मौलाना आजाद राष्ट्रीय प्रौद्योगिकी संस्थान - भोपाल
Maulana Azad National Institute of Technology– Bhopal

TRANSFORMER-BASED TEXT GENERATION WITH MULTIPLE TOKENIZATIONS

Major Project Report

VIII Semester

Under the Guidance of
Dr. SANYAM SHUKLA

SUBMITTED BY

VIVSWAAN SINGH **211112060**

कंप्यूटर विज्ञान एवं अभियांत्रिकी विभाग

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Session: 2023-2024

DECLARATION

I, hereby declare that the following report which is being presented in the Major Project entitled as "**TRANSFORMER-BASED TEXT GENERATION WITH MULTIPLE TOKENIZATIONS**" is an authentic documentation of my own original work to best of my knowledge. The following project and its report, in part or whole, has not been presented or submitted by me for any purpose in any other institute or organization. Any contribution made to the research by others, with whom I have worked **at Maulana Azad National Institute of Technology, Bhopal** or elsewhere, is explicitly acknowledged in the report.

VIVSWAAN SINGH **211112060**

मौलाना आजाद राष्ट्रीय प्रौद्योगिकी संस्थान - भोपाल

Maulana Azad National Institute of Technology– Bhopal

कंप्यूटर विज्ञान एवं अभियांत्रिकी विभाग

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the project entitled **"TRANSFORMER-BASED TEXT GENERATION WITH MULTIPLE TOKENIZATIONS"**

SUBMITTED BY:

VIVSWAAN SINGH **211112060**

is the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering is an authentic work carried out by him under my supervision and guidance.

Dr. Sanyam Shukla
(Major Project Supervisor)

मौलाना आजाद राष्ट्रीय प्रौद्योगिकी संस्थान - भोपाल

Maulana Azad National Institute of Technology– Bhopal

कंप्यूटर विज्ञान एवं अभियांत्रिकी विभाग

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DECLARATION

I hereby declare that the work, which is presented in this Project Report, entitled **"TRANSFORMER-BASED TEXT GENERATION WITH MULTIPLE TOKENIZATIONS"**, in partial fulfilment of the requirements for the award of the degree, submitted in the **Department of Computer Science and Engineering, Maulana Azad National Institute of Technology, Bhopal**. It is an authentic record of my work carried out under the noble guidance of my guide **"Dr. SANYAM SHUKLA"**. The following project and its report, in part or whole, have not been presented or submitted by me for any purpose in any other institute or organization. I hereby declare that the facts mentioned above are true to the best of my knowledge. In case of any unlikely discrepancy that may possibly occur, I will be the ones to take responsibility.

SCHOLAR NAME **Vivswaan Singh**

SCHOLAR NO **211112060**

ACKNOWLEDGEMENT

With due respect, I express my deep sense of gratitude to my respected guide **Dr. Sanyam Shukla**, for his valuable help and guidance. I am thankful for the encouragement that he has given me in completing this project successfully.

I am also grateful to my respected **Director, Prof K. K. Shukla** for permitting me to utilize all the necessary college facilities.

I am also grateful to my respected **HOD, Prof. Deepak Singh Tomar** for permitting me to utilize all the necessary facilities of the college.

I am also thankful to all the other faculty, staff members and laboratory attendants of my department for their kind cooperation and help. Last but certainly not the least; I would like to express my deep appreciation towards my family members and batch mates for providing support and encouragement.

सारांश
ABSTRACT

यह परियोजना एक सरलीकृत GPT-शैली के ट्रांसफॉर्मर भाषा मॉडल का प्रारंभ से कार्यान्वयन प्रस्तुत करती है, जो ऑटोरिग्रेसिव टेक्स्ट जनरेशन के लिए डिज़ाइन किया गया है। इसमें तीन टोकनाइज़ेशन रणनीतियों—कैरेक्टर-लेवल, बाइट-पेयर एनकोडिंग (BPE), और वर्डपीस—का तुलनात्मक विश्लेषण शामिल है। यह अध्ययन करता है कि विभिन्न टोकन ग्रैन्युलरिटी प्रशिक्षण व्यवहार, लॉस और परप्लेक्सिटी जैसे मूल्यांकन मेट्रिक्स, और आउटपुट की स्पष्टता को कैसे प्रभावित करती है। मॉडल को PyTorch में बिना किसी पूर्व-प्रशिक्षित लाइब्रेरी के लागू किया गया था, और सभी टोकनाइज़ेशन विधियाँ एक ही डेटासेट (*Frankenstein* by Mary Shelley) पर लागू की गईं। प्रशिक्षण Google Colab पर GPU समर्थन के साथ किया गया। परिणाम दर्शाते हैं कि कैरेक्टर-लेवल मॉडल तेज़ी से प्रशिक्षित होते हैं, जबकि BPE और वर्डपीस अधिक संरचित और अर्थपूर्ण आउटपुट उत्पन्न करते हैं। यह अध्ययन टोकनाइज़ेशन की भाषा मॉडल के प्रदर्शन में महत्वपूर्ण भूमिका को रेखांकित करता है और कुशल NLP सिस्टम डिज़ाइन के लिए व्यावहारिक मार्गदर्शन प्रदान करता है।

