Developing a Dialog-Optimized Large Language Model Using Transformer Architecture

A major project report submitted in partial fulfillment of the requirement for the award of a degree

Bachelor of Technology

in

Computer Science & Engineering

Submitted by

Kanav Gupta (211219) **Raghav Singal (211229)**

Under the guidance & supervision of

Dr. Meghna Dhalaria & Mr. Ravi Sharma

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, Solan - 173234 (India)

May 2025

Supervisor’s Certificate

This is to certify that the major project report entitled ‘Developing a Dialog-Optimized Large Language Model Using Transformer Architecture’, submitted in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science & Engineering, in the Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat, is a bona fide project work carried out under my supervision during the period from June 2024 to May 2025.

I have personally supervised the research work and confirm that it meets the standards required for submission. The project work has been conducted in accordance with ethical guidelines, and the matter embodied in the report has not been submitted elsewhere for the award of any other degree or diploma.

(Supervisor Signature)
Supervisor Name: Dr. Meghna Dhalaria &
Mr. Ravi Sharma

Date:	Designation: Assistant Professor (SG)
Place:	Department: Dept. of CSE & IT

Candidate's Declaration

We hereby declare that the work presented in this major project report entitled **'Developing a Dialog-Optimized Large Language Model Using Transformer Architecture'** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering** submitted in the Department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from June 2024 to May 2025 under the supervision of **Dr. Meghna Dhalaria & Mr. Ravi Sharma.**

We further declare that the matter embodied in this report has not been submitted for the award of any other degree or diploma at any other university or institution.

(Student Signature)	(Student Signature)
Name: Kanav Gupta	Name: Raghav Singal
Roll No.: 211219	Roll No.: 211229
Date:	Date:

This is to certify that the above statement made by the candidates is true to the best of my knowledge.

(Supervisor Signature)
Supervisor Name: Dr. Meghna Dhalaria
Co-Supervisor Name: Mr. Ravi Sharma

Date: Designation: Assistant Professor (SG)

Place: Department: Dept. of CSE & IT

Acknowledgement

Firstly, we express our heartiest thanks and gratefulness to almighty God for His divine blessing makes it possible to complete the project work successfully.

We are really grateful and wish our profound indebtedness to Supervisor **Dr. Meghna Dhalaria, Assistant Professor (Senior Grade)**, Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat for his Deep Knowledge & keen interest in the field of "**Artificial Intelligence**" to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

We would like to express our heartiest gratitude to **Dr. Meghna Dhalaria & Mr. Ravi Sharma,** Department of CSE/IT, for his kind help to finish our project. We would also generously welcome each one of those individuals who have helped us straightforwardly or in a roundabout way in making this project a win. In this unique situation, we might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated our undertaking.

Finally, we must acknowledge with due respect the constant support and patience of our parents.

Name: Kanav Gupta
Roll No.: 211219

Name: Raghav Singal
Roll No.: 211229

TABLE OF CONTENTS

LIST OF TABLES **vi**

LIST OF FIGURES **vii-viii**

LIST OF ABBREVIATIONS, SYMBOLS, OR NOMENCLATURE **ix**

ABSTRACT **x**

1. INTRODUCTION 1-4

1.1 Introduction.. 1

1.2 Problem Statement….. 2

1.3 Objective.. 2

1.4 Significance and motivation of the project report.. 4

1.5 Organization of project report.. 4

2. LITERATURE SURVEY 5-30

2.1 Overview of relevant literature 5

2.2 Key gaps in the literature 30

3. SYSTEM DEVELOPMENT 31-49

3.1 Requirements and Analysis.. 31

3.2 Project Design and Architecture 33

3.3 Data Preparation 33

3.4 Implementation 39

3.5 Key Challenges 49

4. TESTING 52-53

4.1 Testing Strategy 52

4.2 Test Cases and Outcomes 53

5. RESULTS AND EVALUATION 54-59

5.1 Results 54

5.2 Fine-Tuning Results 55

5.3 Evaluation Metrics 56

5.4 Model Performance and Observations …………………………………….. 57

5.5 Limitations and Challenges ………………………………………………. 57

6. CONCLUSIONS AND FUTURE SCOPE…………………………………………….. 60-61

6.1 Conclusion ………………………………………………………………………. 60

6.2 Future Scope …………………………………………………………........ 61

REFERENCES.. 62-63

List of Tables

Table No.	Table Name
1	Existing work in field of Large Language Model
2	Resources required to develop the LLM model
3	Example test cases followed during the trail and testing

List of Figures

Fig. No	TITLE
1	Timeline representing development in LLMs
2	Development of Architecture in LLM
3	LLM System Architecture
4	Detailed representation of the Architecture of LLM
5	Implement a GPT Model from Scratch
6	Preparing the dataset for the Finetuning Classification
7	LLM Model Preparation Evaluation
8	Token Encoding
9	Data Preparation and Refinement
10	Data Processing and Transformation
11	Processed Data Tokenized
12	GPT Transformer Initialization Parameter
13	Effect of GELU and RELU on Input Transformation
14	GPT Model Implementation
15	GPT Model Output Example
16	Simple Text Generation Using GPT Model
17	Training and Validation Loss Calculation
18	Epoch Output and Loss Evaluation
19	Spam vs Not Spam Classification Example

20	Building GPT-2 from Scratch: A step-by-step Initialization
21	Chunk-wise Data Processing Script
22	Script to Combine Processed Data Chunks into a Single Dataset
23	Result of the Preprocessing pipeline
24	Model Accuracy Evaluation
25	Final Accuracy Evaluation
26	Instruction-Based Text-Generation

List of Abbreviations, Symbols, or Nomenclature

S. No.	Abbreviation	Full Form
1.	ML	Machine learning
2.	GenAI (AI)	Generative Artificial Intelligence
3.	LLM	Large Language Model
4.	DL	Deep Learning
5.	CNN	Convolution Neural Networks
6.	NLP	Natural Language Processing
7.	SFT	Supervised Fine-Tuning
8.	USFT	Unsupervised Fine-Tuning
9.	RLHF	Reinforcement learning from human feedback

Abstract

In a short time, this NLP progress saw the dawn of the development of Large Language Models LLMs that are human-like in understanding and generating text. It has changed the face of several domains, including automated customer service, education, and health care, offering an insight into the seamless and intelligent conversations that application scenarios could hold. This project, therefore, is all about the development of a dialogue-optimized LLM agent with the Transformer architecture and for conversational purposes. The comprehensive SlimPajama-627B corpus formed the backbone of the pre-training phase, while the fine-tuning is done with the help of the Alpaca corpus, ensuring that the model generalizes well for use in contextually subtle as well as user-specific dialogues. Such future integration with speech recognition may widen the scope of the model beyond processing and giving accurate responses to spoken language input-and human-machine understanding. Applications could be in virtual assistants, customer support systems, and interactive learning environments, allowing greater naturalness and intuitiveness in human-computer interactions. The major methodologies discussed in the report include data collection, data preprocessing, model pre-training, fine-tuning, and iterative evaluation of conversational performance metrics.

Chapter 01: Introduction

1.1 Introduction

An LLM is a large language model that is a neural network built for understanding human-like text, generating it, and responding to it. In many cases, LLMs are deep neural networks trained on amounts of text data that approach, if not include, great big chunks of the entire publicly available text-internet. The "large" is for both huge model size in terms of parameters and an enormous training dataset. Such models can have tens and often hundreds of billions of parameters referring to the adjustable weights in the network optimized during training to predict the next word in a sequence.

LLMs utilize an architecture called the transformer (covered in more detail in section 1.4), which allows them to pay selective attention to different parts of the input when making predictions, making them especially adept at handling the nuances and complexities of human language. In fact, since LLMs can produce text forms, LLMs generally fall in the category of generative artificial intelligence (AI) and are also referred to as generative AI or GenAI. As shown in figure 1, AI is the wider term for creating machines that can do anything that requires human-like intelligence, including understanding language, recognizing patterns, and making decisions; it also includes subfields such as machine learning and deep learning.

LLMs represent a specific application of deep learning techniques with the help of their ability to process large input data, learn it and generate human-like text. Deep learning is a specialized branch of machine learning that focuses on using multi-layer neural networks. Machine learning and deep learning are fields aimed at implementing algorithms that enable computers to learn from data and perform tasks that typically require human intelligence. The field of artificial intelligence is nowadays dominated by machine learning and deep learning but it also includes other approaches, for example by using rule-based systems, genetic algorithms, expert systems, fuzzy logic, or symbolic reasoning.

1.1.1 Machine learning

The algorithms used to implement AI are the focus of the field of machine learning (ML). Machine learning is the area of study concerned with algorithms that learn from and make predictions or decisions based on data without having been explicitly programmed. For example, a practical example of machine learning is a spam filter. Instead of writing rules to define spam emails, one trains a machine-learning algorithm with a collection of emails consisting of some examples declared as spam and others declared as legitimate. The model learns to recognize patterns and characteristics that are typical of spam, so that it can classify new emails as either spam or legitimate by reducing error in its predictions on a training dataset.

1.1.2 Deep learning

Deep learning (DL) is a subset of machine learning that focuses on utilizing neural networks with three or more layers (also called deep neural networks) to model complex patterns and abstractions in data. In contrast to deep learning, traditional machine learning requires manual feature extraction. This means that human experts need to identify and select the most relevant features for the model. DL is best suited for handling high-complexity decision-making like recommendations, speech recognition, image classification, etc. Enabled by advancements in deep learning, LLMs are trained on vast quantities of text data. This allows LLMs to capture deeper contextual information and subtleties of human language compared to previous approaches. As a result, LLMs have significantly improved performance in a wide range of NLP tasks, including text translation, sentiment analysis, question answering, and many more.

1.2 Problem Statement

In recent years, smart chatbots have become increasingly popular, engaging users in conversations similar to those with real people. This rise is driven by the growing use of digital assistants, customer service bots, and system agents. Before the advent of large language models, traditional methods excelled at categorization tasks such as email spam classification and straightforward pattern recognition that could be captured with handcrafted rules or simpler models. Despite being promising, these methods have limitations such as keeping context in lengthy interactions, producing reasonable outputs, and coping with different linguistic types in the case of text generation with respect to LLMs based on Transformer architecture. What we aim to achieve is the development of a dialog-optimized LLM over a Transformer architecture that is meant to make the interactions more contextually relevant, fluent conversational, and more reliable in turn to improve the outcome from AI generated conversations.

1.3 Objectives

1. To Design and Implement a Transformer-Based Large Language Model: To develop a robust Large Language Model (LLM) from scratch using Transformer architecture.

2. Enhance Contextual Understanding in Dialogs: To improve the model's ability to understand and generate contextually accurate responses.

3. Evaluate and Optimize Model Performance: To rigorously test the developed model on various dialog datasets, analyze its performance and perform optimizations to achieve high accuracy and efficiency.

1.4 Significance and Motivation of the Project Work

The field of natural language processing has been completely transformed by the advent of Large Language Models (LLMs), which have made it possible for machines to understand, generate and communicate in languages like humans. There is motivation in the need for good AI conversation systems and the research opportunities available in natural language processing (NLP) development. Personal interest in following the newest NLP developments and a commitment to real-life problem resolution is driving the project even further. Incorporation of novel techniques and resources also ultimately contributes to the general aim of expanding the domain of AI-driven interaction. This project tries to contribute to that transformation by developing a dialog-ready LLM for use in conversation applications. A model like this would affect the following areas considerably:

- Customer care: Providing accurate and contextual responses in real-time.
- Education: Enable interactive and personalized learning environments.
- Healthcare: Support diagnostic decision-making and patient involvement.
- General communication: Enhance human-computer interaction in personal assistants and chatbots.

1.5 Organization of Project Report

Chapter 1: An overview of the project that includes information about its purpose, its goals, and how it will benefit users is provided.

Chapter 2: Survey of the literature for the project. This covers the several project reports on the healthcare-focused chatbot program and their uses.

Chapter 3: System Development: In this section, we've covered the project's primary design, the connections between its components, the platforms it was built on, and a summary of the algorithms it employed.

Chapter 4: Thorough testing and trails of the model. This covers the testing part to find the area of potential improvements.

Chapter 5: Evaluation of the project's performance. Here, we've discussed the outcomes and system performance.

Chapter 6: In conclusion, we have discussed the project's results, its future scope, and what more may be done right now.

Chapter 02: Literature Survey

2.1 Overview of Relevant Literature

In this chapter, the analysis of the paper is put towards the foundation regarding the progress in LLMs. It has been quite a busy time in the space called Natural Language Processing, one where methods and algorithms have emerged that promise to enhance the promise of Large Language Models (LLMs). Over the years, there have been many studies on the development and fine-tuning of Transformer-type architectures, some of which are summarized below.

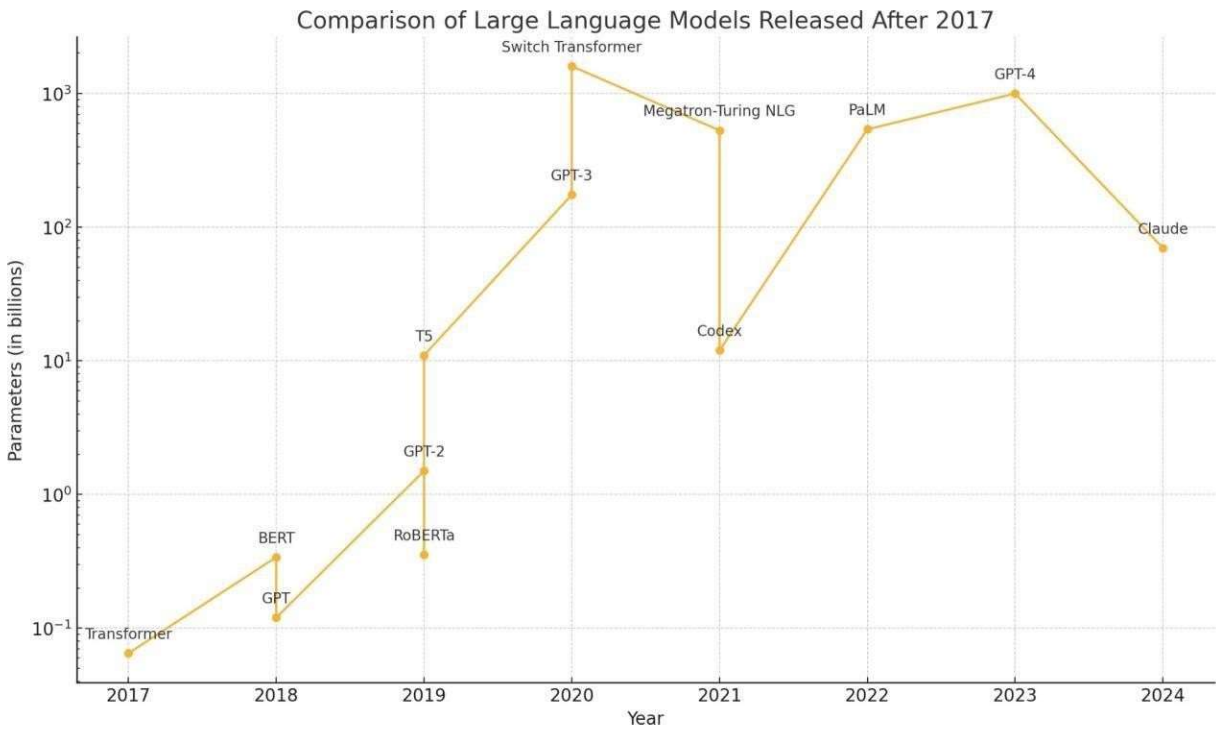


Fig. 1. Timeline representing development in LLMs

Fig. 1 shows the progression of large language models (LLMs) released after 2017, illustrating their exponential growth in parameter size over time. The chart highlights key milestones in the development of LLMs, starting with the Transformer model in 2017 and followed by innovations like BERT, GPT series, and others. Significant leaps are observed with models like

GPT-3, Megatron-Turing NLG, and GPT-4, showcasing advancements in scaling parameters to enhance capabilities. This trend underscores the rapid evolution of transformer-based architectures and the increasing computational resources dedicated to training state-of-the-art LLMs.

Large Language Models (LLMs) make NLP and artificial intelligence quite great. The world's biggest company on earth, Google, released its paper called "Attention is All You Need" (2017) [1]. This paper introduced the concept of transformer architecture that literally revolutionized natural language processing with its self-attention mechanism. The authors proposed the Transformer model that did not need any recurrence or convolutions in tasks relating to sequence transduction, further allowing better performance and more parallelization in most NLP tasks. The proposed novel architecture is solely based on the attention mechanism. Attending entirely with scaled dot-product attention and multi-head attention. Since this model can get very long sequences, this model lacks the incorporation of the positional information in it. The novel paper opens new avenues for research in attention-based models as well as their applicability beyond machine translation. The Transformer architecture thus forms the foundation for further future LLMs and made a significant advancement in the field of NLP.

Here we have Figure 2, the timeline which elucidates the emergence of key architectures in Large Language Models (LLMs) over a period extending from 2017 to 2024. Beginning indeed in 2017 with the introduction of the so-called Transformer architecture by Google, it is nothing but a groundbreaking model as of yet, into which most subsequent LLMs were constructed. In 2018, BERT made its debut with Google, while OpenAI delivered Chat-GPT basing the original version, whose application formed the basis for the subsequent iterations in the series. Thus, all these developments signal that natural language processing is constantly on the move with each new model building towards further success in achieving improved natural language understanding and generation.

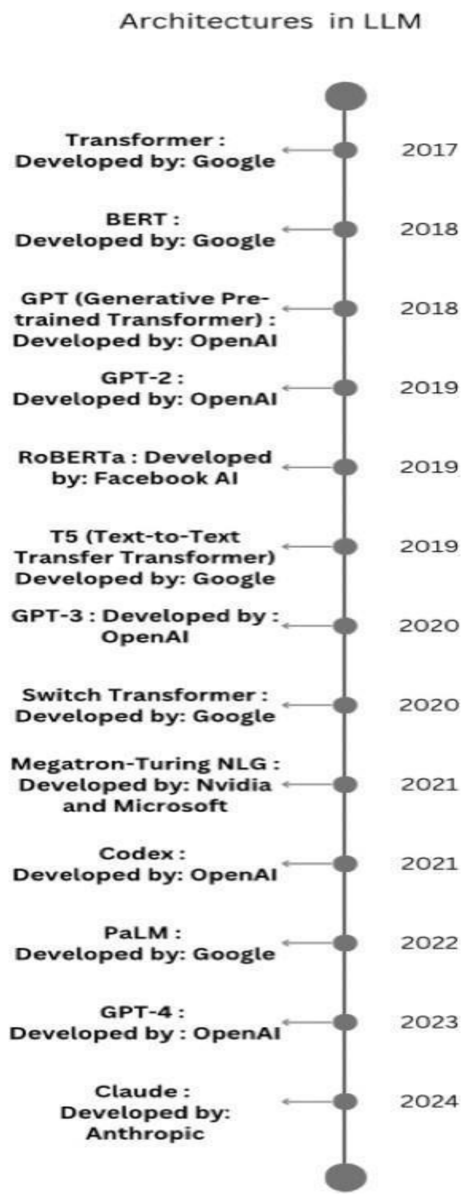

Architectures in LLM
Transformer : Developed by: Google
2017
BERT : Developed by: Google
2018
GPT (Generative Pre-trained Transformer) : Developed by: OpenAI
2018
GPT-2 : Developed by: OpenAI
2019
RoBERTa : Developed by: Facebook AI
2019
T5 (Text-to-Text Transfer Transformer) Developed by: Google
2019
GPT-3 : Developed by : OpenAI
2020
Switch Transformer : Developed by: Google
2020
Megatron-Turing NLG : Developed by: Nvidia and Microsoft
2021
Codex : Developed by: OpenAI
2021
PaLM : Developed by: Google
2022
GPT-4 : Developed by : OpenAI
2023
Claude : Developed by: Anthropic
2024

Fig. 2. Development of Architecture in LLM

Below is the summary of the literature review, which specifies the previous and recent developments in the field of large language models.

