

**TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING**

Khwopa College Of Engineering
Libali, Bhaktapur
Department of Computer and Electronics



**A
MAJOR PROJECT
REPORT ON**

Nepali AI Chatbot using Deep Learning Model

Submitted in partial fulfillment of the requirements for the degree

BACHELOR OF COMPUTER ENGINEERING

Submitted by

Aabishkar Ghimire

KCE077BCT002

Aaryan Raj Daibagya

KCE077BCT003

Arun Kumar Shrestha

KCE077BCT010

Ayushree Kharel

KCE077BCT012

Under the Supervision of
Dr. Ramesh Marikhu

Khwopa College Of Engineering
Libali, Bhaktapur
May, 2025

Copyright

The author has agreed that the library, Khwopa College of Engineering, may make this report freely available for inspection. Moreover, the author has agreed that permission for the extensive copying of this project report for scholarly purposes may be granted by the supervisor who supervised the project work recorded herein or, in their absence, the Head of the Department where the project report was done. It is understood that recognition will be given to the author of the report and to the Department of Computer Engineering, KhCE, for any use of the material of this project report. Copying, publication, or other use of this report for financial gain without approval of the department and the author's written permission is prohibited. Requests for permission to copy or to make any other use of material in this report, in whole or in part, should be addressed to:

Head of Department
Department of Computer Engineering
Khwopa College of Engineering (KhCE)
Liwali,
Bhaktapur, Nepal.

Acknowledgement

We would like to thank **Dr. Ramesh Marikhu** for his wise counsel, inspiring ideas, and invaluable direction, help, and support throughout this project. We also owe a debt of gratitude to **Er. Dinesh Gothe** and **Er. Mukesh Kumar Pokharel** for their tireless efforts at every stage of the project, as well as for their insightful counsel and recommendations.

We are truly grateful for the wisdom and support they have provided us and it is through their efforts that we have been able to bring this project to fruition. On behalf of the entire team, we express our sincere thanks and appreciation for their invaluable contributions.

Aabishkar Ghimire	KCE077BCT002
Aaryan Raj Daibagya	KCE077BCT003
Arun Kumar Shrestha	KCE077BCT010
Ayushree Kharel	KCE077BCT012

Abstract

This project aims to develop a generative language model specifically for the Nepali language, designed for a general-purpose chatbot. Given the limited availability of large-scale datasets and pretrained models for low-resource languages like Nepali, we initially built a Nepali language model from scratch using the GPT-2 (124M) architecture. To enhance its conversational abilities, we further post-trained the model on a diverse Nepali conversational dataset. This approach enables the chatbot to understand and generate contextually relevant responses in Nepali. By leveraging advancements in natural language processing (NLP) and transformer-based architectures, this project addresses the unique linguistic challenges of the Nepali language. Through improved language inclusivity and conversational AI capabilities, this work contributes to NLP research and fosters accessibility for Nepali-speaking users.

Keywords: *Chatbot, Natural Language Processing (NLP), AI, Machine Learning, Nepali Language, Transformer, GPT*

Contents

Copyright	i
Acknowledgement	i
Abstract	iii
List of Tables	vi
List of Figures	viii
List of Symbols and Abbreviation	ix
1 INTRODUCTION	1
1.1 Background	1
1.2 Motivation	1
1.3 Problem Statement	1
1.4 Objective	3
1.5 Scope of project	3
2 LITERATURE REVIEW	4
2.1 NepBERTa: Nepali Language Model Trained on a Large Corpus	4
2.2 NepaliBERT: Pre-training of a Masked Language Model in a Nepali Corpus	4
2.3 GPT-2: Language Models Are Unsupervised Multitask Learners	4
2.4 GPT-3: Language Models Are Few-Shot Learners	4
2.5 GPT-4: Advancements in Transformer-Based AI	5
2.6 Claude: Advancing Conversational AI	5
2.7 Algorithm-Based Chatbot Using Transformer and Sequence-to-Sequence Method	5
2.8 Gemini: A Family of Highly Capable Multimodal Models	5
2.9 TinyBERT: Distilling BERT for Natural Language Understanding . . .	6
2.10 Scaling Laws for Neural Language Models	6
2.11 Development of Pre-trained Transformer-based models for the Nepali language	6
3 THEORETICAL BACKGROUND	9
3.1 Overview of Language Models	9
3.2 Transformer Architecture	9
3.3 Pretraining Paradigms	9
3.4 Challenges in Low-Resource Languages	10
3.5 Dataset Curation and Preprocessing	10
3.6 Model Configuration and Training	10
3.7 Evaluation Metrics	11
4 SYSTEM DESIGN	12
4.1 Requirement Specification	12
4.1.1 Functional Requirements	12
4.1.1.1 Natural Language Understanding (NLU)	12
4.1.1.2 Response Generation	12
4.1.1.3 User Engagement	12
4.1.1.4 Task Execution	12

4.1.2	Non-Functional Requirements	12
4.1.2.1	Accuracy	12
4.1.2.2	Speed	12
4.1.2.3	Scalability	13
4.1.2.4	Usability	13
4.1.2.5	Reliability	13
4.1.2.6	Maintainability	13
4.2	Feasibility Assessment	13
4.2.1	Economic Feasibility	13
4.2.2	Technical Feasibility	13
4.2.3	Operational Feasibility	13
4.3	Proposed System Architecture	14
4.4	Use Case Diagram	14
4.5	Sequence Diagram	15
4.6	Class Diagram	16
4.7	Activity Diagram	17
4.8	Collaboration Diagram	18
4.9	Deployment Diagram	19
5	METHODOLOGY	20
5.1	Training Pipeline	20
5.1.1	Tokenization	20
5.1.2	Model	21
5.1.2.1	Model Parameters (GPT-2 124M Configuration) . . .	23
5.1.2.2	Model Summary	25
5.1.3	Training Configurations	26
5.1.3.1	Hyperparameters	26
5.1.3.2	Loss Function	26
5.1.3.3	Optimizer	27
5.1.3.4	Gradient Clipping	27
5.1.3.5	Learning Rate Scheduling	27
5.1.3.6	Training Performance	28
5.1.4	Cost Optimization	29
5.1.4.1	Gradient Accumulation	29
5.1.4.2	Mixed Precision Training	29
5.1.4.3	Using Numbers as Powers of Two	30
5.1.4.4	Flash Attention	30
5.1.4.5	Fused Adam Optimizer	30
5.1.4.6	torch.compile	30
5.1.4.7	Shared Embedding and lm_head Matrix	30
5.2	Pre-Training	31
5.2.1	Dataset	31
5.2.2	Training	31
5.2.3	Results	31
5.3	Pretraining Outputs	32
5.4	Post Training	33
5.4.1	Dataset	33
5.4.2	Training	34

5.4.3	Results	35
5.5	Post-Training Outputs	36
6	RESULT AND DISCUSSION	37
6.1	Pretraining Results	37
6.2	Post Training Results	38
7	EVALUATION AND COMPARISON	40
7.1	Evaluation	40
7.1.1	Evaluation Setup	40
7.1.2	Results	40
7.1.3	Discussion	40
7.2	Comparison	41
8	SOFTWARE DEPLOYMENT	42
8.1	Architecture Overview	42
8.2	Streamlit UI Deployment	42
8.3	PyTorch Model Serving	42
8.4	Containerization	43
8.5	CI/CD Pipeline	43
8.6	Monitoring & Logging	43
9	CONCLUSION	44
9.1	Limitations	44
9.2	Future Enhancements	44
9.3	Challenges	45
	REFERENCES	47

List of Tables

2.1	Comprehensive Review Matrix for Language Models	8
7.1	Comparison on Training and Validation Loss	41

List of Figures

4.1	System Architecture	14
4.2	System Use Case Diagram	14
4.3	Sequence Diagram	15
4.4	Class Diagram	16
4.5	Activity Diagram	17
4.6	Collaboration Diagram	18
4.7	Deployment Diagram	19
5.1	Vocabulary generated after training the BPE tokenizer.	21
5.2	Tokenization of a Nepali sentence.	21
5.3	GPT 2 Architecture	22
5.4	Model Summary	25
5.5	Cosine Decay Lr Scheduler	28
5.6	Gradient Accumulation	29
5.7	Flash Attention Illustration	30
5.8	Training and validation loss vs. steps.	32
5.9	Full loss curve (pretraining + post-training)	35
5.10	Post-training loss curve	35
6.1	Pretraining Training and Validation Loss Plot	37
6.2	Full loss curve (pretraining + post-training)	38
6.3	Post-training loss curve	38
A.1	Pre-Training Dataset	48
A.2	UI	50

List of Symbols and Abbreviation

AI	Artificial Intelligence
BERT	Bidirectional Encoder Representations from Transformers
GPT	Generative Pre-trained Transformer
ML	Machine Learning
NLP	Natural Language Processing
MLM	Masked Language Modeling

CHAPTER 1

INTRODUCTION

1.1 Background

The rapid advancement of natural language processing (NLP) technologies, exemplified by models such as BERT and GPT-3, has largely benefited high-resource languages, leaving low-resource languages like Nepali underrepresented. Nepali, spoken by millions but lacking extensive digital resources, faces significant challenges in developing sophisticated NLP applications. This project aims to create a generative language model for Nepali language from scratch. By training the model on extensive Nepali text corpora, it will enable the development of a general-purpose chatbot, improving conversational AI capabilities in the Nepali language.

This initiative addresses the unique linguistic challenges posed by Nepali, such as its rich morphology and diverse dialects, while also fostering inclusivity in NLP. It demonstrates a scalable framework that can be extended to other low-resource languages, contributing to global advancements in natural language understanding and generation.

1.2 Motivation

The development of sophisticated natural language processing (NLP) models for the Nepali language has been markedly limited, significantly hindering the advancement of digital communication tools in Nepal. Despite the progress in NLP for high-resource languages, Nepali remains underrepresented, resulting in a substantial gap in language technology. Multilingual models like mBERT and XLM-R, although designed to support multiple languages, perform poorly for Nepali due to their reliance on shared parameters and insufficient Nepali-specific training data.

This inadequacy affects the quality and accuracy of automated systems, including general-purpose chatbots, which are becoming increasingly important for improving digital accessibility and communication. To address these challenges, this project aims to develop a Nepali-specific language model using the GPT architecture. By leveraging the autoregressive capabilities of GPT, this model will be trained from scratch on extensive Nepali text corpora, ensuring it captures the unique linguistic and contextual nuances of the language.

This initiative will not only improve conversational AI for Nepali speakers but also provide a scalable framework for other low-resource languages. It demonstrates the potential of advanced language modeling to bridge technological gaps, enhance inclusivity, and contribute to global advancements in NLP.

1.3 Problem Statement

The lack of advanced natural language processing (NLP) models for the Nepali language has resulted in a significant technological gap that hampers the development

and accessibility of digital communication tools for Nepali speakers. Existing multilingual models, such as mBERT and XLM-R, exhibit poor performance when applied to Nepali due to insufficient training data and reliance on shared parameters that fail to capture the unique linguistic features of the language. This shortfall limits the quality and effectiveness of conversational AI systems, including general-purpose chatbots, which are essential for improving user interaction and digital accessibility in Nepal.

To address this issue, there is an urgent need to develop a dedicated Nepali language model from scratch, leveraging the GPT architecture. Such a model will be trained on extensive Nepali text corpora to accurately understand and generate text, ensuring it accommodates the linguistic and contextual nuances of the language. By overcoming these challenges, this project aims to enhance the conversational AI landscape for Nepali speakers while contributing to the global efforts in developing NLP solutions for other low-resource languages.

1.4 Objective

The objective of this project is:

- To develop a Nepali Language Generative AI chatbot.

1.5 Scope of project

The major scopes of this project are:

- Develop a Nepali Language model from scratch.
- Train on extensive Nepali text corpora.
- Develop a Nepali Language conversational chatbot.

CHAPTER 2

LITERATURE REVIEW

2.1 NepBERTa: Nepali Language Model Trained on a Large Corpus

In this paper [1], NepBERTa is introduced as a BERT-based model trained on an extensive monolingual Nepali corpus, significantly larger than previous datasets. NepBERTa has been evaluated on multiple NLP tasks, including Named-Entity Recognition, Content Classification, POS Tagging, and Categorical Pair Similarity, showing superior performance compared to both prior monolingual and multilingual models. Additionally, this study introduces Nep-gLUE, the first comprehensive evaluation benchmark for Nepali language understanding, facilitating further research and development in this domain.

2.2 NepaliBERT: Pre-training of a Masked Language Model in a Nepali Corpus

NepaliBERT [2] emphasizes the evolution of natural language processing (NLP) for Nepali, transitioning from traditional methods like TF-IDF and Word2Vec to advanced transformer models like BERT. While foundational approaches provided static embeddings, recent works leverage BERT for contextual language understanding. Challenges, such as computational resource limitations and lack of extensive high-quality datasets, remain significant. This paper demonstrates the potential of leveraging BERT-based models for creating robust embeddings and improving NLP applications in the Nepali context.

2.3 GPT-2: Language Models Are Unsupervised Multitask Learners

GPT-2 [3] introduced a generative pre-trained transformer architecture for autoregressive text generation. By training on extensive corpora and leveraging unsupervised learning, GPT-2 demonstrated significant improvements in generating coherent and contextually relevant text. Its architecture was designed for multitask learning without task-specific fine-tuning, making it highly adaptable. GPT-2 laid the foundation for scaling generative models and showcased the potential of large-scale pretraining for diverse NLP tasks.

2.4 GPT-3: Language Models Are Few-Shot Learners

GPT-3 [4] builds on GPT-2's architecture, scaling to 175 billion parameters, making it one of the largest language models to that date. Its ability to perform tasks in zero-shot, one-shot, and few-shot scenarios demonstrates significant advancements in

language understanding and generation. Despite its superior capabilities, GPT-3 highlights challenges such as high computational costs and occasional factual inaccuracies. Its success underscores the importance of scale and generalization in language models.