This project presents a from-scratch implementation of a simplified GPT-style Transformer language model for autoregressive text generation, along with a comparative analysis of three tokenization strategies: Character-Level, Byte-Pair Encoding (BPE), and WordPiece. It examines how different token granularities impact training behavior, evaluation metrics like loss and perplexity, and output coherence. Implemented in PyTorch without pre-trained models, each tokenizer was applied to the same dataset (*Frankenstein* by Mary Shelley) for consistent comparison. Training used GPU acceleration on Google Colab. Results show that while character-level models train faster, BPE and WordPiece produce more structured and meaningful outputs. The findings highlight tokenization's significant role in shaping the performance of generative language models and offer practical guidance for efficient NLP system design.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

BPE	Byte-Pair Encoding
BERT	Bidirectional Encoder Representations from TransformersVector Classification

CHAPTER 1
INTRODUCTION

This chapter introduces the project and outlines its significance, background, relevance, and scope.

The focus of this project is on transformer-based text generation using multiple tokenization strategies, specifically character-level and subword-level tokenization. Transformers have revolutionized natural language processing by enabling models to learn long-range dependencies effectively through self-attention mechanisms, and tokenization plays a key role in how these models interpret and process input text. Instead of relying on pre-trained models or abstracted libraries, this project involves implementing a simplified transformer architecture from scratch to gain a deeper understanding of its inner workings, including attention, positional encoding, and layer-wise processing. By experimenting with different tokenization methods at a low level, I aim to observe and analyze their influence on model behavior, training dynamics, and the overall quality and coherence of generated text. This hands-on approach not only allows for controlled comparisons but also enhances my comprehension of how foundational components like tokenization shape the outputs of generative models.

1.1. Background

Transformer models, such as GPT and BERT, have revolutionized natural language processing with their ability to model long-range dependencies through self-attention mechanisms. A critical aspect of these models is tokenization—the process of converting raw text into tokens that the model can understand. Among the commonly used tokenization methods, Byte-Pair Encoding (BPE) and WordPiece are popular for their ability to efficiently balance vocabulary size and generalization. In contrast, character-level tokenization takes a more granular approach, offering better handling of rare or unseen words. This project explores how these three tokenization strategies—character-level, BPE, and WordPiece—impact the quality of text generation. By implementing a simplified transformer model from scratch, the project provides an in-depth analysis of how each tokenization method influences key components such as attention mechanisms, positional encoding, and layer-wise processing

1.2. Importance

While much of the focus in NLP research is on model architecture, tokenization plays a vital and often underestimated role in determining model performance. The way text is tokenized can significantly influence how well a model learns meaningful patterns, generates coherent text, and generalizes across different domains. This is especially important for generative tasks, where fluency, structure, and consistency of the output depend heavily on the granularity of tokens. By comparing character-level tokenization, BPE, and WordPiece in the same transformer model—implemented from scratch—this project sheds light on how different tokenization strategies affect everything from training stability to output diversity. These insights are crucial not just from a research standpoint but also in real-world applications, where selecting the right tokenization method can have a measurable impact on system performance and efficiency.

1.3. Motivation

Recent advancements in transformer architectures have redefined the landscape of natural language generation. However, much of the attention remains focused on model design, often overlooking the foundational role of tokenization. Tokenization not only dictates how text is fed into a model but also impacts the structure, coherence, and semantics of the generated output. This project is driven by the desire to understand how different tokenization strategies—specifically character-level and subword-level methods like Byte-Pair Encoding and WordPiece—affect the generative performance of transformers. By building a simplified transformer model from the ground up and integrating multiple tokenization techniques, I aim to uncover the nuances that influence training behavior and output quality. This investigation is motivated by a need to bridge the gap between model design and preprocessing strategies, ultimately leading to more interpretable and controllable generation pipelines, especially in scenarios where output quality and efficiency are critical.

CHAPTER 2
PROBLEM STATEMENT

Current approaches to text generation with transformer models predominantly rely on pre-trained architectures, which are heavily influenced by the choice of tokenization methods. While tokenization is a critical preprocessing step, its impact on model behavior and output generation has not been fully explored. Common tokenization strategies such as Byte-Pair Encoding (BPE) and WordPiece are widely used, but they may not always be optimal for every use case, especially when dealing with rare words, out-of-vocabulary terms, or language-specific nuances. Character-level tokenization, although offering finer granularity, presents challenges in terms of vocabulary size and computational efficiency. Existing models tend to overlook the interplay between different tokenization methods and how they influence the underlying transformer architecture.

This project seeks to address these limitations by implementing a simplified transformer model from scratch and experimenting with multiple tokenization strategies, including character-level, Byte-Pair Encoding (BPE), and WordPiece. The goal is to investigate how each tokenization method affects key aspects of the transformer model, such as attention mechanisms, positional encoding, and text generation quality. By comparing these tokenization techniques in a controlled, hands-on environment, this study aims to provide a deeper understanding of the role tokenization plays in shaping model performance, ultimately leading to more efficient and interpretable transformer-based generative models.