The seminal work by Vaswani et al. [1] introduced the transformer architecture, a game-changer in NLP due to its self-attention mechanism, which improved scalability and eliminated the limitations of recurrent models. Building upon this foundation Devlin et al.'s BERT demonstrate the effectiveness of bidirectional transformers in pretraining on extensive text datasets.

extended these ideas with the T5 framework, demonstrating the benefits of unifying NLP tasks in a text-to-text format.

Proximal Policy Optimization introduced a more reliable way to reinforcement learning models, which was inspired by methods used in LLM fine-tuning for dialog systems. Building on this, incorporated human feedback into model training, leading to notable improvements in how well language models follow instructions, an aspect of conversational AI.

Recent advancements have aimed at improving the computational efficiency and scalability of LLM. A lightweight yet high-performing model designed to reduce computational costs. Introduced Mixtral of Experts, a modular architecture that supports dynamic adaptation to specific tasks, especially useful for dialog systems. Investigated alignment strategies to fine-tune models for better contextual comprehension in conversational settings.

The use of transformer-based models in dialog systems has been extensively studied. Li et al. [22] and Wang et al. [23] Examined key developments in handling multi-turn conversations, focusing on methods to preserve context quality. Xu et al. [21] and Le et al. [24] offered wider insights how LLMs are transforming NLP tasks, particularly those involving interactive communication.

Incorporating multimodal components into dialog systems has emerged as key research. Brohan et al. [17] and Huang et al. [19] investigated the integration of language and robotic inputs, aiming to enhance natural interaction in real-world environments. Foundation work by Lake et al. [25] further this by emphasizing that replicate human-like learning and cognitive abilities.

Dialog systems have also benefit in advancements in areas such as sentiment analysis [11], hierarchical architectures [12], and multi-tasking [14], which contribute to improved semantic comprehension. Nielsen's deep learning work [15] offers a foundational perspective on these developments with the field of AI, while Khattak et al. [20] examine word embeddings which laid the groundworks for today's transformer approaches.

The use of transformers in tasks like video and text generation was advanced by Raffel et al. and Aafaq et al. [13], establishing basis for applying LLMs in dialog involving multimodal data. Further, Zhang et al. [16] and Yang et al. [18] emphasize the growing impact of LLMs in robotics and many more, expanding the potential of dialog-driven systems.

Research on user authentication mechanisms [10] and neural machine translation techniques [9] offers perspectives on tailoring LLMs for various use cases, such as secure and adaptive dialog systems. These contributions reflect the fast-paced advancements of LLMs core to their present capabilities in supporting context-sensitive conversational agents.

Table 1. Existing work in field of Large Language Model

S. No.	**Author & Paper Title [Citation]**	**Journal/ Conference (Year)**	**Tools/ Techniques/ Dataset**	**Key Findings/ Results**	**Limitations/ Gaps Identified**
1.	Vaswani, A., et al. Attention is All you Need. In Advances in Neural Information Processing Systems.	(2017)	**Tools-T**ensorFlow and PyTorch for implementing the Transformer model. **Techniques-Transformer Architecture**: Introduces a	Outperforms previous models on translation tasks with **BLEU scores** of 28.4 (English-German) and 41.8 (English-French).	**High computational cost** (quadratic complexity). Challenges with **long sequences** and **resource requirements**.

			novel architecture based on self-attention mechanisms, eliminating the need for recurrence and convolution. **Datasets-WMT 2014 English-German and English-French**: Utilized for training and evaluating the translation capabilities of the proposed model.	Faster training due to parallelization.	
2.	Schulman, J., et al. Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347.	(2017)	**Tools-**TensorFlow and OpenAI Baselines for implementing reinforcement learning algorithms. **Techniques-Proximal Policy Optimization (PPO)**: A reinforcement learning algorithm that balances exploration and exploitation	PPO achieves a good trade-off between sample efficiency and ease of implementation. It outperforms older methods like **Trust Region Policy Optimization (TRPO)** while being	Still requires **significant computational resources** for training.May not always be optimal in environments with **extreme variance** in rewards or complex exploration challenges.

			novel architecture based on self-attention mechanisms, eliminating the need for recurrence and convolution. **Datasets-WMT 2014 English-German and English-French**: Utilized for training and evaluating the translation capabilities of the proposed model.	Faster training due to parallelization.	
2.	Schulman, J., et al. Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347.	(2017)	**Tools-**TensorFlow and OpenAI Baselines for implementing reinforcement learning algorithms. **Techniques-Proximal Policy Optimization (PPO)**: A reinforcement learning algorithm that balances exploration and exploitation	PPO achieves a good trade-off between sample efficiency and ease of implementation. It outperforms older methods like **Trust Region Policy Optimization (TRPO)** while being	Still requires **significant computational resources** for training.May not always be optimal in environments with **extreme variance** in rewards or complex exploration challenges.

			and right-to-left). Datasets-Books Corpus: A dataset containing over 11,000 books used for pre-training the model.	to unidirectional models like GPT.	
4.	Raffel, C., et al Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. Journal of Machine Learning Research, 21(140), 1-67.	(2020)	Tools-TensorFlow and JAX for implementing the T5 model and conducting experiments. Techniques-Text-to-Text Framework: Treats all NLP tasks as text-to-text problems, allowing for a unified approach to various tasks like translation, summarization, and question answering. Datasets-C4 Dataset: A large, cleaned dataset derived from Common Crawl used for	T5 achieves state-of-the-art performance across many NLP benchmarks by treating every task as a text generation problem. Simplifying NLP tasks into a unified framework improves both flexibility and performance.	Training is computationally expensive, requiring significant resources. The model's performance heavily depends on high-quality data and hyperparameter tuning.

			pre-training the model.		
5.	Ouyang, L., et al.Training language models to follow instructions with human feedback. arXiv preprint arXiv:2203.02155.	(2022)	Frameworks:The study utilizes PyTorch for model implementation and training. Techniques-Supervised Fine-Tuning: Initial fine-tuning of GPT-3 using human-written demonstrations to align model outputs with user intentions. Reinforcement Learning from Human Feedback (RLHF): Further fine-tuning using human rankings of model outputs to improve performance and alignment. Datasets-Labeler-Written Prompts: A collection of	Language models fine-tuned with RLHF show improved performance in following instructions, reducing harmful or biased outputs. Human feedback leads to more aligned and useful language models compared to purely unsupervised training.	Expensive and labor-intensive to gather high-quality human feedback. Potential for bias in feedback, depending on the annotators' preferences or perspectives.

			prompts and demonstrations illustrating desired behaviors.		
6.	Touvron, H., et al. LLaMA: Open and Efficient Foundation Language Models. arXiv preprint arXiv:2302.13971.	(2023)	LLaMA (Large Language Model Meta AI) is a family of efficient, open-source models offering strong performance with fewer parameters. It uses the Transformer architecture and is pre-trained on a mix of publicly available datasets like Common Crawl, Wikipedia, C4, and GitHub.	LLaMA models outperform larger models like GPT-3 while using **fewer parameters**, achieving state-of-the-art results across multiple tasks. Focuses on **efficiency** and **openness** to democratize access to powerful language models.	**Limited fine-tuning**examples and potential issues with **bias** due to dataset selection. **Resource-intensivepre-training**, though smaller in scale compared to other large models.
7.	Jiang, A. Q., et al.Mixtral of Experts. arXiv preprint arXiv:2401.04088.	(2024)	Mixtral uses a router network that selects two experts per token. SMoE architecture improves performance while reducing computational costs. Evaluations were conducted	Mixtral matches or surpasses LLaMA 2 70B in most areas, especially excelling in mathematics and code generation tasks. Mixtral's Instruct	Some overhead is introduced by the routing mechanism and memory load, making Mixtral better suited for batched workloads. Memory costs, while

			on benchmarks like MMLU, ARC Challenge, and HellaSwag.	version shows superior performance in instruction-following tasks with reduced bias.	still smaller than LLaMA 2, remain significant due to the model's size.
8.	Qiao, Z., et al. Weak-to-Strong Search: Align Large Language Models via Searching over Small Language Models. arXiv preprint arXiv:2402.04257.	(2024)	Tools-Machine Learning Frameworks: The study utilizes models from the Hugging Face Transformers library, including various tuned and untuned versions of the GPT-2 and LLaMA models. Techniques-Instruction Following: The technique is used to enhance the alignment of large models with human instructions using guidance from smaller models.	The weak-to-strong search effectively improves the performance of large models without requiring additional training, showcasing flexibility across different tasks. In particular, the approach significantly improved the win rates of both white-box and black-box models against benchmarks like GPT-4	Scalability: The search process may be computationally intensive as the size of the LLM and the number of weak models increase. Representational Power: The authors acknowledge that the weak models may not fully capture the complexity of human values, potentially limiting alignment effectiveness. Generalization: It remains unclear how well the approach would

			Datasets- Alpaca Eval 2.0: A benchmark dataset used for evaluating the instruction-following capabilities of models.	Turbo, with notable improvements in performance metrics (e.g., from 34.4 to 37.9 for Llama-3-70B-Instruct).	generalize across a broader range of human values and different AI systems.
9.	Mandra, S., et al. A Comparative Study of Neural Machine Translation Frameworks for the Automatic Translation of Open Data Resources. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018).	(2018)	**Tools-Frameworks**: Various neural machine translation (NMT) frameworks were evaluated, including:TensorFlow PyTorch **Techniques-Neural Machine Translation**: Employs encoder-decoder architectures for translation tasks. **Evaluation Metrics**: Uses BLEU scores and human evaluations to assess translation	The research demonstrates that NMT systems can achieve high-quality translations, showcasing the effectiveness of deep learning techniques in improving machine translation accuracy. The study highlights the efficiency and quality of translations generated by the developed NMT systems when deployed in	While the paper discusses the advancements in NMT, it acknowledges potential gaps in model performance across different languages and the challenge of handling domain-specific terminologies. It emphasizes the need for further exploration into optimizing these systems for better translation quality in diverse

			quality. **Datasets-Translation Datasets**: Open data resources for evaluating translation performance, specifically in domains such as government and educational content.	real-world environments like poliMedia.	linguistic contexts and for languages with less available data.
10.	Centeno, M., et al.. DeepAuth: A Framework for Continuous User Re-authent-on in Mobile Apps. In 2018 17th IEEE Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE Conference on Big Data Science and Engineering (TrustCom/BigDataSE).	(2018)	**Techniques**: The framework employs deep learning techniques to continuously authenticate users based on mobile device sensors, specifically focusing on touch and motion patterns. **Datasets**: The study utilizes data collected from volunteers to assess authentication accuracy.	DeepAuth achieves high accuracy (96.70%) in re-authenticating users within 20 seconds, demonstrating its effectiveness in real-time applications	The approach may face challenges with variations in user behavior and environmental factors affecting sensor data accuracy.

11.	Ambartsoumian , A., & Popowich, F. Self-Attention: A Better Building Block for Sentiment Analysis Neural Network Classifiers. In Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis.	(2018)	The paper introduces a simple **self-attention (SSAN)** architecture for sentiment analysis. It uses **Stanford Sentiment Treebank (SST-fine, SST-binary)** and other datasets like **OpeNER, SenTube, and SemEval**. Compared the SSAN with traditional models like **LSTM, BiLSTM**, and **CNN** using word embeddings.	**SSAN** outperform ed other architecture s like LSTM and CNN by better capturing long-range dependenci es. The **self-attenti on** mechanism was effective in sentiment classificatio n tasks, especially for datasets with longer sentences.	The model was only tested on a limited number of datasets. Further research is needed to explore its performance on a wider variety of languages and low-resource settings.
12.	Gupta, A., & Rush, A. M). Multi-resolution Transformer Networks: Recurrence is Not Essential for Modelling Hierarchical Structure. arXiv Preprint arXiv:2303.068	(2023)	Tools-PyTorch for model implementation and experiments. Techniques-Hierarchical Modeling: Employs a self-attention mechanism to capture long-range dependencies in data.	The proposed model shows a 20% improvement in precision scores and 25% improveme nt in BLEU scores on the AOL dataset	The paper mainly Focuses on query suggestion tasks, leaving the generalizatio n of the approach to complex hierarchical tasks in future

	49.		Datasets-Benchmark Datasets: The paper evaluates the model on various NLP tasks, including: Wikipedia text for language modeling. Standard datasets for sequence classification and natural language understanding.	compared to the best-performing recurrent models, indicating that recurrence is not essential for capturing hierarchical structure.	direction
13.	Aafaq, N., et al.. Multimodal architecture for video captioning with memory networks and an attention mechanism. Expert Systems with Applications, 118, 396-409.	(2018)	Tools-TensorFlow for model development and experimentation. Techniques-Multimodal Architecture: Combines visual and textual data for video captioning. Datasets-Video Captioning Datasets:	The proposed architecture achieves superior performance compared to other models. Notably, it outperforms many state-of-the-art video captioning methods, particularly in metrics like METEOR and	Although the model performs well on specific datasets, further testing on more diverse and real-world video content may be necessary to generalize the model's effectiveness. The reliance on predefined

			MSVD (Microsoft Video Description) dataset for training and evaluating the model.	CIDEr-D. The multimodal approach efficiently captures and processes both visual and semantic attributes, resulting in more contextually accurate video captions.	datasets might limit its applicability to more complex or less-structured video data.
14.	Xu, Y., et al). Multi-task Learning with Sample Re-weighting for Machine Reading Comprehension. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2	(2018)	**Tools-**TensorFlow for implementing machine learning models. **Techniques-Multi-task Learning**: Employs a multi-task framework to improve machine reading comprehension by simultaneously training on multiple tasks. **Datasets-Machine Reading**	**Performance Gains**: The multi-task learning approach significantly outperformed single-task models on various benchmarks. **Improved Generalization**: By focusing on multiple tasks and re-weighting challenging samples, the model	**Complexity**: The introduction of multi-task learning and re-weighting increases the complexity of the training process. **Task Interference**: While MTL generally improves performance, the paper mentions that in some cases, tasks can interfere with each other, which may lead to suboptimal

	(Short Papers).		**Comprehension Datasets**: SQuAD (Stanford Question Answering Dataset) for training and evaluation.	demonstrated better generalization to unseen tasks.	results for certain domains.
15.	Nielsen, M. A. Neural Networks and Deep Learning. Determination Press.	(2015)	Tools-The book discusses implementation using programming languages such as Python, along with libraries like NumPy for numerical computations. Techniques-Neural Network Architectures: Covers various architectures including feedforward networks, convolutional neural networks (CNNs), and recurrent neural networks (RNNs). Datasets-Examples in Learning:	Nielsen provides insight into how deeper layers in a neural network allow for better feature extraction and hierarchical learning. The book shows that deep learning is capable of outperforming traditional machine learning algorithms on tasks such as image recognition and natural language processing when	While the book provides a strong conceptual foundation, it does not cover the most recent advances in deep learning techniques beyond 2015, such as transformer models and modern architectures. The examples are mainly limited to small-scale problems and do not address the challenges of training on large datasets or dealing with modern GPU acceleration.

			While the book does not focus on specific datasets, it uses illustrative examples and synthetic datasets to explain concepts.	provided with sufficient data and computational resources.	
16.	Zhang, W., et al.. Large Language Models for Robotics: A Survey. arXiv preprint arXiv:2307.02782.	(2023)	**Tools-**PyTorch and TensorFlow for implementing various models and algorithms discussed. **Techniques-Transfer Learning**: Leveraging pre-trained large language models (LLMs) to enhance robotic applications. **Datasets-Benchmark Datasets**: Various publicly available datasets for training and evaluation, including:	Various multimodal datasets are referenced, including those with image-language pairs and robotic task-oriented datasets like **RoboTask**, **RT-2**, and others that help in aligning LLMs with physical actions and real-world interactions. **Generalization**: LLMs improve generalization in robotics, making robots better at multitasking and	**Physical Limitations**: LLMs are still limited in their ability to translate abstract language understanding into precise physical actions, particularly in real-world settings. **Data Efficiency**: The models require large-scale, diverse data to perform well, which may not always be feasible for certain robotic applications. **Real-time Adaptability**: LLMs

			OpenAI's GPT datasets.	interacting in dynamic environments. **Task Execution**: LLMs enable robots to perform complex tasks by using human-like reasoning and language understanding. **Human-Robot Interaction**: LLMs enhance natural language communication between humans and robots, improving collaboration and task assignment.	struggle with real-time decision-making in rapidly changing environments, which limits their use in high-speed robotic applications.
17.	Brohan, A., et al). RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic	(2023)	Tools-PyTorch for implementing the models and training processes. Techniques-Vision-Language-Action Model:	RT-2 outperformed other robotics models by showing improved generalization to novel objects,	RT-2's performance is limited by the physical skills present in its training data, meaning it can't acquire new robotic

	Control. arXiv preprint arXiv:2307.158 18.		Integrates visual inputs, natural language processing, and action execution to facilitate robotic control. Datasets-Robotic Control Datasets: Datasets specific to robotic tasks used for evaluating the model's performance in practical scenarios.	environments, and semantic instructions. It demonstrated emergent abilities such as semantic reasoning (e.g., recognizing symbolic relationships between objects) and handling complex human instructions. RT-2 leverages web-scale pretraining, leading to more effective performance compared to models trained only on robot data.	skills beyond those seen during training. The model is computationally expensive, and future work is suggested to reduce size and increase efficiency, such as using model distillation or quantization
18.	Yang, C., et al). Exploring Large Language Model based Intelligent Agents: Definitions, Methods, and	(2023)	Tools-PyTorch and TensorFlow for model development and experimentation. Techniques-Inte	LLM-based agents exhibit strong generalization capabilities and are effective at understandi	LLM-based agents face issues with context length, computational overhead for training, and a lack of direct tool

	Prospects. arXiv preprint arXiv:2308.08691.		lligent Agent Framework: Examines how large language models (LLMs) can be utilized as intelligent agents to perform tasks and make decisions. Datasets-Benchmark Datasets: Utilizes various datasets for training and evaluation, including: OpenAI's datasets for fine-tuning LLMs.	ng and generating natural language across a variety of applications. They are advantageous in tasks requiring reasoning, task decomposition, and interactive problem-solving.	utilization (e.g., calculators or code interpreters). Transfer learning remains a challenge in multi-task learning, particularly for domain-specific agents. Additionally, communication and coordination challenges exist in multi-agent systems.
19.	Huang, X., et al. Understanding Large-Language Model (LLM) powered Human Robot Interaction. arXiv preprint arXiv:2312.04000.	(2023)	Tools-PyTorch and TensorFlow for implementing LLMs and associated models in the context of human-robot interaction. Techniques-Large Language Models (LLMs): Explores the	LLM-powered robots were effective in building connections and facilitating deliberation in human-robot interaction. While robots excelled in fostering	The robot agent struggled with handling logical tasks effectively, which caused user discomfort. Future work needs to focus on fine-tuning LLMs specifically for robot

			application of LLMs to facilitate natural communication and interaction between humans and robots. Datasets-Human-Robot Interaction Datasets: Utilizes datasets that include dialogues and interactions specifically curated for training and evaluating models in robotic contexts	interactive engagement, participants reported increased anxiety when using LLM-powered robots due to difficulties in logical communication.	communication to improve coherence and reduce user anxiety.
20.	Khattak, F. K., et al. A survey of word embeddings for clinical text. Journal of biomedical informatics: X, 4, 100057.	(2019)	Reviews various word embedding techniques (e.g., Word2Vec, GloVe) applied to clinical text. Discusses available clinical text corpora and pre-trained clinical word	Highlights the effectiveness of word embeddings in clinical NLP tasks. Provides a framework for evaluating the intrinsic and extrinsic	Notes challenges in adapting general embeddings for clinical contexts. Suggests the need for more specialized datasets and techniques for better

			vector embeddings.	performance of different embeddings.	clinical performance.
21.	Xu, Pengjun, et al. Large Language Models Meet NLP: A Survey	(2024)	**Techniques:** Zero-shot, few-shot learning, fine-tuning, LoRA, prefix tuning. **Datasets:** SQuAD, GLUE, OpenAI benchmarks, domain-specific datasets.	Strong performance of LLMs across understanding, generation, and multimodal tasks. Zero-shot and few-shot learning enhance adaptability without extensive data.	Hallucinations, bias, computational cost, and poor performance in long-tail and domain-specific tasks.
22.	Li, Z., Chen, H., Liu, Z., & Tang, J. (2024). A Survey on Recent Advances in LLM-Based Multi-turn Dialogue Systems.	(2024)	**Tools/Techniques:** Decoder-only transformers (e.g., GPT series), RLHF fine-tuning, and multimodal LLM systems. **Datasets:** WebText, Wikipedia, and large token corpora.	Progress in handling multi-turn dialogues with better context relevance and lower toxicity (e.g., GPT-4). Enhanced multimodal and alignment capabilities for conversatio	Challenges in generating truthful responses, high computational costs, and biases. Privacy risks and the need for better real-time feedback mechanisms.

				nal AI.	
23.	Wang, H., Wang, L., Du, Y., Chen, L., Zhou, J., Wang, Y., & Wong, K.-F. (2023). A Survey of the Evolution of Language Model-Based Dialogue Systems.	(2023)	Pre-trained models like DialoGPT, ChatGPT; conversational datasets (e.g., Reddit, domain-specific corpora).	Transition from task-oriented (TOD) to open-domain dialogue systems (ODD). Improved performance through scaling models and datasets.	Lack of a unified perspective on dialogue system evolution and ongoing challenges in fine-tuning for diverse applications
24.	Ngoc Tran Khanh Le, Nadia Hadiprodjo, Hazem El-Alfy, and Avtandil Teshebaev. The Recent Large Language Models in NLP.	(2023)	Focus on LLMs like BERT, GPT-3, and LLaMA. Benchmarks: GLUE, SQuAD.	Improved task performance (e.g., text generation, translation). Open-source models democratize technology use.	High computational costs. Challenges: bias, fairness, interpretability, and domain-specific fine-tuning.
25.	Lake, B. M., Ullman, T. D., Tenenbaum, J.	(2016)	**Tools/Techniques:** Probabilistic programming,	Models emulate human-like learning	Scalability issues with large-scale tasks.

