Eötvös Loránd University

Faculty of Informatics

Dept. of Programming Languages and Compilers

Simplified Transformer-Based Language Model for Web-Based User Interaction

Supervisor:
Gregory Morse
Teaching Assistant

Author:
Altynbek kyzy Zhainagul
Computer Science BSc

Budapest, 2024

EÖTVÖS LORÁND UNIVERSITY
FACULTY OF INFORMATICS

Thesis Registration Form

Student's Data:
Student's Name: Altynbek kyzy Zhainagul
Student's Neptun code: V7I2JB

Course Data:
Student's Major: Computer Science BSc
I have an internal supervisor

Internal Supervisor's Name: *Gregory Morse*
Supervisor's Home Institution: ***Department of Programming Languages and Compilers***
Address of Supervisor's Home Institution: ***1117, Budapest, Pázmány Péter sétány 1/C.***
Supervisor's Position and Degree: *PhD student and Teaching Assistant and MSc in Computer Science, ELTE*

Thesis Title: Simplified Transformer-Based Language Model for Web-Based User Interaction

Topic of the Thesis:
(Upon consulting with your supervisor, give a 150-300-word-long synopsis os your planned thesis.)

This thesis aims to develop a simplified language model using a transformer-based architecture, designed for interactive experiences on a web platform. The main goal is to create a website that allows users to select training data sources, observe the model training in real-time, and then interact with the trained model through a conversational interface "(similar to ChatGPT)".
Although the model will not have the extensive parameterization of advanced models, it will serve as an effective demonstration of training and running inference and will be a good example of how these systems work.
The training data for the model will be pre-provided from non-copyrighted, publicly available texts. In addition, the platform will feature an option for users to upload their custom training data, offering a more personalized interaction and learning experience.
The training process will be transparent, with real-time updates based on logs gathered during training. Users will be able to view metrics such as current accuracy/precision/recall. This will provide an insightful glimpse into the model's learning progression and overall performance.
In summary, this thesis is dedicated to the creation of a web-based interactive language model that showcases the essentials of AI conversational systems. This project not only demonstrates the functionality of a language model but also provides a hands-on experience in understanding the training dynamics of AI systems.

Budapest, 2023. 12. 01.

STATEMENT
OF THESIS SUBMISSION AND ORIGINALITY

I hearby confirm the submission of the Bachelor Thesis Work on the Computer Science BSc training with author and title:

Name of Student: Zhainagul Altynbek kyzy

Neptun ID of the Student: V7I2JB

Title of Thesis: Simplified Transformer-Based language Model for Web-Based User Interaction

Supervisor: Gregory Morse

at Eötvös Loránd University, Faculty of Infomatics.

In consciousness of my full legal and disciplinary responsibility I hereby claim that the submitted thesis work is my own original intellectual product, the use of referenced literature is done according to the general rules of copyright.

I understand that in the case of thesis works the following acts are considered plagiarism:

- literal quotation without quotation marks and reference;
- citation of content without reference;
- presenting others’ published thoughts as own thoughts.

Budapest, 2024

Student: Zhainagul:sgd

Contents

Chapter 1

Introduction

1.1 Motivation and Goal

The primary motivation behind this thesis arises from the recognition that large language models (LLMs) are both computationally expensive and require substantial hardware resources, typically involving extensive GPU capacities. These requirements often place LLMs beyond the reach of individuals and institutions with limited resources, making it challenging to engage with state-of-the-art AI technologies and contribute to their development.

The goal of this thesis is to demystify the complexities of LLMs by developing a simplified language model that, while not as powerful or capable as full-scale LLMs, provides a practical and insightful introduction to the mechanics of model training and interaction within a conversational AI context. This project focuses on creating an accessible, visually engaging web-based platform where users can experiment with the training process and interact with the model through a conversational chat interface.

This simplified language model serves as an educational tool, offering a hands-on experience that elucidates the fundamental principles of LLMs without the need for expensive infrastructure. By providing a compact, yet comprehensive visualization of how language models are trained and operate, this thesis aims to make the cutting-edge field of LLMs more accessible and understandable to a broader audience.

Chapter 2

User documentation

This guide is designed to assist users through the installation process, familiarize them with the features, and provide a comprehensive overview of how to effectively utilize the application. Whether you're interested in deploying pre-trained models, training new models, or managing your results, this documentation will provide the necessary guidance to ensure a smooth user experience.

2.1 Hardware requirements

To optimize the performance of the Simplified Transformer-Based Language Model, it is important to consider the hardware capabilities of your system. The following guidelines will help ensure that the model operates efficiently, particularly during tasks such as training and user interactions:

- **CPU and RAM:** The performance of the language model scales with the power of the CPU and the amount of RAM available. Higher-performance CPUs and greater RAM capacities will significantly enhance the model's responsiveness and ability to handle complex computations and larger datasets. Better CPU and RAM are recommended for a smoother and faster user experience. However, the model is designed to be functional even on ordinary computers with standard CPU and RAM configurations; it will perform adequately but at a slower pace.

- **GPU:** For users seeking enhanced interaction and reduced model training times, a dedicated GPU is highly recommended. GPUs are particularly effective at accelerating the kind of parallel processing operations that are fundamental to transformer models. Incorporating a GPU can lead to dramatically faster computation times, enabling more dynamic and responsive interactions with the language model.

By ensuring your system meets or exceeds these general hardware recommendations, you can enhance your experience with the language model, achieving faster response times and more efficient data processing.

2.2 Software requirements

- **Operating System:** The model is compatible with modern versions of Windows, macOS, and Linux.
- **Python:** Python 3.8 or newer is required to support the model and associated libraries.
- **Web Framework:** Flask is used for the backend to handle requests and serve the model's functionalities. React is utilized for the frontend to create a dynamic and responsive user interface.
- **Browser:** A modern web browser such as Google Chrome, Mozilla Firefox, Safari is necessary to interact with the web-based interface of the language model.

Specific versions, additional libraries, and step-by-step installation instructions will be detailed in the Dependencies and Installation Guide section.

2.3 Dependencies and Installation Guide

This guide provides detailed steps for installing and configuring all necessary dependencies for the Simplified Transformer-Based Language Model for Web-Based User Interaction. Follow these instructions to set up your environment for optimal performance.

1. **Prepare Your Operating System:** Ensure your operating system is up-to-date to support all installations described below.
2. **Install Python** Download and install Python 3.8 or newer [1].

3. **Install Required Python Libraries:** Use pip to install all the necessary libraries:

```
pip install torch PyPDF2 tqdm seaborn scikit-learn
matplotlib pandas numpy json pickle sys subprocess lzma
```

Code 2.1: Installation command for required Python packages

4. **Set Up Web Environment:**
   - **Flask:** Ensure Flask is installed to interact with the backend of the application:

   ```
   pip install flask
   ```

   Code 2.2: Command for flask installation

   - **React:** Make sure your environment includes Node.js and npm, which are required to run the React application [2].

5. **Development Tools:** It is recommended to have the following development tools installed for an efficient workflow:
   - Visual Studio Code [3]
   - Git [4]

2.4 Starting the Application

This section guides you through the steps to install and run the project. Ensure that you have completed all previous installation steps before proceeding.

1. **Install the project** [5]

```
git clone <repository-url>
```

Code 2.3: Command for cloning the repository

2. **Install NPM Packages:** Navigate to the project's root directory and install the npm packages required for the React application:

```
npm install
```

Code 2.4: Command for npm dependencies installation

3. **Start the Backend Server:** From the project directory start the Flask Server with:

```
python app.py
```

Code 2.5: Command for starting the flask server

4. **Start the React Application:** In a separate terminal instance run the following command in the root of the React project directory:

```
npm start
```

Code 2.6: Command for compiling the React application

5. **Check the Application:** The website with URL - `http://localhost:3000/` should be accessible now.

2.5 Usage of the System

This is the user guide for interacting with Simplified Transformer-Based Language Model. This section provides comprehensive instructions on how to effectively utilize the various features of our web-based application. If all the conditions that listed above are fulfilled successfully the Chat page will be opened automatically Figure 2.1.

REST APIs

1. /**upload-files:** This route handles file uploads from the Data Preparation Page. It processes and stores the files in a designated directory, making them available for data extraction and model training.

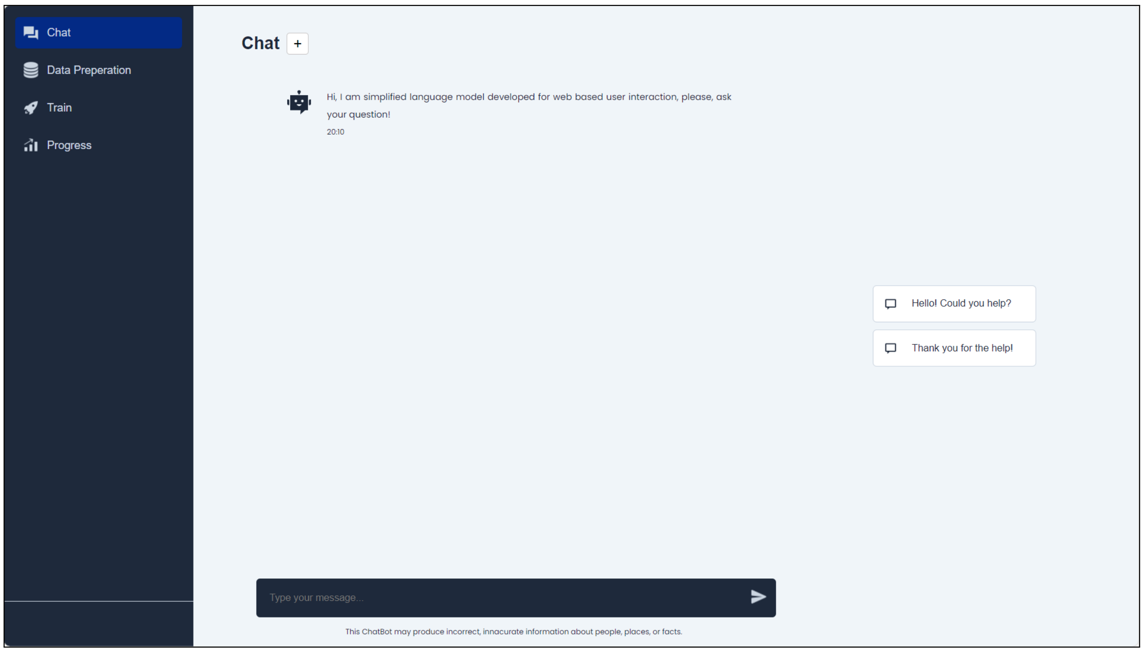
2. /**submit-form:** Handles the submission of form data from the Training Page. This route updates model configurations and hyperparameters stored in ‘`config.json`‘, preparing the system for training based on user-defined settings.

3. /**model-train:** Initiates the model training process. It triggers the ‘`training.py`‘ script, which uses configuration settings and prepared data to train the model, while also streaming back real-time logs to the Progress Page.

4. /**stream:** Facilitates real-time interaction with the trained model on the Chat Page. This route uses ‘`generate.py`‘ to process user queries and send back generated responses, ensuring a dynamic conversational experience.

2.5.1 Chat Page

The Chat page is designed to facilitate a seamless interaction between users and the Simplified Transformer-Based Language Model. This interface serves as the central hub for communication, allowing users to engage directly with the trained language model in a conversational manner.

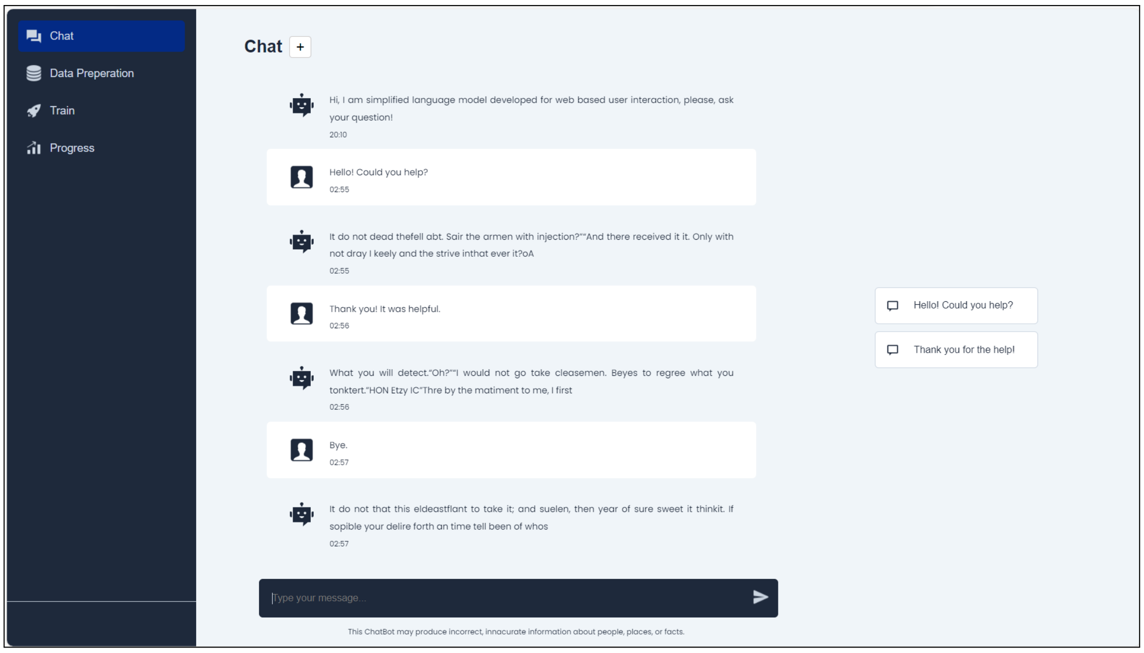


Hi, I am simplified language model developed for web based user interaction, please, ask your question!
20:10

This ChatBot may produce incorrect, innacurate information about people, places, or facts.

Figure 2.1: Chat page initially

It exemplifies the practical application of the trained model, showcasing its ability to understand and respond to user inputs in real time. The Chat window Figure 2.2 displays the ongoing conversation between the user and the model. It allows users to follow the dialogue flow, providing a continuous, context-aware interaction that enhances the user experience.