CHAPTER 3
OBJECTIVES

The primary objective of this project is to design and implement a simplified GPT-style Transformer language model from scratch, in order to better understand the inner workings of transformer-based text generation. This also includes a comparative study of three widely used tokenization strategies—Character-level, Byte-Pair Encoding (BPE), and WordPiece—and their impact on model behavior, training dynamics, and text output quality. The specific objectives are:

- Designing and implementing a simplified GPT-style transformer model from the ground up using PyTorch, without relying on pre-trained libraries.

- To implement a transformer decoder architecture from scratch using PyTorch, including core components such as multi-head self-attention, positional encoding, layer normalization, and feedforward networks.

- To integrate and apply three tokenization methods—Character-level, Byte-Pair Encoding (BPE), and WordPiece—using the HuggingFace Tokenizers library for subword strategies.

- To train the transformer model on a real text corpus and evaluate model performance under each tokenization strategy using metrics such as training loss and perplexity.

- To analyze and compare the output quality, training stability, and efficiency of each tokenizer in the context of generative tasks.

- To gain a deeper practical understanding of the relationship between tokenization, input representation, and generative model output in transformer-based architectures.

CHAPTER 4
LITERATURE SURVEY

2.1. Significance

Understanding the inner workings of transformer-based language models is essential for building efficient and interpretable generative systems. This section explores how implementing a GPT-style transformer from scratch, along with evaluating various tokenization strategies, can provide deeper insights into model behavior. Reviewing existing work on both architectural design and tokenization impact helps highlight key challenges and advancements in developing effective text generation models.

2.2. Reviewed Research Papers

In this section, we discuss key research papers related to the implementation of GPT-style transformer models and of different tokenization methods on text generation.

Table 2.1 Referenced Research Papers

S.no	Authors	Published In	Abstract
1.	A. Vaswani et al.	2017, Advances in Neural Information Processing Systems, vol. 30	Introduces the Transformer model, leveraging self-attention mechanisms to improve NLP tasks.
2.	Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever	2018, OpenAI	Discusses the benefits of generative pre-training for improved language understanding.
3.	T. Brown et al.	2020, arXiv preprint arXiv:2005.14165	Describes how large language models (LLMs) perform well in few-shot learning tasks without fine-tuning.
4.	R. Sennrich, B. Haddow, and A. Birch	2016, ACL	Proposes neural machine translation using subword units to handle rare words effectively.
5.	M. Schuster and K. Nakajima	2012, ICASSP	Focuses on Japanese and Korean voice search using recurrent neural networks

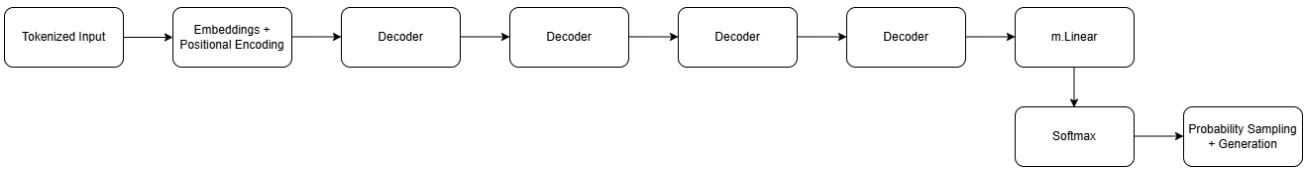
			for improved accuracy. Introduced the WordPiece tokenizer, later used in BERT
6.	J. Devlin et al.	2019, NAACL	Introduces BERT, a transformer-based model pre-trained for deep bidirectional understanding of language.
7.	L. Liu et al.	2020, arXiv preprint arXiv:2004.08249	Investigates challenges in training Transformers, focusing on optimization techniques for stability.
8.	T. Dettmers et al.	2023, arXiv preprint arXiv:2305.14314	Presents QLoRA, an efficient fine-tuning method for quantized large language models, reducing resource needs for training.

CHAPTER 5
PROPOSED METHODOLOGY

This chapter outlines the complete methodology followed to design, implement, and evaluate a simplified GPT-style Transformer language model, with a specific focus on comparing the effects of different tokenization strategies—namely character-level, Byte-Pair Encoding (BPE), and WordPiece—on model performance. The methodology comprises three key components: the model architecture, the tokenization schemes, and the training and evaluation framework. The entire implementation has been done in Python using PyTorch for model building and training, and HuggingFace's tokenizers library for subword-based tokenization. The project was executed on Google Colab to leverage GPU acceleration (via CUDA) for efficient model training.