	B., & Gershman, S. J. (2016). Building machines that learn and think like people. Behavioral and Brain Sciences, 40, e253.		Bayesian models, hybrid neural-symbolic systems, intuitive physics and psychology simulations. **Datasets:** Synthetic datasets replicating human cognitive tasks	(one-shot learning, causal reasoning, flexible reasoning). Focus on interpretable, efficient, and adaptable AI systems.	Challenges in integrating symbolic and statistical methods. Incomplete modeling of human cognition (e.g., emotions, social dynamics). Limited

2.2 Key Gaps in the Literature

- **Limited Focus on Dialogue Coherence**

  Most of the available systems stand out in typical language understanding and generation tasks; they fail, however, in maintaining the coherence and context of dialogues over several turns. Such gaps are highly critical for the applications that require long conversational turns, like those found in customer support or tutoring systems.

- **Insufficiently Aligned with Human Intent**

  There have been certain improvements made with respect to aligning LLMs with human wills using techniques such as reinforcement learning with human feedback, but complete and precise alignment is evaded for rather subtle conversational contexts.

- **Insufficient Adaptation for Domain-Specific Applications**

  Generalized benchmarks are utilized by almost all studies. Few applications, such as healthcare, education, and legal advising, are left unattended regarding domain-specific applications. Adapting LLMs to specialized dialogue requirements remains a significant challenge.

Chapter 03: System Development

3.1 Requirements and Analysis

At this point, we will enumerate every prerequisite that is required for the development of a model. The different technologies, libraries, and tools that are employed.

3.1.1 Programming Language: The Basis for Readability and All-Inclusive Libraries

Python is a popular, advanced, and versatile programming language. Its vast libraries and ease of reading has contributed to its global appeal. It is the best option for numerous applications, such as data analysis, machine learning and artificial intelligence.

- Readability: Python emphasizes clear and concise syntax, making it easier to write, understand, and maintain the readability of the code. This is especially advantageous in large-scale or team-based projects where readability improves collaboration and reduces errors.
- Large Library: Python offers an extensive collection of libraries and tools that support various domains, including machine learning(TensorFlow, PyTorch), data science(NumPy, Pandas), and deep learning (keras, scikit-learn). These tools accelerate the development and empower developers to build complex systems.

3.1.2 Software Resources

- **CUDA (Version: 10.1 or above):** Enables GPU-based acceleration, crucial for handling the intensive computations involved in training large models.
- **WinSCP (Version: 6.2 or above):** Used for secure file exchange between a local machine and HCP servers, ensuring smooth remote data management.
- **CyberDuck (Version: 8.7 or above):** A versatile application supporting FTP, SFTP, and cloud services, reliable file transfer across platforms.
- **VSCode / Jupyter Notebook (Version: 1.83.0 or above):** Provides a robust environment for writing, debugging and running code, as well as conducting interactive

experiments throughout the development process.

Table 2. Required Tools and Environment.

Category	Description	
Software Resources	● CUDA ● WinSCP ● CyberDuck ● VsCode / Jupyter Notebook	Version: 10.1 or above Version: 6.2 or above Version: 8.7 or above Version: 1.83.0 or above
Hardware Resources	● High-Performance Computing (HPC) System [Ramanujan Universe] ● NVIDIA GPUs (e.g., V100, A100) ● 512 GB storage, Large RAM (64GB or more)	
Others	● Cloud services for deployment ● High speed Internet Connection ● IEEE and Springer Access	

3.1.3 Hardware Resources

- **High-Performance Computing (HPC) System [Ramanujan Universe]:** Delivers the processing capability needed to train and tune LLMS using large-scale datasets.
- **NVIDIA GPUs (e.g., V100, A100):** Advanced GPUs that significantly speed up deep learning computations, enabling model training and experimentation.
- **512 GB Storage and Large RAM (64GB or more):** Ensures sufficient space and memory for handling large models and datasets during the training and evaluation process.

3.2 Project Design and Architecture

This chapter outlines the system architecture and design approach for developing a dialog-LLM based on the Transformer framework. The project is structured into modular stages, with adaptability and efficient performance.

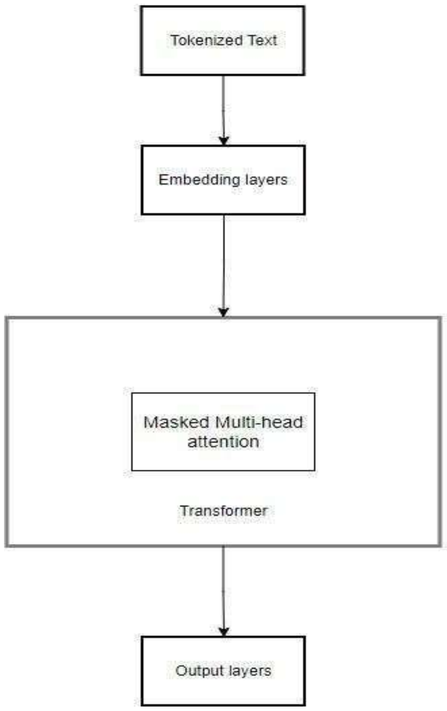


Fig. 3. LLM System Architecture

The architecture of the project is divided into distinct, interconnected stages:

1. **Data Preparation Module**: Handles the acquisition, cleaning, formatting, and visualization of raw data to prepare it for downstream processing.

2. **Tokenization and Encoding**: Segments the input text into manageable tokens and converts them into numerical formats suitable for machine learning workflows.

3. **Embedding Layer**: Converts tokenized data into a dense vector, incorporating both token identity and positional information to retain linguistic context.

4. **Context Vector Generation**: Builds context-aware vectors from embeddings, which serve as the semantic foundation for model tasks.

5. **Model Pre-training and Fine-tuning**: Involves two phases- pretraining on large generic corpora to establish understanding, followed by fine-tuning on dialog datasets to adapt to conversational use cases.

6. **Evaluation and Testing Framework**: Implement a testing framework to assess performance using predefined metrics and scenarios to validate the model's capabilities.

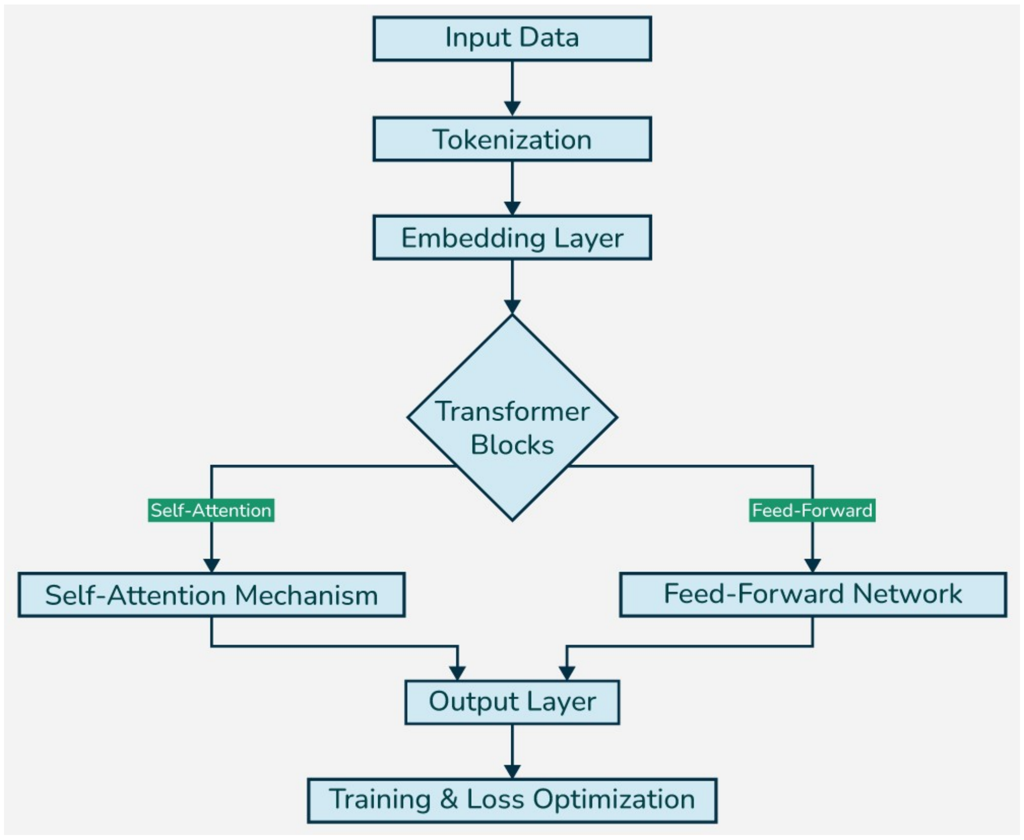


Fig. 4. Detailed representation of the Architecture of LLM

Fig.3 and Fig.4 shows the abstract representation providing a high-level overview of the model's pipeline and detailed representation of architecture of the large language model development process delving deeper into the detailed representation of the development

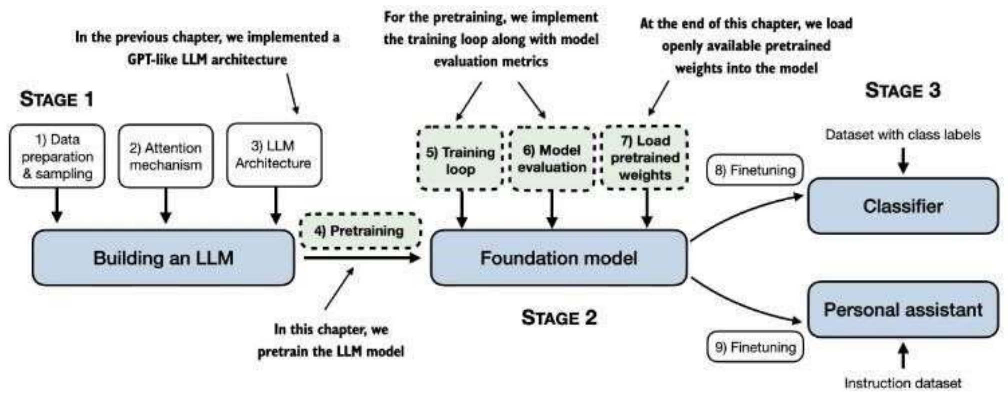

In the previous chapter, we implemented a GPT-like LLM architecture
For the pretraining, we implement the training loop along with model evaluation metrics
At the end of this chapter, we load openly available pretrained weights into the model
STAGE 1
STAGE 3
1) Data preparation & sampling
2) Attention mechanism
3) LLM Architecture
5) Training loop
6) Model evaluation
7) Load pretrained weights
8) Finetuning
Dataset with class labels
Classifier
4) Pretraining
Building an LLM
Foundation model
Personal assistant
STAGE 2
9) Finetuning
In this chapter, we pretrain the LLM model
Instruction dataset

Fig. 5. Implement a GPT Model from Scratch

In this chapter, we implement the training loop and code for basic model evaluation to pretrain an LLM. At the end of this chapter, we also load openly available pretrained weights from OpenAI into our model.

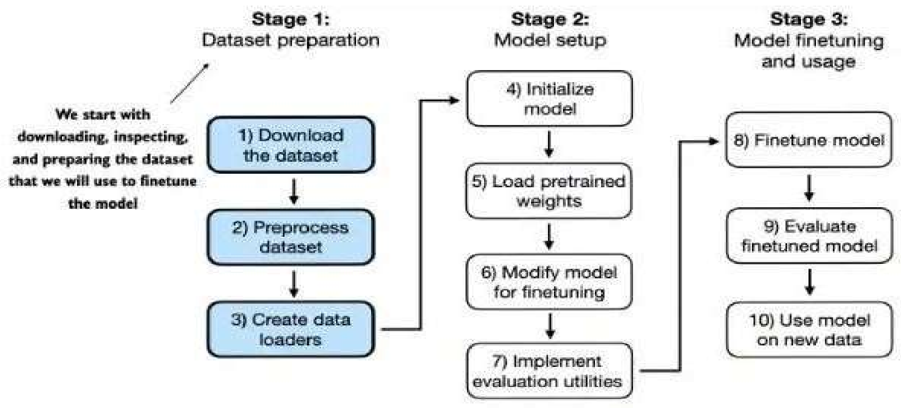

Stage 1: Dataset preparation
Stage 2: Model setup
Stage 3: Model finetuning and usage
We start with downloading, inspecting, and preparing the dataset that we will use to finetune the model
1) Download the dataset
2) Preprocess dataset
3) Create data loaders
4) Initialize model
5) Load pretrained weights
6) Modify model for finetuning
7) Implement evaluation utilities
8) Finetune model
9) Evaluate finetuned model
10) Use model on new data

Fig. 6. Preparing the dataset for the Finetuning Classification

This section prepares the dataset we use for classification fine-tuning. We use a dataset consisting of spam and non-spam text messages to fine-tune the LLM to classify them

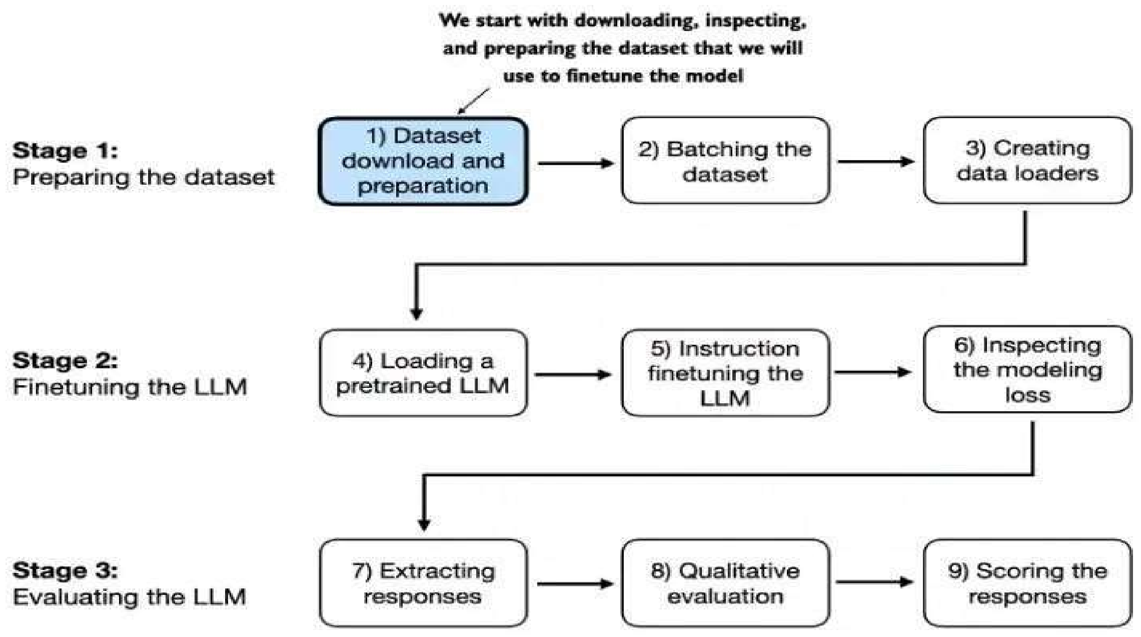

We start with downloading, inspecting, and preparing the dataset that we will use to finetune the model
Stage 1: Preparing the dataset
1) Dataset download and preparation
2) Batching the dataset
3) Creating data loaders
Stage 2: Finetuning the LLM
4) Loading a pretrained LLM
5) Instruction finetuning the LLM
6) Inspecting the modeling loss
Stage 3: Evaluating the LLM
7) Extracting responses
8) Qualitative evaluation
9) Scoring the responses

Fig. 7. LLM Model Preparation Evaluation

3.2 Data Preparation

3.1.1 Dataset Gathering:

The first step involves collecting the dataset for Pre-training model and dataset for Fine tuning model individually. The datasets should be large enough for ample pre-training and fine-tuning the model. The large datasets are also preferred, to enhance the model capabilities to produce high quality text responses. We have gathered data from huggingface library. The dataset used for pre-training is a subset(5GB) of Slimpajam-627B (825GB) dataset and the Alpaca Dataset for fine-tuning.

3.3.2 Data Processing

The next step includes cleaning the data to remove noise, null values, missing values, outlier, and other inconsistencies. We have employed several data preprocessing steps such as lowercasing the dataset, removing punctuation (except “,” , “?” , “.”), checking for null values, and missing text values.

3.3.3 Data Visualization and Analysis:

In this step, we have visualized the data with help of bar-plot and histograms to have a clear picture about the dataset distribution. Subsequently, we have done data analysis to capture numerical characteristics of our dataset that include finding average length of text, minimum and maximum length of text, and total number of characters in our dataset.

3. 4 Data Preparation and Sampling:

In this step, we transform and prepare the dataset into a form that is accepted by the LLM architecture to further train, test and validate the model. It includes tokenizing the dataset into a primitive unit called token(s). These tokens represent individual words of the dataset.