2.5 GPT-4: Advancements in Transformer-Based AI

GPT-4 [5] is a state-of-the-art transformer-based model that builds upon its predecessors with significant improvements in language understanding and generation. Trained with Reinforcement Learning from Human Feedback (RLHF), GPT-4 performs exceptionally well across various NLP benchmarks. While it shares limitations like occasional factual inaccuracies and reasoning errors, GPT-4 represents a leap toward more robust and broadly useful AI systems.

2.6 Claude: Advancing Conversational AI

Claude [6] represents an advancement in conversational AI, leveraging Reinforcement Learning from Human Feedback (RLHF) and safety-focused fine-tuning. Designed to produce coherent, safe, and contextually appropriate outputs, Claude focuses on improving conversational experiences while addressing ethical and safety concerns. Its iterative training process emphasizes collaboration and safe interaction across diverse applications.

2.7 Algorithm-Based Chatbot Using Transformer and Sequence-to-Sequence Method

This paper [7] reviews four main paradigms in chatbot development: Rule-Based Models, Retrieval-Based Models, Learning-Based Models, and Generative-Based Models. It proposes a novel architecture based on conditional Wasserstein GAN (cWGAN) and the transformer model, aiming to overcome sequential constraints and enhance response accuracy. This work provides a comprehensive overview of chatbot development advancements and sets the stage for future innovations.

2.8 Gemini: A Family of Highly Capable Multimodal Models

The "Gemini: A Family of Highly Capable Multimodal Models" paper [8] introduces advanced multimodal models—Gemini Ultra, Pro, and Nano—designed for applications ranging from complex reasoning to on-device tasks. With state-of-the-art performance in benchmarks such as language, image, audio, and video understanding, Gemini Ultra achieves human-expert performance on the MMLU benchmark. The paper emphasizes safety, adversarial testing, and post-training techniques such as supervised fine-tuning and reinforcement learning with human feedback to ensure responsible deployment.

2.9 TinyBERT: Distilling BERT for Natural Language Understanding

This paper [9] introduces a two-stage transformer-based distillation framework to create a smaller, faster version of BERT while maintaining competitive performance. TinyBERT is particularly useful for deployment in resource-constrained environments, making advanced NLP capabilities more accessible.

2.10 Scaling Laws for Neural Language Models

This paper [10] explores the scaling laws governing transformer-based language models, providing insights into the relationship between model size, dataset size, and performance. The findings demonstrate that larger models, when trained on adequately sized datasets, exhibit improved generalization and task performance, forming the theoretical basis for scaling generative models like GPT-2 and GPT-3.

2.11 Development of Pre-trained Transformer-based models for the Nepali language

The paper *Development of Pre-trained Transformer-based Models for the Nepali Language* [11] introduces four new pre-trained models, including parameterized models using BERT [12], RoBERTa [13], GPT-2 124M [3] and GPT-2 124M Instruct. According to the provided data, their pre-trained models outperform existing models in both the BERT and GPT architectures, achieving better scores on the Nep-GLUE and ROUGE benchmarks, respectively. Although this study also uses the GPT architecture, it applies it for data summarization, which differs from our goal of creating a chatbot. This distinction is important because while both applications leverage the power of GPT models, the requirements for conversational agents involve more nuanced, dynamic interactions compared to the relatively static task of summarization.

Title	Dataset Size	Tasks Evaluated	Findings
NepBERTa: Nepali Language Model Trained in a Large Corpus	Largest monolingual Nepali corpus (0.8B words)	Named-Entity Recognition, Content Classification, POS Tagging, Categorical Pair Similarity	Outperformed both monolingual and multilingual models across all tasks. Introduced Nep-gLUE benchmark for comprehensive evaluation.
NepaliBERT: Pre-training of Masked Language Model in Nepali Corpus	Over 94 million tokens	Sentiment Analysis, Text Summarization, Document Classification	Developed embeddings using Word2Vec, Doc2Vec, and BERT architectures. Created a large dataset by scraping news data and combining it with existing datasets.
GPT-2: Language Models Are Unsupervised Multitask Learners	40 GB of internet text (OpenWebText)	Text Generation, Zero-Shot and Few-Shot Tasks	Demonstrated coherent and contextually relevant text generation without fine-tuning. Laid the foundation for scaling generative models for multitask learning.
GPT-3: Language Models Are Few-Shot Learners	570 GB of internet text (Common Crawl, Books, Wikipedia, etc.)	Language Understanding, Text Generation, Few-Shot Tasks	Showed exceptional zero-shot, one-shot, and few-shot learning abilities, emphasizing the importance of scale and pretraining for language models. Highlighted limitations like computational costs and occasional factual inaccuracies.
GPT-4: Advancements in Transformer-Based AI	Not specified	Language Understanding, Text Generation, Reasoning Tasks	Showed improvements in reasoning, coding, and understanding complex instructions. Despite occasional inaccuracies, it represents a significant step forward in AI capabilities.
Claude: Advancing Conversational AI	Not specified	Conversational AI, Safety in Generative Models	Focused on safe and coherent conversation generation. Leveraged RLHF and safety-focused fine-tuning. Emphasized responsible AI deployment in diverse applications.

Algorithm-Based Chatbot Using Transformer and Sequence-to-Sequence Method	Not specified	Chatbot Development, Response Generation	Proposed a novel architecture using cWGAN and transformer model to improve chatbot response accuracy and overcome sequential constraints. Demonstrated advancements in the chatbot development paradigm.
Gemini: A Family of Highly Capable Multimodal Models	Not specified	Image, Audio, Video, and Text Understanding, Multimodal Tasks	Achieved state-of-the-art performance on 30 of 32 benchmarks. Notable for first human-expert performance on the MMLU benchmark. Evaluated extensively for safety and robustness, ensuring compliance with safety standards.
TinyBERT: Distilling BERT for Natural Language Understanding	Not specified	Natural Language Understanding	Introduced a two-stage transformer distillation framework. Achieved competitive performance in resource-constrained environments. Enabled smaller and faster models.
Scaling Laws for Neural Language Models	Not specified	Theoretical Analysis	Explored the relationship between model size, dataset size, and performance. Established scaling laws that informed the development of larger language models like GPT-2 and GPT-3.
Development of Pre-trained Transformer-based models for the Nepali language	27GB	Not Specified	Pre-trained BERT, RoBERTa, GPT-2 124M models on Nepali text to generate a comparatively better result.

Table 2.1: Comprehensive Review Matrix for Language Models

CHAPTER 3

THEORETICAL BACKGROUND

3.1 Overview of Language Models

Modern language modeling has evolved from simple frequency-based methods to deep neural architectures. Each generation has increased the capacity to capture long-range dependencies and richer linguistic patterns.

- **Statistical LM:** n-gram models estimate $P(w_t \mid w_{t-n+1}^{t-1})$; suffer from data sparsity and limited context window [14].
- **Neural LM:** RNN/LSTM capture longer dependencies [15], but still struggle with very long sequences.
- **Pretrained Transformer:** self-attention enables modeling of all token pairs in $O(n^2)$ time, facilitating deep, scalable LM training [16].

3.2 Transformer Architecture

The transformer uses attention mechanisms to directly connect any two positions in the input, bypassing recurrence. This enables massive parallelism and stable gradient flow in very deep models.

- **Self-Attention:** queries, keys, values computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

allowing each token to attend to all others [16].

- **Multi-Head:** multiple parallel attention heads learn complementary representation subspaces.
- **Position Encoding:** adds sinusoidal or learned vectors to inject order information.
- **Layer Norm & Residuals:** stabilize training and enable very deep stacks.

3.3 Pretraining Paradigms

Pretraining objectives shape what the model learns about language. Autoregressive models excel at generation, while masked objectives capture bidirectional context.

- **Autoregressive (e.g., GPT):** predicts next token w_t given $w_{<t}$; aligns naturally with conversational generation [17].
- **Masked LM (e.g., BERT):** predicts masked tokens in bidirectional context; suited for understanding tasks [18].
- **Choice for Nepali:** autoregressive objective prioritized to support fluent, coherent Nepali text generation.

3.4 Challenges in Low-Resource Languages

Nepali lacks the vast, clean corpora available for high-resource languages. We must carefully address orthographic variation, tokenization, and domain coverage to build robust models.

- **Data Scarcity:** limited publicly available Nepali text; potential domain imbalances.
- **Orthographic Variability:** spelling variants and Unicode normalization issues in Devanagari.
- **Tokenization:** subword methods (BPE/WordPiece) must capture agglutinative and compound forms [19].
- **Domain Coverage:** require diverse sources (news, literature, social media) to improve generalization.

3.5 Dataset Curation and Preprocessing

Building a clean, representative dataset is critical. We gather text from multiple Nepali-language sources, then normalize, deduplicate, and tokenize to prepare for model training.

- **Corpus Sources:** Nepali Wikipedia dump, news portals, Common Crawl-filtered Nepali text.
- **Cleaning Steps:** remove boilerplate, normalize Unicode, deduplicate near-duplicates.
- **Subword Vocabulary:** train BPE with 32K–50K merges to balance coverage versus vocabulary size.

3.6 Model Configuration and Training

Selecting appropriate hyperparameters and optimization strategies ensures stable convergence and strong performance on Nepali text generation.

- **Depth & Width:** e.g., 24 transformer layers, hidden size 1024, 16 attention heads.
- **Optimization:** AdamW with linear warm-up (10% of steps) and cosine decay schedule [20].
- **Regularization:** dropout, weight decay, layer dropout, and gradient clipping to prevent overfitting and exploding gradients.

3.7 Evaluation Metrics

We combine automatic and human evaluations to measure perplexity, fluency, and task-oriented performance in Nepali.

- **Perplexity:** standard held-out metric for language models.
- **Human Evaluation:** assess fluency, relevance, cultural appropriateness in Nepali dialogues.
- **Downstream Tasks:** test on Nepali question answering, translation quality (BLEU/ROUGE), and summarization.

CHAPTER 4

SYSTEM DESIGN

4.1 Requirement Specification

4.1.1 Functional Requirements

The chatbot must fulfill the following core functional requirements to ensure effective operation and user interaction:

4.1.1.1 Natural Language Understanding (NLU)

The chatbot should interpret user queries accurately, understand the intent behind each message, and maintain conversational context. This enables personalized responses and task execution.

4.1.1.2 Response Generation

The chatbot should generate contextually appropriate responses based on user intent and extracted information. This will ensure smooth and relevant conversations.

4.1.1.3 User Engagement

The chatbot must engage users through interactive techniques, such as asking questions, offering suggestions, and responding to feedback. This ensures continuous user involvement throughout the interaction.

4.1.1.4 Task Execution

The chatbot must be capable of efficiently executing tasks such as retrieving information, answering queries, and providing services based on user requests.

4.1.2 Non-Functional Requirements

In addition to functional requirements, the system must also meet the following non-functional criteria:

4.1.2.1 Accuracy

The chatbot should maintain high accuracy in understanding user queries and generating responses.

4.1.2.2 Speed

The chatbot must deliver responses quickly to maintain user engagement and satisfaction.

4.1.2.3 Scalability

The system must scale to handle increasing user interactions and complex tasks without performance degradation.

4.1.2.4 Usability

The system should be easy to navigate, with clear user interactions and feedback mechanisms.

4.1.2.5 Reliability

The chatbot must provide consistent, uninterrupted service to ensure high availability and effective user interactions.

4.1.2.6 Maintainability

The chatbot should be designed with modularity for easy updates and troubleshooting.

4.2 Feasibility Assessment

4.2.1 Economic Feasibility

The project is economically feasible, with minimal upfront costs. The dataset is sourced from publicly available platforms, and the required computational resources (laptops and institution-provided GPUs) are already in place.

4.2.2 Technical Feasibility

The technical feasibility is high, as there is access to sufficient data and computational resources (e.g., GPUs) for training. The complexity of scraping and preparing data is manageable with available tools.

4.2.3 Operational Feasibility

The system can be integrated into existing workflows with minimal adaptation required. The user interface will be intuitive, ensuring smooth adoption.

