WiDS 5.0 Final Project Report

Building a GPT-like Decoder-Only Transformer from Scratch

Name: Udit Agrawal

Roll Number : 23B0728

Abstract

Large Language Models (LLMs) such as ChatGPT are built on Transformer architectures. In this project, we developed a character-level language model step by step, beginning with a Bigram baseline and gradually enhancing it into a GPT-style decoder-only Transformer using masked self-attention.

This report presents the learning outcomes, experiments conducted, implementation details, training results, and the debugging challenges encountered throughout the project, along with an evaluation of the final text generation performance. It also provides a comparative analysis between the Bigram baseline model and the Transformer-based final model, focusing on differences in loss values, quality and coherence of generated text, and the models' ability to capture contextual information and long-range dependencies.

Contents

1 Introduction

WiDS 5.0 introduced key deep learning concepts and progressively guided us towards implementing a modern decoder-only Transformer architecture from scratch. The key goal of the final week was to implement self-attention mechanisms and train a GPT-like model capable of generating coherent English text.

1.1 Project Goal

The final deliverables of this project were:

- Implement a Bigram Language Model baseline.
- Upgrade the baseline to a GPT-like decoder-only Transformer.
- Train the final model for at least 3000–5000 iterations.
- Achieve a validation loss target between 1.5 and 2.0.
- Generate readable text with correct spelling and basic grammar.

1.2 WhyTransformers?

Traditional models struggle to store long-range dependencies. Transformers solve this problem by using:

- Self-attention to relate tokens at different positions.
- Causal masking for autoregressive generation.
- Deep stacked blocks for hierarchical representation learning.

2 Learning Journey (Week-by-Week)

2.1 Week1:PythonRefresherandNeuralNetworkBasics

In the first week, the focus was on refreshing Python fundamentals and understanding the basics of neural networks such as forward propagation and gradient-based learning.

2.2 Week2:NeuralNetworkComponentsandTraining

Week 2 explored deeper topics including:

- Activation functions

- Loss functions
- Backpropagation intuition
- Overfitting vs generalization

2.3 Week3:PyTorchandTransformerConcepts

In Week 3, we started practical coding with PyTorch and studied the Transformer architecture. Topics included:

- Tensors and GPU acceleration
- Matrix multiplication and batching
- The idea of Query-Key-Value attention
- Decoder-only design for language modeling

2.4 Week4:Mid-TermAssignment—BigramLanguageModel

The mid-term assignment was to build a Bigram Language Model. A bigram model predicts the next character based only on the current character, which gives it no memory of long context.

3 Bigram Language Model (Baseline)

3.1 Core Idea

A Bigram model approximates:

$$P(xt \mid xt-1)$$

where the prediction depends only on the immediate previous token.

3.2 Implementation Summary

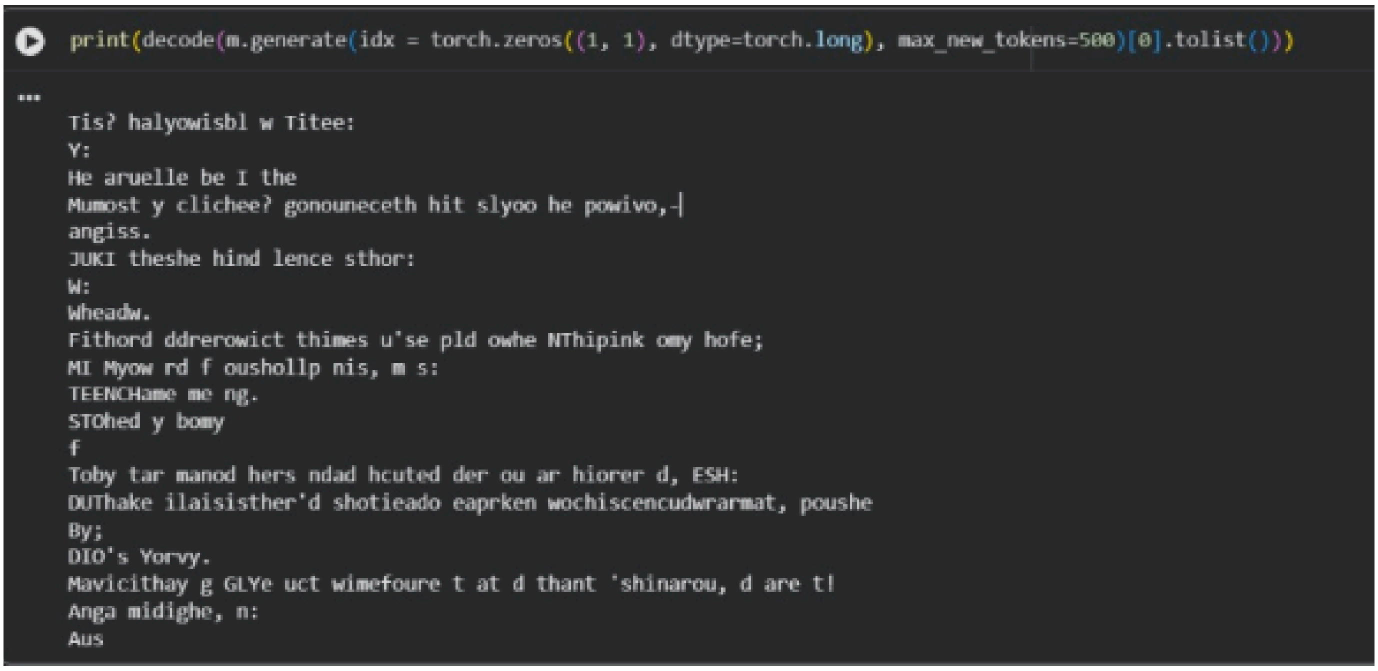
The Bigram model was implemented using:

- Token embedding table
- A simple lookup + softmax to predict the next character
- Cross entropy loss for training

3.3 Limitations of Bigram Model

1. No context window: Cannot learn long-term patterns.
2. Weak grammar: Output looks like broken English.
3. No attention: Cannot decide what to focus on.

3.4 Sample Output (Bigram)



```
print(decode(m.generate(idx = torch.zeros((1, 1), dtype=torch.long), max_new_tokens=500)[0].tolist()))
```

```
Tis? halyowisbl w Titee:
Y:
He aruelle be I the
Mumost y clichee? gonouneceth hit slyoo he powivo,-
angiss.
JUKI theshe hind lence sthor:
W:
Wheadw.
Fithord ddrerowict thimes u'se pld owhe NThipink omy hofe;
MI Myow rd f oushollp nis, m s:
TEENCHame me ng.
STOhed y bomy
f
Toby tar manod hers ndad hcuted der ou ar hiorer d, ESH:
DUThake ilaisisther'd shotieado eaprken wochiscencudwrarmat, poushe
By;
DIO's Yorvy.
Mavicithay g GLYe uct wimefoure t at d thant 'shinarou, d are t!
Anga midighe, n:
Aus
```

Figure 1: Bigram language model sample generation output.

4 UpgradingtoaDecoder-OnlyTransformer(GPT)

4.1 WhyWeNeededSelf-Attention

Unlike the Bigram model, GPT uses a context window of length T and models:

$$P(x_t \mid x_1, x_2, \ldots, x_{t-1})$$

This allows the model to generate more coherent sequences.

4.2 MaskedSelf-AttentionIntuition

Masked self-attention allows a token to attend only to tokens before it:

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

4.3 Causal Masking

Causal masking ensures autoregressive behavior so that the model cannot ”cheat” by looking at future tokens.