Hi, I am simplified language model developed for web based user interaction, please, ask your question!
20:10

Hello! Could you help?
02:55

It do not dead thefell abt. Sair the armen with injection?""And there received it it. Only with not dray I keely and the strive inthat ever it?oA
02:55

Thank you! It was helpful.
02:56

What you will detect."Oh?""I would not go take cleasemen. Beyes to regree what you tonktert."HON Etzy IC"Thre by the matiment to me, I first
02:56

Bye.
02:57

It do not that this eldeastflant to take it; and suelen, then year of sure sweet it thinkit. If sopible your delire forth an time tell been of whos
02:57

This ChatBot may produce incorrect, innacurate information about people, places, or facts.

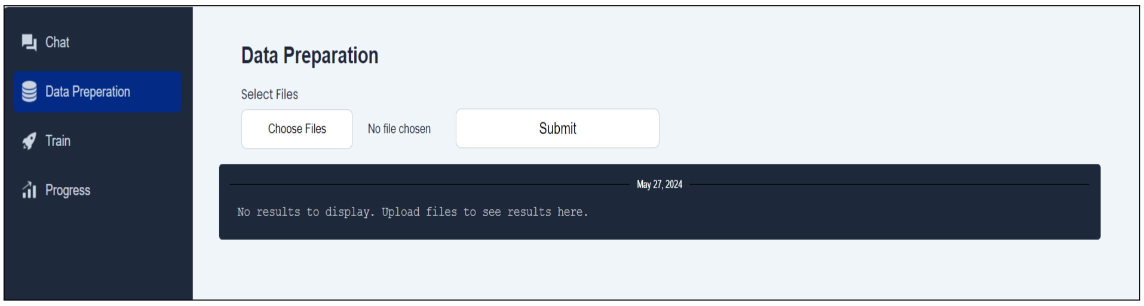
Figure 2.2: Chat page during the conversation

To enhance the model's understanding and the relevance of its responses, users should aim to use clear and direct language in their queries

2.5.2 Data Preparation Page

The Data Preparation page is an essential component of the application, designed to help users easily upload and prepare their text data for the training process. This page supports the initialization of datasets by dividing them into training and validation sets and extracting necessary features such as unique characters and bigrams.

When users first navigate to the Data Preparation page, they are presented with an option to upload their dataset files, which should be in a *.txt* format Figure 2.3.

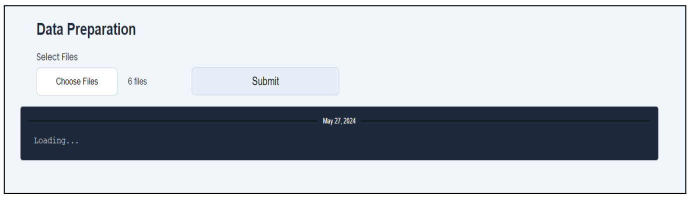


May 27, 2024

No results to display. Upload files to see results here.

Figure 2.3: Data Preparation Page Initially

Once files are submitted, the application processes the data, which includes splitting the dataset into training and validation sets and extracting text features. The process may take some time, during which a loading indicator is displayed Figure 2.4.

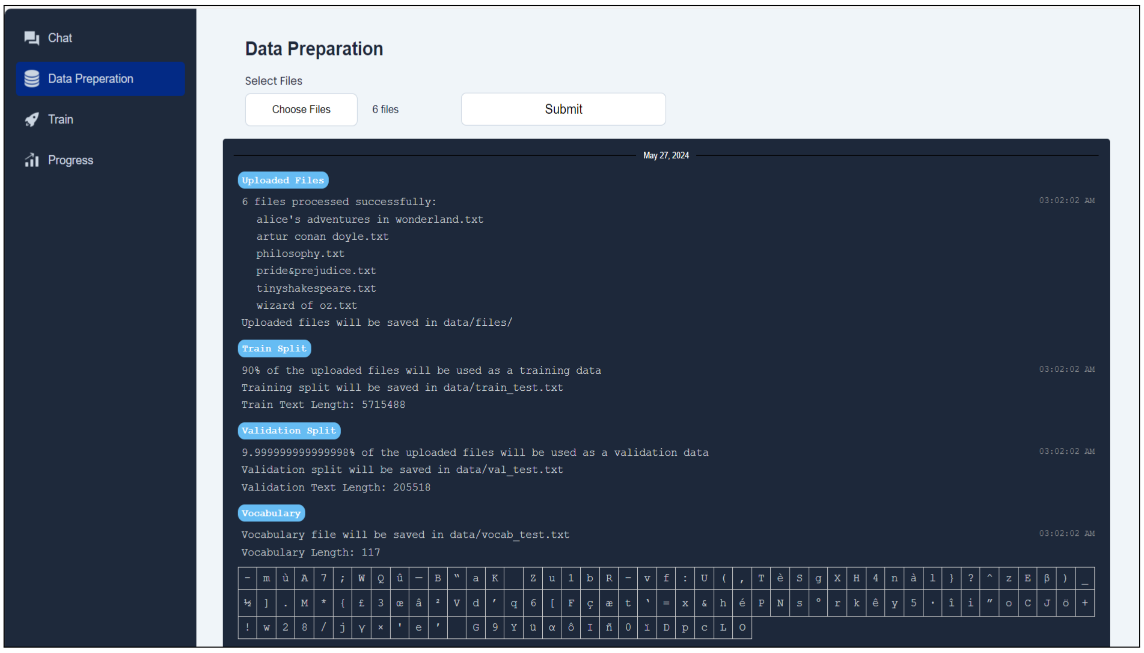


```
Loading...
```

Figure 2.4: Data Preperation Page Loading

After processing, the page displays detailed information about the processed data. This includes:

- **Training and Validation Split:** 90% of the data is used for training, and the remaining 10% for validation Figure 2.5.
- **Unique Characters:** The application identifies and lists all unique characters found in the dataset, which are crucial for later stages of model training Figure 2.5.

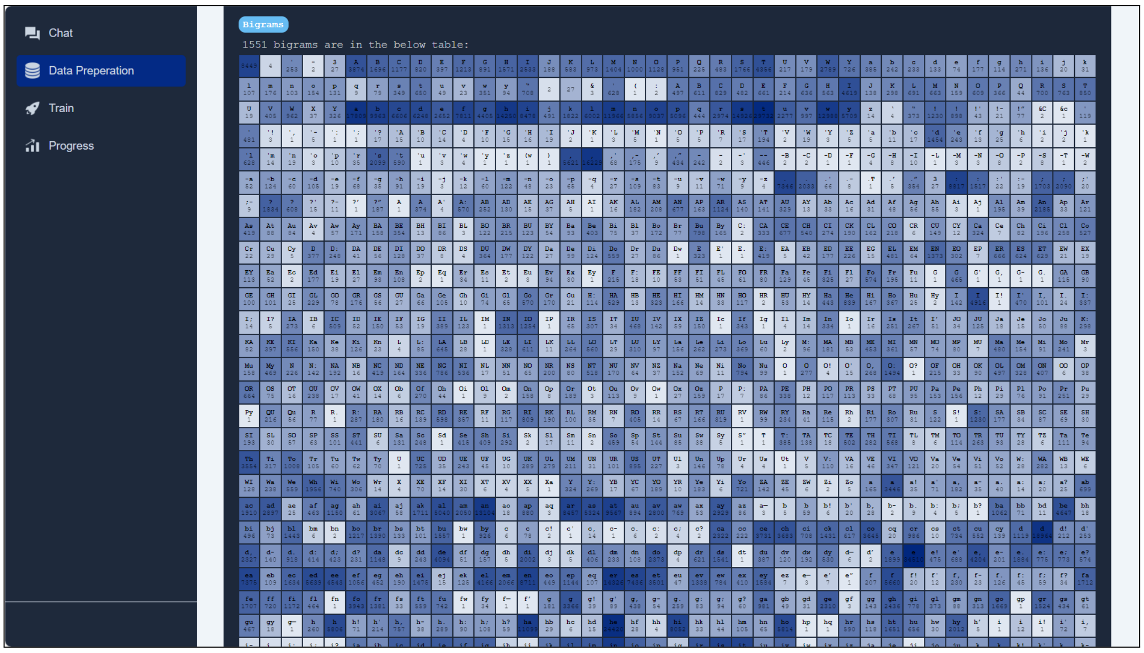


```
Uploaded Files
6 files processed successfully:                                        03:02:02 AM
  alice's adventures in wonderland.txt
  artur conan doyle.txt
  philosophy.txt
  pride&prejudice.txt
  tinyshakespeare.txt
  wizard of oz.txt
Uploaded files will be saved in data/files/
Train Split
90% of the uploaded files will be used as a training data              03:02:02 AM
Training split will be saved in data/train_test.txt
Train Text Length: 5715488
Validation Split
9.999999999999998% of the uploaded files will be used as a validation data   03:02:02 AM
Validation split will be saved in data/val_test.txt
Validation Text Length: 205518
Vocabulary
Vocabulary file will be saved in data/vocab_test.txt                   03:02:02 AM
Vocabulary Length: 117
```

-	m	ù	A	7	;	W	Q	û	—	B	"	a	K		Z	u	1	b	R	–	v	f	:	U	(	,	T	è	S	g	X	H	4	n	à	l	}	?	^	z	E	β	)	_
½	]	.	M	*	{	£	3	œ	â	²	V	d	’	q	6	[	F	ç	æ	t	`	=	x	&	h	é	P	N	s	°	r	k	ê	y	5	·	î	i	″	o	C	J	ö	+
!	w	2	8	/	j	γ	×	'	e	’		G	9	Y	ü	α	ô	I	ñ	0	ï	D	p	c	L	O																		

Figure 2.5: Data Preparation Page After dataset submitted (a)

- **Bigrams Visualization:** A comprehensive display of bigram frequencies within the dataset helps users understand common character combinations Figure 2.6.



1551 bigrams are in the below table:

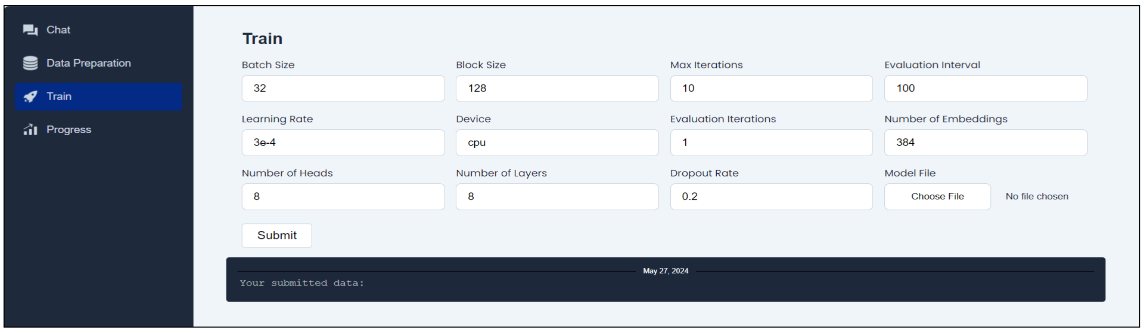
Figure 2.6: Data Preparation Page After dataset submitted (b)

The Data Preparation page automates crucial steps needed for preparing text data for further model training. By handling file uploads, dataset splitting, and feature extraction, it ensures that users can efficiently transition to training their models with well-prepared data.

2.5.3 Train Page

The Train page of the application is crucial for setting up the parameters under which the model will be trained. Here, users can adjust hyperparameters, which are critical in defining how the model learns from the data.

Users are initially presented with a form Figure 2.7 containing various fields for hyperparameter settings. Default values are set based on previously determined optimal settings. The interface provides an option to upload a pre-trained model file *.pk1*, allowing for training continuation or further refinement. If no file is selected, the system uses a default model or creates a new one.



Train

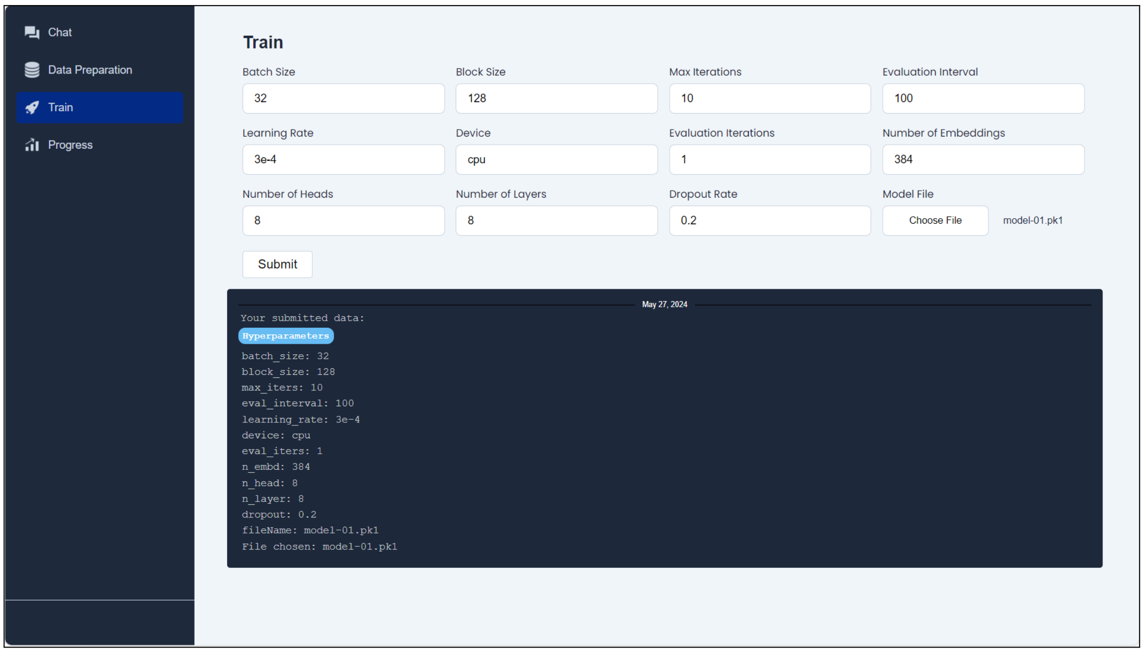
Batch Size	Block Size	Max Iterations	Evaluation Interval
32	128	10	100
Learning Rate	Device	Evaluation Iterations	Number of Embeddings
3e-4	cpu	1	384
Number of Heads	Number of Layers	Dropout Rate	Model File
8	8	0.2	Choose File No file chosen

May 27, 2024

```
Your submitted data:
```

Figure 2.7: Train Page initially

Modify any of the hyperparameters as needed. If no changes are made, the system will proceed with the default settings and the system processes the inputs and provides an overview of the submitted data, which confirms the settings before the actual training begins Figure 2.8.