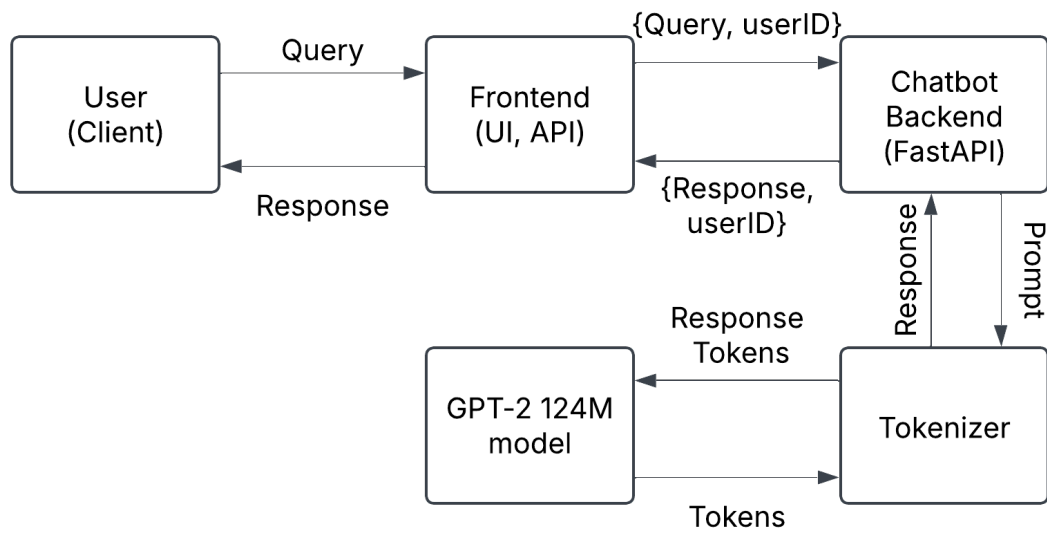


Figure 4.1: System Architecture

4.3 Proposed System Architecture

4.4 Use Case Diagram

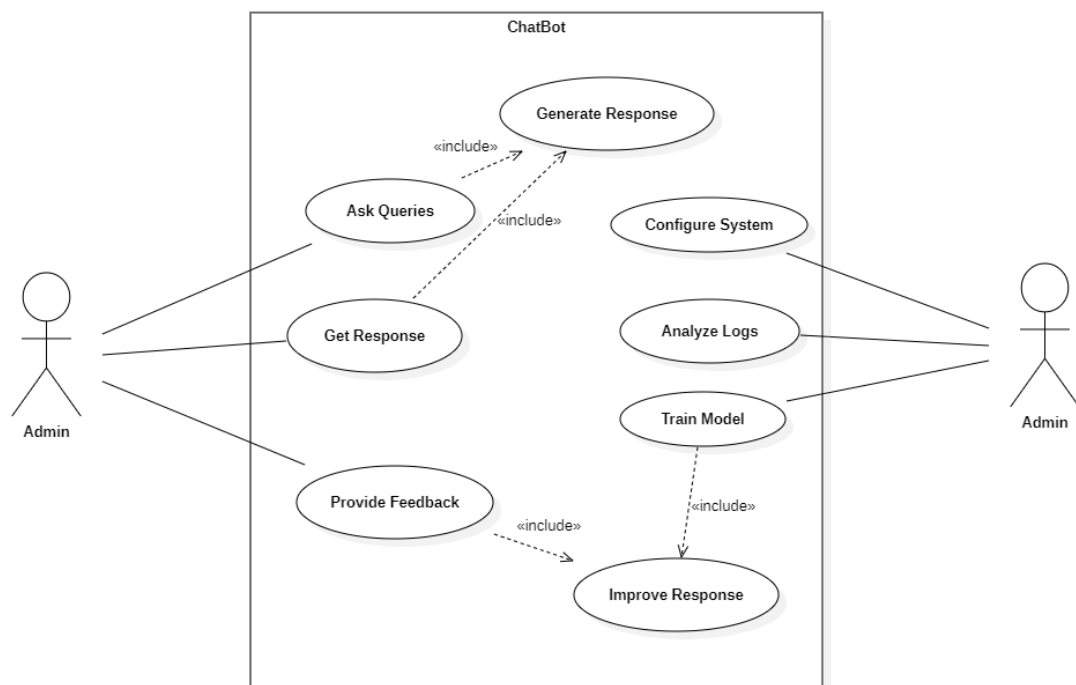


Figure 4.2: System Use Case Diagram

The diagram illustrates the interactions between the primary user, who can ask queries to the chatbot, and the admin, who configures the chatbot's knowledge base and response mechanisms.

4.5 Sequence Diagram

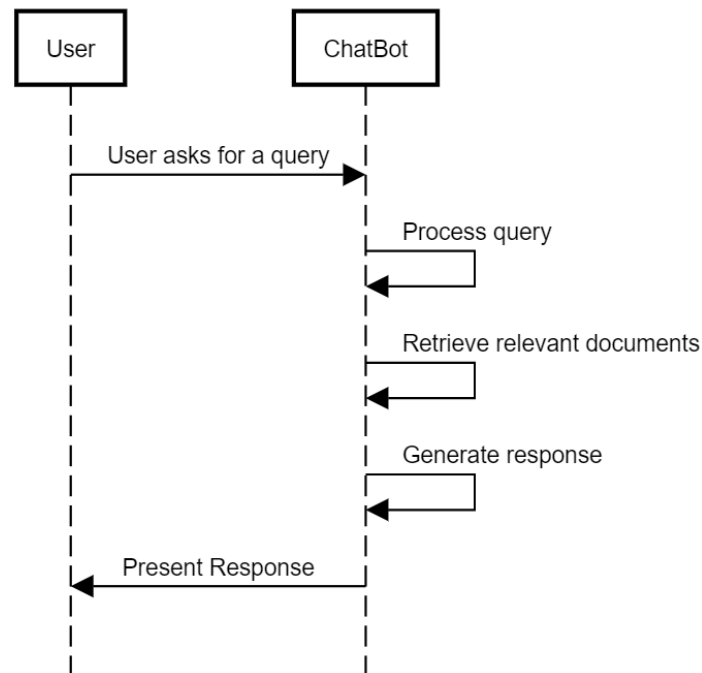


Figure 4.3: Sequence Diagram

A Sequence Diagram illustrates how objects in a system interact over time by showing the order of messages exchanged between them. The sequence diagram here shows the interaction between the user and chatbot over time. Here, when the user submits a query, the chatbot analyzes the query and generates a relevant response based on what it thinks the user is asking and generates the proper response accordingly. The generated response may need formatting. The post-processor applies proper indentation and structuring to make it more readable for the user.

4.6 Class Diagram

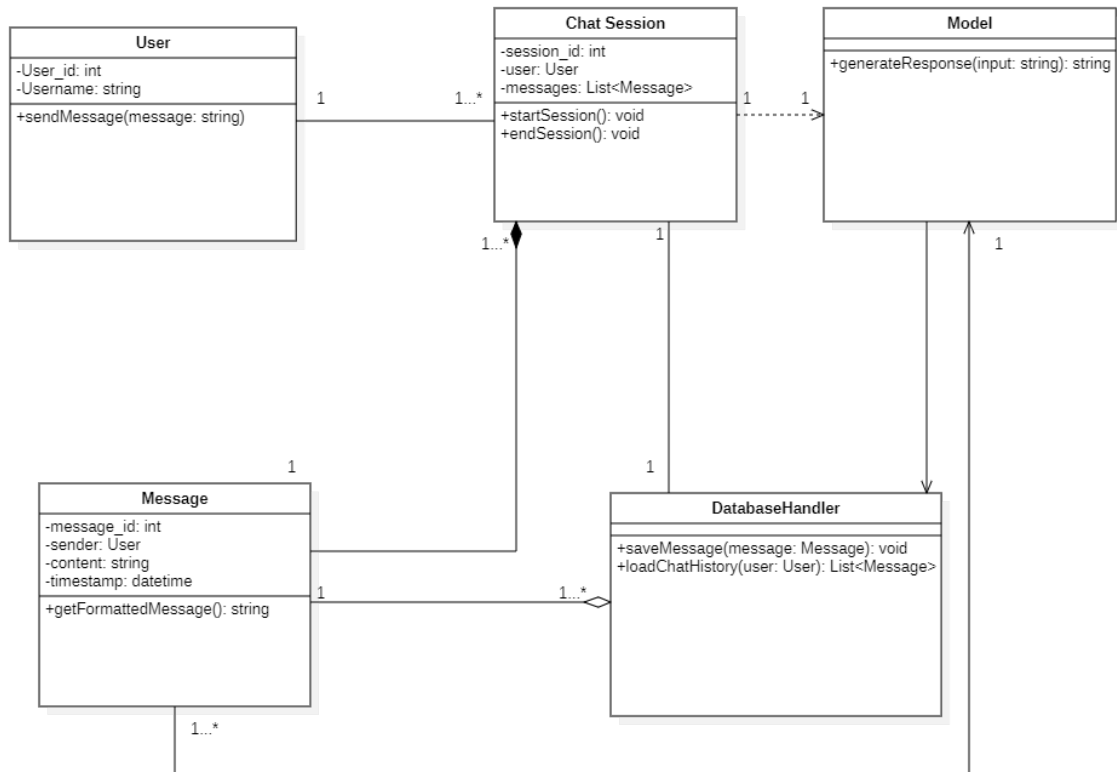


Figure 4.4: Class Diagram

A Class Diagram visualizes the structure of a system by showing its classes, attributes, methods, and relationships between them. The Class Diagram here represents the static structure of a chatbot by showing its main components, their attributes, and methods. Here, the user class contains user id and username to differentiate different users where the user can send messages to the model which goes to the chat session. The chat session contains session id which is for different context the model can get help to know what the user is talking about. The chat sessions are stored in a database for future reference. The model generates a response based on the chat session's context and the latest user message.

4.7 Activity Diagram

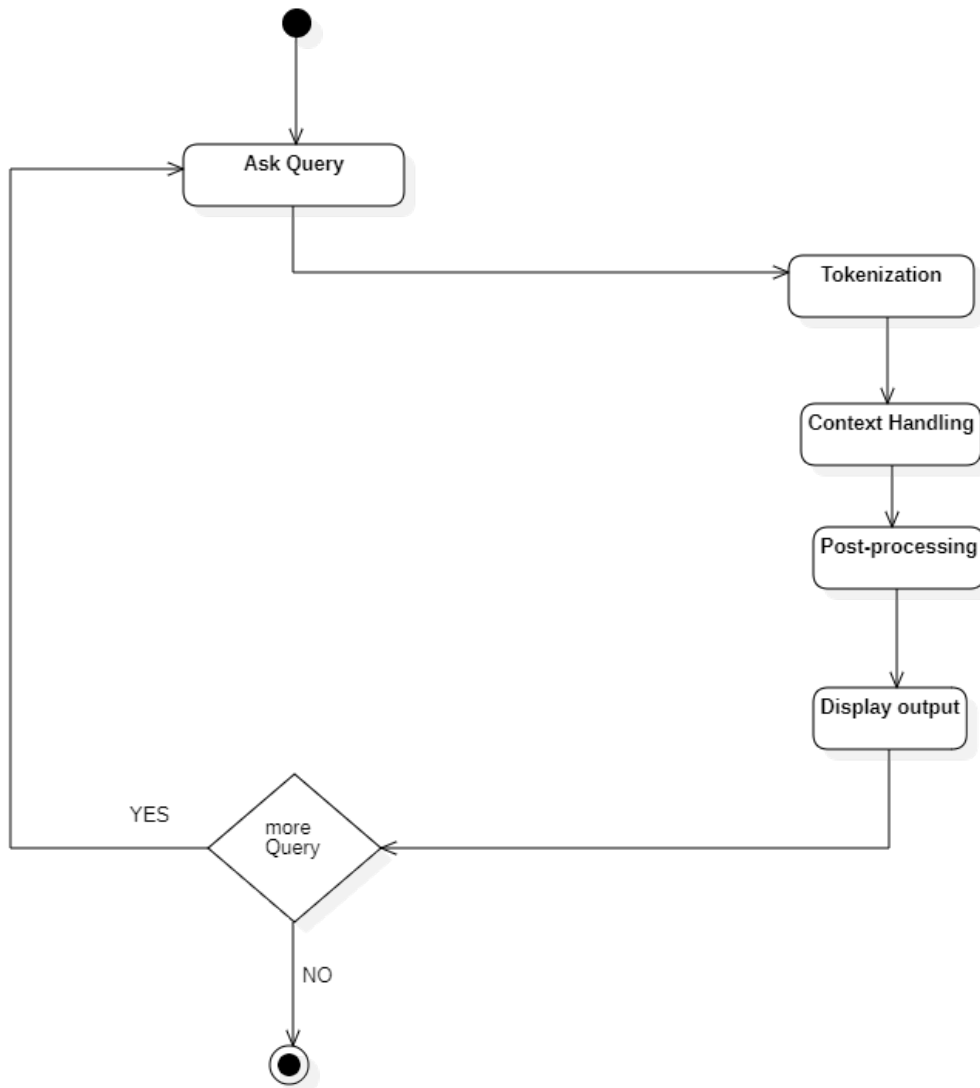


Figure 4.5: Activity Diagram

The Activity Diagram illustrates the step-by-step execution of activities in a process. It includes how the user interacts with the system and how the system processes the query to generate and return a response. Here, when the user submits a query, it goes to pre-processor which is responsible for tokenization, the system extracts relevant context or retrieves related information if needed and The model generates a response, which is then post-processed which is then displayed to the user.

4.8 Collaboration Diagram

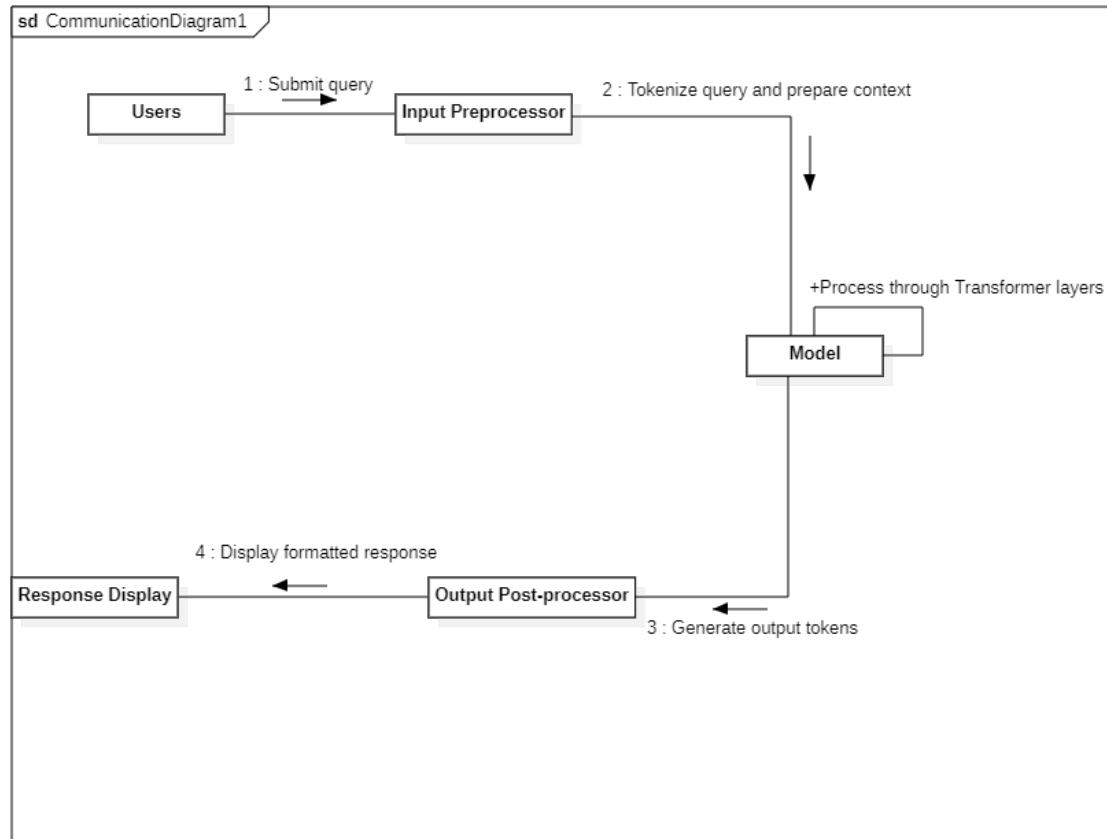


Figure 4.6: Collaboration Diagram

The collaboration diagram illustrates the interaction between the objects in the system and how messages are transferred between them along with the order of interaction between the objects. Here, first, the users submit the query which will go to the pre-processor, which is responsible for tokenization and text normalization before passing it to the model to make it easier for the model. Then, the model passes the preprocessed query through the transformer layers multiple times and generates output tokens which are then displayed with proper formatting.