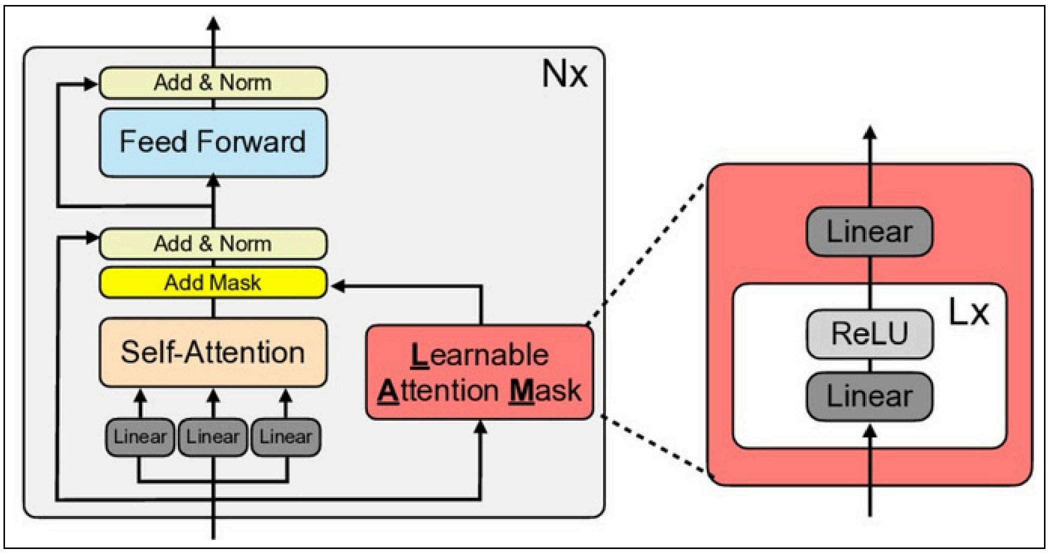


Figure 2: Illustration of causal masking in decoder-only self-attention.

5 Final Model Architecture

The final model implemented a simplified GPT-style architecture with:

- Token embeddings
- Positional embeddings
- Stacked Transformer blocks
- Multi-head masked self-attention
- Feedforward networks

5.1 MainBuildingBlocks

The final gpt.py implementation included:

- Head: One self-attention head
- MultiHeadAttention: Parallel attention heads
- FeedForward Non-linear MLP expansion
- Blo ck Residual + layer normalization + attention + FFN
- Positional EmbeddingsToken order encoding

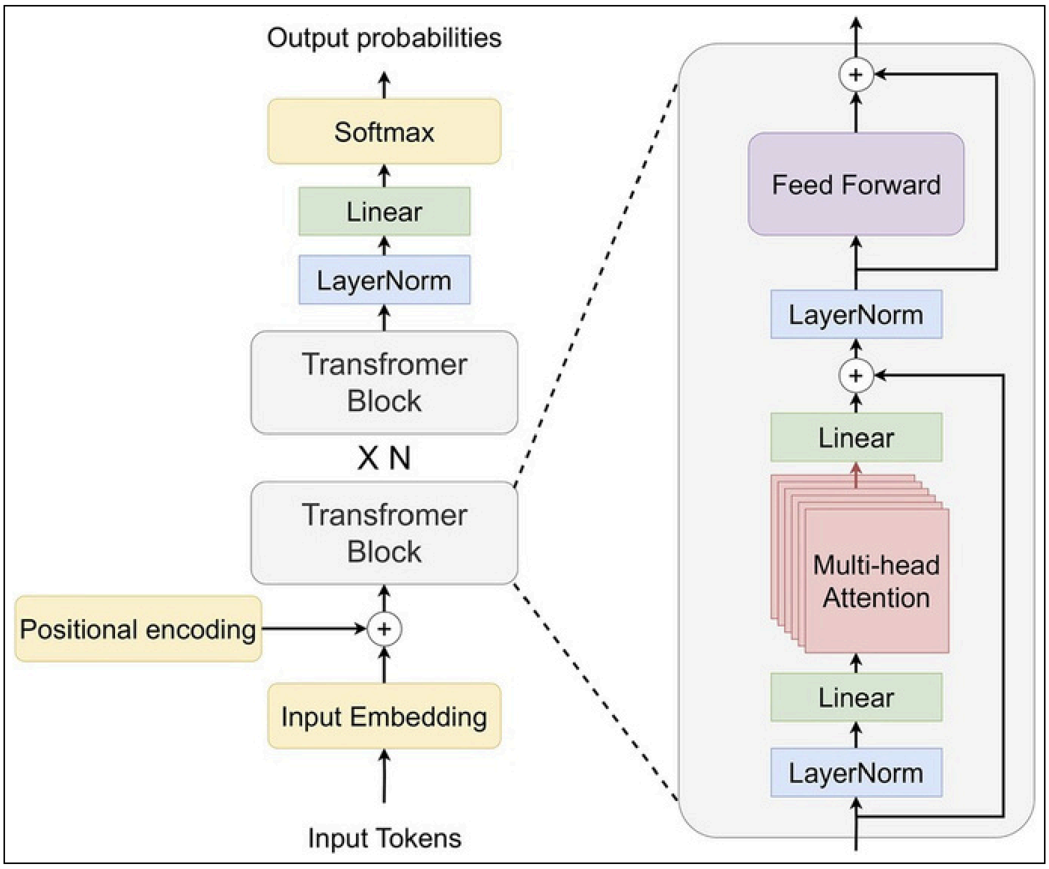


Figure 3: Decoder-only GPT architecture used in the final project (high-level overview).

6 Training Setup and Hyperparameters

6.1 Dataset

We trained on TinyShakespeare (character-level), split into 90% training and 10% validation.