Train

Batch Size	Block Size	Max Iterations	Evaluation Interval
32	128	10	100
Learning Rate	Device	Evaluation Iterations	Number of Embeddings
3e-4	cpu	1	384
Number of Heads	Number of Layers	Dropout Rate	Model File
8	8	0.2	Choose File model-01.pk1

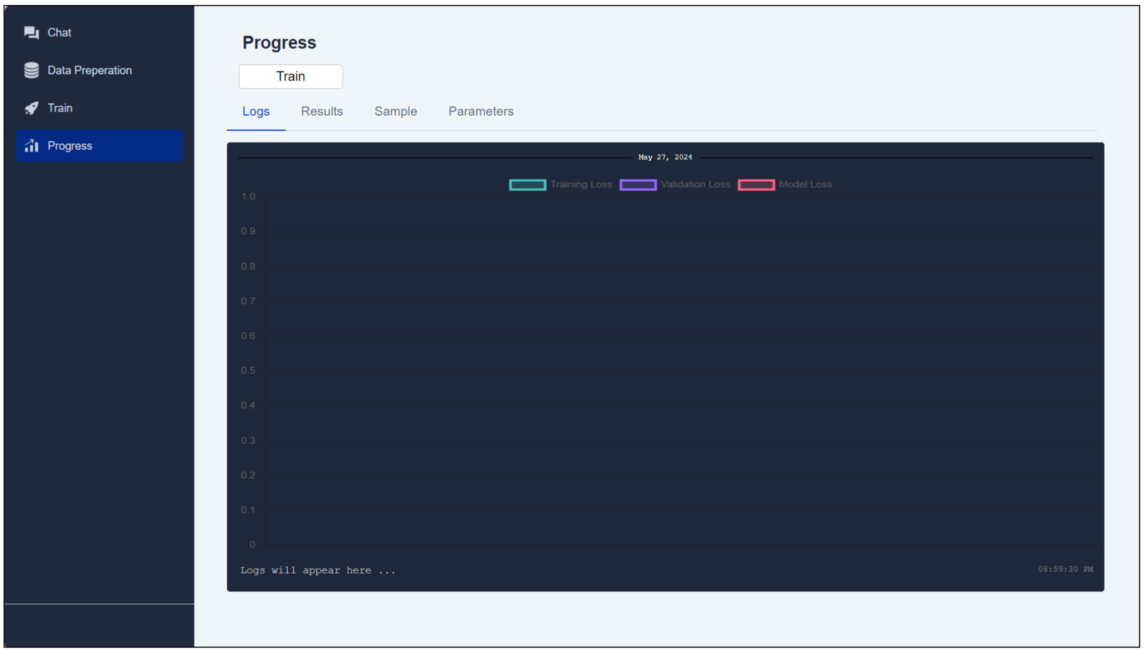
May 27, 2024

```
Your submitted data:
Hyperparameters
batch_size: 32
block_size: 128
max_iters: 10
eval_interval: 100
learning_rate: 3e-4
device: cpu
eval_iters: 1
n_embd: 384
n_head: 8
n_layer: 8
dropout: 0.2
fileName: model-01.pk1
File chosen: model-01.pk1
```

Figure 2.8: Train page after hyperparameters submitted

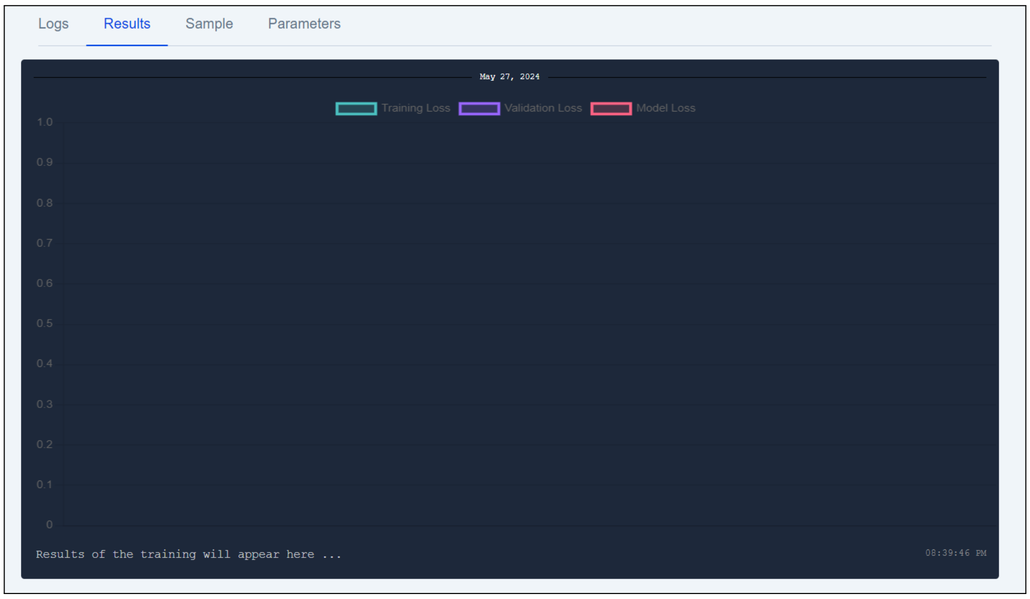
2.5.4 Progress Page

The Progress page is designed to monitor and display the results of the training process in real time. It is structured into several sections—Logs, Results, Sample, and Parameters—each offering specific insights into different aspects of the training cycle Figure 2.9.



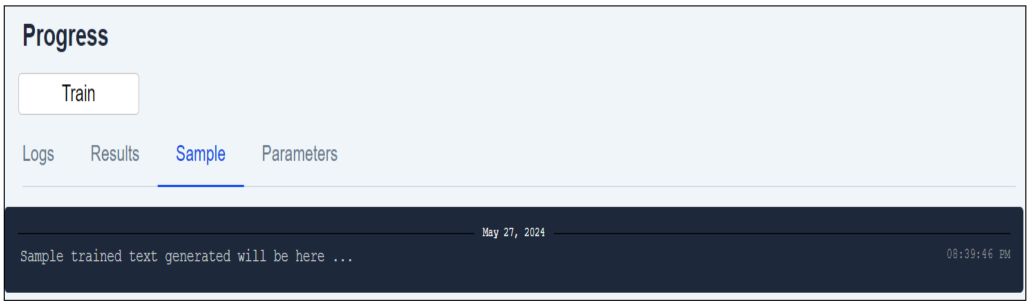
```
Logs will appear here ...                                    08:59:30 PM
```

Figure 2.9: Progress Page, Logs initially



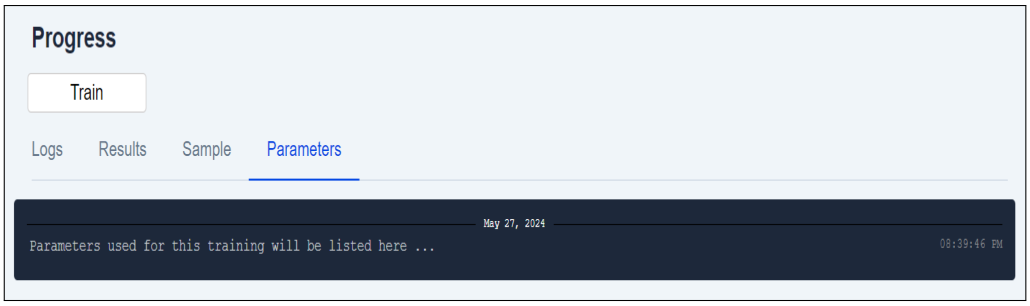
```
Results of the training will appear here ...                 08:39:46 PM
```

Figure 2.10: Progress Page, Results initially



```
Sample trained text generated will be here ...               08:39:46 PM
```

Figure 2.11: Progress Page, Samples initially



```
May 27, 2024
Parameters used for this training will be listed here ...                08:39:46 PM
```

Figure 2.12: Progress Page, Parameters initially

Functionality

1. **Starting the Training:** Upon clicking the *'Train'* button, the system initiates the training process using the data and hyperparameters set up in previous steps. This button activates the sequence of operations that train the model Figure 2.13.

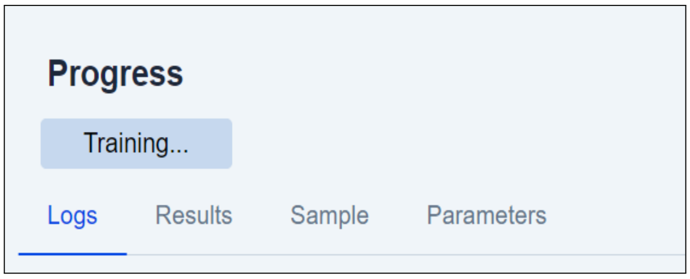
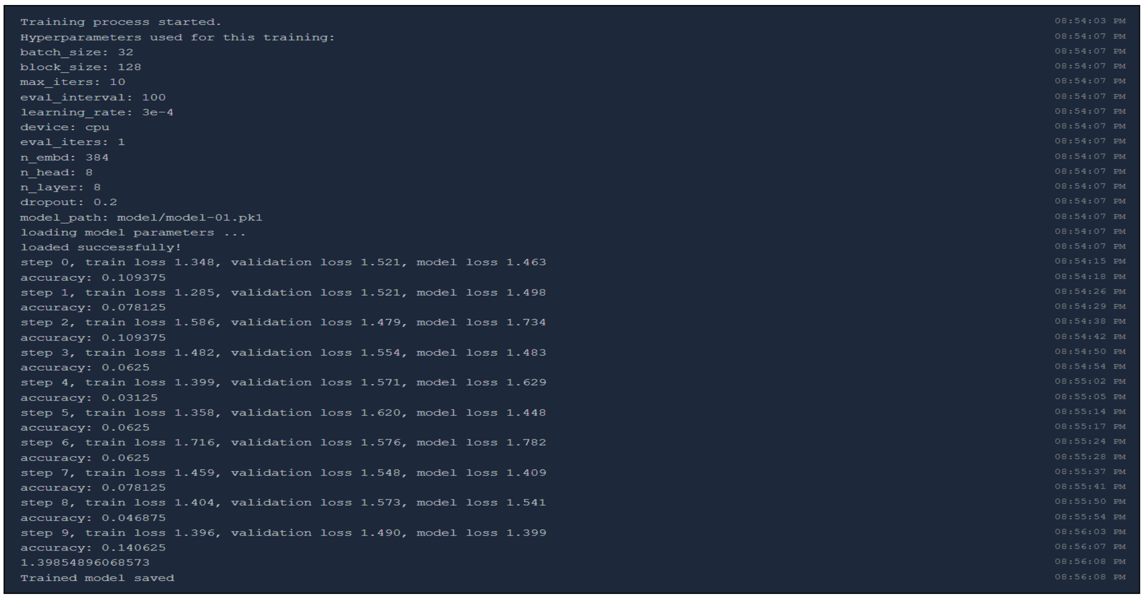


Figure 2.13: Training started

2. **Monitoring Progress:** As the training proceeds, users can watch the Logs tab for detailed progress reports. Each entry includes step-by-step updates on losses and accuracy, giving users a comprehensive view of the model's learning trajectory. The interface updates continuously, providing immediate feedback on the training's progress and outcomes. This setup is especially useful for lengthy training processes, allowing adjustments on the fly to optimize results.



```
Training process started.                                                   08:54:03 PM
Hyperparameters used for this training:                                     08:54:07 PM
batch_size: 32                                                              08:54:07 PM
block_size: 128                                                             08:54:07 PM
max_iters: 10                                                               08:54:07 PM
eval_interval: 100                                                          08:54:07 PM
learning_rate: 3e-4                                                         08:54:07 PM
device: cpu                                                                 08:54:07 PM
eval_iters: 1                                                               08:54:07 PM
n_embd: 384                                                                 08:54:07 PM
n_head: 8                                                                   08:54:07 PM
n_layer: 8                                                                  08:54:07 PM
dropout: 0.2                                                                08:54:07 PM
model_path: model/model-01.pkl                                              08:54:07 PM
loading model parameters ...                                                08:54:07 PM
loaded successfully!                                                        08:54:07 PM
step 0, train loss 1.348, validation loss 1.521, model loss 1.463           08:54:15 PM
accuracy: 0.109375                                                          08:54:18 PM
step 1, train loss 1.285, validation loss 1.521, model loss 1.498           08:54:26 PM
accuracy: 0.078125                                                          08:54:29 PM
step 2, train loss 1.586, validation loss 1.479, model loss 1.734           08:54:38 PM
accuracy: 0.109375                                                          08:54:42 PM
step 3, train loss 1.482, validation loss 1.554, model loss 1.483           08:54:50 PM
accuracy: 0.0625                                                            08:54:54 PM
step 4, train loss 1.399, validation loss 1.571, model loss 1.629           08:55:02 PM
accuracy: 0.03125                                                           08:55:05 PM
step 5, train loss 1.358, validation loss 1.620, model loss 1.448           08:55:14 PM
accuracy: 0.0625                                                            08:55:17 PM
step 6, train loss 1.716, validation loss 1.576, model loss 1.782           08:55:24 PM
accuracy: 0.0625                                                            08:55:28 PM
step 7, train loss 1.459, validation loss 1.548, model loss 1.409           08:55:37 PM
accuracy: 0.078125                                                          08:55:41 PM
step 8, train loss 1.404, validation loss 1.573, model loss 1.541           08:55:50 PM
accuracy: 0.046875                                                          08:55:54 PM
step 9, train loss 1.396, validation loss 1.490, model loss 1.399           08:56:03 PM
accuracy: 0.140625                                                          08:56:07 PM
1.398548960668573                                                           08:56:08 PM
Trained model saved                                                         08:56:08 PM
```

Figure 2.14: Logs after the training with 10 epochs

3. **Visualizing Results:** The Results tab presents a dynamic chart (Figure 2.15) that updates after each training iteration, showing trends in training loss, validation loss, and model loss (Figure 2.16). This visualization helps in quickly assessing the model's performance and adjusting parameters if needed. Additionally, it includes heatmaps (Figure **??**) of the confusion matrix for each iteration, visualizing how well the model predictions align with the actual values.

