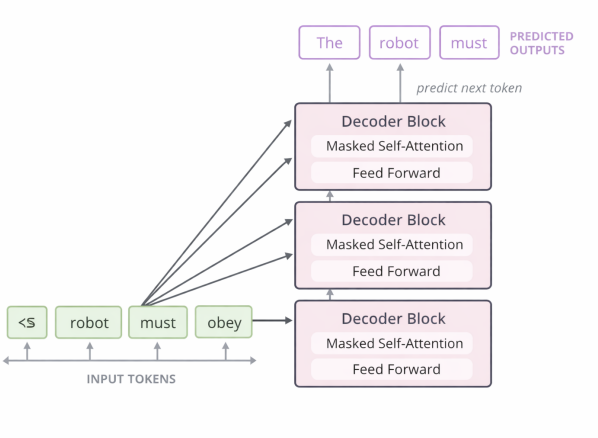
Winters in Data Science (WiDS) 2025

End-Term Report

Coding a ChatGPT like Transformer



Akshat Sharma
Roll No: 23B0311

Mentor: Sandipan

February 2026

Contents

1 Introduction

Natural Language Processing (NLP) is a core domain of artificial intelligence that focuses on enabling machines to understand, generate, and reason over human language. Over the last decade, NLP has undergone a major paradigm shift, transitioning from rule-based and statistical approaches to deep learning-driven architectures capable of learning rich representations directly from data.

One of the most impactful developments in this transition has been the introduction of Transformer-based architectures, particularly decoder-only language models such as GPT. These models form the backbone of modern large language models, including ChatGPT, and have demonstrated remarkable abilities in text generation, reasoning, and general-purpose language understanding.

The objective of this project, titled *Coding a ChatGPT like Transformer*, is not to replicate the scale or performance of production-grade systems, but rather to understand and implement the fundamental architectural principles that make such models possible. The emphasis is placed on conceptual clarity, mathematical understanding, and hands-on implementation of each component from first principles.

This report documents the complete learning journey undertaken during the WiDS program, beginning with foundational machine learning concepts, progressing through probabilistic language models, and culminating in the implementation of a decoder-only Transformer with self-attention. The report also discusses experimental observations, implementation challenges, and key insights gained throughout the process.

2 Foundations and Preliminaries

Before approaching modern Transformer-based language models, it is essential to develop a strong foundation in machine learning and deep learning concepts. This project built upon an understanding of numerical computation using tensors, gradient-based optimization, and neural network training dynamics.

PyTorch was used as the primary deep learning framework due to its flexibility and dynamic computation graph, which is particularly well-suited for research-oriented experimentation and debugging. Working with tensors, understanding broadcasting rules, and tracking tensor shapes across layers were critical skills developed during the early stages of the project.

From an NLP perspective, foundational concepts such as tokenization, vocabulary construction, and representation learning were explored. In this project, character-level tokenization was intentionally chosen over word-level or subword-level approaches. This choice simplified the preprocessing pipeline and allowed the focus to remain on architectural understanding rather than linguistic complexity.

Additionally, fundamental concepts such as loss functions and optimization algorithms were studied. Cross-entropy loss emerged as the standard objective for language modeling tasks, while adaptive optimizers such as Adam and AdamW were used to stabilize and accelerate training.

3 Language Modeling Fundamentals

Language modeling is the task of assigning probabilities to sequences of tokens. Given a sequence of tokens $x_1, x_2, \ldots, x_T$, the goal of a language model is to estimate the joint probability:

$$P(x_1, x_2, \ldots, x_T) \tag{1}$$

Using the chain rule of probability, this joint distribution can be factorized as:

$$P(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_1, \ldots, x_{t-1}) \tag{2}$$

This formulation highlights the autoregressive nature of language modeling, where each token is predicted based on all previous tokens. In practice, neural language models are trained to minimize the negative log-likelihood of the true next token, which corresponds to the cross-entropy loss.