4.9 Deployment Diagram

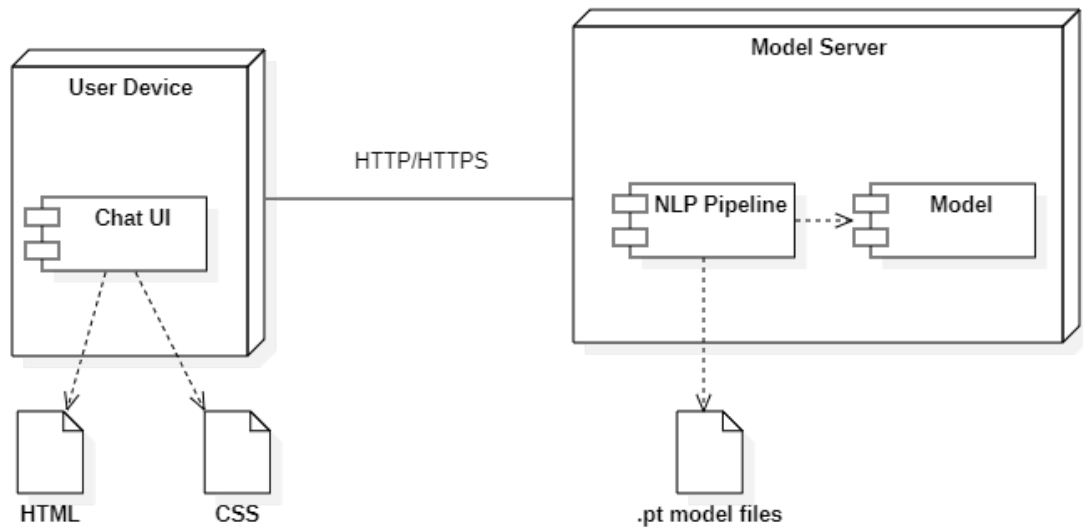


Figure 4.7: Deployment Diagram

The deployment diagram shows the hardware part of the device where software is used. It shows how the software is deployed in hardware nodes and represents the physical architecture along with nodes, containers, components, and artifacts and the connection between the nodes which shows protocols, message types, and data flow direction. Here the chatbot, shows how the user device will contain the chat UI and how it is made up of HTML and CSS. The user device communicates with the model server using HTTP/HTTPS. Then, the user query is passed into NLP pipelines in the server which contains the .pt model files which will return the answer accordingly.

CHAPTER 5

METHODOLOGY

5.1 Training Pipeline

5.1.1 Tokenization

The tokenization phase involves segmenting text into words or subwords and subsequently mapping them to numerical integer representations. In this project, the **BPE tokenizer** [21] has been utilized. This approach ensures reproducibility in normalization and subword segmentation while converting text into an **ID sequence**.

The specific tokenization configuration is as follows:

- **Vocabulary Size** (*vocab_size*): 50,257 representing the total number of unique tokens in the vocabulary.
- **Character Coverage** (*character_coverage*): 0.9995, ensuring nearly all characters are included, thereby minimizing the occurrence of unknown tokens.
- **Unknown Token ID** (*unk_id*): 1, representing the unknown token.
- **Padding Token ID** (*pad_id*): 0, used for sequence padding.
- **Beginning-of-Sequence Token ID** (*bos_id*): 2, indicating the start of a sequence.
- **End-of-Sequence Token ID** (*eos_id*): 3, marking the end of a sequence.
- **User-Defined Symbols** (*user_defined_symbols*): {"<|im_start|>", "<|im_end|>", "<|im_sep|>", "<|system|>", "<|user|>", "<|assistant|>"}, introduced to facilitate structured interactions within the model.

_औजार	-14496
_ब्याब	-14497
इस	-14498
_भान	-14499
_मत्	-14500
_पाइन	-14501
_पश्चिममा	-14502
_राष्ट्रपतिबाट	-14503
नुस्	-14504
_बिहानदेखि	-14505
_अत्त	-14506
हेरी	-14507
_बिउ	-14508
_यती	-14509
_सुह	-14510
_बाइक	-14511
_कतिले	-14512
_गर्छौ	-14513
_विवादले	-14514
_पाठ्यपुस्त	-14515
_उज्	-14516
_अँध्यारो	-14517
_जस्ताको	-14518
_पूजाआजा	-14519
_मृत्युपछि	-14520
_प्रम	-14521
_शिशुको	-14522

Figure 5.1: Vocabulary generated after training the BPE tokenizer.

Sentence to tokenize: हेल्लो मेरो नाम राम हो
 Encoded sentence: ['_हे', 'ल्लो', '_मेरो', '_नाम', '_राम', '_हो']
 Decoded sentence: हेल्लो मेरो नाम राम हो
 Compression ratio: 3.6666666666666665|

Figure 5.2: Tokenization of a Nepali sentence.

5.1.2 Model

For the training of the language model, we opted to recreate the GPT-2 124M [3] architecture from scratch. GPT-2 is a transformer-based generative model recognized for its ability to generate coherent and contextually relevant text. Originally introduced by OpenAI, GPT-2 has demonstrated remarkable capabilities in natural language understanding and generation across a wide range of tasks. Instead of fine-tuning an existing model, we train the GPT-2 124M architecture from scratch on our domain-specific Nepali dataset to ensure effective learning of language structure and context.

All GPT-based models employ an unsupervised pretraining approach on unlabeled text data. Specifically, GPT-2 is trained using *causal language modeling (CLM)*, wherein the model predicts the next token in a sequence based on preceding tokens. Unlike BERT, which learns bidirectional context through *masked language modeling (MLM)*, GPT models follow an autoregressive training paradigm, processing text sequentially from left to right.

The GPT-2 model is built on a **Transformer decoder-only** architecture, which leverages self-attention mechanisms to process input sequences in parallel. This architecture enables the model to capture long-range dependencies and contextual relationships effectively, making it well-suited for generating fluent and meaningful text.

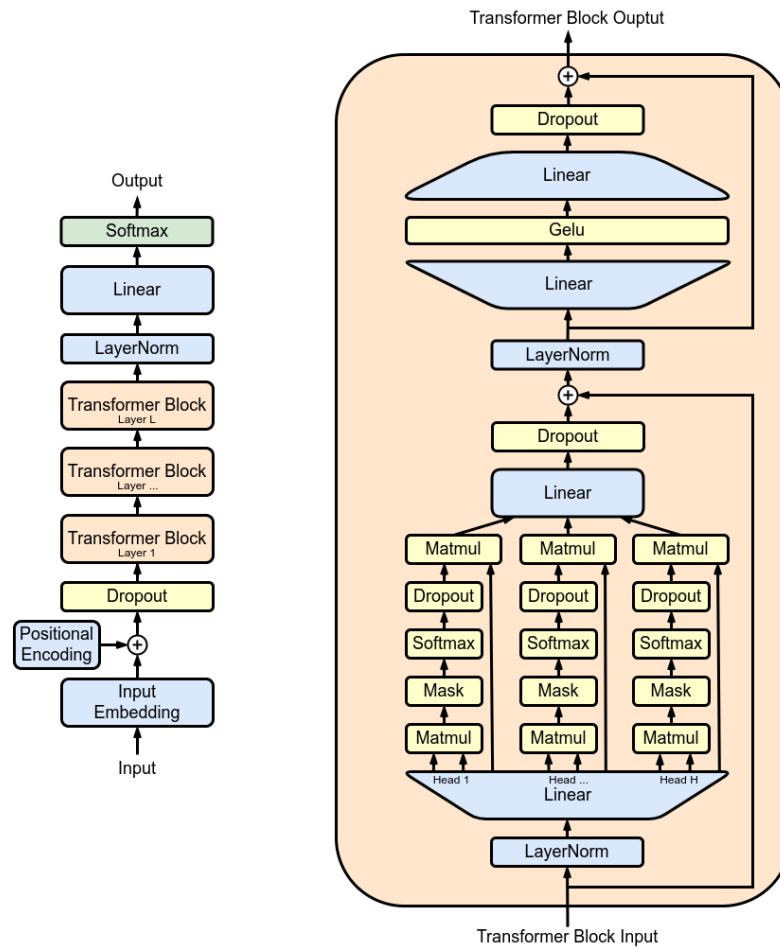


Figure 5.3: GPT 2 Architecture

5.1.2.1 Model Parameters (GPT-2 124M Configuration)

- **Configuration:**

- **Vocabulary size:** $V = 50304$

- * In model training, we used a vocabulary size of 50,304 instead of the standard GPT-2 vocabulary size of 50,257. This choice was made because 50,304 is a multiple of 8, 16, and 32, leading to better performance due to efficient tensor operations.

The extra vocabulary tokens are never activated as they are not present in the dataset. Although memory consumption increases slightly, the overall performance improves due to better hardware utilization.

- **Embedding dimension:** $E = 768$

- **Number of transformer layers:** $L = 12$

- **Feed-Forward hidden size:** $H = 4E = 3072$

- **Embedding and Positional Encoding:**

- **Embedding Layer:**

- * Each token is represented by an E -dimensional vector.

- * **Parameters:**

$$V \times E = 50304 \times 768 = 38\,633\,472$$

- **Positional Encoding:**

- * Each layer has a positional embedding of size E .

- * **Parameters:**

$$L \times E = 12 \times 768 = 9\,216$$

- **Total Embedding-Related Parameters:**

$$38\,633\,472 + 9\,216 = 38\,642\,688$$

- **Transformer Block (Per Layer):**

- **Multi-Head Attention:**

- * GPT-2 uses four linear projections (query, key, value, and output), each with a weight matrix and bias.

- * **Parameters:**

$$4(E^2 + E) = 4E^2 + 4E$$

- * For $E = 768$:

$$4 \times 768^2 + 4 \times 768 = 4 \times 589\,824 + 3\,072 = 2\,359\,296 + 3\,072 = 2\,362\,368$$

- **Feed-Forward Network:**

- * Comprises two linear layers:

- **First Linear Layer (fc):**

$$E \times H + H = 768 \times 3072 + 3072 = 2\,359\,296 + 3072 = 2\,362\,368$$

• **Second Linear Layer (proj):**

$$H \times E + E = 3072 \times 768 + 768 = 2\,359\,296 + 768 = 2\,360\,064$$

* **Total Feed-Forward Parameters:**

$$2\,362\,368 + 2\,360\,064 = 4\,722\,432$$

– **Layer Normalization:**

- * Each transformer block uses two layer normalization modules (one before attention and one before the feed-forward network), with learnable scale and bias parameters.

* **Parameters per block:**

$$2 \times (2E) = 4E = 4 \times 768 = 3\,072$$

– **Total Parameters per Transformer Block:**

- * Summing the components:

$$\begin{aligned} \text{Parameters per Block} &= \text{Multi-Head Attention} + \text{Feed-Forward Network} \\ &\quad + \text{Layer Normalization} \end{aligned}$$

$$= (4E^2 + 4E) + (2EH + E + H) + 4E$$

- * Substituting $H = 4E$:

$$\begin{aligned} &= (4E^2 + 4E) + (2E(4E) + E + 4E) + 4E \\ &= 4E^2 + 4E + 8E^2 + 5E + 4E \\ &= 12E^2 + 13E \end{aligned}$$

- * For $E = 768$:

$$12 \times 768^2 + 13 \times 768 = 12 \times 589\,824 + 9\,984 = 7\,077\,888 + 9\,984 = 7\,087\,872$$

- * Thus, each transformer block has **7,087,872 parameters**.

• **Total Transformer Parameters:**

- For $L = 12$ layers:

$$L \times (12E^2 + 13E) = 12 \times 7\,087\,872 = 85\,054\,464$$

– **Final Layer Normalization:**

- * Applied after all blocks with $2E$ parameters:

$$2E = 2 \times 768 = 1\,536$$

• **Overall Model Parameter Count:**

- Summing all components:

$$\begin{aligned} \text{Total Parameters} &= \text{Embedding Layer} + \text{Positional Encoding} + \text{Transformer Blocks} \\ &\quad + \text{Final Layer Normalization} \end{aligned}$$

$$\begin{aligned} &= (V \times E) + (L \times E) + [L \times (12E^2 + 13E)] + 2E \\ &= 38\,633\,472 + 9\,216 + 85\,054\,464 + 1\,536 \\ &= 123\,698\,688 \end{aligned}$$

- This value approximates the 124M parameters reported for the GPT-2 124M model.

5.1.2.2 Model Summary

Layer (type:depth-idx)	Input Shape	Output Shape	Param #
GPT	[1, 1024]	[1, 1024, 50257]	--
└─ModuleDict: 1-1	--	--	--
└─Embedding: 2-1	[1024]	[1024, 768]	786,432
└─Embedding: 2-2	[1, 1024]	[1, 1024, 768]	38,597,376
└─ModuleList: 2-3	--	--	--
└─Block: 3-1	[1, 1024, 768]	[1, 1024, 768]	7,087,872
└─Block: 3-2	[1, 1024, 768]	[1, 1024, 768]	7,087,872
└─Block: 3-3	[1, 1024, 768]	[1, 1024, 768]	7,087,872
└─Block: 3-4	[1, 1024, 768]	[1, 1024, 768]	7,087,872
└─Block: 3-5	[1, 1024, 768]	[1, 1024, 768]	7,087,872
└─Block: 3-6	[1, 1024, 768]	[1, 1024, 768]	7,087,872
└─Block: 3-7	[1, 1024, 768]	[1, 1024, 768]	7,087,872
└─Block: 3-8	[1, 1024, 768]	[1, 1024, 768]	7,087,872
└─Block: 3-9	[1, 1024, 768]	[1, 1024, 768]	7,087,872
└─Block: 3-10	[1, 1024, 768]	[1, 1024, 768]	7,087,872
└─Block: 3-11	[1, 1024, 768]	[1, 1024, 768]	7,087,872
└─Block: 3-12	[1, 1024, 768]	[1, 1024, 768]	7,087,872
└─LayerNorm: 2-4	[1, 1024, 768]	[1, 1024, 768]	1,536
└─Linear: 1-2	[1, 1024, 768]	[1, 1024, 50257]	38,597,376
Total params: 163,037,184			
Trainable params: 163,037,184			
Non-trainable params: 0			
Total mult-adds (M): 967.56			
Input size (MB): 0.01			
Forward/backward pass size (MB): 1261.05			
Params size (MB): 652.15			
Estimated Total Size (MB): 1913.21			

Figure 5.4: Model Summary

1. Parameter Count Mismatch

During model summary generation, the embedding and output projection layers were counted separately, even though weight tying was used. This caused the total parameter count to appear higher than expected. The increase in vocabulary size also slightly contributes to this.