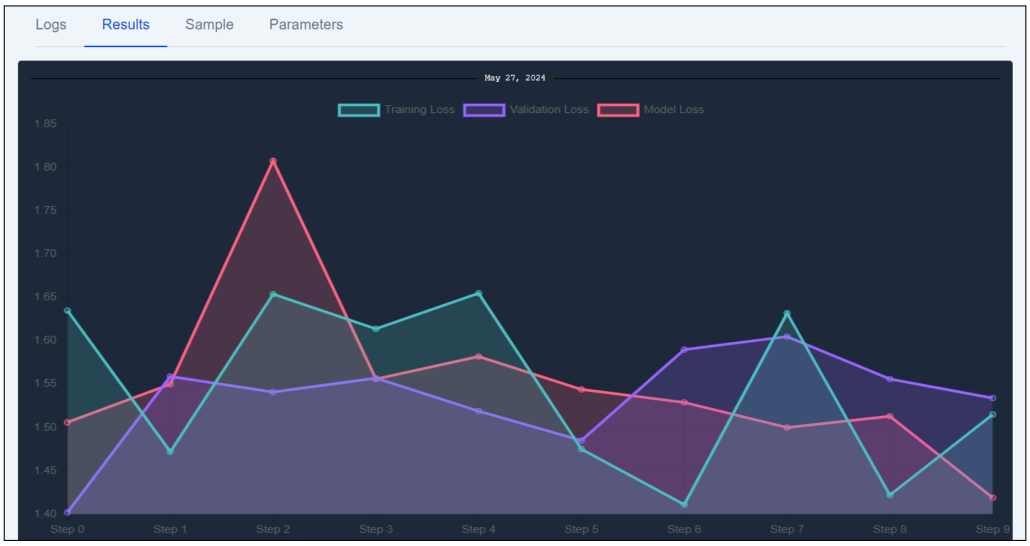
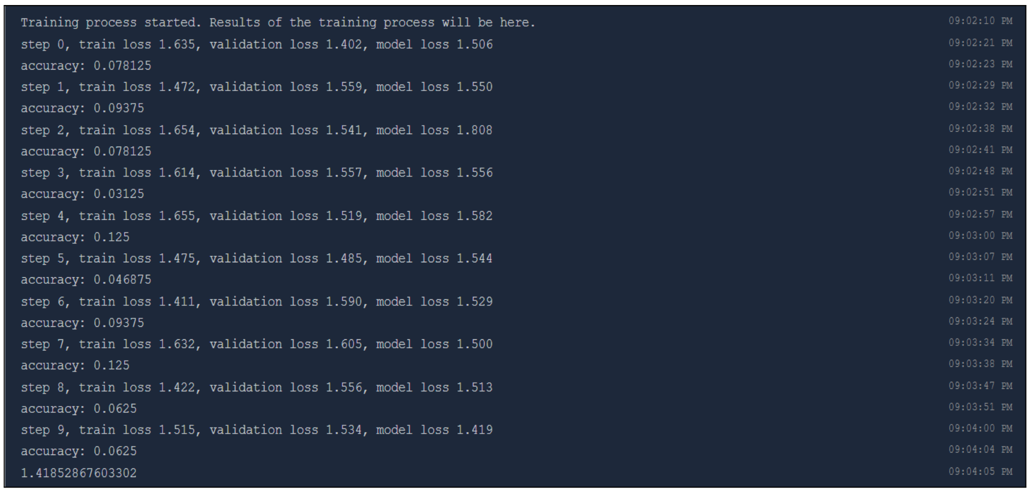
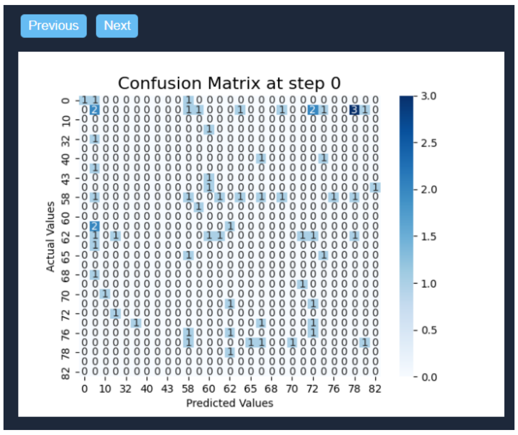


Figure 2.15: Loss chart after step 10

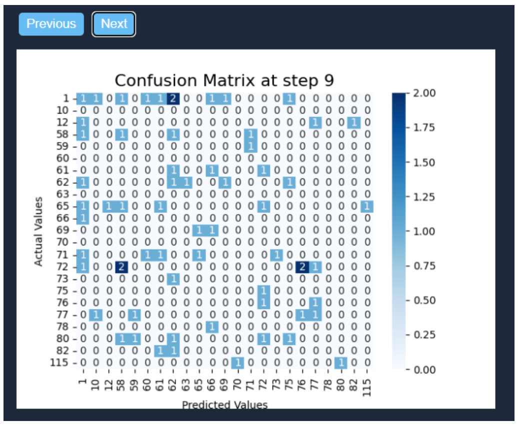


```
Training process started. Results of the training process will be here.      09:02:10 PM
step 0, train loss 1.635, validation loss 1.402, model loss 1.506             09:02:21 PM
accuracy: 0.078125                                                            09:02:23 PM
step 1, train loss 1.472, validation loss 1.559, model loss 1.550             09:02:29 PM
accuracy: 0.09375                                                             09:02:32 PM
step 2, train loss 1.654, validation loss 1.541, model loss 1.808             09:02:38 PM
accuracy: 0.078125                                                            09:02:41 PM
step 3, train loss 1.614, validation loss 1.557, model loss 1.556             09:02:48 PM
accuracy: 0.03125                                                             09:02:51 PM
step 4, train loss 1.655, validation loss 1.519, model loss 1.582             09:02:57 PM
accuracy: 0.125                                                               09:03:00 PM
step 5, train loss 1.475, validation loss 1.485, model loss 1.544             09:03:07 PM
accuracy: 0.046875                                                            09:03:11 PM
step 6, train loss 1.411, validation loss 1.590, model loss 1.529             09:03:20 PM
accuracy: 0.09375                                                             09:03:24 PM
step 7, train loss 1.632, validation loss 1.605, model loss 1.500             09:03:34 PM
accuracy: 0.125                                                               09:03:38 PM
step 8, train loss 1.422, validation loss 1.556, model loss 1.513             09:03:47 PM
accuracy: 0.0625                                                              09:03:51 PM
step 9, train loss 1.515, validation loss 1.534, model loss 1.419             09:04:00 PM
accuracy: 0.0625                                                              09:04:04 PM
1.41852867603302                                                              09:04:05 PM
```

Figure 2.16: Losses, accuracy after 10 epochs



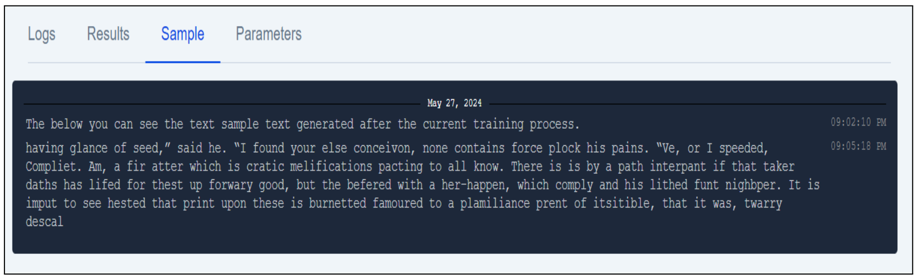
(a) Heatmap for Step 1



(b) Heatmap for Step 10

Figure 2.17: Heatmaps

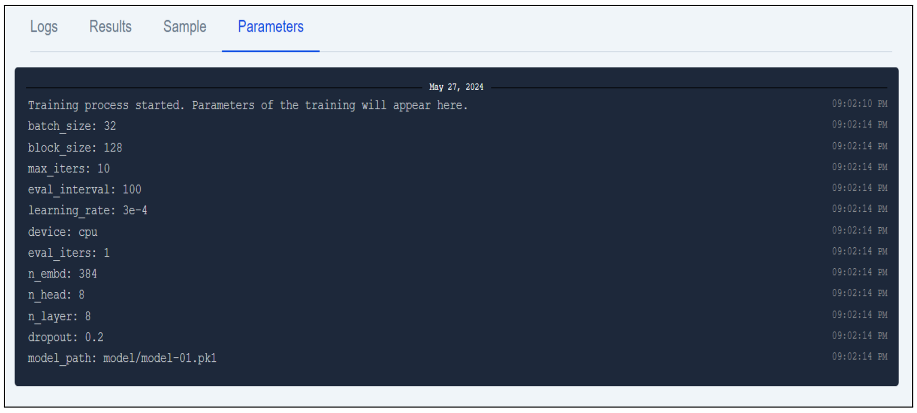
4. **Evaluating Model Output:** The Sample tab displays a snippet of text generated by the model, reflecting its ability to synthesize and contextualize information based on the training it received. This sample (Figure 2.18) is crucial for assessing the qualitative aspect of the model's learning.



```
May 27, 2024
The below you can see the text sample text generated after the current training process.   09:02:10 PM
having glance of seed," said he. "I found your else conceivon, none contains force plock his pains. "Ve, or I speeded,   09:05:18 PM
Compliet. Am, a fir atter which is cratic melifications pacting to all know. There is is by a path interpant if that taker
daths has lifed for thest up forwary good, but the befered with a her-happen, which comply and his lithed funt nighbper. It is
imput to see hested that print upon these is burnetted famoured to a plamiliance prent of itsitible, that it was, twarry
descal
```

Figure 2.18: Sample text generated after the training

5. **Reviewing Parameters:** The Parameters (Figure 2.19) tab reiterates the hyperparameters applied during the training, ensuring that all settings are documented and can be replicated or revised in future training sessions.



```
                                   May 27, 2024
Training process started. Parameters of the training will appear here.    09:02:10 PM
batch_size: 32                                                            09:02:14 PM
block_size: 128                                                           09:02:14 PM
max_iters: 10                                                             09:02:14 PM
eval_interval: 100                                                        09:02:14 PM
learning_rate: 3e-4                                                       09:02:14 PM
device: cpu                                                               09:02:14 PM
eval_iters: 1                                                             09:02:14 PM
n_embd: 384                                                               09:02:14 PM
n_head: 8                                                                 09:02:14 PM
n_layer: 8                                                                09:02:14 PM
dropout: 0.2                                                              09:02:14 PM
model_path: model/model-01.pkl                                            09:02:14 PM
```

Figure 2.19: Sample text generated after the training

This section has walked you through each key component of the Simplified Transformer-Based Language Model application. From initial setup and data preparation to training the model and monitoring its progress, the guidance provided aims to ensure that you can effectively utilize the application to achieve your machine learning objectives.

The application is designed to be user-friendly, allowing both novices and experienced users to navigate and use the features effortlessly.

Through the configuration of hyperparameters and the ability to upload specific datasets, the application offers flexibility to tailor the training process to meet diverse needs and requirements.

The Progress page provides real-time feedback and detailed insights into the training process, enabling users to make informed decisions to optimize model performance.

As you continue to use the application, we encourage you to experiment with different datasets and hyperparameter settings to explore the full potential of the model. The tools provided are designed to help you learn and adapt the model to increasingly complex tasks.

Chapter 3

Developer documentation

This developer documentation is designed to provide a comprehensive guide to the design, implementation, and testing of a simplified transformer-based architecture for enhancing web-based user interactions. This project leverages cutting-edge technologies, including PyTorch [6] for machine learning model development, Flask [7] for backend operations, and React [8] for a dynamic front-end user interface, to create a streamlined conversational AI system.

The purpose of this documentation is to detail the architectural choices, the process of implementation, and the strategies employed in testing the application.

3.1 Design

This section of the developer documentation delves into the architectural blueprint of the simplified transformer-based system designed for enhancing web-based user interactions. It outlines the structural and conceptual designs underpinning the project, providing a comprehensive overview of how each component functions and interacts within the system. Detailed diagrams, including system architecture, component relationships, and data flow diagrams, are included to visually support the descriptions. The design decisions, made to optimize both functionality and efficiency, are thoroughly explained to offer insights into the strategic choices that drive the project. This section aims to articulate the foundational design concepts that guided the development of the transformer-based model and its integration with a web interface, ensuring clarity and coherence in the system's theoretical framework.

3.1.1 System Architecture

This section provides a comprehensive overview of the system architecture for the web-based language model platform. The architecture is designed to handle interactions from data upload and processing to training the model and engaging with it through a conversational interface. Each component of the system—React frontend [8], Flask backend [7], and the PyTorch [6] model — is detailed below, illustrating their interconnected roles and the flow of data across the platform.

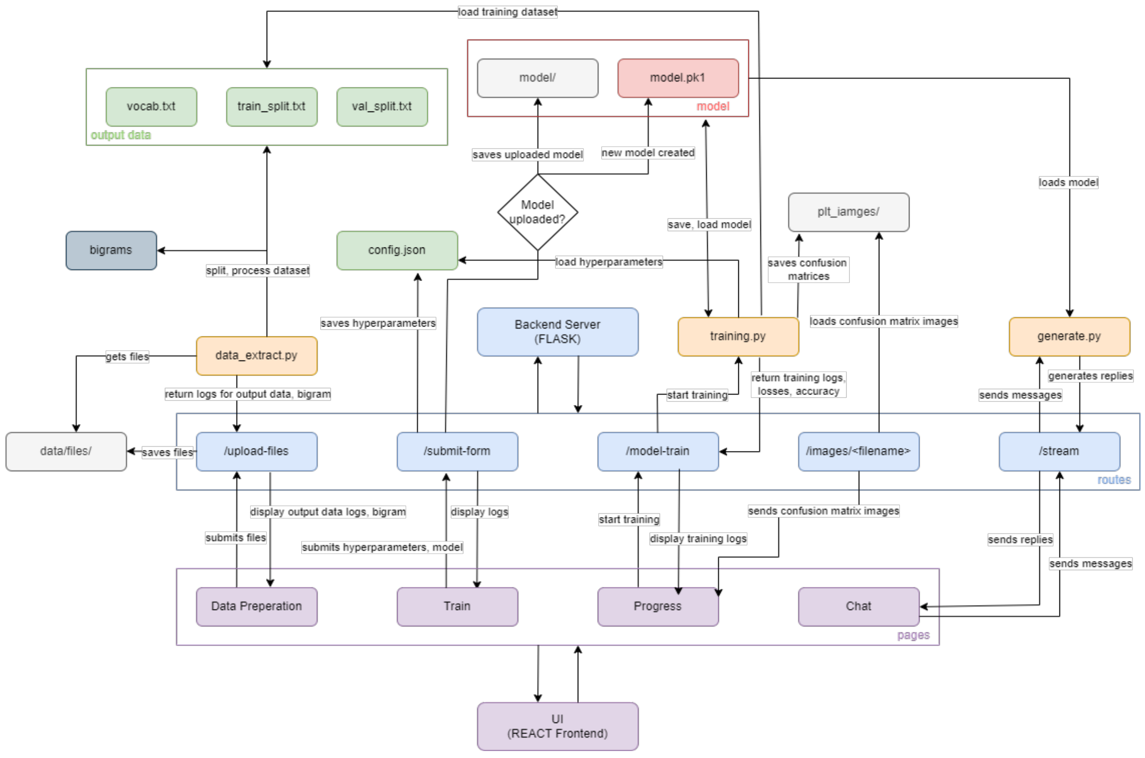


Figure 3.1: Detailed Architecture Diagram

Figure 3.1 visualizes the entire architecture of the platform. This diagram demonstrates the interactions between the frontend interface, the backend server, and the machine learning model, detailing how they work together to provide a seamless user experience.