5.1. Model Architecture

Figure 5.1 GPT Model



The core of the project is a Transformer-based language model inspired by the decoder portion of GPT architectures. Transformers are neural network architectures that use self-attention mechanisms to model dependencies between tokens in a sequence, without relying on recurrence or convolutions. The model was implemented from scratch using PyTorch, without relying on pre-trained models or external libraries for attention or block logic. The architecture consists of the following components:

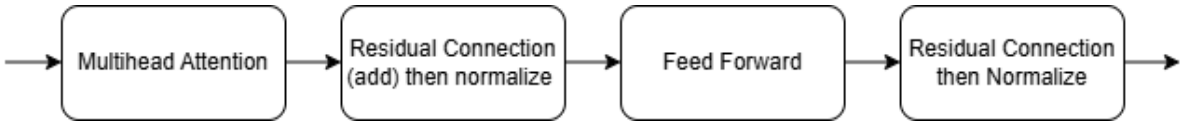
- **Token Embedding Layer**: Each token (whether character or subword) is mapped to a learnable dense vector of fixed size (n_embd). These embeddings serve as the input to the model.

- **Positional Embedding Layer**: Transformers do not inherently understand token order. To inject positional information, a learnable positional embedding is added to each token embedding.

$$Input, x = TokenEmbedding[token] + PositionEmbedding[position]$$

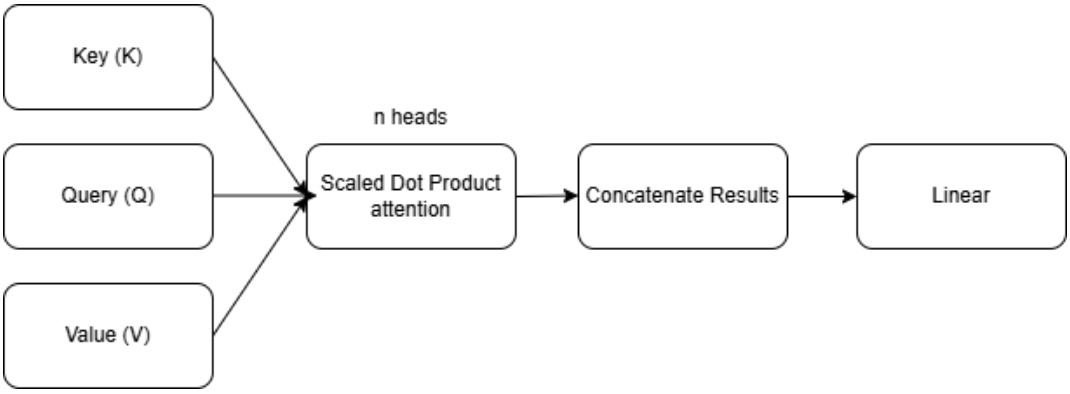
- **Transformer Decoder Blocks**: It is a stack of n_layer blocks, each containing:

Figure 5.2 Decoder



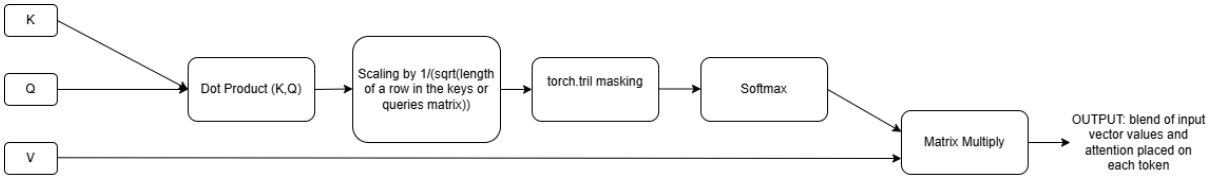
  - **Multi-Head Self-Attention:** Instead of computing a single set of attention weights, the model splits the embedding into multiple "heads," each learning different types of relationships in the data. It consists of n_head attention heads running in parallel whose outputs are concatenated and passed through a linear layer.

Figure 5.3 Multi Head Attention



    - The core operation in a transformer is scaled dot-product self-attention, which allows each token to attend to every previous token in the sequence. For each **input** token embedding x: Query (Q), Key (K), and Value (V) vectors are computed via linear transformations:
      - **Key:** The key vector represents the content of a token that others may attend to. It is what all queries compare themselves against.
      - **Query:** The query vector represents the current token (position) that is trying to gather relevant information from the other tokens in the sequence.
      - **Value:** The value vector holds the actual information to be passed to the next layer if the query decides a key is relevant.

Figure 5.4 Attention Head



- In this autoregressive model, the transformer should not "look ahead" at future tokens during training. Attention scores are masked using a lower-triangular matrix (tril) to prevent the model from accessing future tokens. This ensures the model only attends to the current and previous tokens.
- The raw attention scores are scaled and passed through a softmax function to obtain a probability distribution over which tokens to attend to. We scale down by a factor of square root of size of key matrix to prevent large dot products from dominating softmax. Each row in the resulting matrix is a softmax over the attention scores for a given query token.

$$AttentionWeights = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

- The attention output is the weighted sum of the **value vectors** using the computed attention weights. This produces a new set of representations for each token, informed by all relevant context to the left (in causal/self-attention).

$$AttentionOutput = AttentionWeight.V$$

- **Feedforward Neural Network (FFN):**
  - It is a two-layer MLP with ReLU activation and dropout for regularization. ReLU introduces non-linearity and helps the model learn complex transformations. The hidden layer size taken here is 4× the embedding size.