2. Total Parameters: 163,037,184

This represents the total number of learnable parameters in the model. The increased count results from the separate accounting of tied layers.

3. Trainable Parameters: 163,037,184

All parameters are trainable, meaning the model has full learning capacity but requires more computation.

4. Non-Trainable Parameters: 0

No frozen layers exist, ensuring that all weights contribute to learning.

5. Total Mult-Adds: 967.56M

This refers to the number of multiplication-addition operations per forward pass. With nearly 1 billion operations, the model has a high computational cost and benefits from GPU acceleration.

6. Memory Usage

- **Input Size:** 0.01 MB – The input storage requirement is negligible.
- **Forward/Backward Pass Size:** 1261.05 MB – Over 1GB of memory is needed for activations, making training on GPUs with less than 12GB VRAM difficult.
- **Parameter Storage:** 652.15 MB – The memory required to store model parameters.

- **Estimated Total Size:** 1913.21 MB – The model requires nearly 2GB of memory, necessitating optimizations such as mixed precision training on lower-end hardware.

5.1.3 Training Configurations

The pretraining and post-training phases of our GPT-2 model utilized an optimization strategy that incorporated gradient accumulation, mixed precision training, and gradient clipping to ensure stable and efficient training.

5.1.3.1 Hyperparameters

The following hyperparameters were used during pretraining:

1. **Transformer Layers:** 12 layers stacked to enhance the model’s ability to learn complex representations.
2. **Number of Attention Heads:** Each transformer layer consists of 12 attention heads, enabling the model to focus on different parts of the input sequence.
3. **Embedding Dimension:** Tokens are represented as 768-dimensional vectors.
4. **Batch Size:** Set to 524,288 for stable gradient estimates while ensuring efficient memory utilization.
5. **Learning Rate:** 6e-4, balancing convergence speed and stability.
6. **Weight Decay:** 0.1, applied as a regularization technique to improve generalization.
7. **Warm-up Steps:** 715 steps to stabilize training and prevent divergence.
8. **Optimizer:** AdamW with $\beta_1 = 0.9$ (controls past gradient smoothing) and $\beta_2 = 0.95$ (controls squared gradient smoothing).
9. **Micro Batch Size:** 8, defining the number of samples processed before updating the gradients.
10. **Sequence Length:** 1024 tokens, the maximum input size the model can handle in a single forward pass.

5.1.3.2 Loss Function

The model was trained using the **Cross-Entropy Loss**, a standard loss function for language modeling tasks. Given the predicted logits and the actual target tokens, the cross-entropy loss is computed as:

$$\mathcal{L} = - \sum_i y_i \log(\hat{y}_i) \quad (5.1)$$

where y_i represents the true token distribution and \hat{y}_i represents the predicted token probabilities. This loss function effectively minimizes the divergence between the predicted and actual token distributions.

5.1.3.3 Optimizer

The optimization process was carried out using the **AdamW** [22] optimizer, which is an improved variant of the Adam optimizer with decoupled weight decay regularization. This optimizer helps prevent over-regularization and improves convergence. The weight update rule for AdamW is given by:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad (5.2)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \quad (5.3)$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad (5.4)$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (5.5)$$

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t - \eta \lambda \theta_{t-1} \quad (5.6)$$

where:

- m_t and v_t are first and second moment estimates.
- β_1 and β_2 are decay rates for moment estimates.
- η is the learning rate.
- λ is the weight decay factor.
- g_t represents the gradient at time step t .

5.1.3.4 Gradient Clipping

To prevent exploding gradients, the gradient norm was clipped to a maximum value of 1.0:

$$\theta_t = \theta_t \cdot \min \left(1, \frac{1.0}{\|\theta_t\|} \right) \quad (5.7)$$

This technique ensures stable training by limiting excessively large gradient updates.

5.1.3.5 Learning Rate Scheduling

A dynamic learning rate schedule was implemented to ensure stable and efficient training. The learning rate followed a **linear warmup** phase followed by a **cosine decay** schedule.

Warmup Phase During the first **715 steps**, the learning rate increased linearly from **0** to the maximum learning rate $\eta_{\max} = 6 \times 10^{-4}$. The learning rate during this phase was calculated as:

$$\eta_t = \eta_{\max} \times \frac{t + 1}{\text{warmup_steps}} \quad (5.8)$$

where t represents the current training step.

Cosine Decay Phase After the warmup phase, the learning rate followed a cosine decay schedule down to a minimum learning rate $\eta_{\min} = 0.1 \times \eta_{\max}$ over a total of **21,400 steps**. The learning rate at step t was given by:

$$\eta_t = \eta_{\min} + 0.5 \times (\eta_{\max} - \eta_{\min}) \times \left(1 + \cos \left(\pi \times \frac{t - \text{warmup_steps}}{\text{max_steps} - \text{warmup_steps}} \right) \right) \quad (5.9)$$

where:

- $\eta_{\max} = 6 \times 10^{-4}$ is the maximum learning rate.
- $\eta_{\min} = 6 \times 10^{-5}$ is the minimum learning rate.
- $\text{warmup_steps} = 715$ defines the warmup period.
- $\text{max_steps} = 21,400$ determines the total number of training steps.

Final Learning Rate Phase For steps beyond **19,073**, the learning rate remained constant at η_{\min} , preventing the model from excessive degradation in later training stages.

This learning rate schedule helped ensure a smooth transition from warmup to effective training while preventing sudden drops in learning rate, leading to better model convergence.

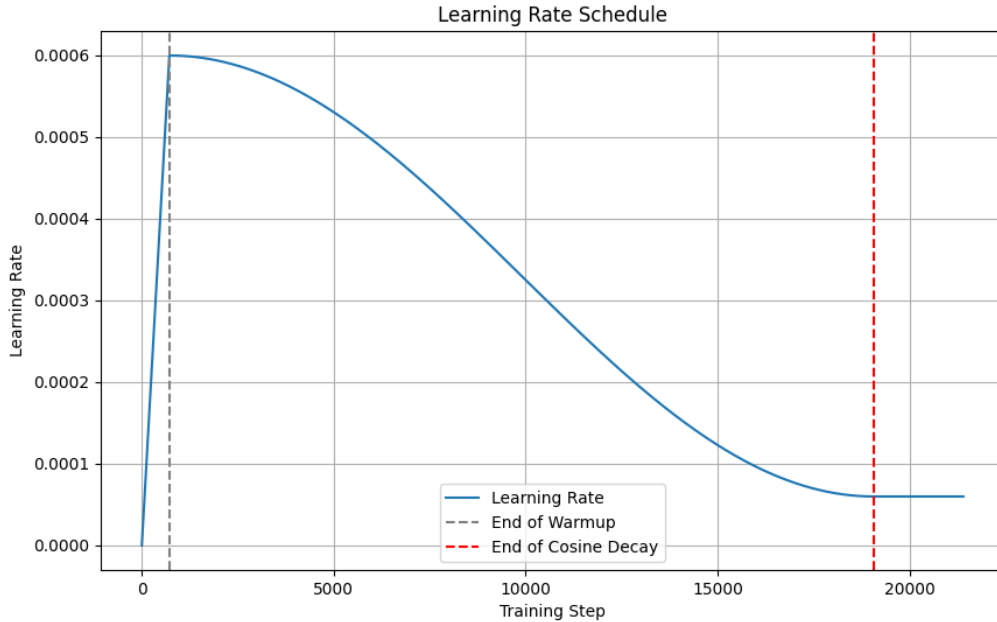


Figure 5.5: Cosine Decay Lr Scheduler

5.1.3.6 Training Performance

The training performance was monitored by tracking key metrics such as:

- Training loss and validation loss.

- Learning rate updates.
- Gradient norms to ensure stability.
- Processing speed in terms of tokens per second.

This approach ensured efficient and stable training, leading to improved performance of the final language model.

5.1.4 Cost Optimization

5.1.4.1 Gradient Accumulation

When training large models, memory constraints often limit the batch size that can be processed at once. **Gradient accumulation** addresses this by computing gradients over several mini-batches and then performing a single weight update. Instead of updating weights after every mini-batch, gradients are accumulated over multiple steps, simulating a larger batch size without exceeding memory limits.

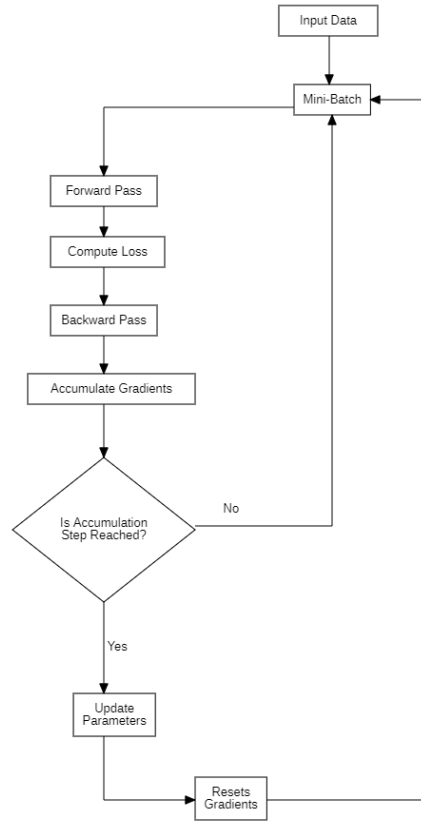


Figure 5.6: Gradient Accumulation

5.1.4.2 Mixed Precision Training

Mixed precision training uses both high-precision (32-bit) and low-precision (bfloat16) arithmetic to speed up computations and reduce memory usage. With `torch.autocast`, operations that are safe to compute in lower precision are automatically cast to bfloat16, while critical operations remain in higher precision to maintain numerical stability. This results in faster computations and lower memory consumption.

5.1.4.3 Using Numbers as Powers of Two

Optimizing array sizes and computational dimensions to be powers of two can lead to significant performance improvements. Many hardware architectures and algorithmic implementations (such as FFTs and matrix multiplications) are optimized for power-of-two sizes, leading to better memory alignment, enhanced caching, and an overall performance improvement of about 30%.

5.1.4.4 Flash Attention

Flash Attention is an optimized attention mechanism that reduces both the memory and computational overhead associated with traditional attention operations in Transformer models. By fusing multiple operations into a single efficient kernel, Flash Attention minimizes memory reads/writes and intermediate computations, achieving up to 7.6x speed improvements.

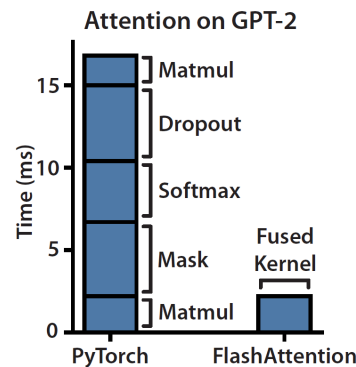


Figure 5.7: Flash Attention Illustration

5.1.4.5 Fused Adam Optimizer

The **fused Adam** optimizer is an enhanced version of the Adam optimizer that combines multiple element-wise operations into a single GPU kernel call. This fusion minimizes kernel launch overhead and reduces memory bandwidth usage, leading to faster and more efficient training compared to the standard Adam implementation.

5.1.4.6 torch.compile

With PyTorch 2.0, **torch.compile** allows models to be compiled into optimized machine code. This process converts the model's computation graph into highly optimized code through various compiler-level optimizations. The result is streamlined execution with reduced Python overhead and up to a 2.3x speed improvement.

5.1.4.7 Shared Embedding and lm_head Matrix

In language models like GPT-2, weight tying is applied by sharing the weights between the input embedding layer and the output projection layer (lm_head). This **shared embedding** strategy reduces the total number of parameters by approximately 30%, saving memory and potentially improving the consistency of word representations during both input encoding and output decoding.

5.2 Pre-Training

5.2.1 Dataset

We utilized **IRIISNEPAL/Nepali-Text-Corpus** [11] dataset from Huggingface for our pre-training. Key details are as follows:

- **Dataset Size:** 27.5GB
- **Dataset Source:** News
- **Total Rows:** 6.39M
- **Total Tokens:** Approximately 1.9 billion tokens
- **Storage Format:** Saved as NumPy arrays after tokenization for efficient loading
- **Sharding:** Each shard contains 100 million tokens
- **Data Split:**
 - **Validation Set:** First shard (100 million tokens)
 - **Training Set:** Remaining shards (approximately 1.8 billion tokens)
- **Validation-to-Training Ratio:** Approximately 1:18 (Validation constitutes about 5.3% of the total data)

5.2.2 Training

We trained the GPT-2 124M model from scratch for 5 epochs. The dataset consists of 1.9 billion tokens, with each batch containing 0.5 million tokens. The training process spanned 19,000 steps.

Each training step took approximately 8500ms, resulting in a total training time of 44.86 hours. The average processing speed was 62k tokens per second.

5.2.3 Results

The training process was monitored using a loss vs. steps curve for both training and validation loss. The validation loss was calculated every 250 steps to make the training efficient.