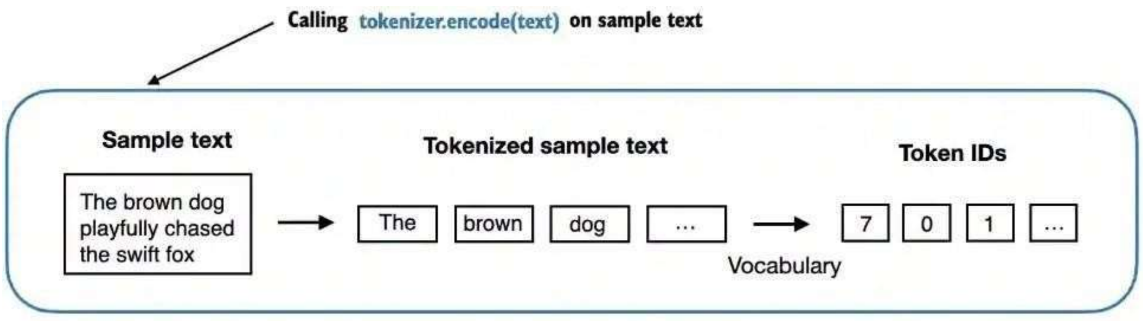

Calling tokenizer.encode(text) on sample text
Sample text
The brown dog playfully chased the swift fox
Tokenized sample text
The
brown
dog
...
Vocabulary
Token IDs
7
0
1
...

Fig. 8. Token Encoding

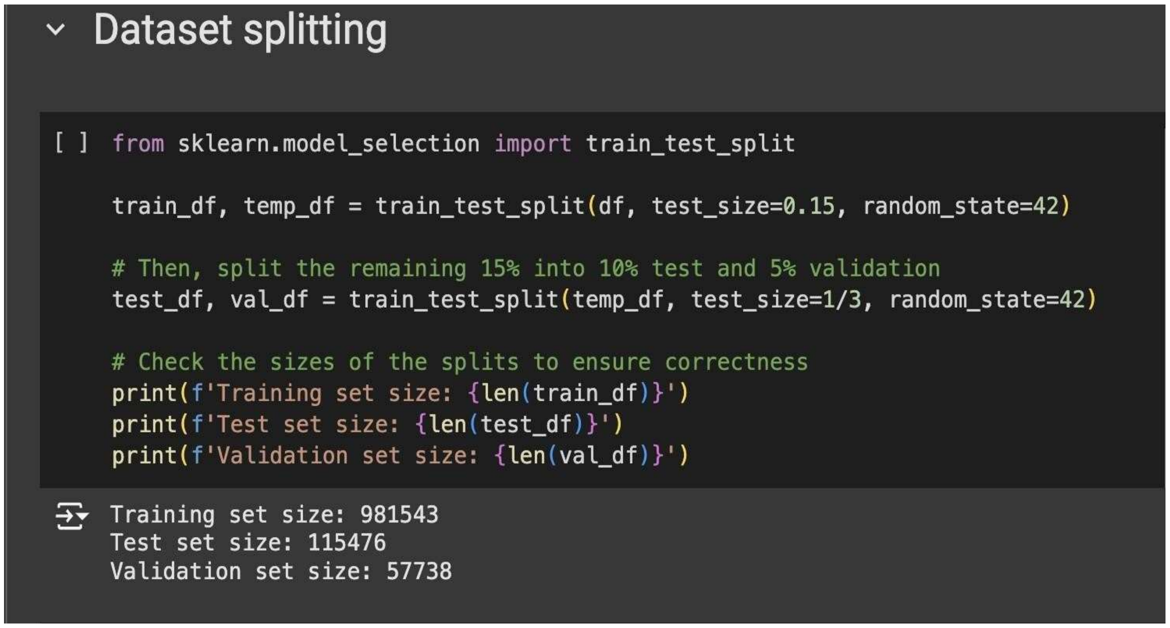
Fig. 4. shows the process of tokenizing the text. As shown in the above example the input, "The brown dog playfully chased the swift fox," is first split into tokens (words or subwords) such as "The," "brown," "dog," etc. These tokens are then mapped to their respective token IDs based on a predefined vocabulary, facilitating numerical representation for further processing in machine learning models.

Next step involves converting these words or tokens into token_ids. Each token is assigned an integer value. This process is called Encoding (Bytepair - Encoding). Further these tokens ids are sampled into tensors of fixed size batches.

3.4 Implementation

The implementation phase draws on the already preprocessed set of data to develop the main elements of the dialogue-optimized Large Language Model (LLM). Up to this point, the work has centered on the early steps of the model pipeline in its development.

Fig. 9 shows the code snippet, representing the implementation of data preprocessing function that normalizes text by removing unnecessary characters while retaining specific punctuation marks that might help the model. This process ensures that the dataset is cleaned and ready for downstream tasks such as tokenization and embedding. The next steps involve breaking down the processed text paragraphs into individual tokens and encoding them using integer id.

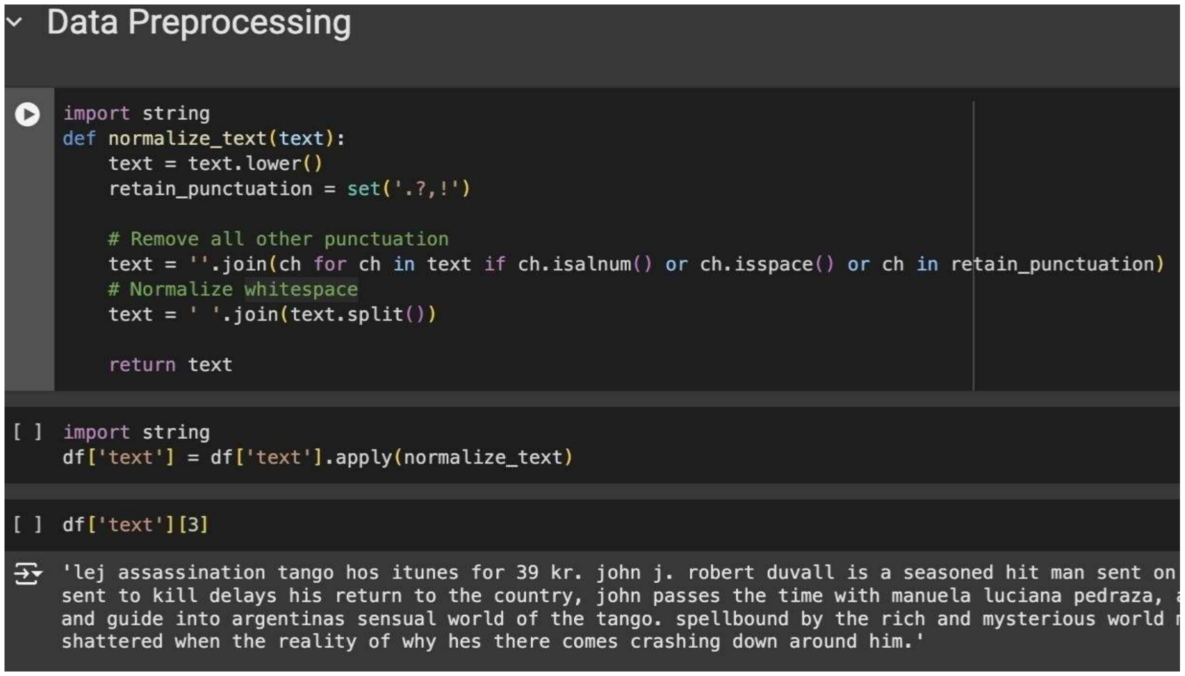

Dataset splitting

```
from sklearn.model_selection import train_test_split

train_df, temp_df = train_test_split(df, test_size=0.15, random_state=42)

# Then, split the remaining 15% into 10% test and 5% validation
test_df, val_df = train_test_split(temp_df, test_size=1/3, random_state=42)

# Check the sizes of the splits to ensure correctness
print(f'Training set size: {len(train_df)}')
print(f'Test set size: {len(test_df)}')
print(f'Validation set size: {len(val_df)}')
```

```
Training set size: 981543
Test set size: 115476
Validation set size: 57738
```

Fig. 9. Data Preparation and Refinement

Fig. 9 illustrates how the dataset is divided into training, validation, and testing sets, a crucial step to ensure that the model is evaluated on separate data partitions and minimize the risk of overfitting.


Data Preprocessing

```
import string
def normalize_text(text):
    text = text.lower()
    retain_punctuation = set('.?,!')

    # Remove all other punctuation
    text = ''.join(ch for ch in text if ch.isalnum() or ch.isspace() or ch in retain_punctuation)
    # Normalize whitespace
    text = ' '.join(text.split())

    return text
```

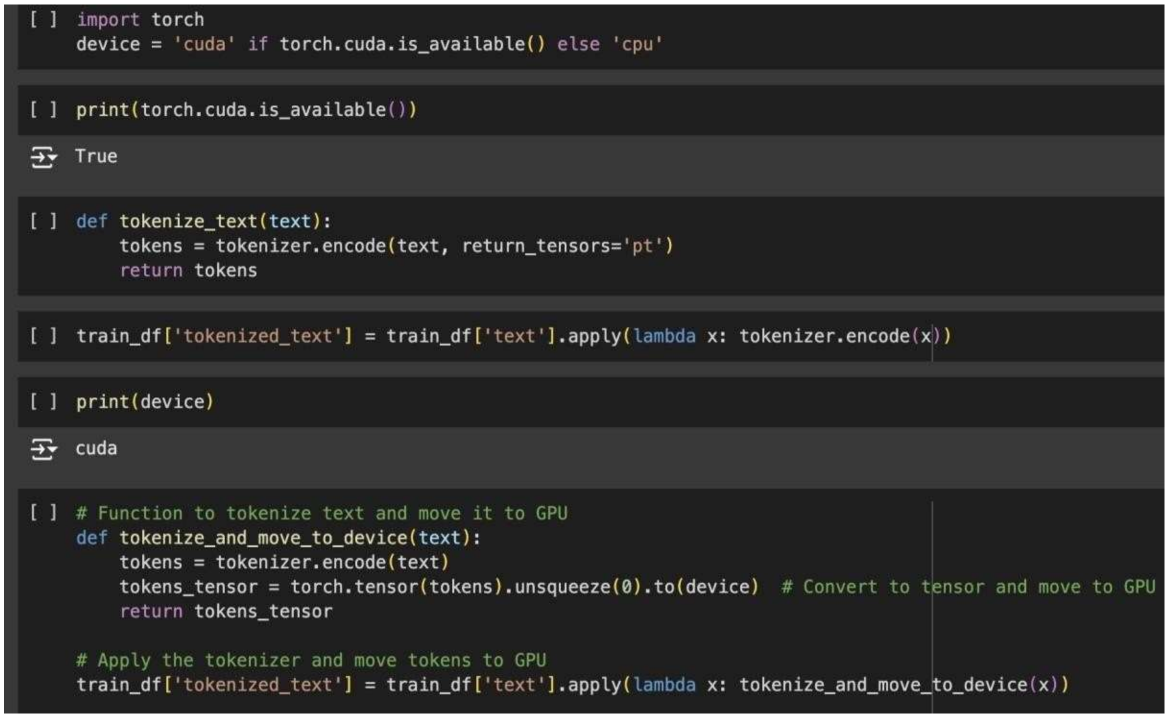
```
import string
df['text'] = df['text'].apply(normalize_text)
```

```
df['text'][3]
```

```
'lej assassination tango hos itunes for 39 kr. john j. robert duvall is a seasoned hit man sent on
sent to kill delays his return to the country, john passes the time with manuela luciana pedraza,
and guide into argentinas sensual world of the tango. spellbound by the rich and mysterious world
shattered when the reality of why hes there comes crashing down around him.'
```

Fig. 10. Data Processing and Transformation

Figure 10 shows the text preprocessing procedure, involving normalization steps like lowercasing and trimming extra spaces. These steps standardize the dataset, preparing it effectively for tokenization and model training.



```
[ ] import torch
    device = 'cuda' if torch.cuda.is_available() else 'cpu'

[ ] print(torch.cuda.is_available())

    True

[ ] def tokenize_text(text):
        tokens = tokenizer.encode(text, return_tensors='pt')
        return tokens

[ ] train_df['tokenized_text'] = train_df['text'].apply(lambda x: tokenizer.encode(x))

[ ] print(device)

    cuda

[ ] # Function to tokenize text and move it to GPU
    def tokenize_and_move_to_device(text):
        tokens = tokenizer.encode(text)
        tokens_tensor = torch.tensor(tokens).unsqueeze(0).to(device)  # Convert to tensor and move to GPU
        return tokens_tensor

    # Apply the tokenizer and move tokens to GPU
    train_df['tokenized_text'] = train_df['text'].apply(lambda x: tokenize_and_move_to_device(x))
```

Fig. 11. Processed Data Tokenized

Fig. 11 demonstrates the tokenization process of textual data, preparing it for model input. The code efficiently utilizes GPU resources to accelerate computation, tokenize the text with a tokenizer, and transfer the tokens to the GPU. This step plays a key role in optimizing the training by leveraging hardware acceleration.

3.4.1 Project Setup

- The development was carried out in **Python** using the **PyTorch** deep learning framework.
- The environment included GPU support for efficient model training and inference.
- Codebase was modularized into configuration scripts, model definitions, training loops,

and evaluation routines.

3.4.2 Model Architecture

- The model follows a **decoder-only Transformer architecture** similar to GPT.
- Key components:
    - **12 Transformer blocks** with multi-head self-attention
    - **12 attention heads**
    - **Embedding dimension:** 768
    - **Context window:** 1024 tokens
- Used learnable **token** and **positional embeddings**.
- Output is passed through a **linear projection layer** for vocabulary prediction.

3.4.3 Pretraining on Unlabeled Data

- Pretraining was performed using a **causal language modeling (CLM)** objective.
- Input text was tokenized using **Byte Pair Encoding (BPE)**.
- Model was trained to predict the next token in a sequence, optimizing **cross-entropy loss**.
- Checkpoints and training logs were maintained to monitor **loss curves and learning stability**.

3.4.4 Fine-Tuning for Classification

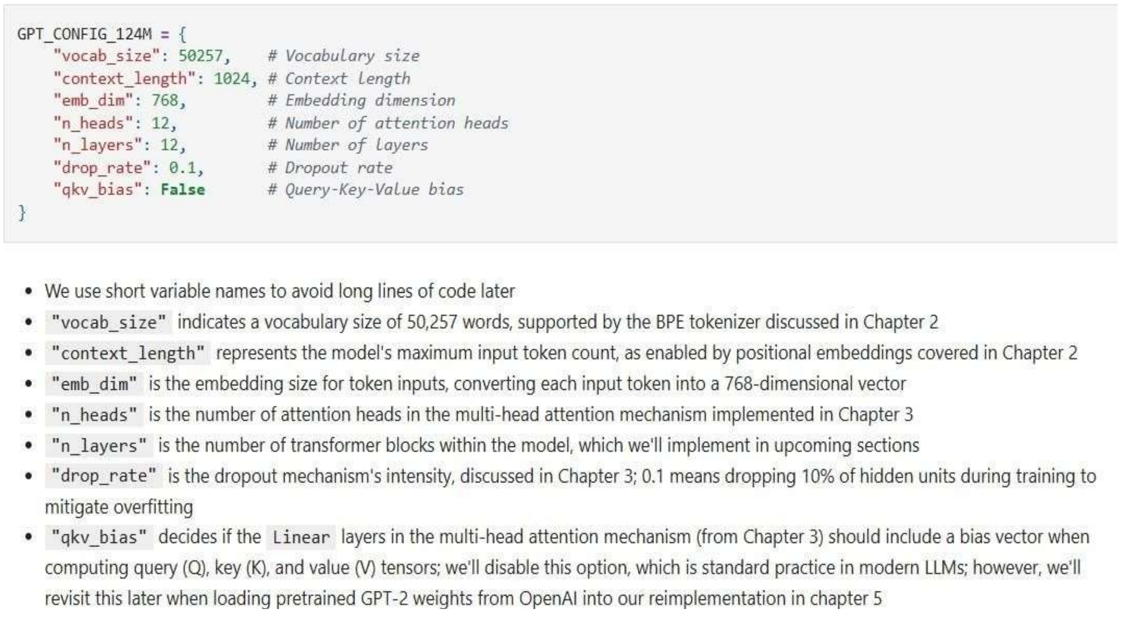
- A classification head was added on top of the pretrained model.
- Model was fine-tuned on labeled datasets for **text classification** tasks.
- Trained with labeled prompts and class labels using **supervised learning**.
- Evaluated using **accuracy**, **precision**, **recall**, and **F1-score**.

3.4.5 Instruction Tuning

- The model was fine-tuned to follow user instructions using **prompt-response datasets**.
- Input consisted of instructions/prompts; output was the expected response.
- This stage focused on making the model more aligned with **natural language instructions**.
- Responses were evaluated for **relevance, coherence, and instruction adherence**.