3.1.2 Detailed Component Analysis

In this section, we explore the roles and functionalities of our system's core components: React [8] for the frontend, Flask [7] for the backend, and PyTorch [6] for the model. Each technology was carefully chosen to enhance performance, scalability, and development efficiency, ensuring the system effectively manages everything from user interactions to complex data processing and model operations. We will discuss why these particular technologies were selected and how they contribute to the platform's overall functionality.

Frontend (React)

React is utilized for the frontend to manage the dynamic user interface requirements efficiently. Its component-based architecture facilitates the rapid development of interactive elements, enhancing the user's ability to manage data uploads and model interactions seamlessly.

The React-based frontend of our application is designed for user-friendly interaction, divided into four main pages. Each page serves a distinct function in facilitating the user's journey from data preparation to interacting with the trained model.

1. **Chat Page:** The Chat Page is the interface where users can engage in conversations with the trained model. This page mimics a chat environment, similar to platforms like ChatGPT, allowing users to input queries and receive responses in real-time, showcasing the model's conversational capabilities.

2. **Data Preparation Page:** On the Data Preparation Page, users can upload their textual data files which are then processed for model training. This page handles the initial steps of data curation, including the segmentation into training and validation splits, and visualization of data statistics like bigrams.

3. **Train Page:** The Training Page allows users to configure and initiate the training process. Users can select hyperparameters, choose to improve an existing model or train a new one from scratch. This page provides options for customization of the model training to suit user preferences or requirements.

4. **Progress Page:** This page displays real-time updates on the training process, including metrics such as loss and accuracy, as well as visual outputs like loss graphs and confusion matrices. It serves as a dashboard for users to monitor and evaluate the model's training progress.

Backend (Flask)

Flask offers the flexibility needed for quick development iterations and is robust enough to handle API requests, manage session data, and serve as the intermediary for all communications between the frontend and the model, ensuring smooth data flow and processing.

The Flask backend is crucial for handling server-side logic, managing data flow, and linking the frontend with the machine learning model. It features several API routes that facilitate various functionalities of the system.

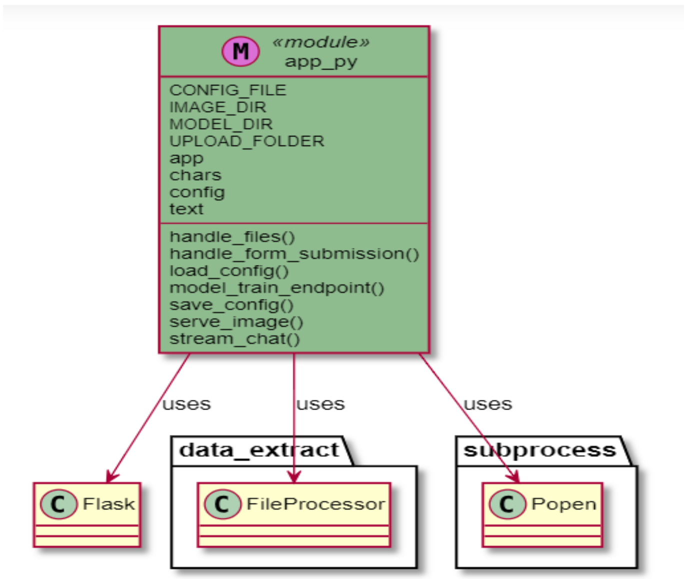


Figure 3.2: FLask Architecture

Model (PyTorch)

PyTorch [6] provides an intuitive framework for building and training the transformer-based model, benefiting from its dynamic computation graph, comprehensive library support, and efficient integration with the Python data ecosystem, crucial for managing complex data processing tasks.

The Transformer Architecture

The transformer architecture is the fundamental building block of all Language Models with Transformers (LLMs). The transformer architecture was introduced in the paper “Attention is all you need,” [9] published in December 2017. The simplified version of the Transformer Architecture looks like this:

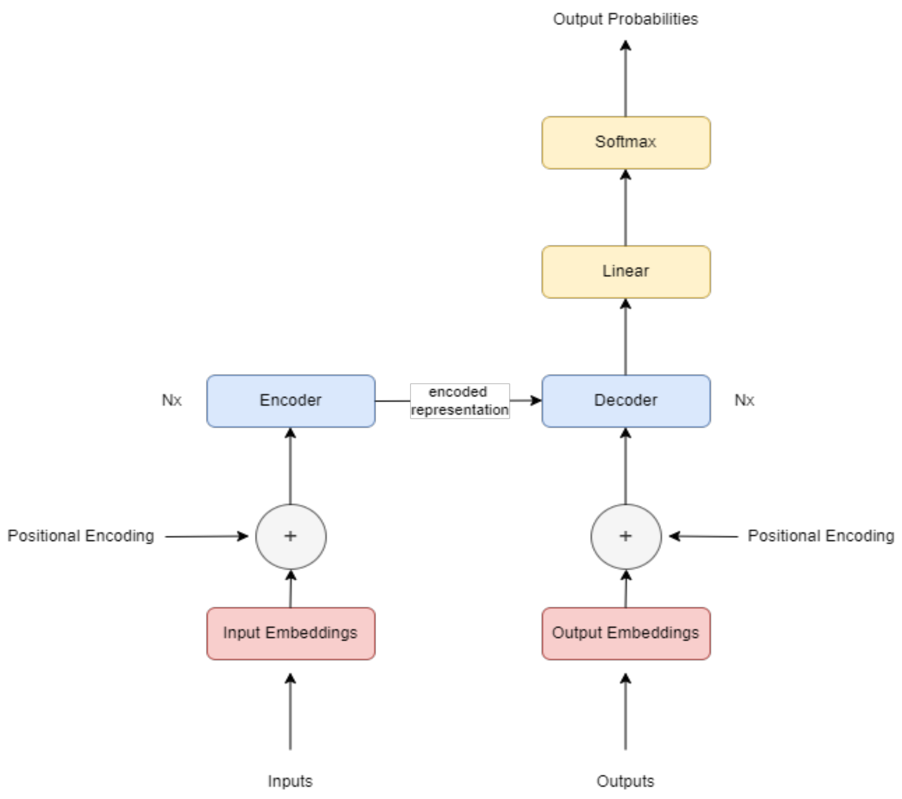


Figure 3.3: Transformer Architecture

There are several important components in transformer architecture as you can see in the above diagram. And these main components are built with the help of PyTorch [6] in our application.

Detailed Description of Transformer Components

Head Class:

The Head class is a fundamental component of the MultiHeadAttention mechanism in our transformer model. It represents a single attention head.

Key, Query, and Value Projections: Each head computes three vectors—key, query, and value—from the input embeddings. These vectors are crucial for determining the attention each element in the input sequence should pay to all other elements.

Attention Weights Calculation: Using the query and key, the head calculates attention scores that determine how much each part of the input sequence contributes to the output. This is adjusted by scaling with the square root of the dimension of the key vectors, a technique to stabilize the gradients during training.

Masking and Softmax: The scores are then masked (to exclude unwanted attention, such as to padding in sequences) and normalized using softmax to ensure they sum to one.

Output Computation: Finally, the attention weights are used to compute a weighted sum of the value vectors, which becomes the output of the head. The dropout applied after computing the softmax probabilities helps prevent the model from becoming overly reliant on specific paths, enhancing its generalization capabilities.

Class MultiHeadAttention:

The MultiHeadAttention class manages multiple attention heads, combining their outputs to allow the model to jointly attend to information from different representation subspaces.

Parallel Attention Heads: It runs several Head instances in parallel, each providing a unique contribution to the final output, allowing the model to capture a richer array of information.

Concatenation and Projection: After obtaining outputs from each head, the results are concatenated and linearly transformed to produce the final output of the multi-head attention block. This class leverages the diversity of attention mechanisms to enhance the model's ability to focus on various parts of the input data differently, which is particularly beneficial in tasks involving complex dependencies.

Class FeedForward:

The FeedForward class represents a position-wise feedforward neural network, which is applied to each position separately and identically.

Layer Structure: Consists of two linear transformations with a ReLU activation in between. The first linear layer expands the input dimension, allowing the network to represent more complex patterns.

Non-linearity: The ReLU activation introduces non-linear capabilities to the model, enabling it to learn more complex functions.

Dropout: Includes dropout for regularization, which helps in preventing overfitting during training. The expansion and subsequent compression of dimensions allow the network to mix information at each position effectively, essential for transforming the representation after aggregating information through attention.

Class Block:

The Block class encapsulates a full block of the transformer architecture, containing a multi-head attention layer followed by a position-wise feedforward network.

Layer Normalization: Each sub-layer (attention and feedforward) in a block is followed by layer normalization, which helps stabilize the training process by normalizing the layer outputs.

Residual Connections: Implements residual connections around each of the two sub-layers, facilitating deeper networks by allowing gradients to flow through the network directly. Residual connections help in mitigating the vanishing gradient problem in deep networks, enabling the model to learn effectively even with many layers.

In the following diagram you can see the detailed interaction of all components:

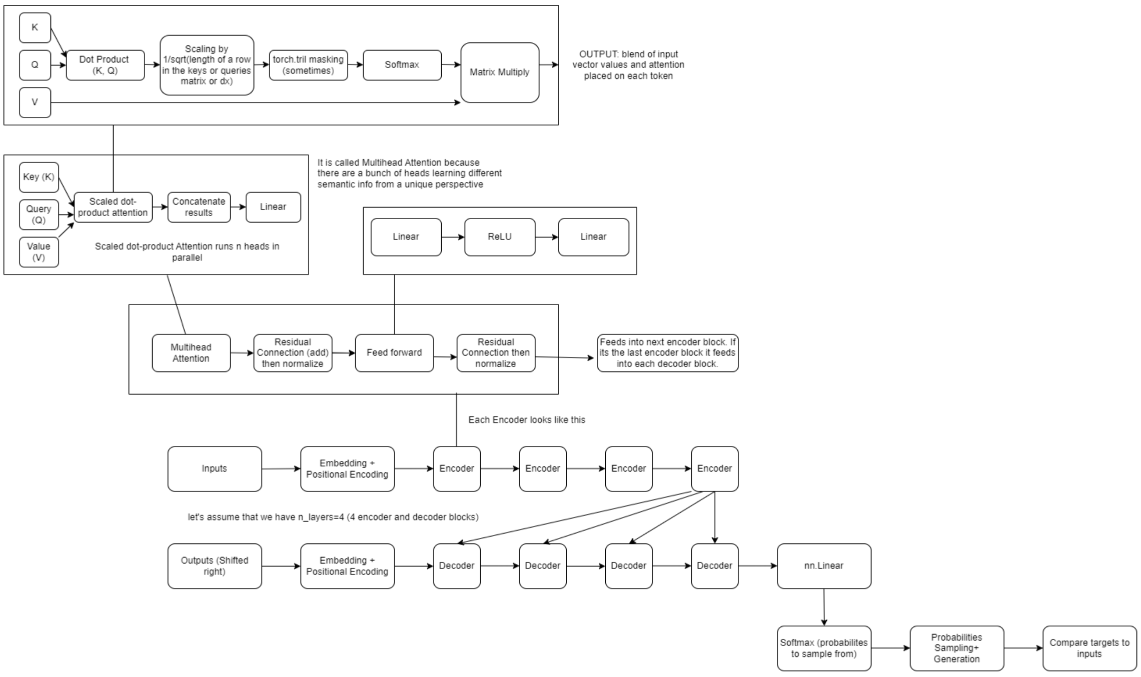

K
Q
V
Dot Product (K, Q)
Scaling by 1/sqrt(length of a row in the keys or queries matrix or dx)
torch.tril masking (sometimes)
Softmax
Matrix Multiply
OUTPUT: blend of input vector values and attention placed on each token
Key (K)
Query (Q)
Value (V)
Scaled dot-product attention
Concatenate results
Linear
Scaled dot-product Attention runs n heads in parallel
It is called Multihead Attention because there are a bunch of heads learning different semantic info from a unique perspective
Linear
ReLU
Linear
Multihead Attention
Residual Connection (add) then normalize
Feed forward
Residual Connection then normalize
Feeds into next encoder block. If its the last encoder block it feeds into each decoder block.
Each Encoder looks like this
Inputs
Embedding + Positional Encoding
Encoder
Encoder
Encoder
Encoder
let's assume that we have n_layers=4 (4 encoder and decoder blocks)
Outputs (Shifted right)
Embedding + Positional Encoding
Decoder
Decoder
Decoder
Decoder
nn.Linear
Softmax (probabilites to sample from)
Probabilities Sampling+ Generation
Compare targets to inputs

Figure 3.4: Transformer Architecture

Model Persistence (‘`model.pk1`‘)

The ‘`model.pk1`‘ file plays a key role in our system as the serialized form of the trained model. This file stores the state of the neural network, including all parameters and learned weights, enabling persistence and reusability of the model.