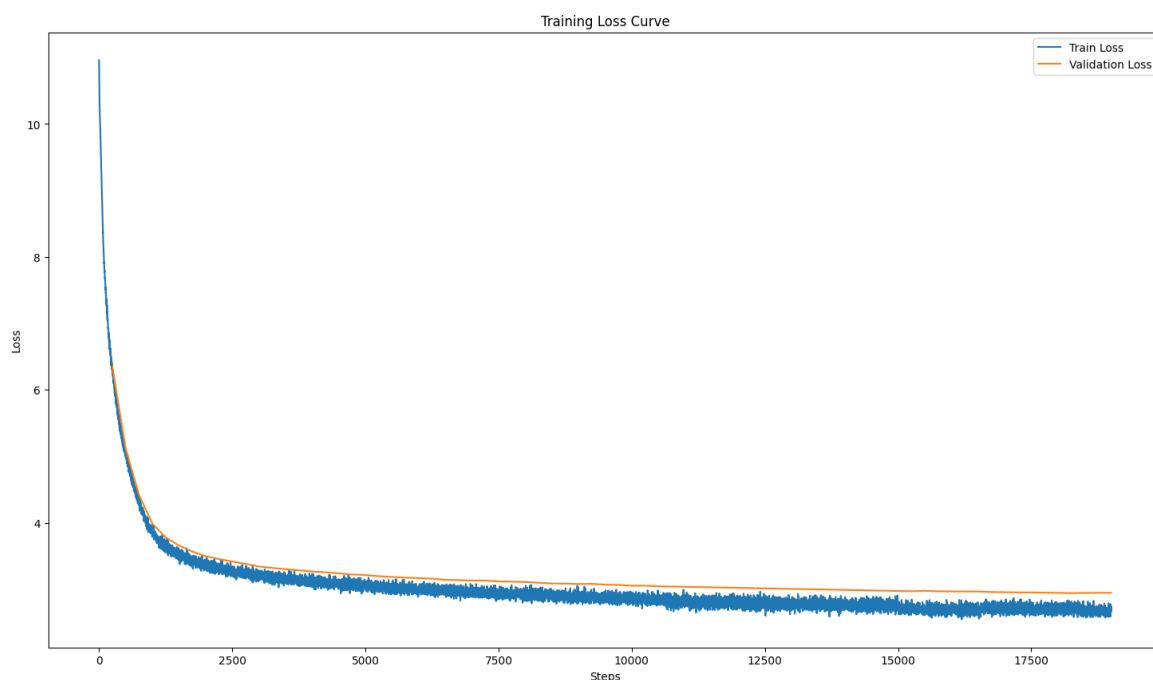


Figure 5.8: Training and validation loss vs. steps.

5.3 Pretraining Outputs

After pre-training the GPT-2 model with the large corpus, it was proficient at predicting the next word.

Here are the pre-training outputs with different temperatures:

- Prompt:** नमस्ते, म एक भाषा मोडेल हुँ,
Temperature: 0.7
Generated: नमस्ते, म एक भाषा मोडेल हुँ, म एक अंग्रेजीभाषी मान्छे । म विद्यार्थी हुँदा मेरो घरको आँगनमा हुने एउटा अंग्रेजी भाषाको गीत सुनेको थिएँ, जसमा मैले त्यही अंग्रेजीको अनुवाद बोलेको थिएँ । अंग्रेजी
- Prompt:** नमस्ते, म एउटा
Temperature: 0.8
Generated: नमस्ते, म एक भाषा मोडेल हुँ, जुन नेपालीभाषीको पहिचान र अधिकार हो। म नेपालीभाषीको पहिलो र प्रमुख पहिचान पनि हुँ। र, मलाई थाहा छ, मेरो परिचय नेपाली भाषाको
- Prompt:** नमस्ते, म एउटा
Temperature: 0.9
Generated: नमस्ते, म एक भाषा मोडेल हुँ, म एक फरक भाषा अनुवादकको रूपमा छु । त्यसैले मलाई ती गुरुहरूले तपाईंको रचना र प्रेरणाहरूको प्रयोग गरेर आफ्नो भाषामा अनुवाद गर्न प्रेरित गर्नुभएको कुरा आज पनि

5.4 Post Training

5.4.1 Dataset

We translated UltraChat_200k [23] dataset using google translate to Nepali language for our post-training. Key details are as follows:

- **Dataset Size:** 3.5 GB
- **Dataset Source:** Synthesized using ChatGPT
- **Total Conversations:** 207,865
- **Total Tokens:** 0.3B tokens
- **Storage Format:** Saved as NumPy arrays after tokenization for efficient loading
- **Sharding:** Each shard contains 60 million tokens
- **Data Split:**
 - **Validation Set:** First shard (60 million tokens)
 - **Training Set:** Remaining shards (approximately 240 million tokens)
- **Validation-to-Training Ratio:** Approximately 1:5 (Validation constitutes about 20% of the total data)

Below is an example of a **formatted input sample for post training**:

```
<|im_start|><|system|><|im_sep|>तपाईं हर्के हुनुहुन्छ, एक सहयोगी र मैत्रीपूर्ण एआई  
सहायक। तपाईं स्पष्ट र संक्षिप्त रूपमा उत्तर दिनुहोस्, तर वार्तालापलाई प्राकृतिक बनाउनुहोस्। यदि  
प्रयोगकर्ताले तथ्य सोधे भने, सही जानकारी दिनुहोस्। यदि विचार सोधे भने, तटस्थ रहनुहोस्। स्पष्टता  
आवश्यक परे, शिष्ट रूपमा सोध्नुहोस्। हानिकारक, पक्षपाती वा अनैतिक उत्तरहरू नदिनुहोस्। <|im_end|>  
<|im_start|> <|user|> <|im_sep|>नेपालको राजधानी कहाँ छ? <|im_end|> <|im_start|>  
<|assistant|><|im_sep|> नेपालको राजधानी काठमाण्डौ हो। <|im_end|>
```

This example demonstrates the required input structure for the dataset, ensuring clarity and uniformity in the training data.

5.4.2 Training

We resumed training from the saved checkpoint of the pretraining phase for 5 epochs. Given that the post-training dataset contains 0.3 billion tokens and each batch comprises 0.5 million tokens, the training process spanned 2,400 steps, continuing from the pretraining step count of 19,000 up to 21,400 steps.

At the same training rate as the pretraining phase, it took 5.67 hours to complete post-training making the total training time approx. 50.53 hours on 3080ti GPU.

5.4.3 Results

The training process was monitored using the same approach as in the pretraining phase.

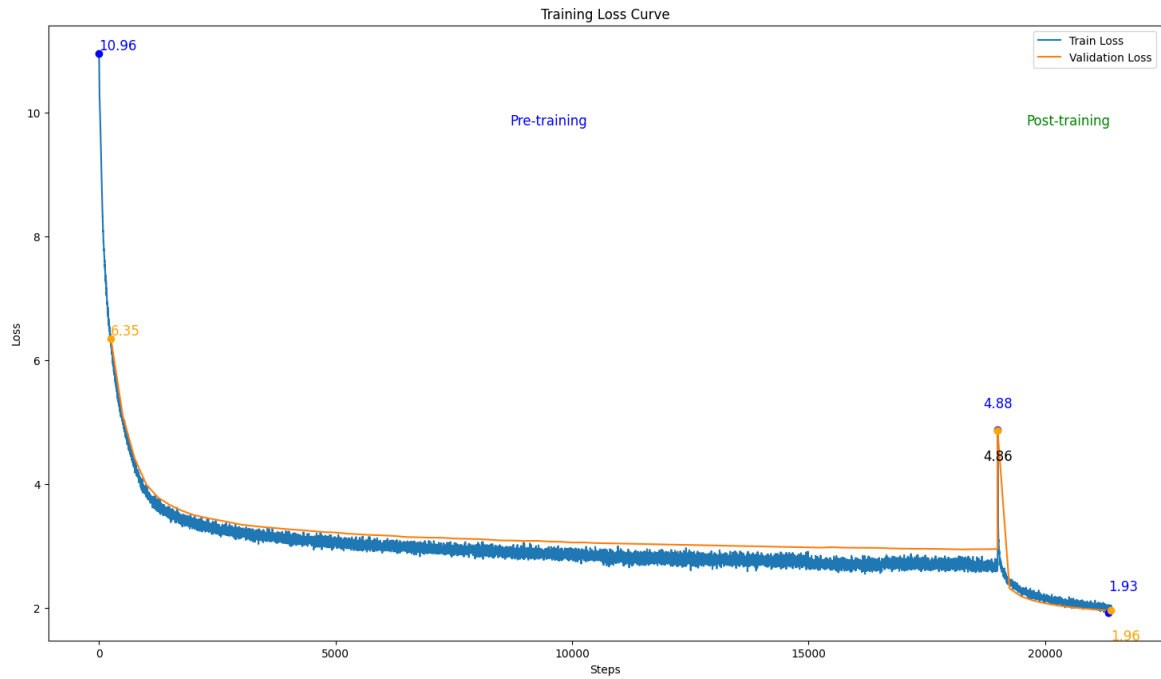


Figure 5.9: Full loss curve (pretraining + post-training)

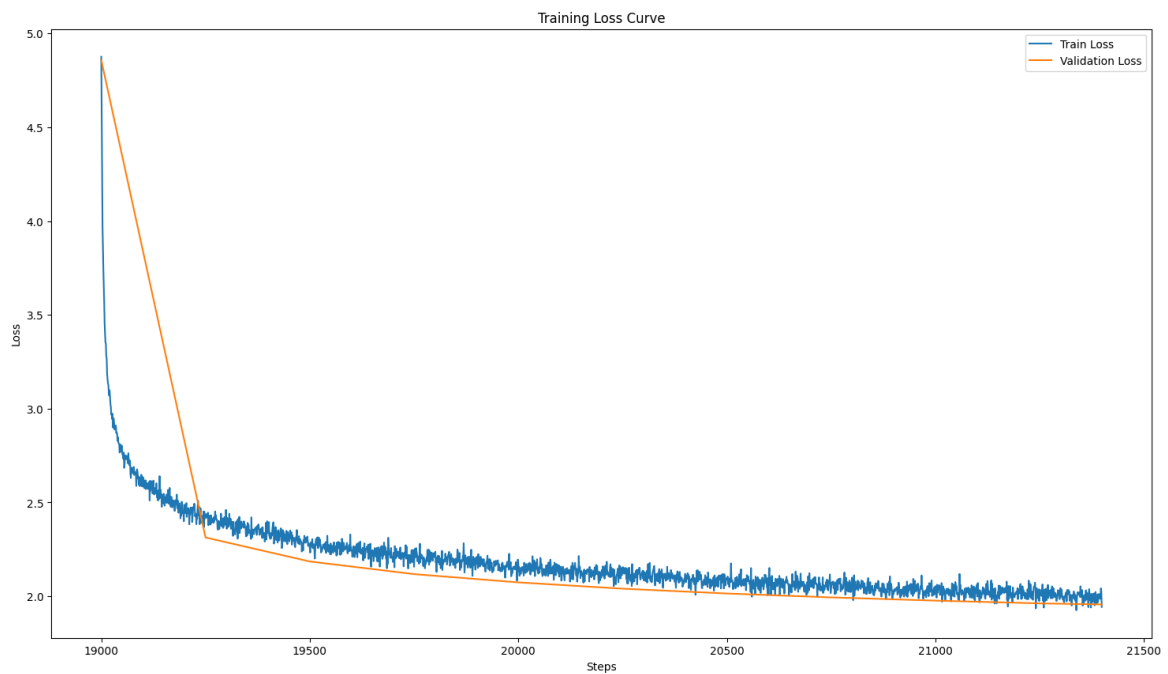


Figure 5.10: Post-training loss curve

5.5 Post-Training Outputs

After post-training the GPT-2 pre-trained model with the conversation dataset, it was ready to answer user queries.

Here are the post-training outputs with different temperatures:

- **Prompt:** तिम्रो नाम के हो?

Temperature: 0.7

Generated: मसँग व्यक्तिगत नाम वा पाठ छैन। यद्यपि, दिइएको जानकारीको आधारमा, दिइएको जानकारीको आधारमा, यो भन्न सकिन्छ कि जोन स्मिथ अङ्ग्रेजी बोल्ने शिक्षक थिए, जसलाई उनले मा अङ्ग्रेजी कक्षाको अध्यक्षको रूपमा नियुक्त गरेका थिए।

- **Prompt:** तिम्रो नाम के हो?

Temperature: 0.8

Generated: मसँग कुनै व्यक्तिगत जानकारी छैन। के तपाईं दिइएको पाठको आधारमा उसको नाम के हो भन्ने बारे विशेष जानकारी प्रदान गर्न सक्नुहुन्छ

- **Prompt:** तिम्रो नाम के हो?