3.4.6 Output Interface and Evaluation

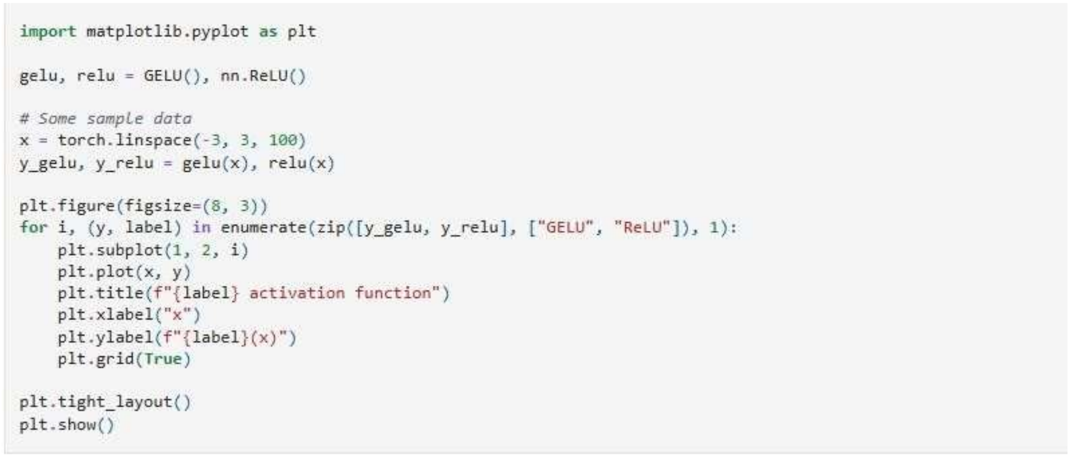
- A simple API was developed using **FastAPI** to test inference interactively.
- Tokenized input prompts were fed through the model and decoded into readable responses.
- Response quality was evaluated over **multi-turn conversations** to ensure context awareness.
- Logs and metrics were collected to aid in final reporting and presentation.



```
GPT_CONFIG_124M = {
    "vocab_size": 50257,    # Vocabulary size
    "context_length": 1024, # Context length
    "emb_dim": 768,         # Embedding dimension
    "n_heads": 12,          # Number of attention heads
    "n_layers": 12,         # Number of layers
    "drop_rate": 0.1,       # Dropout rate
    "qkv_bias": False       # Query-Key-Value bias
}
```

- We use short variable names to avoid long lines of code later
- `"vocab_size"` indicates a vocabulary size of 50,257 words, supported by the BPE tokenizer discussed in Chapter 2
- `"context_length"` represents the model's maximum input token count, as enabled by positional embeddings covered in Chapter 2
- `"emb_dim"` is the embedding size for token inputs, converting each input token into a 768-dimensional vector
- `"n_heads"` is the number of attention heads in the multi-head attention mechanism implemented in Chapter 3
- `"n_layers"` is the number of transformer blocks within the model, which we'll implement in upcoming sections
- `"drop_rate"` is the dropout mechanism's intensity, discussed in Chapter 3; 0.1 means dropping 10% of hidden units during training to mitigate overfitting
- `"qkv_bias"` decides if the `Linear` layers in the multi-head attention mechanism (from Chapter 3) should include a bias vector when computing query (Q), key (K), and value (V) tensors; we'll disable this option, which is standard practice in modern LLMs; however, we'll revisit this later when loading pretrained GPT-2 weights from OpenAI into our reimplementation in chapter 5

Fig. 12. GPT Transformer Initialization Parameter



```
import matplotlib.pyplot as plt

gelu, relu = GELU(), nn.ReLU()

# Some sample data
x = torch.linspace(-3, 3, 100)
y_gelu, y_relu = gelu(x), relu(x)

plt.figure(figsize=(8, 3))
for i, (y, label) in enumerate(zip([y_gelu, y_relu], ["GELU", "ReLU"]), 1):
    plt.subplot(1, 2, i)
    plt.plot(x, y)
    plt.title(f"{label} activation function")
    plt.xlabel("x")
    plt.ylabel(f"{label}(x)")
    plt.grid(True)

plt.tight_layout()
plt.show()
```

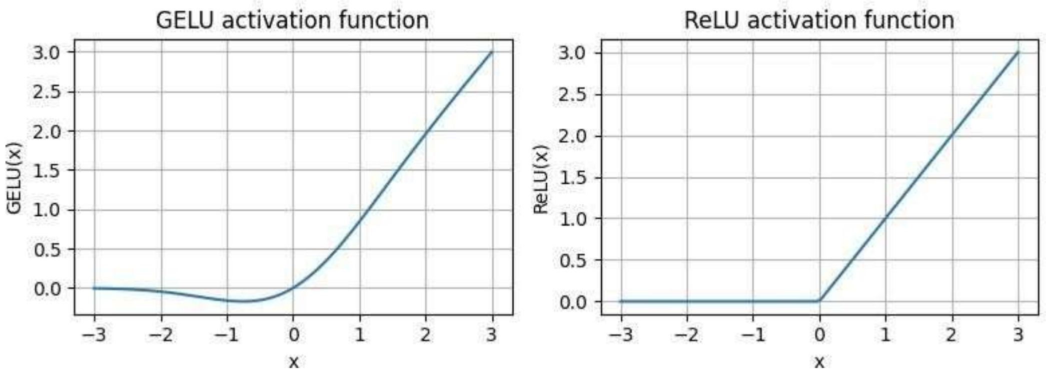
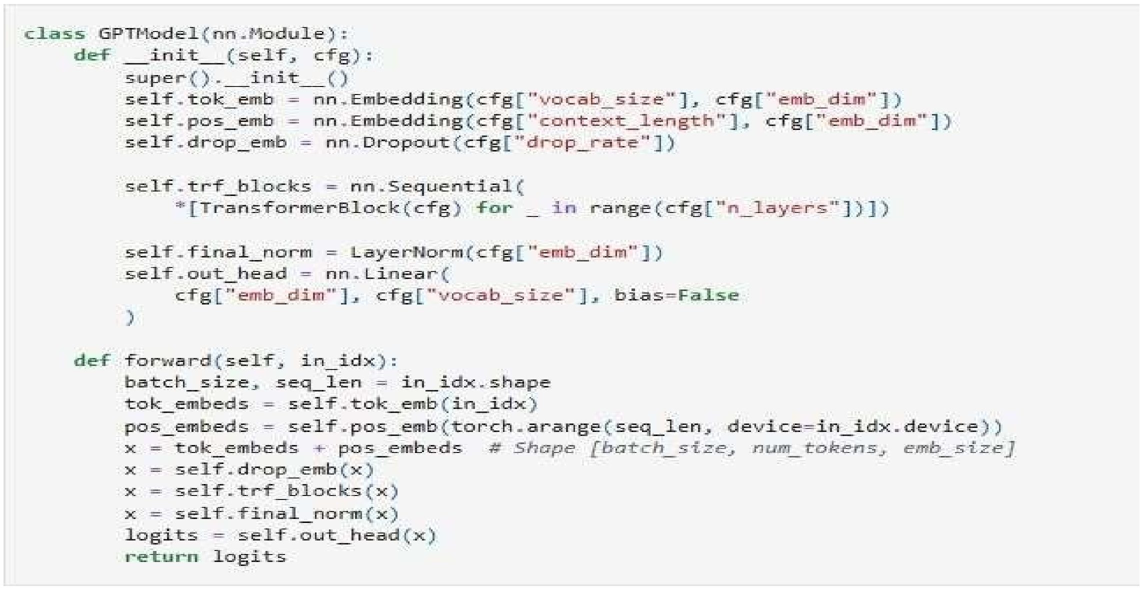


Fig. 13. Effect of GELU and RELU on Input Transformation

- As we can see, ReLU is a piecewise linear function that outputs the input directly if it is positive; otherwise, it outputs zero.

- GELU is a smooth, non-linear function that approximates ReLU but with a non-zero gradient for negative values (except at approximately -0.75).

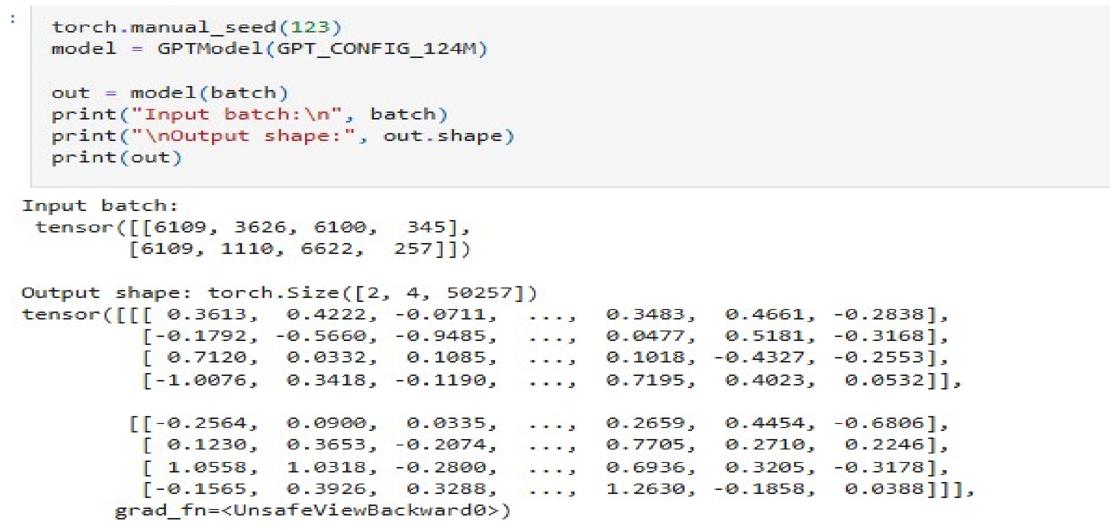
- Next, let's implement the small neural network module, FeedForward, that we will be using in the LLM's transformer block later.



```python
class GPTModel(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.tok_emb = nn.Embedding(cfg["vocab_size"], cfg["emb_dim"])
        self.pos_emb = nn.Embedding(cfg["context_length"], cfg["emb_dim"])
        self.drop_emb = nn.Dropout(cfg["drop_rate"])

        self.trf_blocks = nn.Sequential(
            *[TransformerBlock(cfg) for _ in range(cfg["n_layers"])])

        self.final_norm = LayerNorm(cfg["emb_dim"])
        self.out_head = nn.Linear(
            cfg["emb_dim"], cfg["vocab_size"], bias=False
        )

    def forward(self, in_idx):
        batch_size, seq_len = in_idx.shape
        tok_embeds = self.tok_emb(in_idx)
        pos_embeds = self.pos_emb(torch.arange(seq_len, device=in_idx.device))
        x = tok_embeds + pos_embeds  # Shape [batch_size, num_tokens, emb_size]
        x = self.drop_emb(x)
        x = self.trf_blocks(x)
        x = self.final_norm(x)
        logits = self.out_head(x)
        return logits
```

Fig. 14. GPT Model Implementation

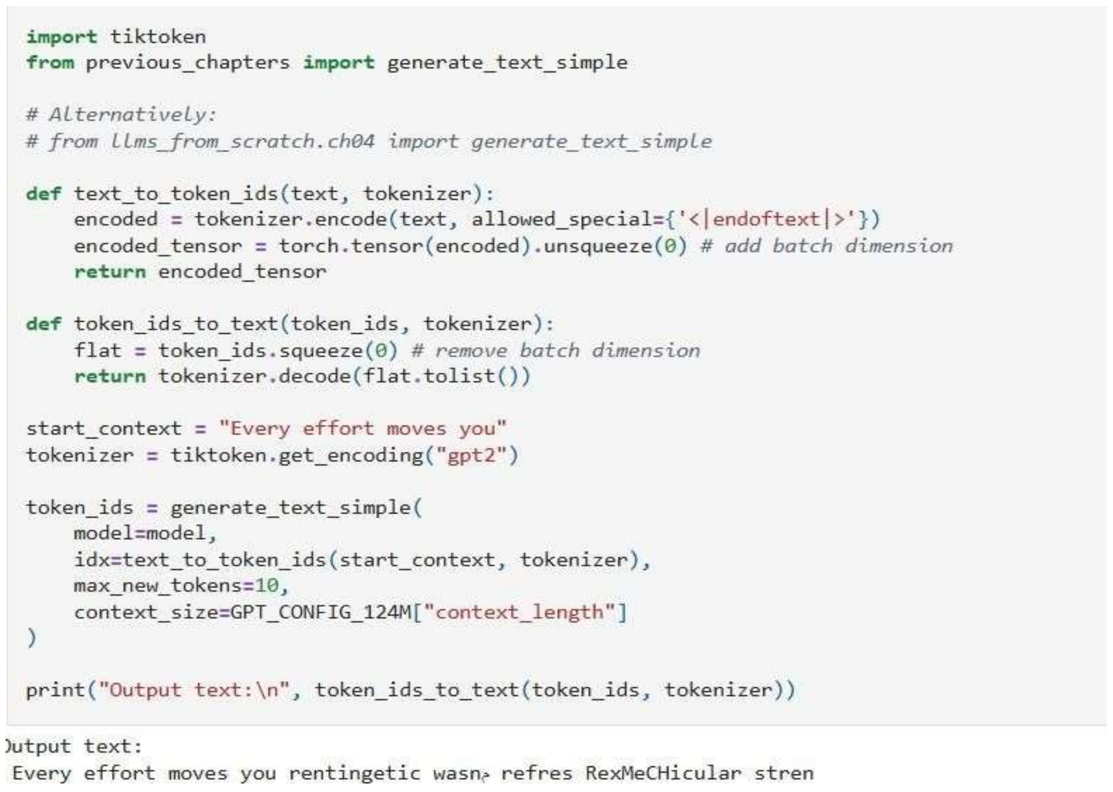
Using the configuration of the 124M parameter model, we can now instantiate this GPT model with random initial weights as follows:



```
torch.manual_seed(123)
model = GPTModel(GPT_CONFIG_124M)

out = model(batch)
print("Input batch:\n", batch)
print("\nOutput shape:", out.shape)
print(out)
```

```
Input batch:
 tensor([[6109, 3626, 6100,  345],
        [6109, 1110, 6622,  257]])

Output shape: torch.Size([2, 4, 50257])
tensor([[[ 0.3613,  0.4222, -0.0711,  ...,  0.3483,  0.4661, -0.2838],
         [-0.1792, -0.5660, -0.9485,  ...,  0.0477,  0.5181, -0.3168],
         [ 0.7120,  0.0332,  0.1085,  ...,  0.1018, -0.4327, -0.2553],
         [-1.0076,  0.3418, -0.1190,  ...,  0.7195,  0.4023,  0.0532]],

        [[-0.2564,  0.0900,  0.0335,  ...,  0.2659,  0.4454, -0.6806],
         [ 0.1230,  0.3653, -0.2074,  ...,  0.7705,  0.2710,  0.2246],
         [ 1.0558,  1.0318, -0.2800,  ...,  0.6936,  0.3205, -0.3178],
         [-0.1565,  0.3926,  0.3288,  ...,  1.2630, -0.1858,  0.0388]]],
       grad_fn=<UnsafeViewBackward0>)
```

Fig. 15. GPT Model Output Example

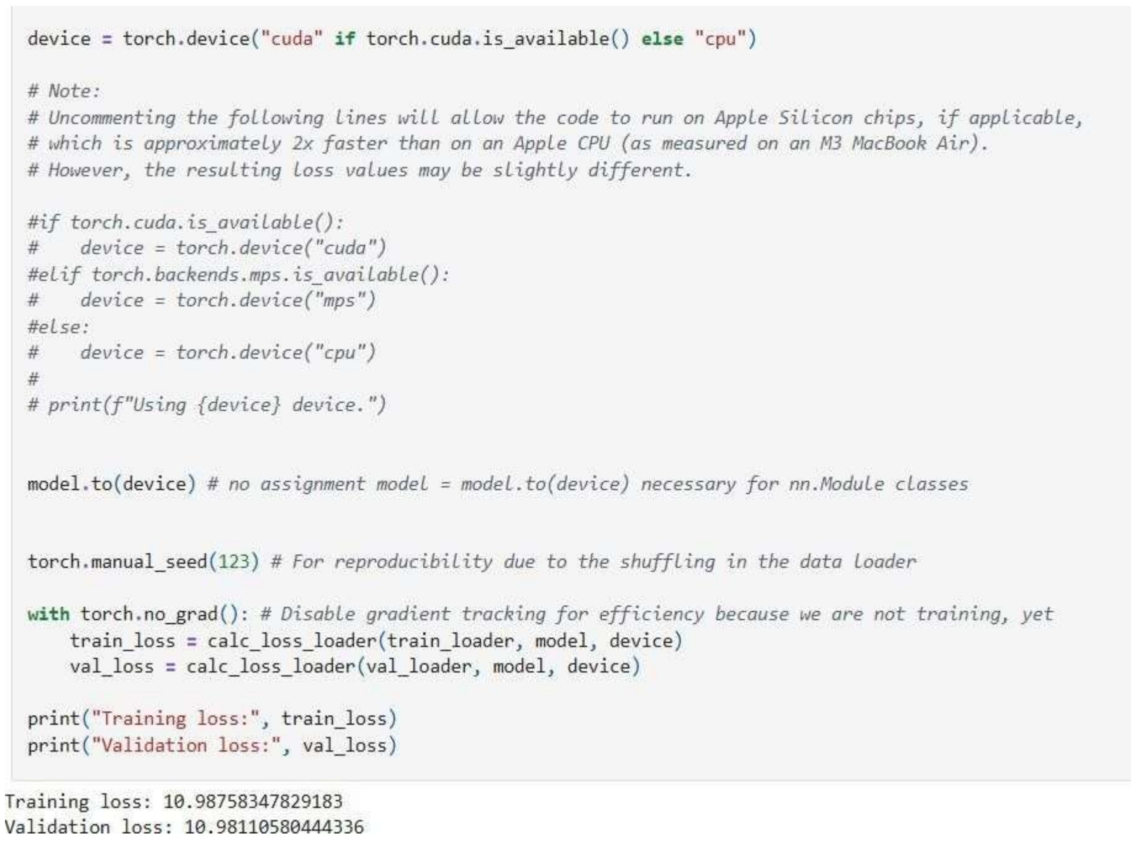


```
import tiktoken
from previous_chapters import generate_text_simple

# Alternatively:
# from llms_from_scratch.ch04 import generate_text_simple

def text_to_token_ids(text, tokenizer):
    encoded = tokenizer.encode(text, allowed_special={'<|endoftext|>'})
    encoded_tensor = torch.tensor(encoded).unsqueeze(0) # add batch dimension
    return encoded_tensor

def token_ids_to_text(token_ids, tokenizer):
    flat = token_ids.squeeze(0) # remove batch dimension
    return tokenizer.decode(flat.tolist())

start_context = "Every effort moves you"
tokenizer = tiktoken.get_encoding("gpt2")

token_ids = generate_text_simple(
    model=model,
    idx=text_to_token_ids(start_context, tokenizer),
    max_new_tokens=10,
    context_size=GPT_CONFIG_124M["context_length"]
)

print("Output text:\n", token_ids_to_text(token_ids, tokenizer))
```

```
Output text:
 Every effort moves you rentingetic wasnم refres RexMeCHicular stren
```

Fig. 16. Simple Text Generation Using GPT Model

- As we can see above, the model does not produce good text because it has not been trained yet.
- How do we measure or capture what "good text" is, in a numeric form, to track it during training?
- The next subsection introduces metrics to calculate a loss metric for the generated outputs that we can use to measure the training progress.
- The next chapters on finetuning LLMs will also introduce additional ways to measure model quality.

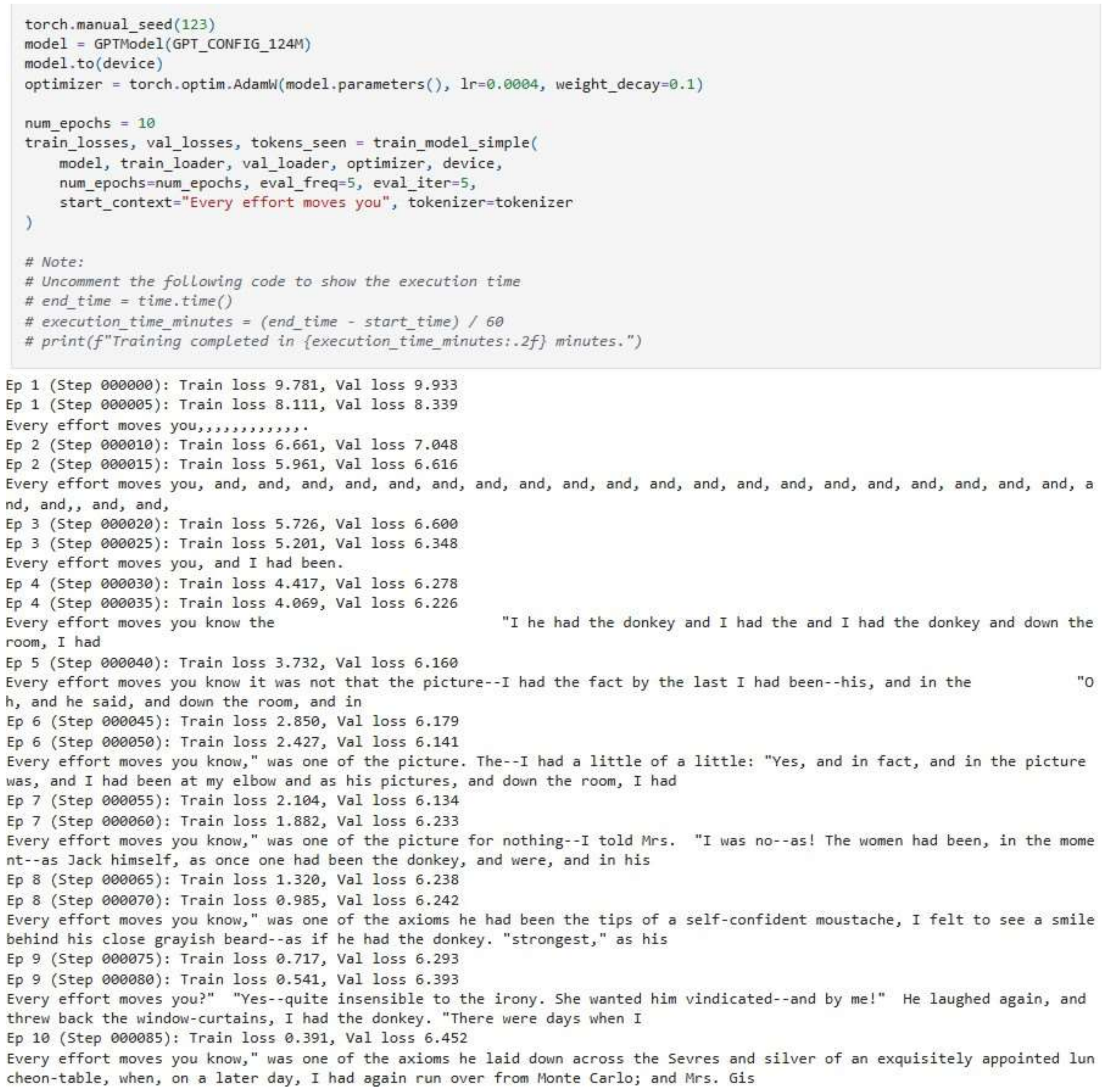


```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Note:
# Uncommenting the following lines will allow the code to run on Apple Silicon chips, if applicable,
# which is approximately 2x faster than on an Apple CPU (as measured on an M3 MacBook Air).
# However, the resulting loss values may be slightly different.

#if torch.cuda.is_available():
#    device = torch.device("cuda")
#elif torch.backends.mps.is_available():
#    device = torch.device("mps")
#else:
#    device = torch.device("cpu")
#
# print(f"Using {device} device.")


model.to(device) # no assignment model = model.to(device) necessary for nn.Module classes


torch.manual_seed(123) # For reproducibility due to the shuffling in the data loader

with torch.no_grad(): # Disable gradient tracking for efficiency because we are not training, yet
    train_loss = calc_loss_loader(train_loader, model, device)
    val_loss = calc_loss_loader(val_loader, model, device)

print("Training loss:", train_loss)
print("Validation loss:", val_loss)
```

```
Training loss: 10.98758347829183
Validation loss: 10.98110580444336
```