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(x_t \mid x_1, \ldots, x_{t-1}) \tag{3}$$

This objective forms the basis for both simple probabilistic models and large-scale Transformer-based architectures.

4 Bigram Language Model

4.1 Motivation

The first language model implemented in this project was a Bigram Language Model. A bigram model predicts the next token using only the immediately preceding token, making it the simplest non-trivial autoregressive model.

Formally, the bigram assumption simplifies the conditional probability as:

$$P(x_t \mid x_1, \ldots, x_{t-1}) \approx P(x_t \mid x_{t-1}) \tag{4}$$

Despite its simplicity, this model provides valuable insight into probabilistic text generation and serves as an important baseline.

4.2 Model Architecture

The bigram model was implemented using an embedding table that directly maps each token to a vector of logits representing the probability distribution over the vocabulary. Conceptually, this embedding table can be interpreted as a learned transition matrix between characters.

Given an input token index i, the model outputs:

$$\text{logits} = W_i \tag{5}$$

where $W \in \mathbb{R}^{V \times V}$ and V is the vocabulary size.

The logits are converted into probabilities using the softmax function:

$$P(x_{t+1} \mid x_t) = \text{softmax}(\text{logits}) \tag{6}$$

4.3 Results and Observations

The bigram model successfully learned character-level transition probabilities and was able to generate text that superficially resembled English. However, the generated output lacked coherence, grammatical structure, and long-term consistency.

The output generated by the Bigram Language Model demonstrates the model's inability to capture long-range dependencies, resulting in incoherent and unstructured text.

Figure 1: Sample text generated by the Bigram Language Model

This behavior is expected, as the model has no memory beyond a single character. The limitations observed here directly motivated the exploration of architectures capable of modeling longer-range dependencies.

5 Motivation for Attention-Based Models

Natural language exhibits dependencies that span across words, sentences, and even entire documents. Fixed-context models such as bigrams or n-grams are fundamentally incapable of capturing such long-range relationships.

Historically, Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks were introduced to address this issue by maintaining hidden states across time steps. While effective, these architectures suffer from sequential computation bottlenecks and difficulty in parallelization.

Attention mechanisms provide an alternative by allowing models to dynamically select relevant parts of the input sequence when computing representations. This enables both efficient parallel computation and improved modeling of long-range dependencies, making attention a cornerstone of modern NLP architectures.

6 Self-Attention Mechanism

6.1 Intuition

Self-attention allows each token in a sequence to attend to all other tokens when computing its representation. Instead of processing tokens sequentially, the model evaluates relationships between all token pairs simultaneously.

Each token is projected into three vectors:

- Query (Q): represents what the token is looking for
- Key (K): represents what the token offers
- Value (V): represents the information content

6.2 Mathematical Formulation

The scaled dot-product attention mechanism is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{7}$$

where d_k is the dimensionality of the key vectors. The scaling factor stabilizes gradients during training.

6.3 Causal Masking

In decoder-only architectures, future tokens must not influence current predictions. This is enforced using a causal mask that prevents attention to future positions.
Figure 2 illustrates the internal computation performed by the self-attention mechanism for a single token in a sequence.