Temperature: 0.9

Generated: मेरो नाम डेभिड हो, र म तपाईंलाई यसमा मद्दत गर्न पाउँदा खुसी छु। के तपाईंसँग मेरो हजुरआमाको जन्म मिति समावेश गर्ने सुन्दर छुट्टी बिदाको योजना छ

CHAPTER 6

RESULT AND DISCUSSION

6.1 Pretraining Results

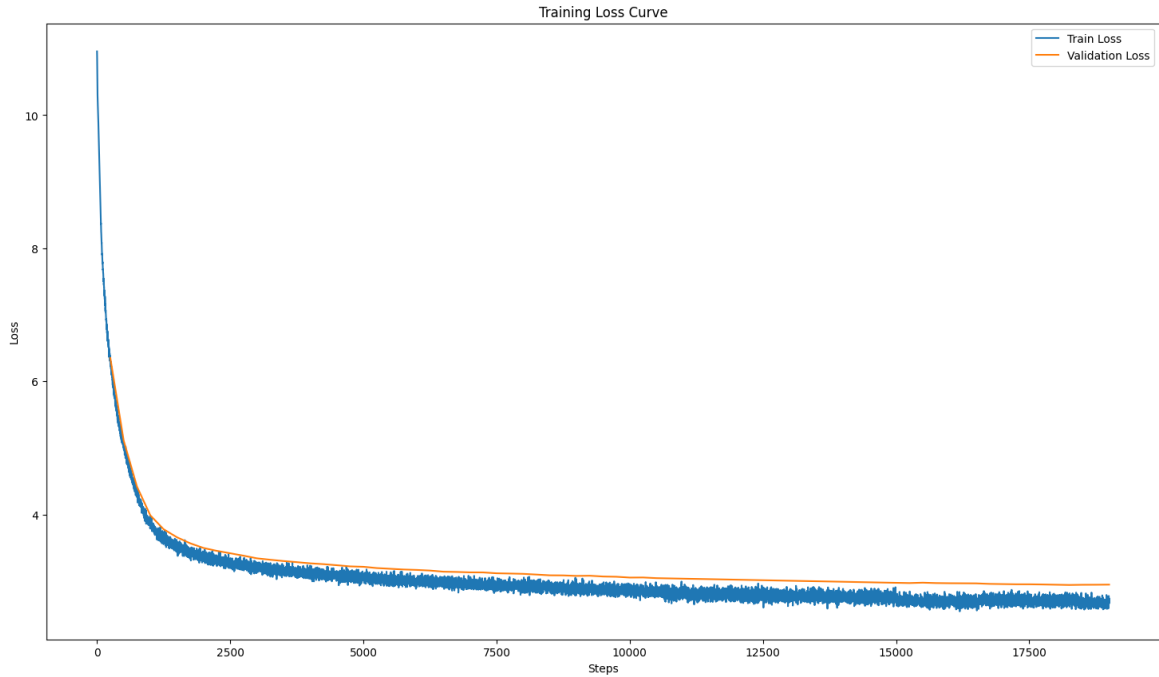


Figure 6.1: Pretraining Training and Validation Loss Plot

It is evident that both training and validation losses decrease rapidly in the initial stages, indicating that the GPT2-124M model efficiently learns fundamental linguistic patterns. At the start of training, the maximum training loss was 10.96, while the maximum validation loss was 6.35. The lower initial validation loss is attributed to the fact that validation was first performed after 250 training steps, by which time the model had already begun learning basic representations.

As training progressed, the losses continued to decrease, with the minimum training loss reaching 2.5479 and the minimum validation loss reaching 2.9445 over 19,000 steps. The relatively small gap between training and validation losses suggests minimal overfitting and indicates that the model generalizes well to unseen data. However, the plateau observed in later stages suggests diminishing returns from continued training under the current hyperparameter settings.

6.2 Post Training Results

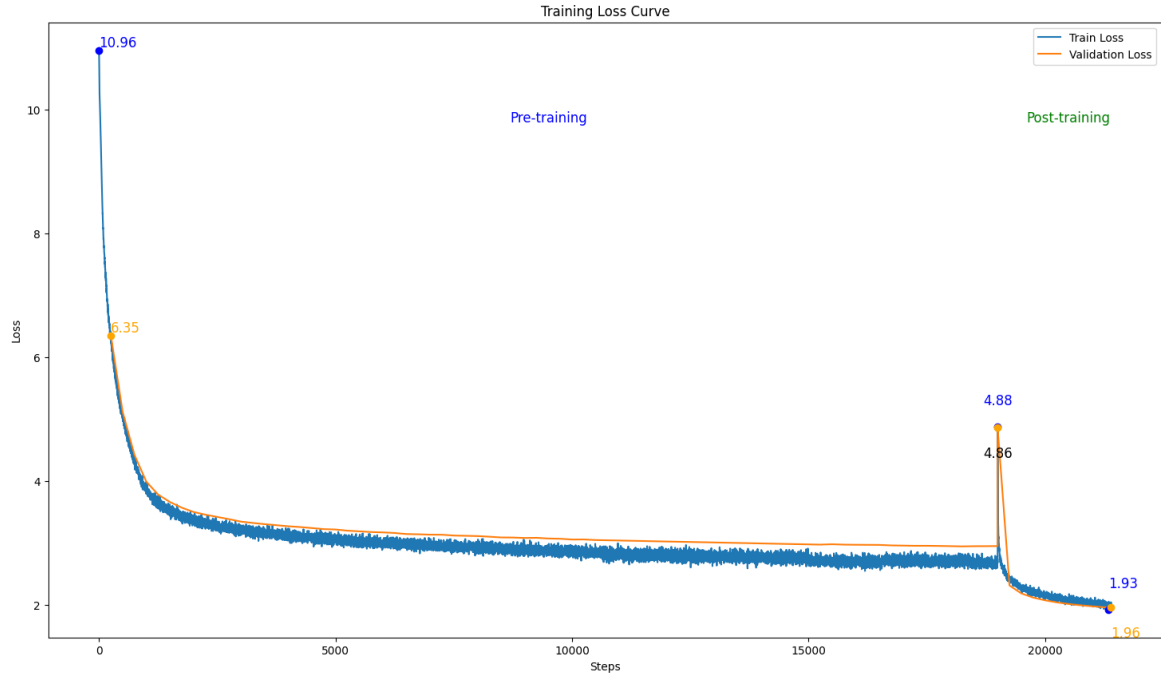


Figure 6.2: Full loss curve (pretraining + post-training)

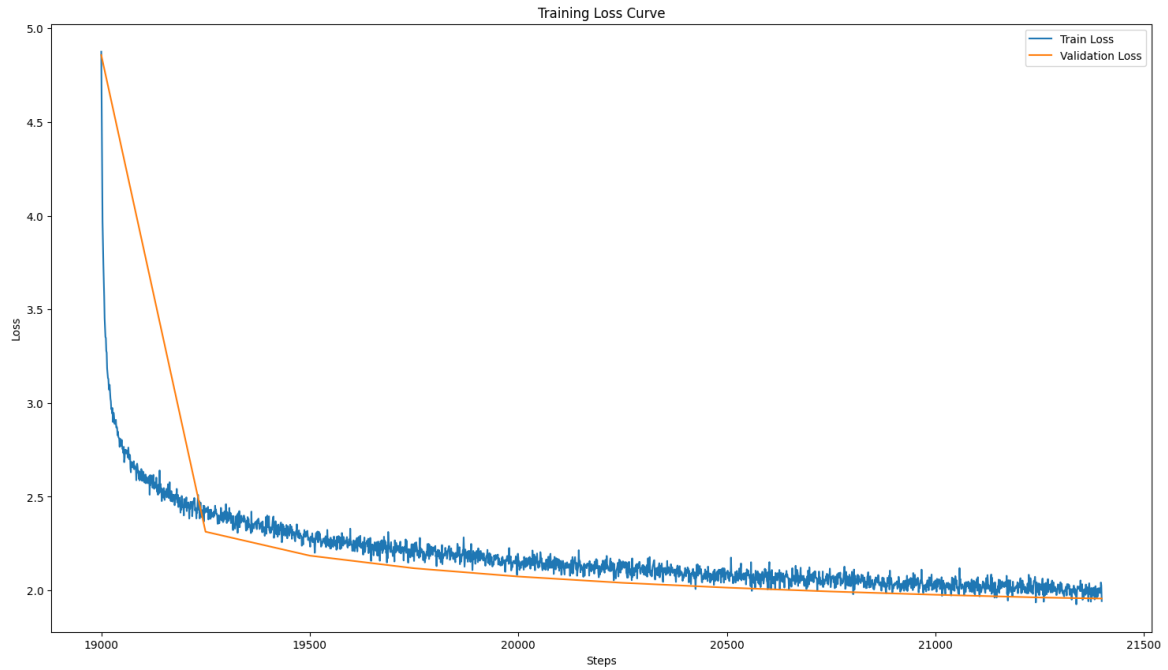


Figure 6.3: Post-training loss curve

During post-training, the dataset was switched to a conversational dataset at 19,000 training steps, resulting in an initial jump in losses, with the training loss increasing to 4.88 and the validation loss to 4.86. This increase was expected, as the model had to adapt to a new data distribution.

Following this transition, both losses exhibited a rapid decline, indicating that the model quickly adapted to the new dataset. Over the course of 21,400 post-training steps, the training loss reached a minimum of 1.93, while the validation loss reached 1.96. The close alignment between training and validation losses suggests effective learning with minimal overfitting.

Although the loss has significantly decreased, it is still converging, albeit at a smaller rate. Further training might lead to additional improvements, allowing the model to refine its understanding of conversational patterns further. Overall, the post-training phase successfully enhanced the model’s ability to handle dialogue-based tasks, improving its contextual awareness and response generation capabilities.

CHAPTER 7

EVALUATION AND COMPARISON

7.1 Evaluation

7.1.1 Evaluation Setup

To assess the performance of the trained model, we evaluated it on the HellaSwag [24] validation dataset, a benchmark designed to test commonsense reasoning. Since our model was trained for Nepali text generation, we translated the dataset into Nepali using Google Translate to align with the model’s linguistic capabilities.

7.1.2 Results

The model achieved an accuracy of 0.2541, correctly predicting 2,552 out of 10,042 examples. This result indicates that the model successfully captures some degree of commonsense reasoning but still has room for improvement.

7.1.3 Discussion

The accuracy of 25.41% suggests that the model has learned some meaningful patterns from the training data but struggles with more complex reasoning tasks. Compared to state-of-the-art language models, this performance is relatively low, likely due to several factors:

- **Limited Pretraining Data:** The model was trained from scratch on a relatively small dataset, which may have limited its ability to develop robust representations.
- **Translation Artifacts:** Since the dataset was translated using Google Translate, potential translation errors or unnatural phrasing might have introduced noise, affecting performance.
- **Model Size:** The GPT2-124 architecture is relatively small compared to larger transformer models, limiting its ability to capture complex relationships within text.
- **Domain Shift:** HellaSwag is originally designed for English commonsense reasoning, and some cultural or linguistic nuances might not transfer well into Nepali.

Although the current accuracy is modest, further improvements could be achieved by additional fine-tuning on high-quality Nepali datasets, employing better translation techniques, or increasing model capacity. Despite these limitations, the evaluation demonstrates that the model has learned to some extent and provides a foundation for future improvements.

7.2 Comparison

In this domain, there exists a significant research gap, as most published papers have not conducted a thorough evaluation of their models. Instead, they primarily report training and validation loss without providing comprehensive performance metrics.

In contrast, our study includes a comparative analysis between our model and several well-known models in this field. The following table presents the training and validation loss for these models:

Model	PARAMS	Training Loss	Validation Loss
distilgpt-nepali [25]	88.2M	3.3968	3.2705
GPT-2 [11]	124M	3.001	-
GPT-2 (Ours)	124M	2.5478	2.9445
GPT-2-Finetuned (Ours)	124M	1.9255	1.9563

Table 7.1: Comparison on Training and Validation Loss

The results in Table 7.1 indicate that our fine-tuned GPT-2 model achieves the lowest training and validation loss compared to other models. While the base GPT-2 model outperforms distilgpt-nepali, our fine-tuned GPT-2 further improves performance, reducing both training and validation loss significantly. This demonstrates the effectiveness of fine-tuning in enhancing model performance for the given task.

CHAPTER 8

SOFTWARE DEPLOYMENT

8.1 Architecture Overview

Before detailing each component, our deployment stack follows a simple client–server design:

- **UI Layer:** Streamlit app exposing chat interface and controls.
- **Model Layer:** PyTorch transformer served via a lightweight API.
- **Infrastructure:** Docker containers orchestrated on a VM or cloud instance.
- **CI/CD & Monitoring:** Automated builds, tests, and logs for reliability.

8.2 Streamlit UI Deployment

We chose Streamlit for rapid prototyping and easy browser access.

- **App Structure:**
 - `app.py`: defines layout, input widgets, and callback logic.
 - `requirements.txt`: pins Streamlit and supporting libs.
- **Serving:**
 - Launch via `streamlit run app.py --server.port $PORT$`.
 - Bind to host `0.0.0.0` for external access.
- **UI Features:**
 - Chat window with scrollbar.
 - Model-parameter sliders (e.g. temperature, max tokens).
 - Feedback buttons for human evaluation.

8.3 PyTorch Model Serving

The transformer implementation is wrapped in a lightweight REST API.

- **Model Loading:**
 - Load checkpoint in `model.py` using `torch.load()`.
 - Move model to GPU if available, else CPU fallback.
- **API Layer:**
 - Use FastAPI for `/generate` endpoint.

- Accept JSON payload: prompt, generation parameters.
- Return JSON: generated text, latency metrics.
- **Error Handling:**
 - Catch CUDA out-of-memory and fallback or return meaningful error code.
 - Validate input sizes and types before inference.

8.4 Containerization

Encapsulate UI and model into Docker for consistent environments.

- **Dockerfiles:**
 - `Dockerfile.ui`: installs Python, Streamlit, copies `app.py`.
 - `Dockerfile.api`: installs PyTorch, FastAPI/Flask, copies model code and weights.
- **Docker Compose:**
 - Define two services—`ui` and `api`—on a shared network.
 - Expose ports (e.g. 8501 for UI, 8000 for API).

8.5 CI/CD Pipeline

Automate builds, tests, and deployments via GitHub Actions.

- **Build & Test:**
 - Lint Python code and run unit tests on commits.
 - Build Docker images on merge to `main`.
- **Deployment:**
 - Push images to Docker registry.
 - Trigger rolling update on target server or cloud.

8.6 Monitoring & Logging

Ensure uptime and performance tracking.

- **Logs:** capture Streamlit and API logs to stdout, aggregate with ELK/Promtail.
- **Metrics:** collect inference latency, error rates; visualize with Prometheus / Grafana.
- **Alerts:** set thresholds for high latency or OOM errors to notify maintainers.

CHAPTER 9

CONCLUSION

Our study marks a significant advancement in Natural Language Processing (NLP) for the Nepali language through the pretraining and fine-tuning of a GPT-2-based model. By leveraging the UltraChat_200k [23] dataset, which have significantly improved the model’s ability to generate contextually relevant, coherent, and natural responses. This work contributes to the growing demand for language models tailored to low-resource languages like Nepali, addressing the challenges of limited, high-quality datasets for conversational AI.

The fine-tuning process has greatly enhanced the model’s ability to perform well on dialogue-based tasks, including understanding context, managing conversational turn-taking, and generating appropriate responses. These improvements make the model a valuable tool for a wide range of applications, such as chatbots. Additionally, the success of this model underscores the potential for adapting transformer-based architectures like GPT-2 to underrepresented languages, opening new opportunities for the development of effective and scalable conversational AI in languages with fewer resources. This research represents a step forward in bridging the language technology gap and advancing the field of NLP for Nepali and similar low-resource languages.

