEnd/Core/database.py**:

- Review and update these configurations as necessary to ensure they are properly configured and secured.

Launch the Backend Server

Start the backend server using FastAPI which will serve as the API backend for your frontend application:

- Navigate to the directory containing **main.py**.
- Start the FastAPI application using **uvicorn main:app --reload**. The --reload option makes the server restart after code changes, which is useful during development.

Start the Angular Frontend

Navigate to the frontend directory and start the Angular application:

- Change directory to the frontend application.
- Install npm packages with **npm install** and start the Angular application with **npm start**.This will compile the Angular application and typically serve it on **http://localhost:4200**.

Testing the Application

Thoroughly test the application by navigating through the frontend and ensuring that all functionalities such as user registration, login, prediction generation, and history retrieval are working as expected.

Evaluation and Optimization or Future Works

System Performance Evaluation

The performance evaluation of the generative AI lyric platform is conducted by comparing the output quality of the custom Transformer model with that of established models like OpenAI's GPT. The primary metrics for evaluation include accuracy of lyrics relevance, user engagement, and system response time. This comparative analysis helps identify the custom model's strengths and areas for improvement. User feedback plays a crucial role in this process, providing insights into the user experience and satisfaction, which are critical for iterative improvements to the system.

Key Evaluation Metrics

- **Lyrics Relevance**: How closely the generated lyrics match the context and style expected by users.
- **User Engagement**: User interaction rates with the generated content, including likes, shares, and retention metrics.
- **Response Time**: The speed at which the system responds to lyric generation requests, crucial for user satisfaction in interactive applications.

Model Optimization

Optimization efforts focus on enhancing the custom Transformer model to improve its performance in generating contextually relevant and creative lyrics. Techniques such as hyperparameter tuning, model architecture adjustments, and training with augmented datasets are employed.

Optimization Strategies

- **Hyperparameter Tuning**: Adjusting learning rates, batch sizes, and attention layers to optimize training outcomes.
- **Expanded Training Dataset**: Incorporating a broader range of music genres and languages to enhance the model's diversity and generalization capabilities.
- **Regularization Techniques**: Implementing dropout and other regularization methods to prevent overfitting and promote model robustness.

Future Enhancements

Looking forward, several enhancements are proposed to extend the platform's capabilities and scalability:

- **Multimodal Data Integration**: Future versions could integrate audio and image data to create a richer dataset for model training. This integration would allow the model to generate lyrics that not only reflect the text input but also adapt to the mood and style indicated by music and album art.
- **Support for Multiple Languages**: Expanding the model to support multiple languages would significantly increase its accessibility and usability across different geographical regions. This requires training the model with a diverse dataset encompassing various languages and cultural contexts.
- **Scalability Improvements**: As user base and data volume grow, scaling the infrastructure to handle increased traffic and data processing needs will be essential. Implementing cloud solutions, optimizing database operations, and using load balancing could address these needs effectively.
- **Advanced AI Models**: Incorporating state-of-the-art AI models that handle complex patterns in music generation could further enhance the quality of output. Research into AI models that can understand and mimic artistic style could lead to more innovative and personalized lyric generation.
- **Feedback Loop Mechanism**: Establishing a systematic feedback loop where user interactions directly influence model training can make the AI more responsive to user preferences and trends in music.

Conclusion

The project successfully developed a robust generative AI platform designed to enhance creative expression by producing synchronized lyrics through advanced artificial intelligence techniques. This achievement marks a significant advancement in the creative industries, blending cutting-edge technology with artistic creation to provide a tool that augments human creativity with AI-driven insights and capabilities.

Achievements of the Project

- **Innovative Integration of AI Models**: At the core of the platform lies the integration of a custom Transformer model and OpenAI's GPT, tailored for generating creative and contextually relevant lyrics. This integration demonstrates the project's innovative approach to leveraging existing AI technologies while enhancing them with custom adaptations to meet specific creative needs.
- **Development of a User-Friendly Interface**: The implementation of an Angular-based frontend ensures that the platform is accessible and user-friendly, enabling artists and creators to interact seamlessly with the AI without needing technical expertise. This accessibility broadens the tool's appeal and utility across a diverse user base.
- **Real-Time Lyric Generation**: One of the standout features of the project is its capability to generate lyrics in real-time, providing immediate creative output that artists can use as inspiration or directly incorporate into their work. This feature

significantly enhances the creative workflow, making it more dynamic and responsive.
- **Multi-Modal Data Handling**: Although the current focus is on text-based lyric generation, the foundational work laid down by this project sets the stage for future multi-modal integration, where audio and visual data can also be used to influence and guide the lyric generation process.

Impact on Creative Industries

The implications of this project for the creative industries are profound. By automating one aspect of the creative process, the platform allows artists to focus on other creative endeavors, thereby enhancing overall productivity and creative output. Moreover, the AI-generated lyrics can inspire new creative directions and interpretations, enriching the creative landscape.

- **Enhancement of Creative Expression**: The platform provides tools that support artists in overcoming creative blocks and experimenting with new styles and themes, thereby expanding their creative boundaries.
- **Democratization of Music Production**: By providing an easy-to-use platform that requires no advanced technical skills, the project democratizes music production, enabling emerging artists and smaller studios to produce high-quality creative content that competes with larger, more established entities.
- **Cultural and Linguistic Diversity**: As the platform evolves to support multiple languages, it has the potential to embrace cultural diversity, promoting a broader range of voices and perspectives within the creative industries.

Reinforcing the Contribution to AI and Creative Technology

This project not only advances the field of AI in creative contexts but also serves as a benchmark for future explorations into how AI can be harmoniously integrated with human creativity. It stands as a testament to the potential of AI to not only mimic human capabilities but to genuinely augment and enhance the human creative process.

Future Potential and Impact

Looking forward, the project is poised to continue evolving, with potential expansions into audio and visual modalities and more sophisticated AI models that could further refine the quality and relevance of the generated content. The ongoing development will likely focus on enhancing the platform's understanding of complex artistic nuances, making its output increasingly indistinguishable from human-generated content.

In conclusion, the project represents a significant leap forward in the convergence of technology and art, highlighting the limitless possibilities when human creativity is amplified by artificial intelligence. This platform not only provides practical tools for today's artists but also inspires future innovations at the intersection of technology and creative expression.

References

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).
2. Radford, A., et al. (2018). Improving Language Understanding by Generative Pre-Training.
3. Sebastián Ramírez. FastAPI Documentation. [Online]. Available: https://fastapi.tiangolo.com/
4. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners. OpenAI Blog.
5. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805.
6. MongoDB. (n.d.). MongoDB Manual. [Online]. Available: https://docs.mongodb.com/manual/
7. Angular. (n.d.). Angular Documentation. [Online]. Available: https://angular.io/docs
8. OpenAI. (n.d.). OpenAI API. [Online]. Available: https://beta.openai.com/docs/
9. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language Models are Few-Shot Learners. arXiv preprint arXiv:2005.14165.
10. Pydantic. (n.d.). Pydantic Documentation. [Online]. Available: https://pydantic-docs.helpmanual.io/
11. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.