Post-training, the model is serialized into ‘`model.pk1`‘ using Python’s ‘`pickle`‘ [10] module, which allows us to save the model’s state efficiently. This file can be reloaded to resume training, conduct further improvements, or directly for inference on the Chat Page without needing to retrain from scratch.

3.1.3 Data Flow and Integration

This section outlines the data lifecycle within our system, from initial input through processing to output generation. It details the roles of various scripts and components in managing data flow and transforming inputs into actionable outputs.

Data Processing in Backend (‘`data-extract.py`‘)

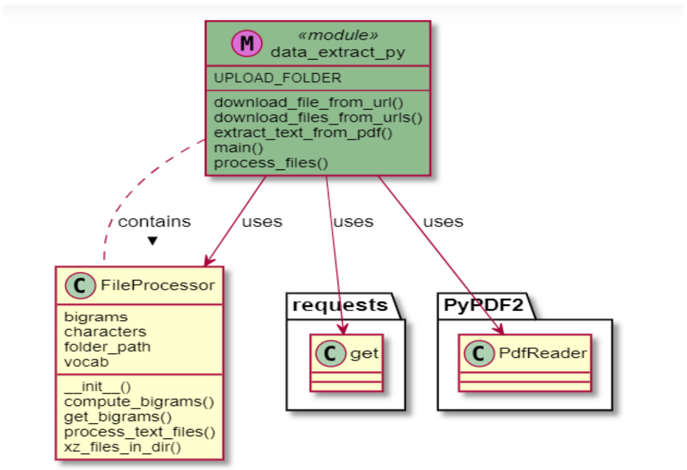


Figure 3.5: Data Processing Architecture

The ‘`data-extract.py`‘ script is critical in the initial stages of data handling. It processes user-uploaded text files by validating and preparing them for model training. This involves parsing the files, extracting and cleaning text, and splitting data into training and validation datasets. It also constructs the vocabulary that the model will use, ensuring that all inputs are tokenized effectively for training.

Model Training Dynamics (‘`training.py`‘)

‘`training.py`‘ manages the model’s training process. It loads processed data and model configurations, orchestrating the training loop. This script handles batch processing, computes loss, updates model parameters, and logs training metrics such as accuracy, precision, and recall. The script ensures that the model training is not only efficient but also transparent, allowing real-time tracking of the model's performance.

Response Generation and Interaction (‘generate.py‘)

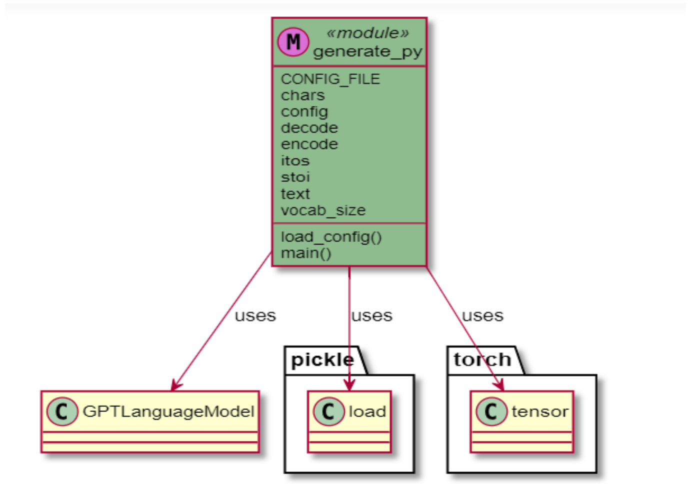


Figure 3.6: Response Generation Architecture

The ‘generate.py‘ script is utilized during the interaction phase, where the trained model generates responses based on user inputs. This script loads the trained model, processes incoming queries by converting them into a suitable format, and uses the model to predict responses. The generated outputs are then formatted back into human-readable text, facilitating real-time conversations with users on the platform.

Understanding the data flow within our system clarifies how each component and script contributes to the overall functionality. From data preparation to interactive response generation, each step is crucial for maintaining the integrity and efficiency of the model training and user interaction processes.

3.1.4 Scalability and Flexibility

This thesis introduces a basic transformer-based architecture intended to demonstrate the principles of language model training and interaction. While the design theoretically supports scalability and flexibility, practical implementations to actualize these aspects have not been developed due to the project's educational and demonstrative focus.

Scalability Concerns:

The application was developed with a focus on functionality and educational value rather than on handling large-scale operations or rapid adaptability. Given the project's scope, the current system does not include an automated scaling solution, which could lead to performance bottlenecks under increased loads. While the architecture was conceptualized with modular components in both the frontend and backend, allowing for independent scaling, this has not been fully realized in practice.

Flexibility Limitations

Despite the intention to allow for dynamic updates and modifications through a service-oriented architecture and configurable parameters, the actual system does not fully support on-the-fly changes without potential disruptions.

3.1.5 Security and Limitations

As of now, the application does not include specific security measures. The focus has primarily been on building a functional prototype to illustrate the workings of a simplified language model. This absence of security features means the application should not be used with sensitive or personal data under any circumstances.

The lack of security measures is a significant oversight, given the importance of data protection and system integrity in today's digital environment.

Areas for improvement

This section outlines the known limitations and faults of our system, underscoring areas where the design and implementation fell short of professional standards or did not completely meet the intended goals.

Scalability: The current deployment setup is static and not designed to dynamically scale; it serves only the minimum required load.

Dependency on Third-Party Libraries: The system relies on third-party libraries, which can introduce vulnerabilities and bugs that are out of our control. This dependency also means that updates to these libraries can potentially break existing functionalities. This reliance is acknowledged as a potential source of security and operational risks but was deemed acceptable for the educational intent of the project.

Limited Testing and Lack of Responsiveness: The testing focused only on basic functionality, and the UI was not optimized for mobile, different size devices, reflecting the project's limited use-case and development resources.

This section acknowledges the limitations in the design and implementation of the project. While these limitations are significant from a professional software development perspective, they are consistent with the educational and demonstrative objectives of this thesis. This honest evaluation highlights areas for potential enhancement should the project scope expand beyond its current educational framework.

3.2 Implementation

The implementation section details the technical specifics of how the application's features and components were developed. It focuses on the practical application of the design principles outlined in earlier sections and describes the technologies, frameworks, and coding practices employed to build the application. This section also provides code snippets, configurations, and descriptions of the system architecture that support the functionality of the application.

3.2.1 Data Processing

The Data Processing module is central to preparing raw text data for training our language model. This module handles the downloading, cleaning, and organization of `‘.txt‘` files into a structured format that is suitable for efficient model training. Implemented in Python [1], it leverages a variety of libraries to manage and process textual data effectively.

In the initial phases of the project, we utilized a large dataset of .xz compressed files. However, the decompression and processing of these files were extremely time-consuming, leading us to optimize our approach by shifting to plain text .txt files and focusing on a selective set of vocabulary to enhance performance.

Training and Validation Split

To ensure the robustness of our model, we implemented a systematic split of the data into training and validation sets (Code 3.1). This strategy is crucial for evaluating the model's performance accurately and avoiding overfitting.

```
def process_text_files(self, output_file_train, output_file_val,
    vocab_file, train_file_percentage):
     files = [f for f in os.listdir(self.folder_path) if f.endswith
         (".txt")]
     total_files = len(files)
     split_index = int(total_files * train_file_percentage)  # 90%
         for training
     files_train = files[:split_index]
     files_val = files[split_index:]

     with open(output_file_train, "w", encoding="utf-8") as
         outfile_train, \
```

```python
         open(output_file_val, "w", encoding="utf-8") as outfile_val:
        for i, filename in enumerate(files):
            file_path = os.path.join(self.folder_path, filename)
            with open(file_path, "r", encoding="utf-8") as infile:
                text = infile.read()
                if i < split_index:
                    outfile_train.write(text)
                else:
                    outfile_val.write(text)
                self.vocab.update(set(text))
```

Code 3.1: Split provided dataset into training, validation splits

Vocabulary Set Creation and Bigram Computing

Alongside optimizing file handling, we introduced a curated vocabulary set to limit the scope of our model, enhancing both memory management and processing speed. Additionally, we compute bigrams from the text to aid in understanding language patterns and improving predictive accuracy.

```python
def compute_bigrams(self):
    characters_set = set(self.characters)
    N = torch.zeros((len(characters_set), len(characters_set)), dtype=torch.int32)

    for ch1, ch2 in zip(self.characters, self.characters[1:]):
        ix1, ix2 = map(self.bigram_stoi.get, (ch1, ch2))
        N[ix1, ix2] += 1

    self.bigrams = {(self.bigram_itos[i], self.bigram_itos[j]): N[i, j].item()
                    for i in range(len(characters_set)) for j in range(len(characters_set))}

    # Saving the vocabulary and bigrams
    with open(vocab_file, "w", encoding="utf-8") as vfile:
        for char in sorted(self.vocab):
            vfile.write(char + "\n")
```

Code 3.2: Vocabulary Set Creation and Bigram Computing

3.2.2 Hyperparameters Setup

Effective tuning of hyperparameters is critical to optimizing the performance of neural networks. In this project, the hyperparameters such as batch size, sequence block size, learning rate, and the architectural specifics like the number of heads and layers in the transformer model can be adjusted. To facilitate experimentation and allow users to iteratively improve their models, hyperparameters are made configurable via the UI. Users can submit their preferred settings through a form, which are then saved to a configuration file (config.json). This config is dynamically loaded each time the model is trained, enabling flexible adjustments without altering the core code.

Effective tuning of hyperparameters is critical to optimizing the performance of neural networks. In this project, the hyperparameters such as batch size, sequence block size, learning rate, and the architectural specifics like the number of heads and layers in the transformer model can be adjusted. To facilitate experimentation and allow users to iteratively improve their models, hyperparameters are made configurable via the UI. Users can submit their preferred settings through a form, which are then saved to a configuration file (config.json). This config is dynamically loaded each time the model is trained, enabling flexible adjustments without altering the core code.

```python
import json

CONFIG_FILE = "configurations/config.json"

def load_config():
    """ Load the configuration from a file. """
    with open(CONFIG_FILE, "r") as f:
        return json.load(f)

def save_config(config):
    """ Save the updated configuration to a file. """
    with open(CONFIG_FILE, "w") as f:
        json.dump(config, f, indent=4)
```

Code 3.3: Load, Save hyperparameters on config.json

3.2.3 Model Implementation

This section outlines the implementation of our transformer-based language model, designed for natural language processing. The model comprises various components such as embeddings, multi-head attention mechanisms, and transformer blocks, each crucial for handling sequence data efficiently. We'll explore each component's role and configuration within our PyTorch [6] framework, detailing how they contribute to the model's ability to learn and generate text.

Token and Positional Embeddings

Embeddings transform the input tokens and their positions within a sequence into dense vectors of fixed size. This enables the model to interpret the inputs and understand the sequence's structure.

```python
class GPTLanguageModel(nn.Module):
    def __init__(self, vocab_size, config):
        super().__init__()
        self.token_embedding_table = nn.Embedding(vocab_size,
            config['n_embd'])
        self.position_embedding_table = nn.Embedding(config['
            block_size'], config['n_embd'])
        self.blocks = nn.Sequential(*[Block(config['n_embd'],
            config['n_head']) for _ in range(config['n_layer'])])
        self.ln_f = nn.LayerNorm(config['n_embd'])
        self.lm_head = nn.Linear(config['n_embd'], vocab_size)

    def forward(self, x):
        tokens_embeddings = self.token_embedding_table(x)
        positions = torch.arange(x.size(1), device=x.device).
            unsqueeze(0)
        position_embeddings = self.position_embedding_table(
            positions)
        x = tokens_embeddings + position_embeddings
        x = self.blocks(x)
        x = self.ln_f(x)
        logits = self.lm_head(x)
        return logits
```

Code 3.4: GPTLanguageModel Class

Multi-Head Attention Mechanism

The Multi-Head Attention mechanism allows the model to focus on different positions of the input sequence simultaneously, which is critical for understanding complex dependencies.

Head

Defines a single attention head within a Multi-Head Attention layer.

```
class Head(nn.Module):
    def __init__(self, head_size):
        super().__init__()
        self.key = nn.Linear(config['n_embd'], head_size, bias=False)
        self.query = nn.Linear(config['n_embd'], head_size, bias=False)
        self.value = nn.Linear(config['n_embd'], head_size, bias=False)
        self.dropout = nn.Dropout(config['dropout'])

    def forward(self, x):
        k = self.key(x)
        q = self.query(x)
        v = self.value(x)
        weights = torch.matmul(q, k.transpose(-2, -1)) * (1. / math.sqrt(k.size(-1)))
        weights = F.softmax(weights, dim=-1)
        weights = self.dropout(weights)
        return torch.matmul(weights, v)
```

Code 3.5: Head Class

Multi-Head Attention

Combines multiple attention heads to enhance the model's ability to focus on different parts of the input sequence.

```
class MultiHeadAttention(nn.Module):
    def __init__(self, num_heads, head_size):
        super().__init__()
        self.heads = nn.ModuleList([Head(head_size) for _ in range(num_heads)])
        self.proj = nn.Linear(head_size * num_heads, config['n_embd'])
        self.dropout = nn.Dropout(config['dropout'])

    def forward(self, x):
        x = torch.cat([h(x) for h in self.heads], dim=-1)
        return self.dropout(self.proj(x))
```

Code 3.6: MultiHead Class

Feed-Forward Network

Each attention output is processed through a Feed-Forward Network which introduces additional non-linearity into the computation, enhancing the model's ability to represent complex functions.

```
class FeedForward(nn.Module):
    def __init__(self, n_embd):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(n_embd, 4 * n_embd),
            nn.ReLU(),
            nn.Linear(4 * n_embd, n_embd),
            nn.Dropout(config['dropout'])
        )

    def forward(self, x):
        return self.net(x)
```

Code 3.7: FeedForward Class

Transformer Block

Combines the components of Multi-Head Attention and Feed-Forward Network into a single processing block, each of which operates on the embeddings to generate outputs used by subsequent blocks.

```
class Block(nn.Module):
    def __init__(self, n_embd, n_head):
        super().__init__()
        head_size = n_embd // n_head
        self.sa = MultiHeadAttention(n_head, head_size)
        self.ffwd = FeedForward(n_embd)
        self.ln1 = nn.LayerNorm(n_embd)
        self.ln2 = nn.LayerNorm(n_embd)

    def forward(self, x):
        y = self.sa(x)
        x = self.ln1(x + y)
        y = self.ffwd(x)
        x = self.ln2(x + y)
        return x
```

Code 3.8: Block Class

3.2.4 Text Generation

The text generation functionality leverages the trained model to generate responses based on input prompts. This process involves several steps:

Model Loading

The system checks if a pre-trained model is available and loads it. If not available, it initializes a new instance of the GPTLanguageModel.

```
def load_model():
    try:
        with open(config["model_path"], "rb") as f:
            model = pickle.load(f)
        print("Model loaded successfully!")
    except FileNotFoundError:
        print("Model file not found, initializing new model!")
        model = GPTLanguageModel(vocab_size)
    return model
```

Code 3.9: Load Model

Text Encoding

Input text is encoded into a sequence of integers using a mapping that converts each character to its corresponding index in a pre-defined vocabulary.

```
def encode(text):
    return [stoi[char] for char in text]
```

Code 3.10: Text Encoding

Token-by-Token Generation

The model generates text one token at a time, which allows for considering the full context of the input provided so far.

```python
def generate_text(model, input_text):
    encoded_input = torch.tensor([encode(input_text)], dtype=torch.
       long)
    generated_indices = model.generate_token_by_token(encoded_input
       , max_new_tokens=150)
    return ''.join([decode(idx) for idx in generated_indices])
```

Code 3.11: Generate text

Decoding

Converts the sequence of integer indices back to characters to form the output string.

```python
def decode(indices):
    return ''.join([itos[index] for index in indices])
```

Code 3.12: Decoding

Execution

Putting it all together to handle the model interaction.

```python
if __name__ == "__main__":
    model = load_model()
    model.to(config["device"])
    input_text = sys.argv[1]  # Assuming the input is passed as a
       command-line argument
    generated_text = generate_text(model, input_text)
    print(generated_text)
```

Code 3.13: Execution

3.2.5 Challenges and Shortcomings

During the development of this project, I encountered several challenges and shortcomings that influenced both the process and the outcome of the implementation. Understanding and implementing the transformer architecture was a significant challenge due to its complexity, requiring a deep dive into attention mechanisms and neural network integrations.