9.1 Limitations

- Constrained to a formal, news-like style of language due to its reliance on a news-based dataset
- The model may struggle to handle regional dialects and informal speech variations in Nepali.
- Complex inflections and word forms in Nepali might not be fully captured by the model.
- There are no standard evaluation dataset for Nepali language.
- The validation dataset not has been properly translated. Since English and Nepali are not directly translatable word-for-word, partial English sentences often result in completely different Nepali translations, which can confuse the model.

9.2 Future Enhancements

- Training on a Larger and More Diverse Dataset
- Increasing Model Architecture Parameters
- Enhancing Reinforcement Learning with Human Feedback (RLHF)
- Multimodal Integration
- Optimizing Model Efficiency with LoRA and QLoRA

9.3 Challenges

- Nepali has limited annotated datasets, making it difficult to train models with accurate, diverse, and unbiased text.
- Training high-parameter models requires expensive GPUs/TPUs, which are often not readily available in low-resource settings.
- Some datasets are translated using Google Translate, which may introduce inaccuracies, loss of contextual meaning, and biases.
- Unlike high-resource languages, Nepali lacks robust NLP tools, tokenizers, embeddings, and benchmarks.
- There is insufficient data for specialized domains like medical, legal, financial, and technical fields, limiting the model's effectiveness in real-world applications.

REFERENCES

- [1] Sulav Timilsina, Milan Gautam, and Binod Bhattarai. NepBERTa: Nepali language model trained in a large corpus. In *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing*, pages 273–284, Online only, November 2022. Association for Computational Linguistics.
- [2] Shushanta Pudasaini, Subarna Shakya, Aakash Tamang, Sajjan Adhikari, Sunil Thapa, and Sagar Lamichhane. Nepalibert: Pre-training of masked language model in nepali corpus. pages 325–330, 10 2023.
- [3] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. OpenAI.
- [4] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.
- [5] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [6] Anthropic. Claude: An ai assistant by anthropic, 2023. <https://www.anthropic.com/index/claude>.
- [7] Nick Shin and Yanjia Chen. Algorithm-based chatbot using transformer and sequence to sequence method, 03 2021.
- [8] Gemini Team and et al. Gemini: A family of highly capable multimodal models, 2024.
- [9] Lifeng Shang Xin Jiang Xiao Chen Linlin Li Fang Wang¹ Xiaoqi Jiao¹, Yichun Yin² and Qun Liu. Tinybert: Distilling bert for natural language understanding. 16 Oct 2020.
- [10] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- [11] Prajwal Thapa, Jinu Nyachhyon, Mridul Sharma, and Bal Krishna Bal. Development of pre-trained transformer-based models for the nepali language, 2024.
- [12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018.
- [13] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. 2019.
- [14] Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393, 1999.

- [15] Tomas Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, page 1045–1048, 2010.
- [16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.
- [17] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. Technical Report 2018-06, OpenAI, 2018.
- [18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, page 4171–4186. Association for Computational Linguistics, 2019.
- [19] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, page 1715–1725. Association for Computational Linguistics, 2016.
- [20] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *International Conference on Learning Representations (ICLR)*, 2019.
- [21] Philip Gage. A new algorithm for data compression. *The C Users Journal*, 12(2):23–38, 1994.
- [22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [23] Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations, 2023.
- [24] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019.
- [25] Utsav Maskey. Distilgpt2-nepali, 2022. <https://huggingface.co/Sakonii/distilgpt2-nepali>.

APPENDIX

Datasets

Dataset Viewer			Auto-converted to Parquet	API	Embed	Data Studio
Split (2) train · 5.2M rows						
Search this dataset						
index int64	Article string · lengths	Source string · classes				
1	2	99 values				
5,176,754	बिहीबार दिउँसो खोला तने क्रममा एक जना बगेर बेपत्ता भएको प्रहरीले जनाएको छ। शिवतासक्षी नगरपालिका-१० सुकुम्वासी बस्तीका ४३ वर्षीय साहेब सरदार माइ खोला तने क्रममा दिउँसो १ बजेदेखि...	nepalkhabar.com				
6,241,372	नेकपा एमालेका अध्यक्ष केपी शर्मा ओलीले मंसिर ४ गते सम्पन्न प्रतिनिधिसभा र प्रदेशसभा चुनावको मतपरिणामले मुलुक अस्थिरतातिर धकेलिने संकेत देखिएको बताएका छन् । मंसिर ४ गते सम्पन्न...	shilapatra.com				
4,574,058	अदालतको आदेशपछि हिरासतमुक्त भएका सांसद शर्मा आइतबार बिहान एमाले अध्यक्ष केपी शर्मा ओलीलाई भेट्न बालकोट पुगेका हुन् । ओलीसँग भेटेपछि बाहिरिने क्रममा प्रधानमन्त्री पुष्पकमल दाहाल...	hamrobiratnagar.com				
3,325,703	नेकपा एकीकृत समाजवादीले बैतडीमा सङ्गठन विस्तार अभियानलाई तीव्रता दिएको छ ।	dristinews.com				
5,627,533	निगमले बुधबार सूचना निकालेर पेट्रोलियम पदार्थको आपूर्ति व्यवस्थामा प्रतिकूल प्रभाव पार्ने पिडिओ नलिने, भारतीय आयल कर्पोरेशन (आइओसी)मा इन्धन लिन नजाने, डिपोमा इन्धन लिएर नआउने र...	ujyaaloonline.com				
5,582,018	नुवाकोट जिल्लाका १२ स्थानीय तहको एक विदुर नगरपालिकाले शिक्षा र स्वास्थ्य र पर्यटन प्रवर्द्धन पहिलो प्राथमिकतामा राखेको छ ।नगरप्रमुखमा निर्वाचित भएपछि राजन श्रेष्ठले वैदेशिक रोजगारीबाट...	gorkhapatraonline.com				
< Previous 1 2 3 ... 52,000 Next >						

Figure A.1: Pre-Training Dataset

```
{
  "prompt": "संसारका शीर्ष 10 सबैभन्दा पर्यावरण-मैत्री शहरहरू र तिनीहरूको नवीकरणीय ऊर्जा पहलहरूको बारेमा कम्तिमा 1000 शब्दहरूको एक व्यापक ब्लग पोस्ट",
  "prompt_id": "6db663a4d2671b41e0038c43c39f79cf909b10987dc5954286228079c2c78207",
  "messages": [
    {
      "content": "संसारका शीर्ष 10 सबैभन्दा पर्यावरण-मैत्री शहरहरू र तिनीहरूको नवीकरणीय ऊर्जा पहलहरूको बारेमा कम्तिमा 1000 शब्दहरूको एक व्यापक ब्लग",
      "role": "user"
    },
    {
      "content": "हालैका वर्षहरूमा, जलवायु परिवर्तनको हानिकारक प्रभावहरू विरुद्ध लड्न दिगो शहरहरूको विकासको आवश्यकताको बारेमा बढ्दो चेतना बढेको छ।",
      "role": "assistant"
    },
    {
      "content": "यो शीर्ष पर्यावरण-मैत्री शहरहरू र तिनीहरूको नवीकरणीय ऊर्जा पहलहरूको बारेमा साँच्चै जानकारीमूलक पोस्ट हो। म तिनीहरूको फोहोर व्यवस्थापन",
      "role": "user"
    },
    {
      "content": "बिल्कुल! फोहोर व्यवस्थापन दिगोपनको महत्वपूर्ण पक्ष हो किनभने यसले शहरहरूलाई तिनीहरूको कार्बन फुटप्रिन्ट कम गर्न र स्रोतहरू बर्बाद हुन न",
      "role": "assistant"
    },
    {
      "content": "यी पर्यावरण-मैत्री शहरहरूमा फोहोर व्यवस्थापन रणनीतिहरूको बारेमा थप जानकारीको लागि धन्यवाद। ऊर्जा खपत र कार्बन उत्सर्जन कम गर्न दिगो",
      "role": "user"
    },
    {
      "content": "पक्कै पनि! हरियो भवनहरू दिगोपनको महत्वपूर्ण पक्ष हुन्, किनकि तिनीहरूले ऊर्जा खपतलाई उल्लेखनीय रूपमा घटाउन र कार्बन उत्सर्जनलाई क",
      "role": "assistant"
    }
  ],
}
```

Post Training Dataset

Interaction

1. User inputs their query.

2. Chatbot will generate response as per the query of the user.

UI

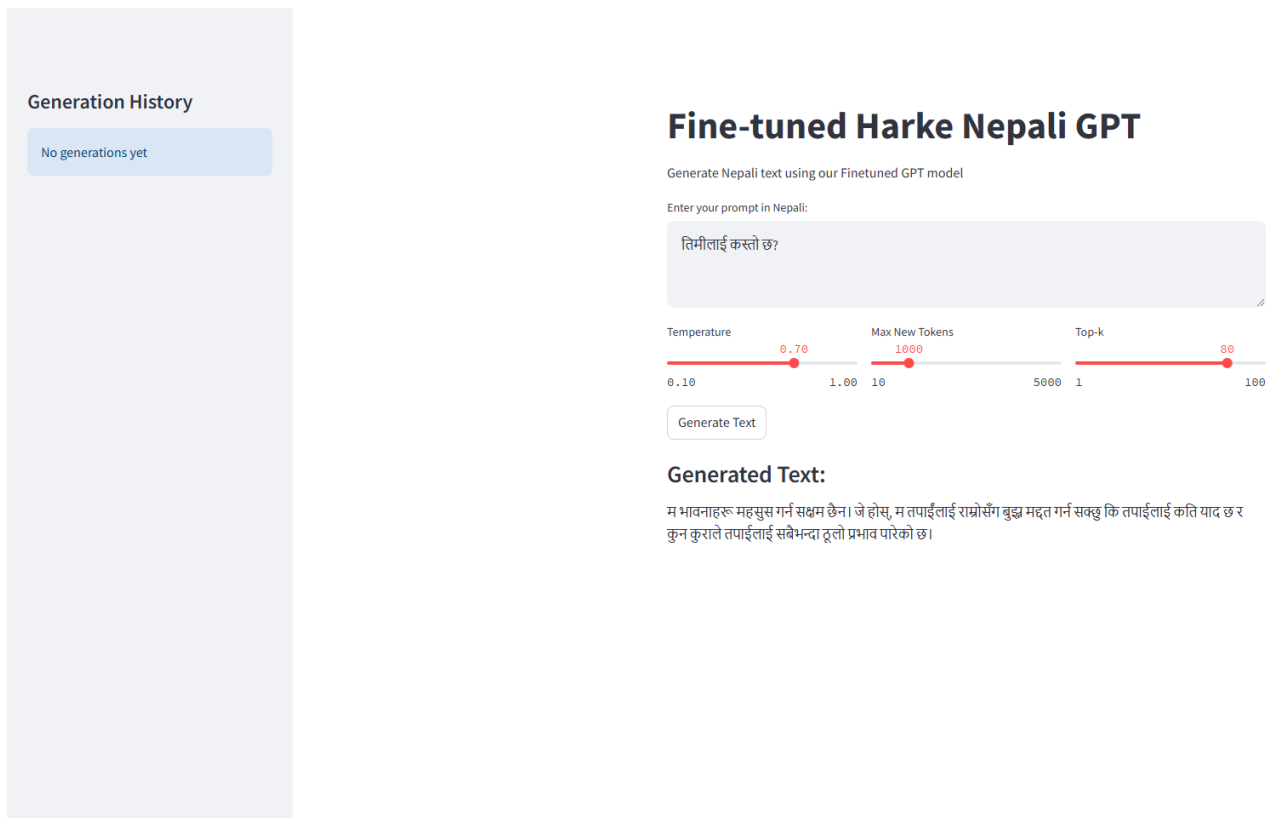






Figure A.2: UI




Plagiarism Report



Match Groups

-  **38 Not Cited or Quoted 7%**
Matches with neither in-text citation nor quotation marks
-  **0 Missing Quotations 0%**
Matches that are still very similar to source material
-  **0 Missing Citation 0%**
Matches that have quotation marks, but no in-text citation
-  **0 Cited and Quoted 0%**
Matches with in-text citation present, but no quotation marks

Top Sources

-  **6%** Internet sources
-  **2%** Publications
-  **0%** Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Internet	
	www.coursehero.com	2%
2	Internet	
	arxiv.org	<1%
3	Internet	
	aclanthology.org	<1%
4	Internet	
	ireasoning.sourceforge.net	<1%
5	Internet	
	www.slideshare.net	<1%
6	Publication	
	Hafiz Adnan Ashraf, Jiajun Li, Zeyu Li, Azam Sohail, Raza Ahmed, Muhammad Ha...	<1%
7	Internet	
	core.ac.uk	<1%
8	Internet	
	elibrary.tucl.edu.np	<1%
9	Internet	
	vuml.sourceforge.net	<1%
10	Publication	
	Adrian David Cheok, Emma Yann Zhang. "From Turing to Transformers: A Compr...	<1%

11	Internet	alumni-portal.sasin.edu	<1%
12	Internet	www.lavoisier.fr	<1%
13	Internet	repositorium.uminho.pt	<1%
14	Internet	bitweb.ict.op.ac.nz	<1%
15	Internet	www.ecva.net	<1%
16	Internet	medium.com	<1%
17	Internet	repositori.udl.cat	<1%
18	Publication	Mireshghallah, Fatemehsadat. "Auditing and Mitigating Safety Risks in Large Lan...	<1%
19	Internet	Inu.diva-portal.org	<1%
20	Publication	Deb, Dipok. "Application and Analysis of Machine Learning and Deep Learning AI...	<1%
21	Publication	Zheng Yang, Bing Han, Xinbo Gao, Zhi-Hui Zhan. "Eye-movement-prompted large ...	<1%
22	Publication	Chen, Mike. "Changing the Narrative Perspective: A New Language Processing Ta...	<1%
23	Publication	Xiaochun Wu, Ning Guo. "MGSLU-Net: a lightweight network for efficient detectio...	<1%
24	Internet	opus4.kobv.de	<1%