Figure 5.5 Feed Forward Layer

- **Residual Connections:**
  - The original input is added back to the output of the sub-layer.

- **Layer Normalization:**
  - Applied after residual connections for training stability.

Each major sub-layer (attention, feedforward) is wrapped with Residual Connections and Layer Normalization to improve gradient flow and training stability.

$$x = LayerNorm(x + SelfAttention(x))$$
$$x = LayerNorm(x + FeedForward(x))$$

- **Dropout:**
  - Dropout is applied after the attention and feedforward layers to prevent overfitting, especially important given the relatively small dataset. It randomly disables a fraction of the neurons during training.

- **Final Linear Layer**: Maps the output of the decoder stack to the vocabulary size, producing logits over the possible next tokens.
- **Loss Function**: The model uses cross-entropy loss for training, calculated between the predicted logits and the true next tokens.

5.2. Tokenization Techniques

To study the effect of tokenization on model training and generation quality, three distinct tokenization strategies were applied:

a) Character-Level Tokenization

- Each character from the input text is treated as an individual token.
- A vocabulary is built by extracting the set of unique characters from the dataset.
- Tokenization and detokenization are performed using custom mappings (str_to_int and int_to_str dictionaries).

b) Byte-Pair Encoding (BPE)

- Implemented using HuggingFace's Tokenizer with a BPE model backend.
- Trained on the dataset using BpeTrainer, configured with a vocabulary size of 5000.
- Common character pairs are iteratively merged based on frequency to create a compact and expressive vocabulary.

c) **WordPiece Tokenization**

- Implemented using HuggingFace's Tokenizer with a WordPiece model backend.
- Trained using WordPieceTrainer, also with a vocabulary size of 5000.
- Merges are selected based on maximizing the likelihood of the training data under a language model, rather than simple frequency.

Each tokenization method was applied to the same dataset (frankenstein.txt) and used to generate separate training and validation data tensors.

5.3. Dataset, Preprocessing and Training

The raw text data was sourced from Mary Shelley's Frankenstein, processed as plain text. After tokenization, the resulting token sequences were split into training (75%) and validation (25%) sets. Input-target pairs were prepared using sliding windows of fixed sequence length (block_size = 126), where the input is a chunk of tokens and the target is the next token sequence. The raw text model was trained using **PyTorch's AdamW optimizer**, with a learning rate of 3e-5 and weight decay of 0.1 for regularization.

- **Training loop**:
  - Mini-batch size: 256
  - Number of iterations: 1000
  - Training and validation loss were logged every 100 iterations
  - Intermediate logs were printed every 50 iterations for monitoring
- **Evaluation Metrics**:
  - **Cross-Entropy Loss**: Used during training
  - **Perplexity**: Computed from validation loss to measure model's confidence in its predictions
- **Early Generation Tests**:
  - After initial training, the model was prompted with a zero-token or seed word (e.g., "monster") to observe generation quality.

5.4. Text Generation

The trained model is capable of autoregressive text generation. At each step, the model predicts the next token based on the current sequence. The predicted token is appended to the

input, and the process is repeated. A softmax distribution is used to sample the next token. A maximum token limit is set to avoid infinite generation and control output length.
Each tokenizer's model was used to generate multiple sequences for qualitative comparison.

5.5. Tools and Frameworks used

The following tools, libraries, and platforms were used throughout the implementation, experimentation, training, and analysis phases of the project:

- **Python**: The primary programming language used for the entire implementation, including model architecture, tokenization, training, and evaluation.
- **PyTorch**: Used to build the transformer model from scratch. It facilitated tensor operations, automatic differentiation, neural network layers, training loop management, and loss computation.
- **HuggingFace tokenizers Library**: Utilized for implementing and training Byte-Pair Encoding (BPE) and WordPiece tokenizers. Provided access to low-level tokenization control, custom vocabulary training, and integration with Python code.
- **Matplotlib**: Used for visualizing training and validation loss curves, helping in tracking model convergence and comparing tokenizer performance graphically.
- **Google Colab**: Served as the cloud-based development and training environment. Provided access to free GPU acceleration, which significantly reduced model training time and enabled larger batch sizes.
- **CUDA (via Colab GPU runtime)**: Enabled GPU-based matrix operations and neural network training, greatly accelerating the training process compared to CPU-only execution.
- **Torchvision (optional dependency)**: Although not directly used in model logic, it was preinstalled in the Colab environment and available for visualization or dataset utility extensions.
- **NumPy**: Used implicitly through PyTorch for tensor initialization, reshaping, and numerical operations such as random sampling or vector transformation.
- **math:** For numerical calculations (e.g., computing perplexity via exponentiation).
- **os:** For file and path handling.
- **Text Data**: The dataset used was a plain text version of *Frankenstein* by Mary Shelley, processed manually for use in character and subword tokenization experiments.

These tools were carefully selected to balance flexibility, control, and computational efficiency, and they collectively enabled end-to-end execution of the project from architecture design to training and evaluation.