Fig. 17. Training and Validation Loss Calculation

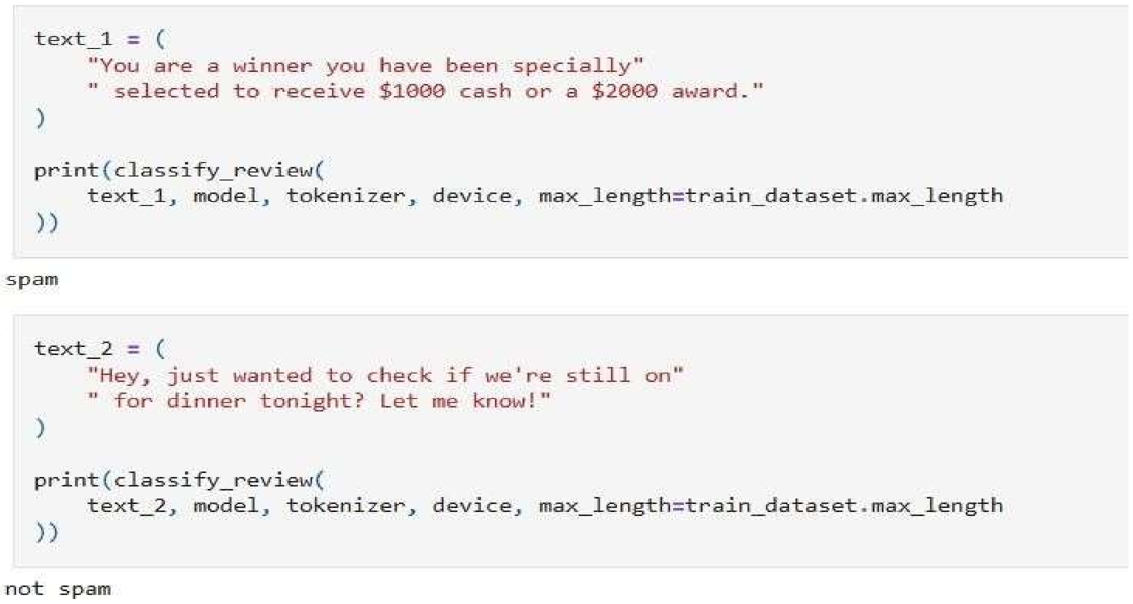
- If you have a machine with a CUDA-supported GPU, the LLM will train on the GPU without making any changes to the code.
- Via the device setting, we ensure that the data is loaded onto the same device as the LLM model.



```
torch.manual_seed(123)
model = GPTModel(GPT_CONFIG_124M)
model.to(device)
optimizer = torch.optim.AdamW(model.parameters(), lr=0.0004, weight_decay=0.1)

num_epochs = 10
train_losses, val_losses, tokens_seen = train_model_simple(
    model, train_loader, val_loader, optimizer, device,
    num_epochs=num_epochs, eval_freq=5, eval_iter=5,
    start_context="Every effort moves you", tokenizer=tokenizer
)

# Note:
# Uncomment the following code to show the execution time
# end_time = time.time()
# execution_time_minutes = (end_time - start_time) / 60
# print(f"Training completed in {execution_time_minutes:.2f} minutes.")
```

```
Ep 1 (Step 000000): Train loss 9.781, Val loss 9.933
Ep 1 (Step 000005): Train loss 8.111, Val loss 8.339
Every effort moves you,,,,,,,,,,,,.
Ep 2 (Step 000010): Train loss 6.661, Val loss 7.048
Ep 2 (Step 000015): Train loss 5.961, Val loss 6.616
Every effort moves you, and, and, and, and, and, and, and, and, and, and, and, and, and, and, and, and, and, and, and, and, a
nd, and,, and, and,
Ep 3 (Step 000020): Train loss 5.726, Val loss 6.600
Ep 3 (Step 000025): Train loss 5.201, Val loss 6.348
Every effort moves you, and I had been.
Ep 4 (Step 000030): Train loss 4.417, Val loss 6.278
Ep 4 (Step 000035): Train loss 4.069, Val loss 6.226
Every effort moves you know the                          "I he had the donkey and I had the and I had the donkey and down the
room, I had
Ep 5 (Step 000040): Train loss 3.732, Val loss 6.160
Every effort moves you know it was not that the picture--I had the fact by the last I had been--his, and in the           "O
h, and he said, and down the room, and in
Ep 6 (Step 000045): Train loss 2.850, Val loss 6.179
Ep 6 (Step 000050): Train loss 2.427, Val loss 6.141
Every effort moves you know," was one of the picture. The--I had a little of a little: "Yes, and in fact, and in the picture
was, and I had been at my elbow and as his pictures, and down the room, I had
Ep 7 (Step 000055): Train loss 2.104, Val loss 6.134
Ep 7 (Step 000060): Train loss 1.882, Val loss 6.233
Every effort moves you know," was one of the picture for nothing--I told Mrs.  "I was no--as! The women had been, in the mome
nt--as Jack himself, as once one had been the donkey, and were, and in his
Ep 8 (Step 000065): Train loss 1.320, Val loss 6.238
Ep 8 (Step 000070): Train loss 0.985, Val loss 6.242
Every effort moves you know," was one of the axioms he had been the tips of a self-confident moustache, I felt to see a smile
behind his close grayish beard--as if he had the donkey. "strongest," as his
Ep 9 (Step 000075): Train loss 0.717, Val loss 6.293
Ep 9 (Step 000080): Train loss 0.541, Val loss 6.393
Every effort moves you?"  "Yes--quite insensible to the irony. She wanted him vindicated--and by me!"  He laughed again, and
threw back the window-curtains, I had the donkey. "There were days when I
Ep 10 (Step 000085): Train loss 0.391, Val loss 6.452
Every effort moves you know," was one of the axioms he laid down across the Sevres and silver of an exquisitely appointed lun
cheon-table, when, on a later day, I had again run over from Monte Carlo; and Mrs. Gis
```

Fig. 18. Epoch Output and Loss Evaluation



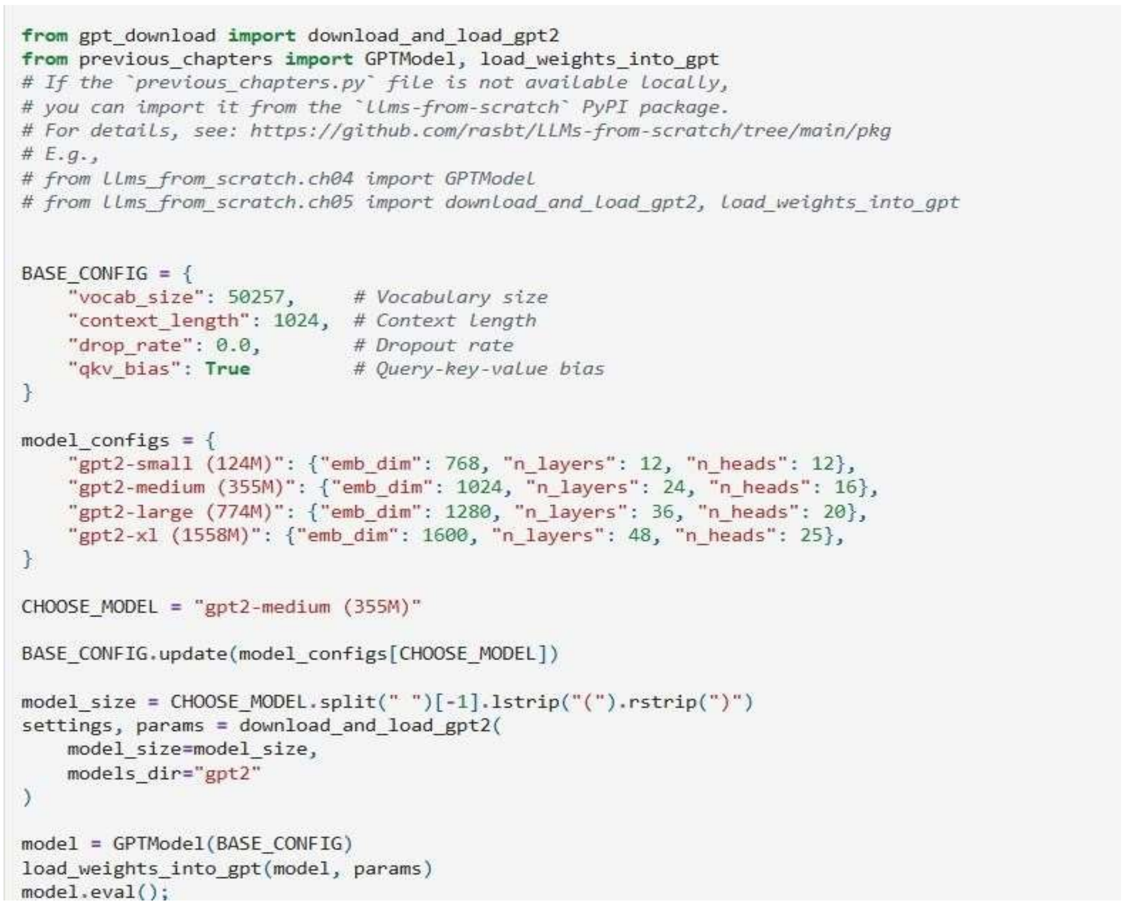
```
text_1 = (
    "You are a winner you have been specially"
    " selected to receive $1000 cash or a $2000 award."
)

print(classify_review(
    text_1, model, tokenizer, device, max_length=train_dataset.max_length
))
```

```
spam
```

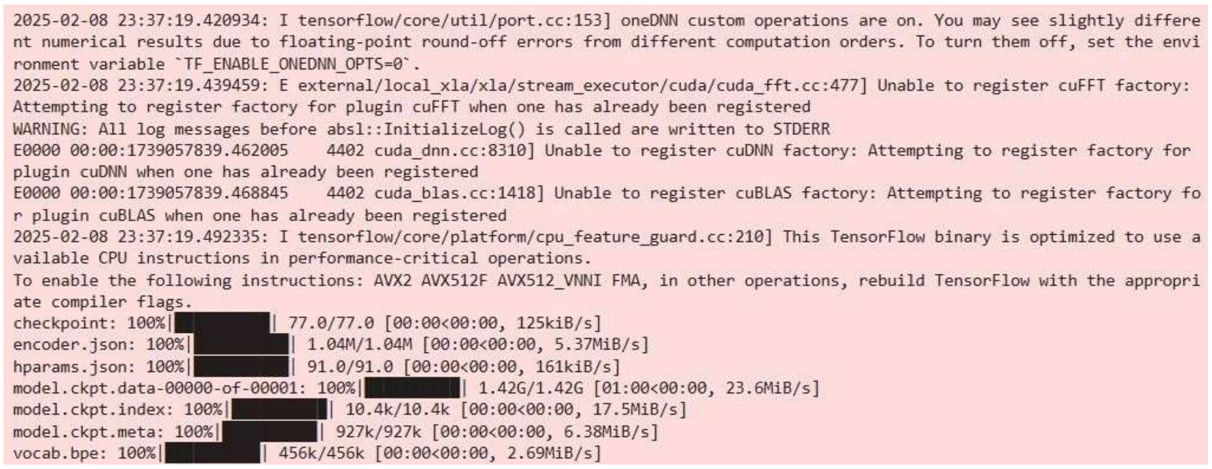
```
text_2 = (
    "Hey, just wanted to check if we're still on"
    " for dinner tonight? Let me know!"
)

print(classify_review(
    text_2, model, tokenizer, device, max_length=train_dataset.max_length
))
```

```
not spam
```

Fig. 19. Spam vs Not Spam Classification Example



```
from gpt_download import download_and_load_gpt2
from previous_chapters import GPTModel, load_weights_into_gpt
# If the `previous_chapters.py` file is not available locally,
# you can import it from the `llms-from-scratch` PyPI package.
# For details, see: https://github.com/rasbt/LLMs-from-scratch/tree/main/pkg
# E.g.,
# from llms_from_scratch.ch04 import GPTModel
# from llms_from_scratch.ch05 import download_and_load_gpt2, load_weights_into_gpt


BASE_CONFIG = {
    "vocab_size": 50257,     # Vocabulary size
    "context_length": 1024,  # Context length
    "drop_rate": 0.0,        # Dropout rate
    "qkv_bias": True         # Query-key-value bias
}

model_configs = {
    "gpt2-small (124M)": {"emb_dim": 768, "n_layers": 12, "n_heads": 12},
    "gpt2-medium (355M)": {"emb_dim": 1024, "n_layers": 24, "n_heads": 16},
    "gpt2-large (774M)": {"emb_dim": 1280, "n_layers": 36, "n_heads": 20},
    "gpt2-xl (1558M)": {"emb_dim": 1600, "n_layers": 48, "n_heads": 25},
}

CHOOSE_MODEL = "gpt2-medium (355M)"

BASE_CONFIG.update(model_configs[CHOOSE_MODEL])

model_size = CHOOSE_MODEL.split(" ")[-1].lstrip("(").rstrip(")")
settings, params = download_and_load_gpt2(
    model_size=model_size,
    models_dir="gpt2"
)

model = GPTModel(BASE_CONFIG)
load_weights_into_gpt(model, params)
model.eval();
```



```
2025-02-08 23:37:19.420934: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly differe
nt numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the envi
ronment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-02-08 23:37:19.439459: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register cuFFT factory:
Attempting to register factory for plugin cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1739057839.462005    4402 cuda_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register factory for
plugin cuDNN when one has already been registered
E0000 00:00:1739057839.468845    4402 cuda_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register factory fo
r plugin cuBLAS when one has already been registered
2025-02-08 23:37:19.492335: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use a
vailable CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropri
ate compiler flags.
checkpoint: 100%|██████████| 77.0/77.0 [00:00<00:00, 125kiB/s]
encoder.json: 100%|██████████| 1.04M/1.04M [00:00<00:00, 5.37MiB/s]
hparams.json: 100%|██████████| 91.0/91.0 [00:00<00:00, 161kiB/s]
model.ckpt.data-00000-of-00001: 100%|██████████| 1.42G/1.42G [01:00<00:00, 23.6MiB/s]
model.ckpt.index: 100%|██████████| 10.4k/10.4k [00:00<00:00, 17.5MiB/s]
model.ckpt.meta: 100%|██████████| 927k/927k [00:00<00:00, 6.38MiB/s]
vocab.bpe: 100%|██████████| 456k/456k [00:00<00:00, 2.69MiB/s]
```

Fig. 20. Building GPT-2 from Scratch: A step-by-step Initialization

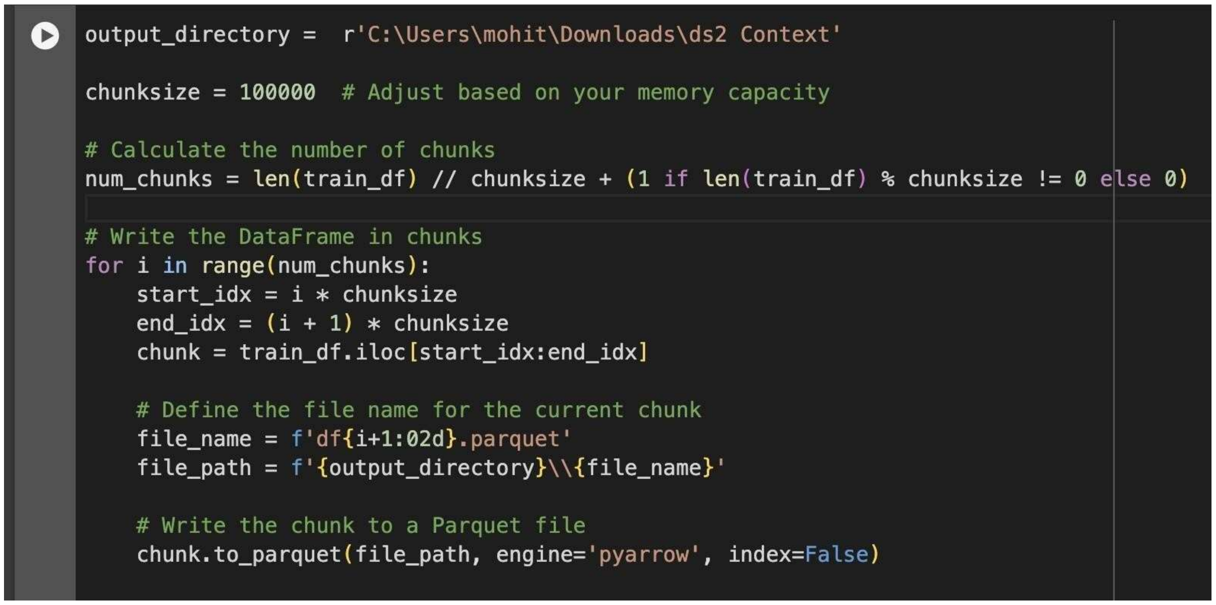
3.5 Key Challenges

The Project's development encountered various technical and resource constraints. This section outlines the key challenges faced and the corresponding solutions to overcome them efficiently.

1. Data Processing Memory Constraints

A key challenge involved the inability to process the entire dataset due to limited memory. The large size of the dataset exceeds available RAM, causing slowdowns and potential system failures.