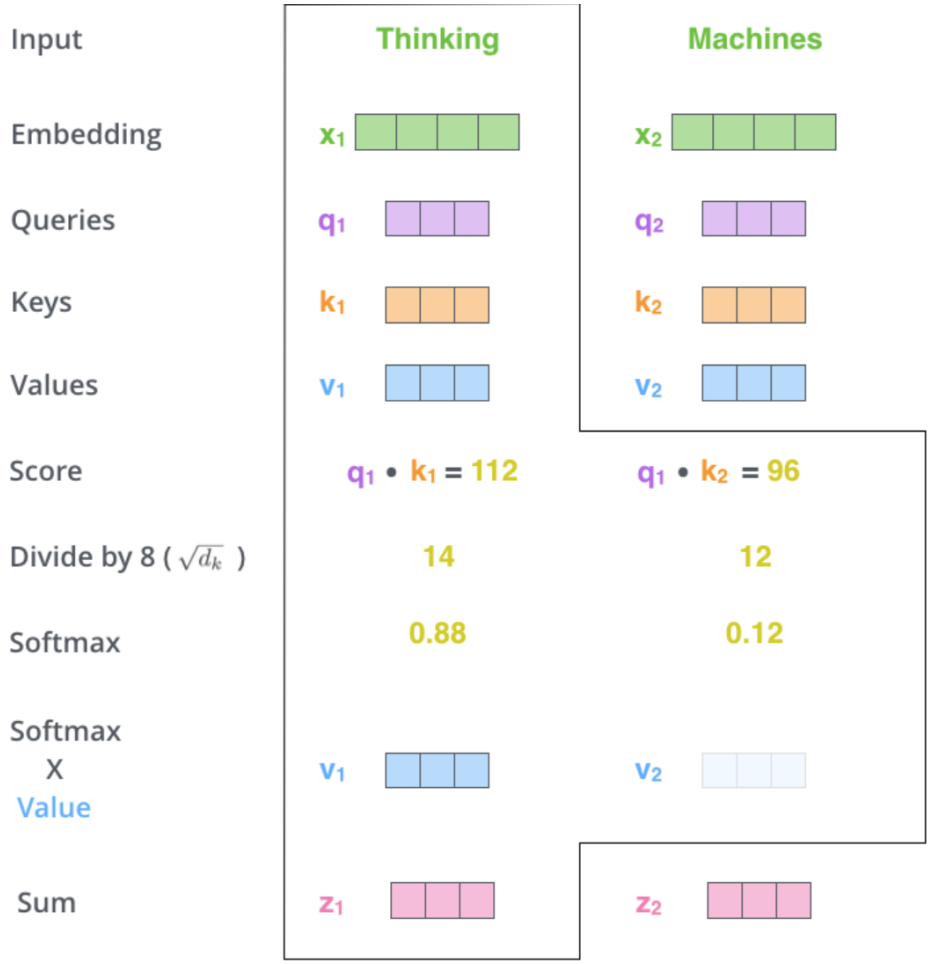


Figure 2: Self-attention mechanism illustrating Query, Key, and Value projections, scaled dot-product attention, and weighted aggregation of values

The figure is adapted from Jay Alammar's The Illustrated Transformer and is included for conceptual clarity.

7 Decoder-Only Transformer Architecture

The full model implemented in this project follows a GPT-style decoder-only Transformer architecture. Each Transformer block consists of the following components:

- Multi-head masked self-attention
- Feedforward neural network
- Residual connections
- Layer normalization

Multi-head attention allows the model to capture diverse relational patterns in parallel. Residual connections improve gradient flow, while layer normalization stabilizes training.

Positional embeddings are added to token embeddings to inject information about token order, since attention alone is permutation-invariant.