CHAPTER 6
PROJECT PLAN

This chapter outlines the structured approach taken to complete the project, divided into multiple phases corresponding to key milestones in the development of a transformer-based language model and the comparative analysis of different tokenization strategies. Each phase was planned and executed in a sequential and iterative manner, ensuring a balance between theoretical understanding and hands-on experimentation. The timeline reflects the actual progression of the project, executed over several weeks using an iterative and research-driven workflow.

Phase 1: Research and Literature Review

- **Objective**: To build a foundational understanding of transformer architectures, language modeling, and tokenization strategies.
- **Activities**:
    - Studied transformer architectures with a focus on decoder-only models (GPT-style).
    - Explored core components such as self-attention, feedforward networks, positional embeddings, and LayerNorm.
    - Reviewed literature on subword tokenization techniques including Byte-Pair Encoding (BPE) and WordPiece.
    - Analyzed relevant papers such as *Attention Is All You Need*, *GPT*, *BERT*, and tokenization-related works.
- **Outcome**: Gained clarity on model structure and established the significance of tokenization in generative models.

Phase 2: Design and Planning

- **Objective**: To translate theoretical insights into a concrete project structure and define the scope of implementation.
- **Activities**:
    - Decided on implementing a simplified GPT-style transformer architecture from scratch.

- Planned the inclusion of three tokenization methods: character-level, BPE, and WordPiece.
- Selected the dataset (*Frankenstein* by Mary Shelley) for experimentation.
- Chose PyTorch for model development, HuggingFace's tokenizers for subword tokenization, and Google Colab for training with GPU support.

- **Outcome**: A clear project architecture and experimental setup plan was established.

Phase 3: Implementation

- **Objective**: To build all core components of the transformer model and integrate tokenization pipelines.
- **Activities**:
  - Implemented token and positional embedding layers.
  - Developed the self-attention mechanism with attention masking using torch.tril.
  - Constructed multi-head attention and feedforward networks.
  - Added LayerNorm, residual connections, and dropout for regularization.
  - Integrated three tokenization strategies: Character-level (custom mapping), BPE and WordPiece (using HuggingFace training APIs)
  - Prepared training and validation datasets for each tokenizer type.
- **Outcome**: A complete, functional GPT-style model with flexible tokenization support was built from the ground up.

Phase 4: Training and Testing

- **Objective**: To train the model on the selected dataset under different tokenization strategies and monitor performance.
- **Activities**:
  - Trained models on character, BPE, and WordPiece tokenized data using the same architecture and hyperparameters.
  - Used AdamW optimizer with a learning rate of 3e-5 and block_size = 126.
  - Computed and logged training and validation cross-entropy loss and perplexity every 100 iterations.
  - Implemented a text generation function to sample outputs from trained models.
  - Visualized loss curves and evaluated convergence and generation quality.

- **Outcome**: Observed decreasing loss and perplexity values indicating successful learning and identified differences in learning dynamics and output fluency across tokenization methods.

Phase 5: Evaluation and Comparative Analysis

- **Objective**: To compare the performance of the three tokenization strategies in terms of model learning and generation quality.
- **Activities**:
    - Analyzed loss and perplexity trends across character-level, BPE, and WordPiece models.
    - Evaluated the coherence, diversity, and structure of generated text samples.
    - Compared the training efficiency and semantic consistency of each approach.
- **Outcome**: Character-level tokenization produced simpler and repetitive output but trained faster. BPE and WordPiece offered more coherent and structured generation, with slower but more stable training curves.

Phase 6: Finalization and Documentation

- **Objective**: To compile the results, prepare documentation, and present the work in a well-organized report.
- **Activities**:
    - Documented methodology, experiments, and results into structured report chapters.
    - Created comparative summaries and formatted evaluation results for the final report.
    - Drafted sections on problem statement, objectives, literature review, and methodology.
    - Prepared visualizations and sample outputs for inclusion in the final document.
- **Outcome**: Completion of a comprehensive and well-structured project report, demonstrating technical depth, originality, and academic rigor.

The entire code is uploaded at https://github.com/Vivaswaan/gpt_project/tree/main

CHAPTER 7
EXPERIMENTAL RESULT AND ANALYSIS

This chapter presents the experimental results obtained by training the simplified GPT-style transformer model using three different tokenization strategies—Character-Level, Byte-Pair Encoding (BPE), and WordPiece—and evaluates the impact of each method on model performance, training dynamics, and text generation quality. The analysis is both **quantitative**, using loss and perplexity metrics, and **qualitative**, using sample outputs to assess fluency, coherence, and structure. The chapter also outlines the expected practical outcomes and relevance of the findings in broader NLP contexts.

7.1. Experimental Setup

- **Dataset**: The complete text of Frankenstein by Mary Shelley.
- **Training-Validation Split:** 75% training, 25% validation.
- **Model Hyperparameters**:
    - Embedding Size, n_embd = 256
    - Number of layers, n_layer = 4
    - Number of heads, n_head = 4
    - Dropout = 0.4
- Training Iterations = 1000
- Batch size = 256
- **Optimizer:** AdamW (lr = 3e-5, weight decay = 0.1)
- **Training Platform:** Google Colab with CUDA GPU

Each tokenizer model (Character, BPE, WordPiece) was trained separately using the same model architecture and hyperparameters to ensure a fair comparison.