Additionally, implementing real-time feedback on training metrics for each iteration to monitor the model's learning dynamically, which required modifications to standard training procedures.

Another technical challenge was mastering the nuances of Python and PyTorch, especially in tensor manipulation and batch processing, which proved essential for efficient data handling and accurate model computations.

One specific Python-related challenge involved the text generation process. Initially, the output appeared as a disjointed series of tokens because each token was followed by a newline character. This made the generated text look more like a random combination of characters rather than coherent sentences. By modifying the output processing to strip newline characters, the generated text was transformed into continuous, readable text.

Among the shortcomings, the lack of computational resources was a major limitation, without access to a GPU, the model could not be trained extensively, limiting its sophistication and accuracy. Currently, the model operates only with pre-training and lacks fine-tuning on specific tasks, which restricts its ability to engage in smart, context-aware interactions. This is further compounded by the limited training data, which fails to capture the diversity and complexity of natural language fully, potentially affecting the model's ability to generalize across real-world scenarios. These experiences underline the iterative nature of machine learning projects and highlight the significant impact of resource availability on achieving advanced AI capabilities.

3.3 Testing

Testing is a crucial phase in the development of any software, particularly when dealing with an application as complex as a language model. This section outlines the various manual testing strategies employed to ensure the reliability and performance of the simplified language model. These tests were designed to measure the system's functionality, performance under different conditions, and the overall user experience. By meticulously documenting the results, we can assess how well the model performs in real-world scenarios and identify areas for future improvement.

3.3.1 Implemented Tests

Automated testing in this project is implemented to ensure that the core functionalities of the backend, particularly those related to the machine learning model's operations, behave as expected. These tests help verify the integrity and accuracy of the data processing and training processes, which are critical to the project's success.

Testing is conducted using the Python 'unittest' framework, providing a robust structure for defining test cases. The tests automate the validation of batch processing, loss computation, and the training loop, ensuring that each component functions correctly even as new changes are introduced into the codebase.

Configuration Loading Test:

This test verifies that system configurations can be accurately loaded from a JSON file, which is crucial for maintaining consistent training settings and model parameters.

```
def test_load_config(self):
    config = load_config()
    self.assertIsInstance(config, dict)
```

Code 3.14: Configuration Loading Test

Batch Processing Test:

This test checks whether the model can handle a batch of input data correctly, ensuring that the dimensions of the output match the configured batch size and that a loss value is calculable, indicating that the model's forward pass is functioning.

```
def test_batch_processing(self):
    model = GPTLanguageModel(vocab_size=100)
    batch_x, batch_y = get_batch("train")
    logits, loss = model(batch_x, batch_y)
    self.assertEqual(batch_x.shape[0], config["batch_size"])
    self.assertIsInstance(loss, torch.Tensor)
```

Code 3.15: Batch Processing Test

Loss Estimation Test:

Ensures that the model computes the loss for its outputs against the provided targets, which is essential for tracking the model's learning progress and optimizing performance.

```
def test_loss_estimation(self):
    model = nn.Linear(10, 2)
    criterion = nn.CrossEntropyLoss()
    inputs = torch.randn(config["batch_size"], 10)
    targets = torch.randint(0, 2, (config["batch_size"],))
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    self.assertIsInstance(loss, torch.Tensor)
```

Code 3.16: Loss Estimation Test

Training Iteration Test: Simulates a single training iteration to ensure that the model can perform a forward pass, compute the loss, backpropagate errors, and update its parameters successfully. This test is vital for confirming that the training loop is robust and error-free.

```
def test_training_iteration(self):
    model = GPTLanguageModel(vocab_size=10)
    optimizer = torch.optim.Adam(model.parameters())
    batch_x, batch_y = get_batch("train")
    optimizer.zero_grad()
    logits, loss = model(batch_x, batch_y)
    loss.backward()
    optimizer.step()
    self.assertIsInstance(logits, torch.Tensor)
    self.assertIsInstance(loss, torch.Tensor)
```

Code 3.17: Training Iteration Test

3.3.2 Manual Testing

Model Performance Evaluation

Training, Validation, Model Loss and Accuracy Tracking: To make sure the model was learning correctly, we tracked its training loss, model loss, and accuracy over time. This helped us see how well the model was improving and if we needed to make any adjustments to get better results.

Epoch	Training Loss	Validation Loss	Model Loss	Accuracy
1	3.877	3.797	4.798	0.016
10	2.887	2.858	3.114	0.078
50	2.586	2.518	2.522	0.078
100	2.539	2.486	2.757	0.046
200	2.459	2.403	2.430	0.031
300	2.293	2.241	2.310	0.078
400	2.170	2.120	2.190	0.078
500	1.959	1.949	2.027	0.125
700	1.837	1.831	1.814	0.062
1000	1.451	1.541	1.726	0.156

Table 3.1: Tracking of training, validation, model loss, and accuracy over epochs

Hyperparameter Tuning:

Throughout the development of our model, extensive experiments were conducted to determine the influence of various hyperparameters on the model's performance and efficiency. These experiments tested different settings, including learning rate, number of layers, and batch size, across various configurations.

After careful analysis and evaluation, the optimal hyperparameters for our model, particularly suited for CPU-based computation, were established. These parameters were selected to balance training efficiency and model accuracy effectively. The chosen configuration is as follows:

- Batch Size: 32
- Block Size: 128
- Maximum Iteration: 1000
- Evaluation Interval: 100
- Learning Rate: 0.0003

- Evaluation Iterations: 100
- Embedding Dimension: 384
- Number of Attention Heads: 8
- Number of Transformer Layers: 8 Dropout Rate: 0.2

This configuration has been found to offer the best trade-off between training duration and accuracy under the constraints of our CPU setup. It reflects a comprehensive approach to harnessing our resources efficiently, ensuring that the model performs optimally without necessitating the computational power of a GPU. This setup aligns with our goal to make advanced machine learning techniques more accessible and manageable within the constraints of typical hardware environments.

File Format, Size Efficiency:

To ensure efficient training and resource management, the impact of different file formats on the system's performance was analyzed. Initially, large .xz files were used, which significantly increased training times and system resource usage. To optimize performance, the approach was shifted towards using smaller .txt files. This section details the observations made regarding training time, system resource usage (SRU), and overall efficiency with these changes.

File Format	Data Size (GB)	Training Time (min)	epochs
.xz (initial)	12	2880	3000
.txt (optimized)	3	215	3000

Table 3.2: Comparison of File Format Impact on Model Training

Time Efficiency Testing

Training Time Analysis: To thoroughly evaluate the efficiency of the training process on a CPU, we assessed how the training duration scales with the number of iterations. This detailed analysis helps to pinpoint potential performance bottlenecks and demonstrates the necessity of efficient computation. Below, the table presents our findings, highlighting the impact of iteration count on the training duration using a CPU.

Table 3.3: Training Time Analysis Across Different Hardware

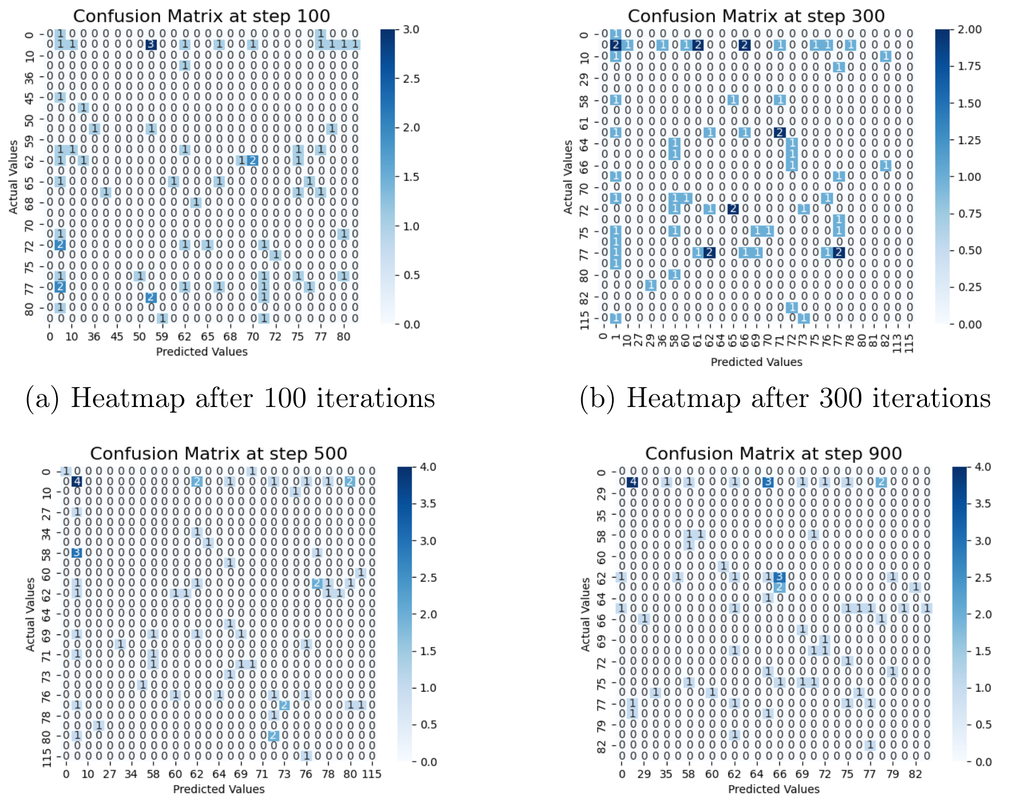
Iteration Count	**Average Training Time (s)**	**Overall Training Time (s)**
1	2.37	2.37
10	3.54	35.46
50	2.64	132.21
100	2.70	270.64
200	2.53	507.78
300	2.67	803.81
400	2.69	1078.21
500	2.75	1375.58
700	2.37	1665.58
1000	1.96	1966.58

Token Generation Time: To understand the responsiveness and scalability of the language model in practical applications, we measured the time required to generate text for different numbers of tokens. This testing provides insight into the performance of the model during interactive sessions, reflecting how quickly the system can produce responses under varying loads.

Number of Tokens	**Average Generation Time (s)**
10	1
50	3
100	5
200	9-12

Table 3.4: Token Generation Time Analysis

Confusion Matrices Visualization: To effectively illustrate the model's predictive performance during training, confusion matrices were generated at key intervals using Matplotlib [11]. These heatmaps visually display the accuracy of predictions across different classes, highlighting improvements and ongoing challenges in the model's classification capabilities. Also sample text generated after specific iterations are also displayed. It helps us to see how the text generation improves after each iteration.