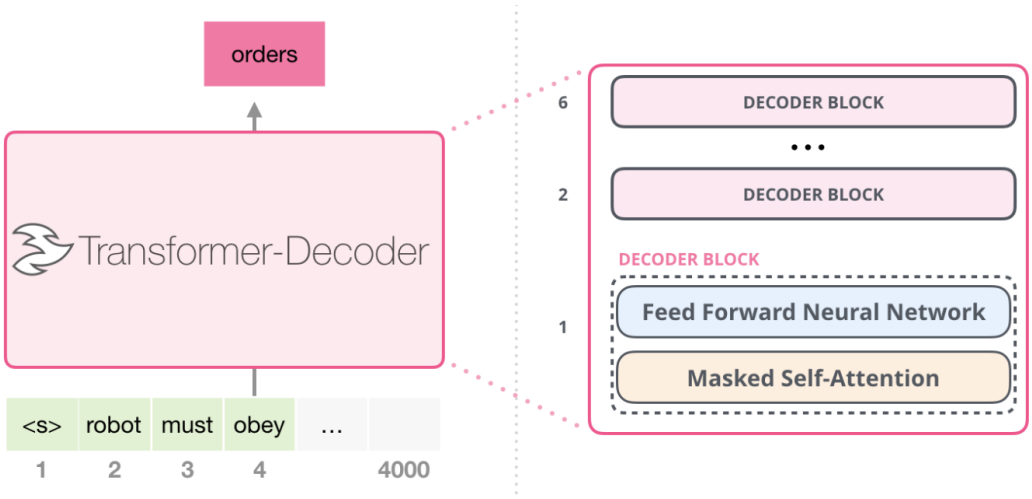


Figure 3: Decoder-only Transformer architecture showing stacked decoder blocks with masked self-attention and feed-forward networks

The figure is adapted from Jay Alammar's The Illustrated GPT-2 and is used for conceptual explanation of the decoder-only Transformer architecture.

8 Training Setup and Experiments

The model was trained on the Tiny Shakespeare dataset using character-level tokenization. Training involved splitting the data into training and validation sets to monitor generalization performance.

The AdamW optimizer was used due to its effective handling of weight decay and stable convergence properties. Hyperparameters such as batch size, learning rate, and context length were tuned empirically to balance training stability and computational constraints.

8.1 Dataset

The model was trained on the *Tiny Shakespeare* dataset, which consists of the complete works of William Shakespeare compiled into a single text corpus. The dataset was processed at the character level to simplify tokenization and focus on understanding the

Transformer architecture rather than linguistic preprocessing. The text was split into training and validation subsets, with the training set used for parameter optimization and the validation set used to monitor generalization performance during training.

Figure 4 illustrates the convergence behavior of the model, showing stable training and validation loss across iterations.