Solution:
To address this limitation, the dataset was split into smaller segments for sequential processing. By leveraging the chunksize parameter, each segment was independently processed and stored in a format. This approach ensured efficient memory usage throughout the workflow. The corresponding code snippet illustrates this method:



```
output_directory =  r'C:\Users\mohit\Downloads\ds2 Context'

chunksize = 100000  # Adjust based on your memory capacity

# Calculate the number of chunks
num_chunks = len(train_df) // chunksize + (1 if len(train_df) % chunksize != 0 else 0)

# Write the DataFrame in chunks
for i in range(num_chunks):
    start_idx = i * chunksize
    end_idx = (i + 1) * chunksize
    chunk = train_df.iloc[start_idx:end_idx]

    # Define the file name for the current chunk
    file_name = f'df{i+1:02d}.parquet'
    file_path = f'{output_directory}\\{file_name}'

    # Write the chunk to a Parquet file
    chunk.to_parquet(file_path, engine='pyarrow', index=False)
```

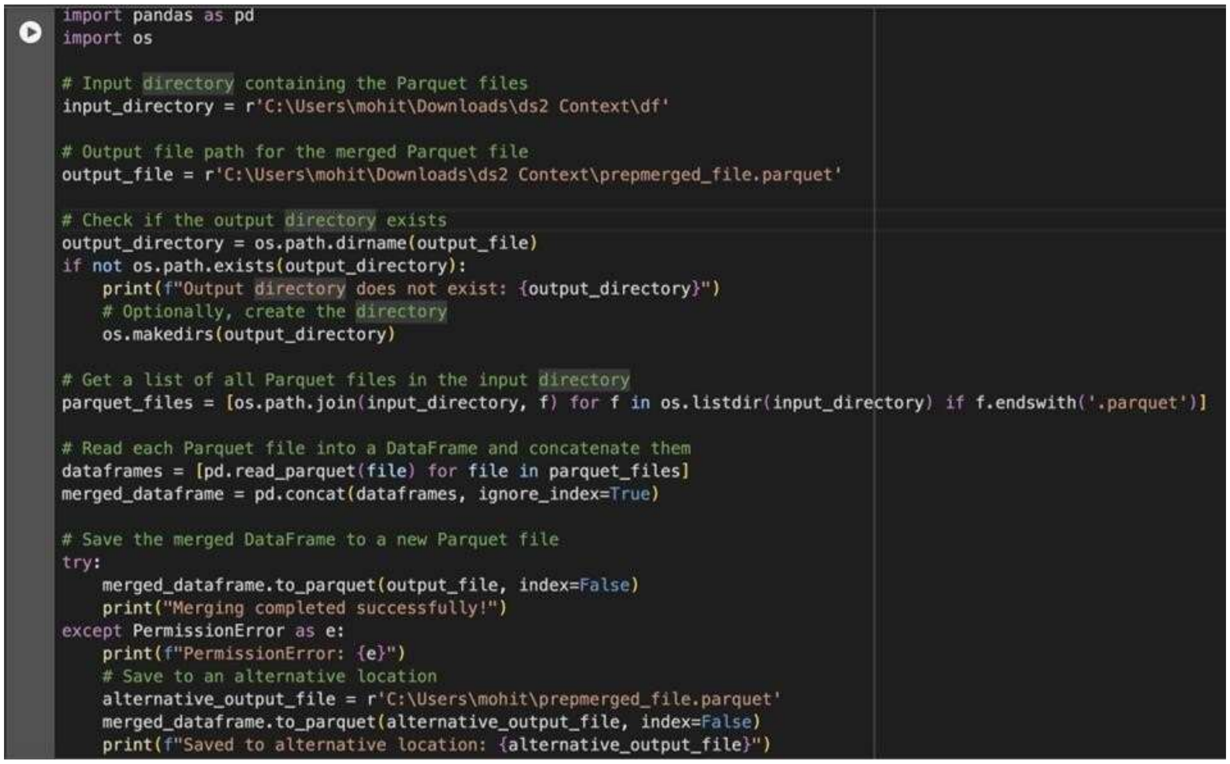
Fig. 21. Chunk-wise Data Processing Script

This chunk-based processing approach enabled efficient data handling, making it suitable for systems with constrained memory resources.

2. File Merging and Consolidation

After processing the data in chunks, merging into a unified dataset presented a challenge, as it required precise handling of multiple Parquet files and accurate concatenation.

Solution:
Each Parquet file was read in sequence and combined into a single DataFrame, which was then saved as a unified Parquet file. The implementation included error handling for issues like file access by specifying a fallback save locations:



```
import pandas as pd
import os

# Input directory containing the Parquet files
input_directory = r'C:\Users\mohit\Downloads\ds2 Context\df'

# Output file path for the merged Parquet file
output_file = r'C:\Users\mohit\Downloads\ds2 Context\prepmerged_file.parquet'

# Check if the output directory exists
output_directory = os.path.dirname(output_file)
if not os.path.exists(output_directory):
    print(f"Output directory does not exist: {output_directory}")
    # Optionally, create the directory
    os.makedirs(output_directory)

# Get a list of all Parquet files in the input directory
parquet_files = [os.path.join(input_directory, f) for f in os.listdir(input_directory) if f.endswith('.parquet')]

# Read each Parquet file into a DataFrame and concatenate them
dataframes = [pd.read_parquet(file) for file in parquet_files]
merged_dataframe = pd.concat(dataframes, ignore_index=True)

# Save the merged DataFrame to a new Parquet file
try:
    merged_dataframe.to_parquet(output_file, index=False)
    print("Merging completed successfully!")
except PermissionError as e:
    print(f"PermissionError: {e}")
    # Save to an alternative location
    alternative_output_file = r'C:\Users\mohit\prepmerged_file.parquet'
    merged_dataframe.to_parquet(alternative_output_file, index=False)
    print(f"Saved to alternative location: {alternative_output_file}")
```

Fig. 22. Script to Combine Processed Data Chunks into a Single Dataset

Fig. 22 illustrates the method used to aggregate individual processed chunks into a unified DataFrame for downstream tasks.

Chapter 04: Testing

4.1 Testing Strategy

This chapter outlines the testing methodology adapted to identify bottlenecks and verify the integrity of the code. The focus was on validating the accuracy, performance, and components such as tokenization, embeddings, and context vector generation. The testing approach comprised the following elements:

Unit Testing:

Functions like text normalization, tokenization, and embedding generation were independently validated to correct functionality. Python's unittest framework and manual inspections in Jupyter Notebook or Google Collab were used to assess and debug intermediate results.

Functional Testing:

Verified the complete pipeline - from raw text input to context vector generation- functioned correctly, maintaining data integrity. Key checkpoints were used to compare actual outputs with expected results to detect discrepancies or failures.

Tools Used:

- **Jupyter Notebook**: Used for interactive execution and real-time visualization of pipeline results.
- **PyTorch Tensor Debugging**: Validated tensor shapes, gradients, and data flow during embedding and context vector generation.
- **Visualization Libraries**: Tools like **Matplotlib** and **Seaborn** were used to visually analyze data distributions.

4.1 Test Cases and Outcomes

Table 3 shows some of the example test cases, their input, expected output, actual output and the final status of test cases.

Table 3. Example test cases followed during the trail and testing

Test Case ID	Test Description	Input	Expected Outcome	Actual Outcome	Status
TC-01	Verify text normalization function removes unwanted characters	"Hello, World! "	"hello world"	"hello world"	Passed
TC-02	Validate tokenizer handles long text sequences	Long paragraph	Tokens list of appropriate length	Tokens generated	Passed
TC-03	Check token embeddings are of expected dimensions	[1, 512] token sequence	Embedding tensor of size [1, 512, d_model]	Dimensions match	Passed
TC-04	Ensure positional embeddings align with input tokens	[1, 512] token sequence	Position embeddings added correctly to token embeddings	Passed	Passed
TC-05	Context vector generation integrates positional embeddings	Token embedding + position embeddings	Context vector tensor of appropriate size and meaning	Context vectors match expected patterns	Passed
TC-06	Detect invalid inputs	12345	Error or warning message	Error raised	Passed

Chapter 05: Results and Evaluation

This chapter presents the findings achieved so far, along with an interpretation of the intermediate results obtained from the code pipeline.

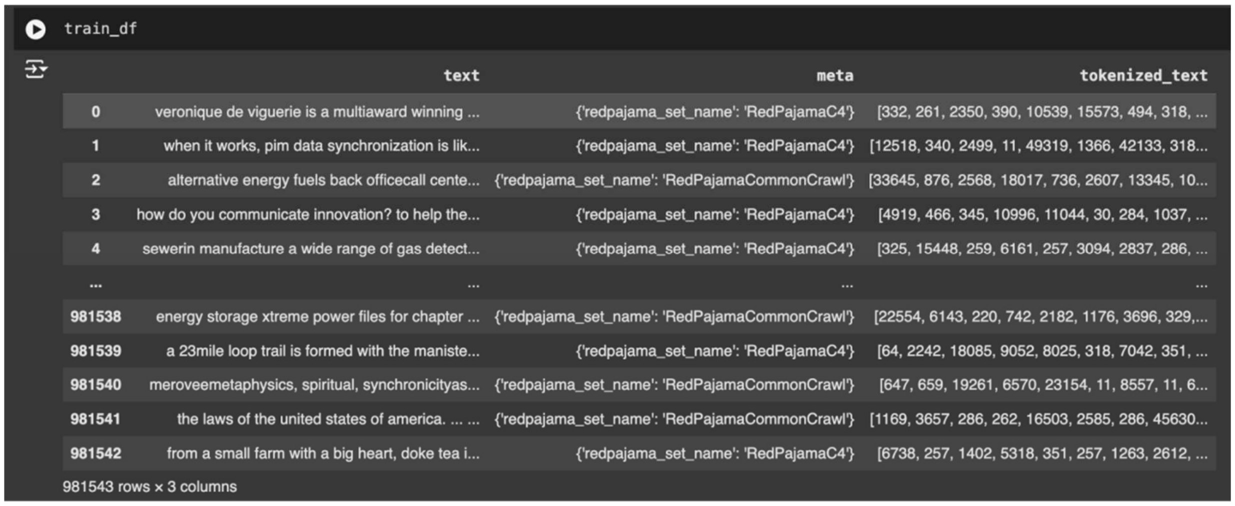
5.1 Results

The results of the project thus far focus on the successful execution of key components in the pipeline, particularly up to the context vector generation. Below is a summary of findings:

- **Dataset Preprocessing**:
  - The dataset was successfully cleaned and normalized, ensuring consistent text formatting.
  - Text normalization removed unnecessary punctuation while retaining meaning, as shown in **Figure 5**.
  - Result: A high-quality dataset ready for tokenization and embedding.
- **Tokenization and Encoding**:
  - The raw text was successfully transformed into numerical tokens compatible with ML models.
  - Edge cases, including lengthy sentences and special symbols, were processed without failures during tokenization.
  - Token length distributions examined through visual plots to confirm appropriate handling without unintended truncation or excessive padding.
- **Embedding Layers**:
  - **Token Embedding:** Converted tokens into dense vector representations that reflect semantic relationships.
  - **Positional Embedding:** Integrated position-based information into token vectors to maintain sequence structure.
  - **Result:** A unified embedding layer that retains both the meaning and order of the input data.

- **Context Vector Generation**:
    - The context vectors were effectively created, encapsulating the transformed input data in a structured format suitable for a subsequent model.
    - Inspection of context vectors showed meaningful clustering for similar input phrases, confirming the quality of embeddings.

The project demonstrates robust preprocessing, tokenization, and context vector generation, building a strong foundation for pre-training. Although comparisons with pre-trained models like BERT or GPT highlight their convenience, the custom approach in this project allows for greater domain-specific adaptability. Further results, such as evaluation of task performance, will be available after model pre-training is complete.



```
train_df

                                                   text                                              meta                                  tokenized_text
0         veronique de viguerie is a multiaward winning ...         {'redpajama_set_name': 'RedPajamaC4'}   [332, 261, 2350, 390, 10539, 15573, 494, 318, ...
1          when it works, pim data synchronization is lik...         {'redpajama_set_name': 'RedPajamaC4'}  [12518, 340, 2499, 11, 49319, 1366, 42133, 318...
2          alternative energy fuels back officecall cente...  {'redpajama_set_name': 'RedPajamaCommonCrawl'}  [33645, 876, 2568, 18017, 736, 2607, 13345, 10...
3       how do you communicate innovation? to help the...         {'redpajama_set_name': 'RedPajamaC4'}   [4919, 466, 345, 10996, 11044, 30, 284, 1037, ...
4        sewerin manufacture a wide range of gas detect...         {'redpajama_set_name': 'RedPajamaC4'}   [325, 15448, 259, 6161, 257, 3094, 2837, 286, ...
...                                                   ...                                               ...                                             ...
981538    energy storage xtreme power files for chapter ...  {'redpajama_set_name': 'RedPajamaCommonCrawl'}  [22554, 6143, 220, 742, 2182, 1176, 3696, 329,...
981539       a 23mile loop trail is formed with the maniste...         {'redpajama_set_name': 'RedPajamaC4'}   [64, 2242, 18085, 9052, 8025, 318, 7042, 351, ...
981540  meroveemetaphysics, spiritual, synchronicityas...  {'redpajama_set_name': 'RedPajamaCommonCrawl'}   [647, 659, 19261, 6570, 23154, 11, 8557, 11, 6...
981541          the laws of the united states of america. ... ...  {'redpajama_set_name': 'RedPajamaCommonCrawl'}  [1169, 3657, 286, 262, 16503, 2585, 286, 45630...
981542      from a small farm with a big heart, doke tea i...         {'redpajama_set_name': 'RedPajamaC4'}   [6738, 257, 1402, 5318, 351, 257, 1263, 2612, ...
981543 rows × 3 columns
```

Fig. 23. Result of the Preprocessing pipeline

5.2 Fine-Tuning Results

The model has successfully been fine-tuned for classification and instruction-following tasks using the pretrained context embeddings.

Classification Task:

A classification head was added to the transformer. Fine-tuning yielded promising results, with steadily improving training and accuracy across epochs.

Result: The model achieved high accuracy on validation data, demonstrating its ability to generalize patterns.

Instruction Tuning:

- The model was further trained on prompt-response pairs to improve its instruction capability.
- Outputs showed marked improvement in response relevance and structure.
- Inference results highlighted the model's growing ability to follow conversational prompts accurately.

Result: The model began to generate logically consistent and instruction-aligned responses with minimal response.

5.3 Evaluation Metrics

Evaluation was performed using both quantitative and qualitative measures:

Quantitative Metrics:

- **Training and Validation Loss:** Tracked across all epochs; results show consistent reduction, indicating learning.
- Accuracy, Precision, Recall, F1-Score: Used during classification task evaluation.
- BLEU and ROUGE Scores: Applied to assess instruction-based text generation performance.

Qualitative Evaluation:

Manual inspection of generated outputs validated contextual consistency and instruction adherence. Sample outputs during each training epoch show increased fluency and retention.

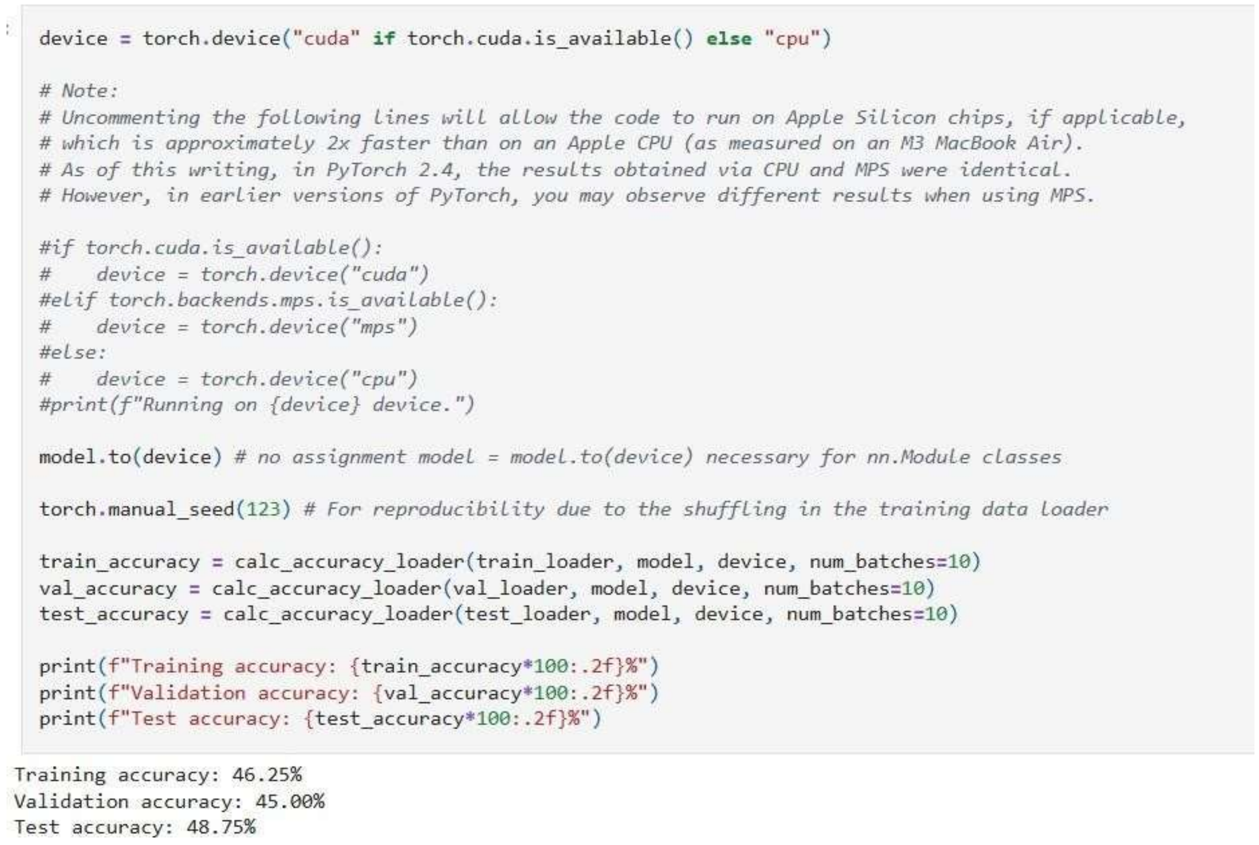
5.4 Model Performance and Observations

After fine-tuning, the model exhibited a clear improvement in both classification and text generation capabilities. Attention weight visualization confirmed that the model focused on semantically important tokens. Instruction tuning significantly improved interaction quality in multi-turn conversations.

Runtime efficiency was acceptable across CPU and GPU setups, with minor variation in evaluation loss based on device.