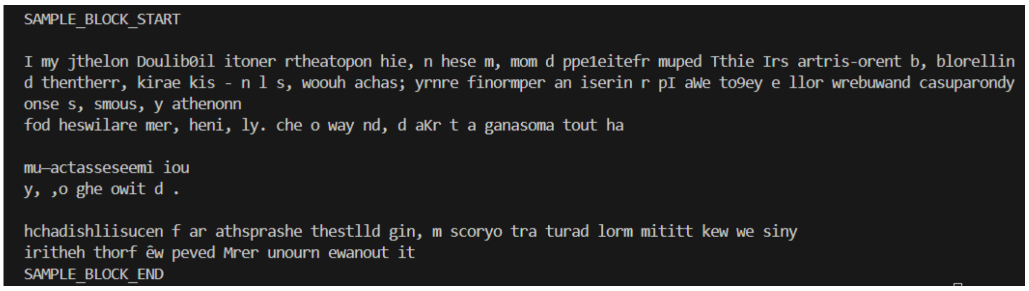
(a) Heatmap after 100 iterations

(b) Heatmap after 300 iterations

(c) Heatmap after 500 iterations

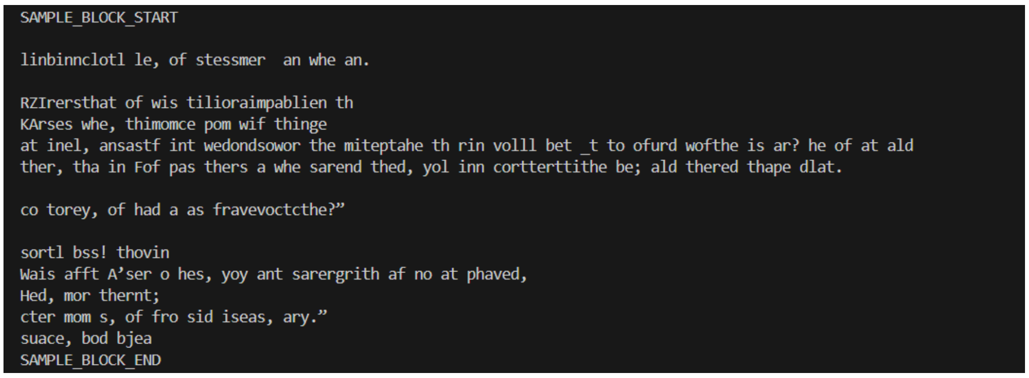
(d) Heatmap after 1000 iterations

Figure 3.7: Heatmaps



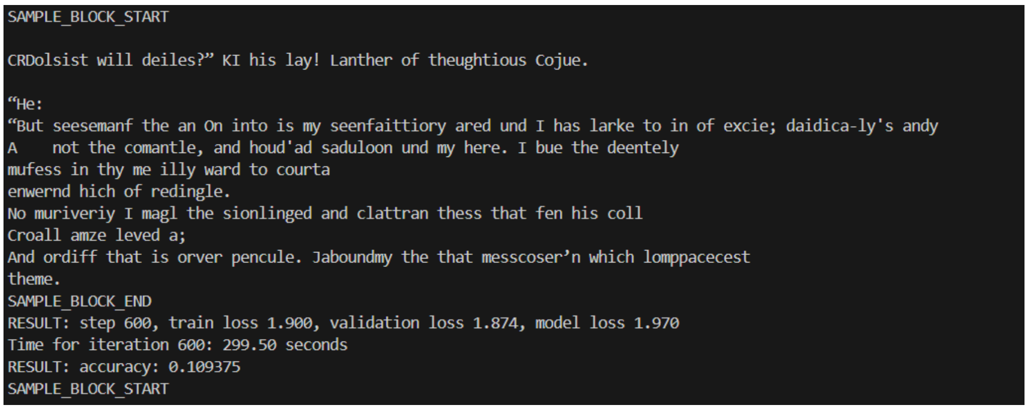
```
SAMPLE_BLOCK_START

I my jthelon Doulib0il itoner rtheatopon hie, n hese m, mom d ppe1eitefr muped Tthie Irs artris-orent b, blorellin
d thentherr, kirae kis - n l s, woouh achas; yrnre finormper an iserin r pI aWe to9ey e llor wrebuwand casuparondy
onse s, smous, y athenonn
fod heswilare mer, heni, ly. che o way nd, d aKr t a ganasoma tout ha

mu—actasseseemi iou
y, ,o ghe owit d .

hchadishliisucen f ar athsprashe thestlld gin, m scoryo tra turad lorm mititt kew we siny
iritheh thorf êw peved Mrer unourn ewanout it
SAMPLE_BLOCK_END
```

(a) Sample text generated after 100 iterations



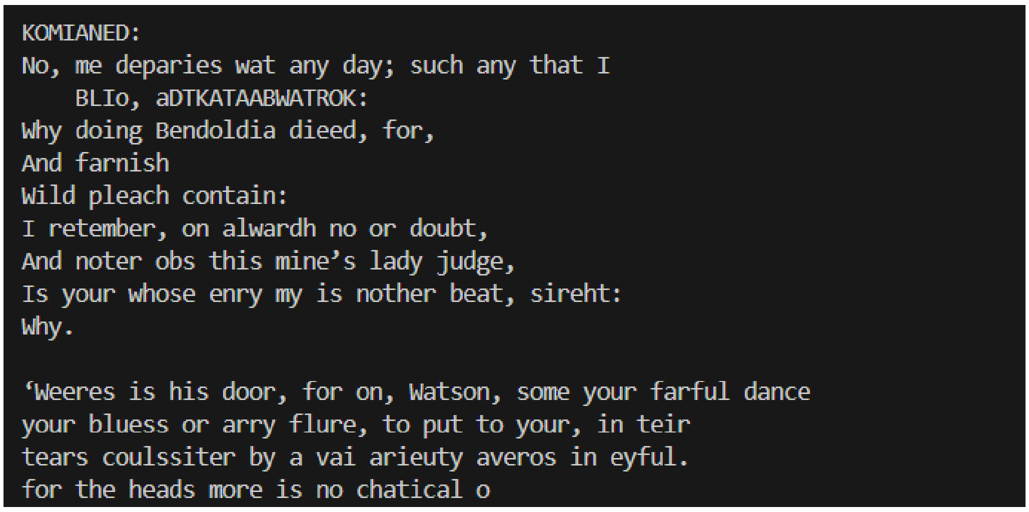
```
SAMPLE_BLOCK_START

linbinnclotl le, of stessmer  an whe an.

RZIrersthat of wis tilioraimpablien th
KArses whe, thimomce pom wif thinge
at inel, ansastf int wedondsowor the miteptahe th rin volll bet _t to ofurd wofthe is ar? he of at ald
ther, tha in Fof pas thers a whe sarend thed, yol inn cortterttithe be; ald thered thape dlat.

co torey, of had a as fravevoctcthe?”

sortl bss! thovin
Wais afft A’ser o hes, yoy ant sarergrith af no at phaved,
Hed, mor thernt;
cter mom s, of fro sid iseas, ary.”
suace, bod bjea
SAMPLE_BLOCK_END
```

(b) Sample text generated after 300 iterations



```
SAMPLE_BLOCK_START

CRDolsist will deiles?” KI his lay! Lanther of theughtious Cojue.

“He:
“But seesemanf the an On into is my seenfaittiory ared und I has larke to in of excie; daidica-ly's andy
A    not the comantle, and houd'ad saduloon und my here. I bue the deentely
mufess in thy me illy ward to courta
enwernd hich of redingle.
No muriveriy I magl the sionlinged and clattran thess that fen his coll
Croall amze leved a;
And ordiff that is orver pencule. Jaboundmy the that messcoser’n which lomppacecest
theme.
SAMPLE_BLOCK_END
RESULT: step 600, train loss 1.900, validation loss 1.874, model loss 1.970
Time for iteration 600: 299.50 seconds
RESULT: accuracy: 0.109375
SAMPLE_BLOCK_START
```

(c) Sample text generated after 500 iterations



```
KOMIANED:
No, me deparies wat any day; such any that I
    BLIo, aDTKATAABWATROK:
Why doing Bendoldia dieed, for,
And farnish
Wild pleach contain:
I retember, on alwardh no or doubt,
And noter obs this mine’s lady judge,
Is your whose enry my is nother beat, sireht:
Why.

‘Weeres is his door, for on, Watson, some your farful dance
your bluess or arry flure, to put to your, in teir
tears coulssiter by a vai arieuty averos in eyful.
for the heads more is no chatical o
```

(d) Sample text generated after 1000 iterations

Figure 3.8: Sample texts generated

3.3.3 User Interface Testing

Response Time: Testing the responsiveness of our system involved manual tests on different user interface actions. We specifically focused on how long it takes for the system to respond after user inputs through the UI, such as submitting forms for hyperparameter adjustments or uploading training data files. This included timing how quickly the system processes different sizes of training data files and how it handles dynamic adjustments to the model's settings via the UI.

UI Action	Average Response Time	Notes
Form Submission	2 seconds	Consistent without delays.
File Upload (Small Files)	24 seconds	Quick processing due to smaller vocabularies.
File Upload (Large Files)	37 seconds	Slower processing time as increased bigram computation requires more time to handle larger data sets.

Table 3.5: UI Response Times for Different Actions

3.3.4 Test Plan for User Interface

1. Chat Page

Description	Steps to Execute	Expected Result	Pass Criteria
Initial Load Test	Open the Chat Page and observe the interface.	The chat interface loads without any errors.	The interface appears correctly and is fully functional.
Message Submission Test	Type a message into the chat input box and send it.	The message appears in the chat window followed by a model response.	The message and response are displayed correctly without huge delay.

Description	**Steps to Execute**	**Expected Result**	**Pass Criteria**
Response Time Test	Time the interval from sending a message to receiving a response.	The response is received within 7 seconds maximum.	Response time is less than 7-10 seconds consistently.
Refresh Persistence Test	Send messages, refresh the Chat Page, then observe the chat window.	No previous messages are displayed after refresh.	The chat history does not persist after page refresh, as expected.
Input Validation Test	Enter various inputs including empty, long texts, and special characters.	The system correctly handles each input type.	All valid messages are accepted.

2. Data Preparation Page

Description	**Steps to Execute**	**Expected Result**	**Pass Criteria**
File Upload Functionality	Select multiple text files and click 'Submit'.	Files are uploaded and a confirmation message is displayed.	All selected files are listed as successfully uploaded.
File Processing Validation	After uploading, observe processed file details.	Details about file processing, including train and validation splits, are displayed.	Correct information about each file's processing is shown without errors.
Visualization Display	Check the display of vocabulary and bigram tables.	Tables should be properly formatted and populated with data from the uploaded files.	Tables display correctly with accurate data representation.

Description	**Steps to Execute**	**Expected Result**	**Pass Criteria**
Performance Measurement	Measure the time taken from file upload to data visualization display.	All processes from upload to display are completed within a reasonable timeframe.	The entire operation from file selection to data visualization does not exceed 60 seconds.
Error Handling	Try uploading non-text files or very large files.	System should display an error message for unsupported or overly large files.	Appropriate error messages are shown and no system crashes occur.

3. Train Page

Description	**Steps to Execute**	**Expected Result**	**Pass Criteria**
Hyperparameter Configuration	Input different values for each hyperparameter field and submit the form.	Parameters are accepted and confirmation of submitted data is displayed.	Submitted parameters match the displayed configuration in the confirmation section.
Model File Upload	Choose a model file to upload and submit the form.	The model file is uploaded and a message confirming the model file upload is displayed.	The system acknowledges the uploaded model file without errors.
Training Initiation	Submit the form with all required fields filled.	Training process starts and training metrics start updating on the Progress page.	Training begins without errors and metrics such as loss and accuracy are displayed as expected.

Description	**Steps to Execute**	**Expected Result**	**Pass Criteria**
Error Handling in Model	Choose not '.pk1' file format.	Error message is displayed for invalid file format.	On top of the page relevant error message displayed.
Response to Configuration Changes	Change hyperparameters several times and submit each time.	Each submission correctly updates the training configuration.	The system must consistently update and reflect new configurations without requiring a restart or causing errors.

4. Progress Page

Description	**Steps to Execute**	**Expected Result**	**Pass Criteria**
Training Metrics Update Test	Initiate training and observe metrics updating in real-time on the logs tab.	Metrics including loss and accuracy should update periodically without delay.	Training metrics are accurately reported in real-time with no delays.
Results Tab Functionality Test	Switch to the results tab after some iterations have completed.	Display accumulated training and validation loss metrics.	The displayed metrics match the expected values based on the training output.
Sample Generation Test	Navigate to the sample tab post-training.	Display the text sample generated by the model.	The generated text should be coherent and correspond to the training data style.

Description	**Steps to Execute**	**Expected Result**	**Pass Criteria**
Parameter Visibility Test	Check the parameters tab for the training setup.	All used parameters such as batch size, block size, and learning rate are visible and correct.	The parameters displayed should match those configured before training began.
Real-time Monitoring Test	Observe the progress of the training through the logs.	The logs should display a sequential account of the training process.	Logs must correctly reflect the training progression with accurate timestamps.

This comprehensive manual testing documentation will not only demonstrate the thoroughness of your testing but also provide insights into potential improvements and adjustments for future versions of your application. Including visual aids such as screenshots, graphs, and tables will enhance the clarity and usefulness of the documentation.

Chapter 4

Conclusion

This thesis was inspired by the high computational costs and substantial hardware requirements of large language models (LLMs), which often make them inaccessible to those without significant resources. By developing a simplified language model, this project aimed to make the underlying processes of LLMs understandable and accessible, even for users with limited technical facilities. Although the model presented does not match the full capabilities of advanced LLMs, it effectively demonstrates the essential aspects of AI training and operation within a conversational context.

The web-based platform developed through this thesis allows users to engage directly with the model training process and interact with the trained model, providing a tangible experience that demystifies the complexities of LLMs. While acknowledging its limitations in scalability and security, the project serves as a foundational educational tool that elucidates the principles of machine learning and natural language processing in an interactive manner.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Gregory Morse, for his constant support and guidance throughout this project. His advice was invaluable, and his encouragement helped me navigate through many challenges.

A big thank you to Andrej Karpathy as well. His clear and informative YouTube lectures on transformer architectures were incredibly helpful and made complex ideas much easier to understand.

Finally, I must thank ChatGPT for being such a reliable helper during this project. Whether it was fixing bugs, refining ideas, or just providing quick answers, ChatGPT made everything a lot smoother. It’s somewhat amusing yet profoundly impactful to realize that a project aimed at simplifying language models benefited so greatly from an advanced LLM. This experience has been a full-circle moment, highlighting the practical utility and transformative potential of the very technologies we study.

Bibliography

[1] Python Software Foundation. *Python Installation Guide*. Accessed: 2024-05-26. 2024. URL: https://www.python.org/downloads/.

[2] Node.js Foundation. *Node.js Installation Guide*. Accessed: 2024-05-26. 2024. URL: https://nodejs.org/en/download/.

[3] Microsoft. *Visual Studio Code Installation Guide*. Accessed: 2024-05-26. 2024. URL: https://code.visualstudio.com/docs/setup/setup-overview.

[4] Git Documentation. *Git Installation Guide*. Accessed: 2024-05-26. 2024. URL: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git.

[5] Zhainagul Altynbek kyzy. *Simplified Transformer-Based Architecture for Web-Based User Interaction*. Accessed: 2024-05-26. 2024. URL: https://github.com/zhainagulaltynbekk/simplified-transformer-based-language-model.

[6] *PyTorch*. URL: https://pytorch.org/get-started/locally/.

[7] Flask. *Flask Documentation*. 2010. URL: https://flask.palletsprojects.com/en/3.0.x/.

[8] *React*. URL: https://react.dev/.

[9] Google Researchers. *Attention Is All You Need*. 2019. URL: https://arxiv.org/pdf/1706.03762.

[10] *Pickle Module Documentation*. URL: https://docs.python.org/3/library/pickle.html.

[11] *Matplotlib*. URL: https://matplotlib.org/.

List of Figures

List of Tables

List of Codes