7.2. Evaluation Metrics

- **Cross Entropy Loss**: Used as the primary training objective. Lower values indicate better alignment between predicted and actual next-token distributions.

$$Loss, L = -\sum_{i=1}^{N} y_i \log \hat{y}_i$$

- **Perplexity:** Calculated as the exponential of the average cross-entropy loss. Represents the model's uncertainty in predicting the next token. Lower perplexity indicates better model confidence and language understanding.

$$Perplexity, ppl = e^{(Loss)}$$

7.3. Results

The tables below summarize the losses and perplexities observed before and after 1000 training iterations for each tokenization method:

Table 7.1 Change in Loss

Tokenization Method	Initial Train Loss	Final Train Loss	Initial Test Loss	Final Test Loss
Character-Level	4.484	2.426	4.481	2.417
Byte-Pair Encoding	8.575	5.032	8.577	5.600
WordPiece	8.568	5.000	8.568	5.545

Table 7.2 Change in Perplexity

Tokenization Method	Initial Train Perplexity	Final Train Perplexity	Initial Test Perplexity	Final Test Perplexity
Character-Level	88.60	11.31	88.32	11.21
Byte-Pair Encoding	5295.35	153.22	5310.76	270.54
WordPiece	5260.34	148.41	5259.53	256.06

Figure 7.1 Change in Loss for Character Level Tokenization

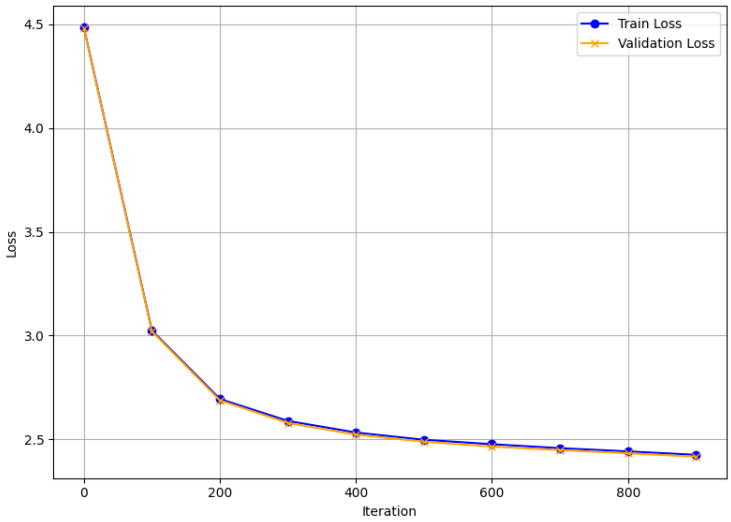


Figure 7.2 Change in Loss for BPE Tokenization

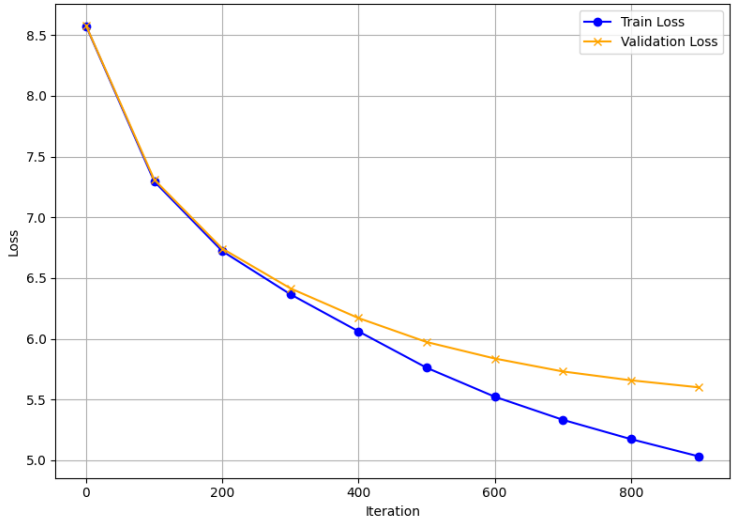
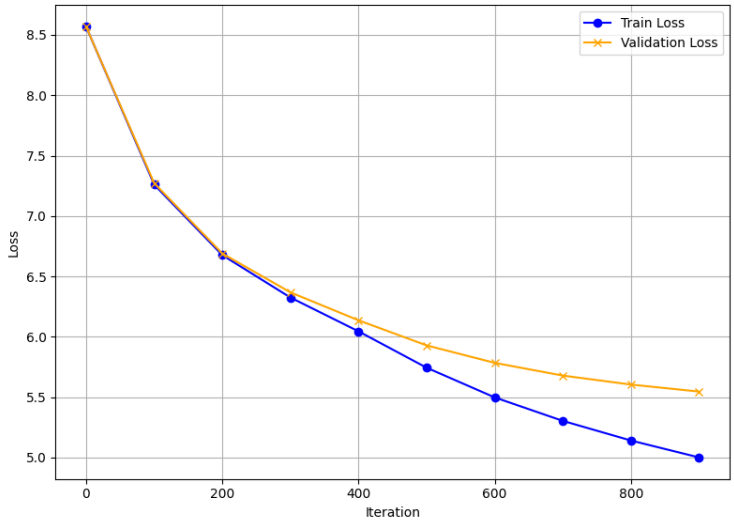