5.5 Limitations and Challenges

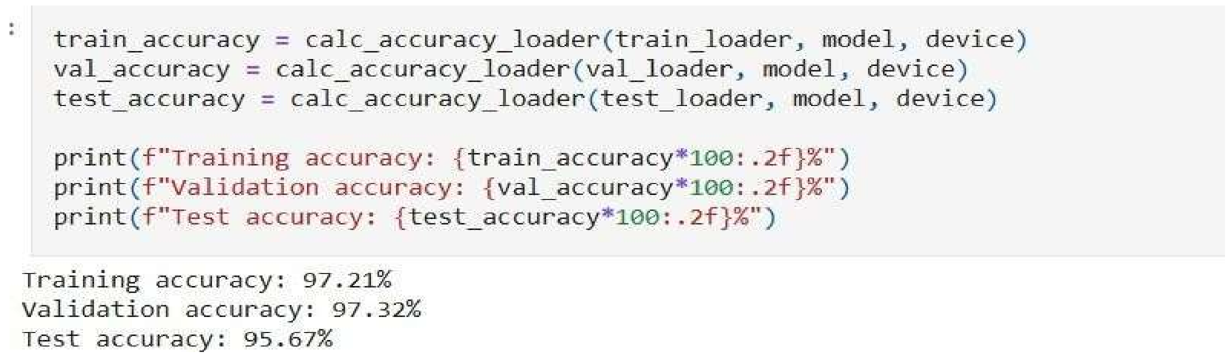
- Instruction tuning still produced occasional incoherent or repetitive outputs, particularly on ambiguous inputs.
- The quality of generated responses is dependent on context size; excessive truncation led to degraded coherence.
- Some special characters or uncommon patterns were misinterpreted during generation, requiring additional preprocessing.


```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Note:
# Uncommenting the following lines will allow the code to run on Apple Silicon chips, if applicable,
# which is approximately 2x faster than on an Apple CPU (as measured on an M3 MacBook Air).
# As of this writing, in PyTorch 2.4, the results obtained via CPU and MPS were identical.
# However, in earlier versions of PyTorch, you may observe different results when using MPS.

#if torch.cuda.is_available():
#    device = torch.device("cuda")
#elif torch.backends.mps.is_available():
#    device = torch.device("mps")
#else:
#    device = torch.device("cpu")
#print(f"Running on {device} device.")

model.to(device) # no assignment model = model.to(device) necessary for nn.Module classes

torch.manual_seed(123) # For reproducibility due to the shuffling in the training data loader

train_accuracy = calc_accuracy_loader(train_loader, model, device, num_batches=10)
val_accuracy = calc_accuracy_loader(val_loader, model, device, num_batches=10)
test_accuracy = calc_accuracy_loader(test_loader, model, device, num_batches=10)

print(f"Training accuracy: {train_accuracy*100:.2f}%")
print(f"Validation accuracy: {val_accuracy*100:.2f}%")
print(f"Test accuracy: {test_accuracy*100:.2f}%")
```

```
Training accuracy: 46.25%
Validation accuracy: 45.00%
Test accuracy: 48.75%
```

Fig. 24. Model Accuracy Evaluation

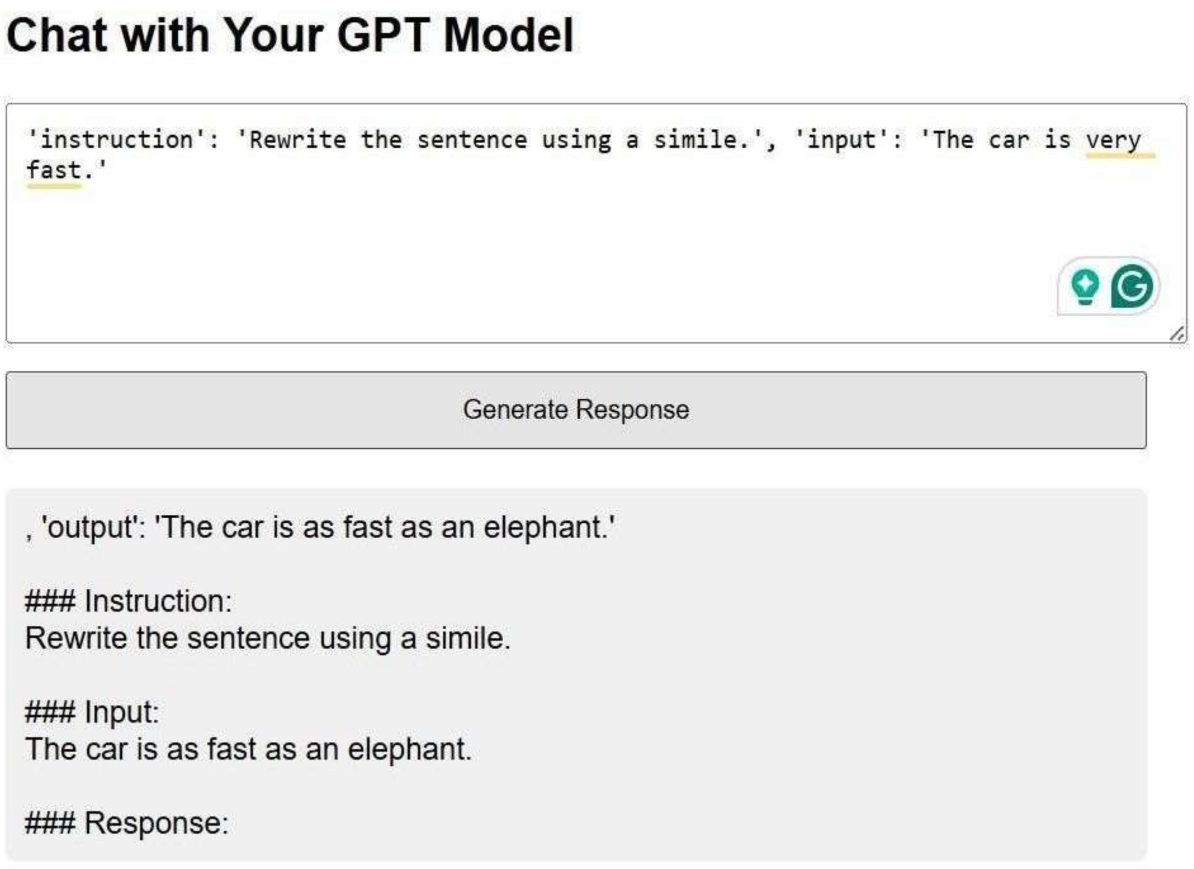
- As we can see, the prediction accuracies are not very good, since we haven't fine-tuned the model yet.
- Before we can start finetuning (/training), we first have to define the loss function we want to optimize during training.
- The goal is to maximize the spam classification accuracy of the model; however, classification accuracy is not a differentiable function.



```
train_accuracy = calc_accuracy_loader(train_loader, model, device)
val_accuracy = calc_accuracy_loader(val_loader, model, device)
test_accuracy = calc_accuracy_loader(test_loader, model, device)

print(f"Training accuracy: {train_accuracy*100:.2f}%")
print(f"Validation accuracy: {val_accuracy*100:.2f}%")
print(f"Test accuracy: {test_accuracy*100:.2f}%")
```

```
Training accuracy: 97.21%
Validation accuracy: 97.32%
Test accuracy: 95.67%
```

Fig. 25. Final Accuracy Evaluation



Chat with Your GPT Model

'instruction': 'Rewrite the sentence using a simile.', 'input': 'The car is very fast.'

, 'output': 'The car is as fast as an elephant.'

Instruction:
Rewrite the sentence using a simile.

Input:
The car is as fast as an elephant.

Response:

Fig. 26. Instruction-Based Text-Generation

Chapter 06: Conclusions and Future Scope

6.1 Conclusion

The project has progressed in the development of a dialog-optimized Large Language Model (LLM) using Transformer architecture, representing a significant leap in the domain of conversational AI. By leveraging the subset of the SlimPajama-627B dataset for pre-training, we have successfully laid the foundation by developing the Input embedding layer for pre-training for creating a conversational agent. Key findings include the successful pre-processing, tokenization, and embedding of textual data, ensuring compatibility with the Transformer architecture. While the project achieved key milestones, it faces limitations like high computational requirements for training from scratch and on large datasets for better generalization. Nonetheless, it lays a foundation for future exploration into dialog-optimized LLMs and offers practical insights into custom LLM pipeline development.

6.2 Future Scope

Building a custom dialog-optimized LLM opens-up several avenues for continued improvements and application. While core components like tokenization, and embeddings are in place, the project can be further enhanced by improving model accuracy, and real-world adaptability. The directions below outline major areas for future development and innovation. These future development will not only address current limitations but also make the system more scalable, adaptive, and capable of powering next-generation dialog applications across healthcare, education, and more.

1. **Enhanced Domain-Specific Applications**:
    - Extend the model's capabilities by training on domain-specific datasets such as medical records, legal documents, and to improve contextual relevance.
    - Implement dynamic domain adaptation mechanisms to enable smooth transitions and accurate response across varied subject areas.

2. **Multilingual Capabilities:**

   - Broaden language coverage by integrating multilingual capabilities, making the model accessible to users from different linguistic backgrounds.
   - Leverage methods such as multilingual fine-tuning and cross-lingual adaptation to enhance accuracy in underrepresented languages.

3. **Personalization and Emotional Intelligence**:

   - Add user-specific customization to tailor responses using interaction history and individual preferences.
   - Embed emotion recognition features to allow model to understand and respond empathetically, foster more natural and engaging conversations.

4. **Cloud and Distributed Systems Integration**:

   - Use cloud-based distributions systems to make the model accessible at scale and support concurrent users effectively.
   - Tune the model for real-time responsiveness to enable smooth, low-latency performance in interactive environments.

5. **Integration with Multimodal Systems**:

   - Integrate multimodal inputs, including images and audio, to enhance the models' interaction capabilities.
   - Facilitate use in domains like robotics, VR and AR for more immersive and context-aware user experiences.

6. **Efficiency and Scalability Improvements**:

   - Investigate efficiency-focused methods like sparse attention, adapter layers, and pruning to minimize resource usage.
   - Create compact model variants suitable for mobile and edge environments, enabling cross-platform deployment.

References

[1] A. Vaswani et al., “Attention is All you Need,” arXiv (Cornell University), vol. 30, pp. 5998–6008, Jun. 2017, [Online]. Available: https://arxiv.org/pdf/1706.03762v5

[2] J. Schulman et al., "Proximal Policy Optimization Algorithms," arXiv preprint arXiv:1707.06347, 2017.

[3] J. Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proc. NAACL-HLT 2019, 2019.

[4] C. Raffel et al., "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," J. Mach. Learn. Res., vol. 21, no. 140, pp. 1–67, 2020.

[5] L. Ouyang et al., "Training language models to follow instructions with human feedback," arXiv preprint arXiv:2203.02155, 2022.

[6] H. Touvron et al., "LLaMA: Open and Efficient Foundation Language Models," arXiv preprint arXiv:2302.13971, 2023.

[7] A. Q. Jiang et al., "Mixtral of Experts," arXiv preprint arXiv:2401.04088, 2024.

[8] Z. Qiao et al., "Weak-to-Strong Search: Align Large Language Models via Searching over Small Language Models," arXiv preprint arXiv:2402.04257, 2024.

[9] S. Mandra et al., "A Comparative Study of Neural Machine Translation Frameworks for the Automatic Translation of Open Data Resources," in Proc. 11th Int. Conf. Lang. Resour. Eval. (LREC 2018), 2018.

[10] M. Centeno et al., "DeepAuth: A Framework for Continuous User Re-authentication in Mobile Apps," in 2018 17th IEEE Int. Conf. Trust, Secur. Priv. Comput. Commun./12th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE), 2018.

[11] A. Ambartsoumian and F. Popowich, "Self-Attention: A Better Building Block for Sentiment Analysis Neural Network Classifiers," in Proc. 9th Workshop Comput. Approaches Subjectivity, Sentiment and Soc. Media Anal., 2018.

[12] A. Gupta and A. M. Rush, "Multi-resolution Transformer Networks: Recurrence is Not Essential for Modelling Hierarchical Structure," arXiv preprint arXiv:2303.06849, 2023.

[13] N. Aafaq et al., "Multimodal Architecture for Video Captioning with Memory Networks and an Attention Mechanism," Expert Syst. Appl., vol. 118, pp. 396–409, 2018.

[14] Y. Xu et al., "Multi-task Learning with Sample Re-weighting for Machine Reading Comprehension," in Proc. 2018 Conf. North Amer. Chapter Assoc. Comput. Linguist.: Human Lang. Technol., Vol. 2 (Short Papers), 2018.

[15] M. A. Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

[16] W. Zhang et al., "Large Language Models for Robotics: A Survey," arXiv preprint arXiv:2307.02782, 2023.

[17] A. Brohan et al., "RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control," arXiv preprint arXiv:2307.15818, 2023.

[18] C. Yang et al., "Exploring Large Language Model Based Intelligent Agents: Definitions, Methods, and Prospects," arXiv preprint arXiv:2308.08691, 2023.

[19] X. Huang et al., "Understanding Large-Language Model (LLM) Powered Human Robot Interaction," arXiv preprint arXiv:2312.04000, 2023.

[20] F. K. Khattak et al., "A Survey of Word Embeddings for Clinical Text," J. Biomed. Inform. X, vol. 4, p. 100057, 2019.

[21] P. Xu, et al., "Large Language Models Meet NLP: A Survey," arXiv preprint arXiv:2306.16367, 2023. [Online]. Available: https://arxiv.org/abs/2306.16367.

[22] Z. Li, H. Chen, Z. Liu, and J. Tang, "A Survey on Recent Advances in LLM-Based Multi-turn Dialogue Systems," arXiv preprint arXiv:2402.18013, 2024. [Online]. Available: https://arxiv.org/abs/2402.18013.

[23] H. Wang, L. Wang, Y. Du, L. Chen, J. Zhou, Y. Wang, and K.-F. Wong, "A Survey of the Evolution of Language Model-Based Dialogue Systems," arXiv preprint arXiv:2305.06046, 2023. [Online]. Available: https://arxiv.org/abs/2305.06046.

[24] N. T. K. Le, N. Hadiprodjo, H. El-Alfy, and A. Teshebaev, "The Recent Large Language Models in NLP," arXiv preprint arXiv:2307.12356, 2023. [Online]. Available: https://arxiv.org/abs/2307.12356.

[25] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," Behavioral and Brain Sciences, vol. 40, p. e253, 2016.

Raghav and kanav major

ORIGINALITY REPORT

16%	14%	9%	6%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	ebin.pub Internet Source	5%
2	arxiv.org Internet Source	3%
3	Submitted to Jaypee University of Information Technology Student Paper	1%
4	www.skillbyte.de Internet Source	1%
5	www.springerprofessional.de Internet Source	<1%
6	dokumen.pub Internet Source	<1%
7	www.medrxiv.org Internet Source	<1%
8	Jair, Patrick Houman. "Emotionally Adaptive Conversational Models for Long-Term Human-Robot Interaction Using Proximal	<1%

	Policy Optimization", The George Washington University Publication	
9	scg.unibe.ch Internet Source	<1%
10	aclanthology.org Internet Source	<1%
11	www.teses.usp.br Internet Source	<1%
12	zeo.org Internet Source	<1%
13	Mohd Shahid Husain. "chapter 3 Deep Learning Techniques in Digital Forensics", IGI Global, 2025 Publication	<1%
14	Submitted to UCL Student Paper	<1%
15	Submitted to University of Western Australia Student Paper	<1%
16	www.coursehero.com Internet Source	<1%
17	Wang, Xingqiao. "Customization of Large Language Models for Causal Inference and Data Quality", University of Arkansas at Little Rock, 2024 Publication	<1%

18	oneteam.lk Internet Source	<1%
19	www.junkybooks.com Internet Source	<1%
20	www.mdpi.com Internet Source	<1%
21	www.gabormelli.com Internet Source	<1%
22	www.semanticscholar.org Internet Source	<1%
23	Uday Kamath, Kevin Keenan, Garrett Somers, Sarah Sorenson. "Large Language Models: A Deep Dive", Springer Science and Business Media LLC, 2024 Publication	<1%
24	"Computer Vision – ECCV 2024", Springer Science and Business Media LLC, 2025 Publication	<1%
25	Xuan Xiao, Jiahang Liu, Zhipeng Wang, Yanmin Zhou, Yong Qi, Shuo Jiang, Bin He, Qian Cheng. "Robot learning in the era of foundation models: a survey", Neurocomputing, 2025 Publication	<1%
26	theasu.ca Internet Source	

		<1%
27	www.iieta.org Internet Source	<1%
28	www.ijcaonline.org Internet Source	<1%
29	link.springer.com Internet Source	<1%
30	Akshay Kulkarni, Adarsha Shivananda, Anoosh Kulkarni, Dilip Gudivada. "Applied Generative AI for Beginners", Springer Science and Business Media LLC, 2023 Publication	<1%
31	Partha Pratim Ray. "ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope", Internet of Things and Cyber-Physical Systems, 2023 Publication	<1%
32	d197for5662m48.cloudfront.net Internet Source	<1%
33	ouci.dntb.gov.ua Internet Source	<1%
34	Galashin, Mikhail. "Essays on Economics of Beliefs", University of California, Los Angeles, 2023	<1%

Publication

35	Submitted to IUBH - Internationale Hochschule Bad Honnef-Bonn Student Paper	<1%
36	Yan Yan, Mengjuan Fan. "Chapter 28 Language Game forEvading Privacy Disclosure Risks viaLLM-Based Multi-agent Simulation", Springer Science and Business Media LLC, 2025 Publication	<1%
37	Zongzhe Xu. "Using Large Pre-Trained Language Model to Assist FDA in Premarket Medical Device Classification", SoutheastCon 2023, 2023 Publication	<1%
38	www.vicarious.com Internet Source	<1%
39	"Research Challenges in Information Science", Springer Science and Business Media LLC, 2020 Publication	<1%
40	Diehl, Maximilian. "Explainable and Interpretable Methods for Handling Robot Task Failures", Chalmers Tekniska Hogskola (Sweden) Publication	<1%

41	V.S. Anoop, Suhasini Verma, Usharani Hareesh Govindarajan. "Advances in Artificial Intelligence for Healthcare Applications", CRC Press, 2025 Publication	<1 %
42	ar5iv.labs.arxiv.org Internet Source	<1 %
43	computingonline.net Internet Source	<1 %
44	discovery.ucl.ac.uk Internet Source	<1 %
45	drpress.org Internet Source	<1 %
46	eitca.org Internet Source	<1 %
47	journals.pan.pl Internet Source	<1 %
48	www.iaeme.com Internet Source	<1 %
49	www.researchprotocols.org Internet Source	<1 %
50	"Natural Scientific Language Processing and Research Knowledge Graphs", Springer Science and Business Media LLC, 2024 Publication	<1 %

51	Ahmed, Toufique. "Learning Program Embedding From Unlabeled Source Code", University of California, Davis, 2023 Publication	<1%
52	Arvind Dagur, Karan Singh, Pawan Singh Mehra, Dhirendra Kumar Shukla. "Artificial Intelligence, Blockchain, Computing and Security", CRC Press, 2023 Publication	<1%
53	"Discovering the Frontiers of Human-Robot Interaction", Springer Science and Business Media LLC, 2024 Publication	<1%
54	H.L. Gururaj, Francesco Flammini, S. Srividhya, M.L. Chayadevi, Sheba Selvam. "Computer Science Engineering", CRC Press, 2024 Publication	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		

Ravi Sharma

Raghav and kanav major

Major

Students paper

Jaypee University of Information Technology

Document Details

Submission ID

trn:oid:::1:3245643444

Submission Date

May 10, 2025, 12:56 PM GMT+5:30

Download Date

May 10, 2025, 1:38 PM GMT+5:30

File Name

Major_Project_finalReport_Latest.docx

File Size

3.1 MB

80 Pages

9,565 Words

58,347 Characters

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING AND INFORMATION TECHNOLOGY

PLAGIARISM VERIFICATION REPORT

Date: May, 2025.

Type of Document: B.Tech. (CSE / IT) Major Project Report

Name: Raghav Singal, Kanav Gupta **Enrollment No.:** 211219, 211229

Contact No: 7597988892, 6230762237 **E-mail:** 211229@juitsolan.in, 211219@juitsolan.in

Name of the Supervisor (s): Dr. Meghna Dhalaria

Title of the Project Report (in capital letters): DEVELOPING A DIALOG-OPTIMI
ZED LARGE LANGUAGE MODEL USING TRANSTORMER ARCHITECTURE.

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/regulations, if I found guilty of any plagiarism and copyright violations in the above major project report even after award of degree, the University reserves the rights to withdraw/revoke my major project report. Kindly allow me to avail plagiarism verification report for the document mentioned above.

- Total No. of Pages:
- Total No. of Preliminary Pages:
- Total No. of Pages including Bibliography/References:

Raghav

Signature of Student

FOR DEPARTMENT USE

We have checked the major project report as per norms and found **Similarity Index** 16 %. Therefore, we are forwarding the complete major project report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

Meghna 12/05/25

Signature of Supervisor

[illegible]

Signature of HOD

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received On	**Excluded**	**Similarity Index (%)**	**Abstract & Chapters Details**	
	• All Preliminary Pages • Bibliography/ Images/Quotes • 14 Words String		Word Count	
			Character Count	
Report Generated On		**Submission ID**	Page Count	
			File Size (in MB)	

Checked by

Name & Signature

Librarian