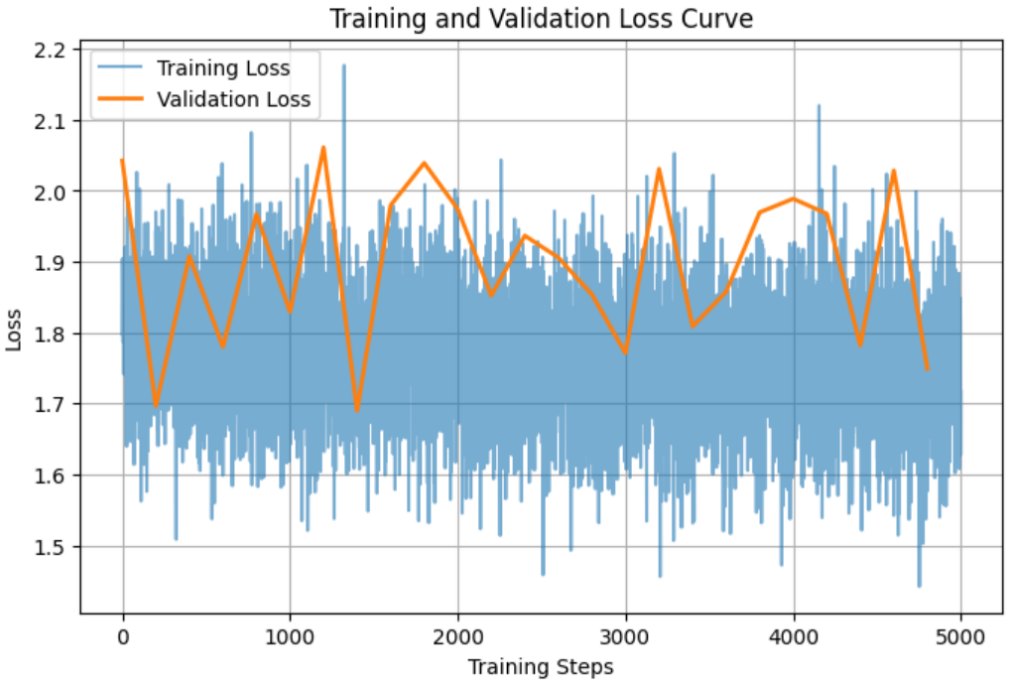


Figure 4: Training and validation loss over training iterations

9 Results and Text Generation

After training for several thousand iterations, the model achieved a stable validation loss in the range of approximately 1.5 to 2.0. The generated text demonstrated coherent spelling, consistent formatting, and emerging grammatical structure.
Figure 5 presents a sample of text generated by the trained Transformer model, demonstrating its ability to capture long-range contextual dependencies and produce coherent English-like text.

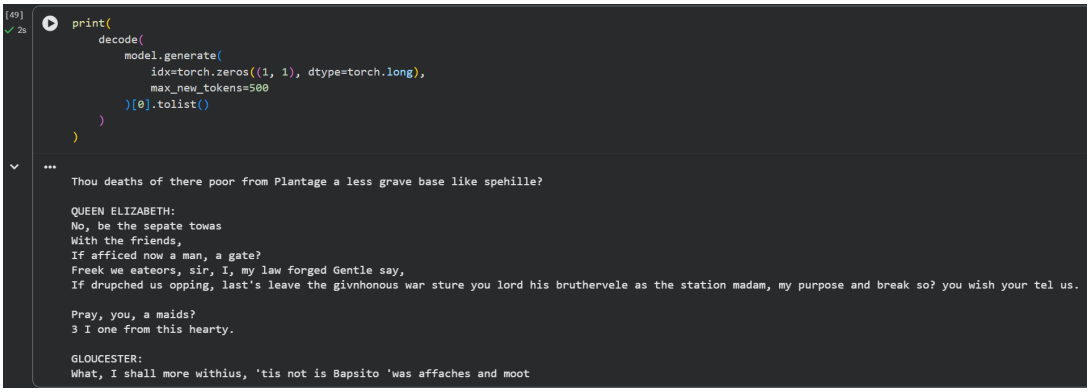


Figure 5: Sample text generated by the trained decoder-only Transformer model

While the output does not match human-level fluency, it clearly reflects the model's ability to leverage contextual information beyond immediate neighbors.

10 Challenges and Learnings

One of the most significant challenges encountered during this project was managing tensor dimensions across multiple attention heads and layers. Debugging shape mismatches required careful inspection of intermediate tensor shapes during the forward pass.

Additionally, GPU memory limitations necessitated careful tuning of batch size and context length. These challenges reinforced the importance of understanding both the mathematical and computational aspects of deep learning systems.

Overall, this project provided deep insight into transformer internals, attention mechanisms, and practical model debugging strategies.

11 Limitations and Future Work

The primary limitations of this project stem from dataset size and computational constraints. Training on larger datasets and scaling the model would likely improve text coherence and diversity.

Future work could explore token-level modeling, larger context windows, and fine-tuning on downstream NLP tasks such as summarization or question answering.

12 Conclusion

This project successfully demonstrated the end-to-end implementation of a ChatGPT-like decoder-only Transformer from scratch. By progressing from simple probabilistic models to attention-based architectures, the project achieved its core goal of developing a strong conceptual and practical understanding of modern language models.

The knowledge gained through this process provides a solid foundation for further exploration of large-scale NLP systems and deep learning research.

13 References

1. Vaswani et al., *Attention Is All You Need*, Advances in Neural Information Processing Systems (NeurIPS), 2017.
2. Andrej Karpathy, *Let's Build GPT from Scratch*, YouTube, 2023.
3. Jay Alammar, *The Illustrated Transformer*, available at: https://jalammar.github.io/illustrated-transformer/
4. Jay Alammar, *The Illustrated GPT-2*, available at: https://jalammar.github.io/illustrated-gpt2/