Figure 7.3 Change in Loss for WordPiece Tokenization



7.4. Observations

The Character-Level model trained faster and achieved lower loss and perplexity due to its smaller vocabulary and more frequent token reuse. BPE and WordPiece models took longer to converge but generated more semantically meaningful and coherent text. Similarly, Perplexity was significantly higher for subword tokenizations due to their larger vocabularies and sparse distributions, but this did not directly translate into worse generation quality.

In Character-Level, limited vocabulary led to repetitive or random token choices. Outputs obtained from BPE were grammatically structured, with recognizable English word fragments which shows model learned syntactic patterns better than character-level. Outputs

obtained from WordPiece were fragmented with subword pieces (For eg.: ##ness, ##inc), which is expected with WordPiece. and shows partial understanding of word formation and sentence structure.

Table 7.3 Pros and Cons of Different Tokenization Techniques

Tokenization Method	**Advantages**	**Disadvantages**
Character-Level	Simpler, faster to train, small vocab size.	Lower semantic quality, struggles with meaningful output beyond 1–2 tokens.
Byte-Pair Encoding	Balance between character and word-level representations, good structural coherence.	Slightly slower training, may merge irrelevant sequences due to frequency-only merging.
WordPiece	Stronger semantic generalization, ideal for tasks involving classification or nuanced meaning.	Higher perplexity, introduces subword fragments in generation.

These trade-offs highlight that tokenization is not just a preprocessing step but a critical component that shapes how the model learns and expresses information.

7.3. Practical Relevance

By implementing and analysing the transformer architecture and tokenization strategies from the ground up, this project enhances fundamental understanding of deep learning-based NLP systems. The findings are relevant for resource-constrained environments where training from scratch is required. The tokenizer comparison can inform decisions in real-world systems where trade-offs between memory, performance, and accuracy matter (e.g., mobile NLP apps, embedded AI). The controlled comparison of tokenizers in a custom model adds practical value to tokenization-focused research in generative modelling. The generated outputs and training behaviour reflect patterns similar to larger models, validating the architecture even at a small scale. Potential extensions include scaling the model to larger datasets, integrating pretraining + fine-tuning, or evaluating additional tokenization methods (e.g., Unigram, SetencePiece).

CHAPTER 8
CONCLUSION

This project explored the development lifecycle of a transformer-based text generation model, from architectural implementation to experimental evaluation using multiple tokenization strategies. Adopting a bottom-up approach, it clarified core elements of modern NLP systems—such as self-attention, positional embeddings, and autoregressive decoding—while emphasizing the critical role of tokenization in shaping model outcomes. By comparing character-level tokenization with subword methods like BPE and WordPiece, the study highlighted trade-offs in vocabulary size, training speed, and output quality. These results affirm that tokenization is a foundational design choice in generative modeling.

The application of this work lies in enabling more resource-efficient NLP systems, particularly where pre-trained models are not feasible. From educational tools that teach model internals to lightweight generative systems on edge devices, the ability to train compact, custom transformers with task-specific tokenization has clear value. Its flexibility also makes it suitable for creative applications like writing assistants or chatbot engines.

The utility of this project extends to informing researchers and developers on tokenizer selection for different use cases. Supporting multiple tokenizers within one architecture allows controlled experimentation and clearer understanding of how preprocessing affects learning and generation. This framework is also well-suited for hands-on educational environments.

Future work may include scaling to larger datasets, integrating pretraining and fine-tuning phases, and exploring additional tokenizers such as Unigram or SentencePiece. Enhancements like attention visualization or intermediate output inspection could further increase interpretability and align the project more closely with production-grade NLP systems.

REFERENCES

1. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention Is All You Need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

2. A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving Language Understanding by Generative Pre-Training," *OpenAI*, 2018.

3. T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan *et al.*, "Language Models are Few-Shot Learners," *arXiv preprint arXiv:2005.14165*, 2020.

4. Yafen Ye, Junbin Gao, Yuanhai Shao, Chunna Li, Yan Jin, Xiangyu Hua, "Robust support vector regression with generic quadratic nonconvex ε-insensitive loss", Applied Mathematical Modelling, Volume 82, June 2020 R. Sennrich, B. Haddow, and A. Birch, "Neural Machine Translation of Rare Words with Subword Units," in *Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016.

5. M. Schuster and K. Nakajima, "Japanese and Korean Voice Search," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012.

6. J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proc. of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

7. L. Liu, H. Jiang, M. He, W. Chen, X. Liu, J. Gao, and J. Han, "Understanding the Difficulty of Training Transformers," *arXiv preprint arXiv:2004.08249*, 2020.

8. T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs," *arXiv preprint arXiv:2305.14314*, 2023.