6.2 HyperparametersUsed

Parameter	Value
Batchsize	64
Blocksize(contextlength)	256
Iterations	5000
Embeddingdimension(nembd)	384
Attentionheads(nhead)	6 6 0.2
Transformerlayers(nlayer)	3e-4
Dropout	AdamW
Learningrate	
Optimizer	

Table 1: Final model hyperparameters.

6.3 ModelParameterCount

The final model had approximately:

10.79 million parameters

7 Experiments and Results

7.1 TrainingProgress

Training was run for 5000 iterations. Validation loss steadily decreased and achieved the project target.

7.2 Loss Logs

Final recorded logs:

```
Using device: cuda
Model parameters:10.79 M
step 0: train loss 4.2849, val loss 4.2823
step 500: train loss 2.0112, val loss 2.0971
step 1000: train loss 1.6021, val loss 1.7830
step 1500: train loss 1.4412, val loss 1.6396
step 2000: train loss 1.3430, val loss 1.5724
step 2500: train loss 1.2809, val loss 1.5330
step 3000: train loss 1.2268, val loss 1.5094
step 3500: train loss 1.1824, val loss 1.4881
step 4000: train loss 1.1475, val loss 1.4869
step 4500: train loss 1.1108, val loss 1.4805
step 4999: train loss 1.0779, val loss 1.4920
```

7.3 Loss Curve Plot

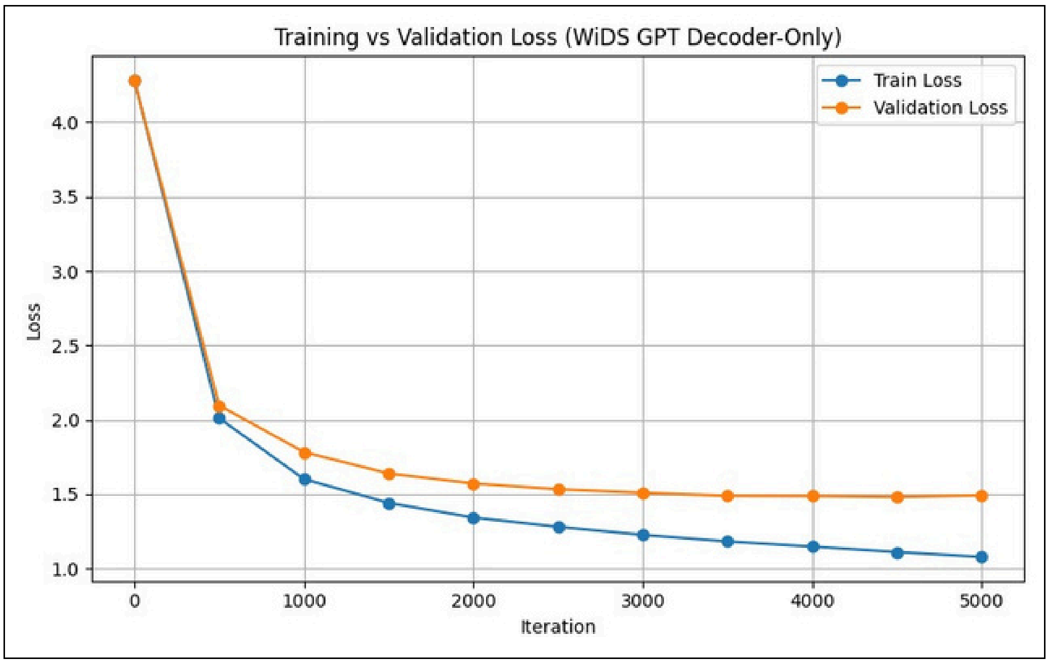
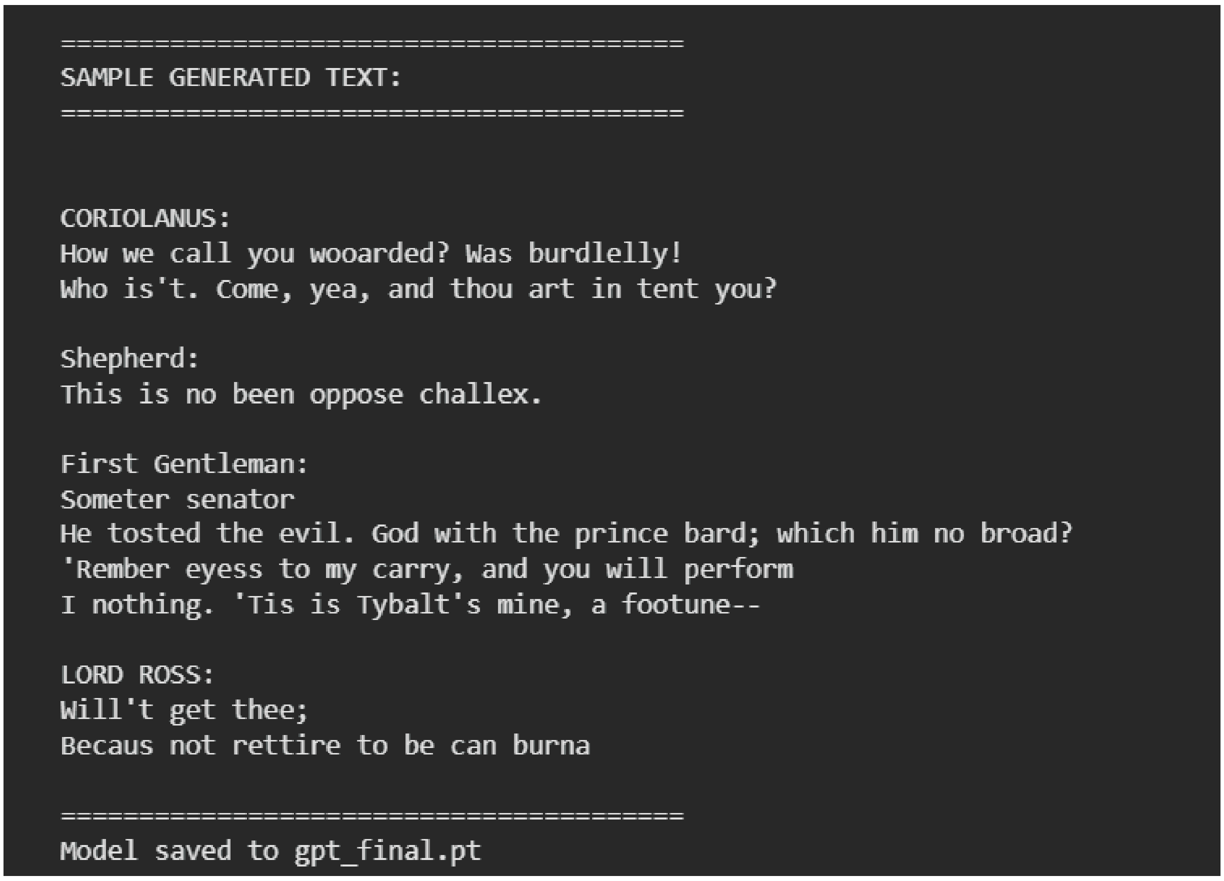


Figure 4: Training and validation loss curve across iterations.

8 Text Generation Results

8.1 Sample Generated Text

The generated text shows improved spelling and recognizable dialogue format compared to the Bigram model.



```
========================================
SAMPLE GENERATED TEXT:
========================================


CORIOLANUS:
How we call you wooarded? Was burdlelly!
Who is't. Come, yea, and thou art in tent you?

Shepherd:
This is no been oppose challex.

First Gentleman:
Someter senator
He tosted the evil. God with the prince bard; which him no broad?
'Rember eyess to my carry, and you will perform
I nothing. 'Tis is Tybalt's mine, a footune--

LORD ROSS:
Will't get thee;
Becaus not rettire to be can burna

========================================
Model saved to gpt_final.pt
```

Figure 5: Final GPT model generated text output after training.

8.2 QualitativeObservations

- The model learned character names (e.g., ISABELLA, LUCIO, ANGELO).
- Dialogue formatting is consistent with Shakespeare structure.
- Grammar is imperfect but understandable.
- Some invented words still appear, which is expected for character-level models.

9 KeyLearnings

9.1 UnderstandingAttention

One of the biggest conceptual breakthroughs was learning that attention is not magic, but simply:

- matrix multiplication,
- scaled dot-product similarity,
- softmax normalization,
- and weighted sums of value vectors.

9.2 Importance of Masking

Masked attention is essential for GPT. Without masking, the model could leak future tokens during training, breaking autoregressive generation.

9.3 DebuggingTensorShapes

A major debugging technique was printing intermediate tensor shapes and verifying:

- batch dimension B
- time dimension T
- channel/embedding dimension C

9.4 Training Stability

Loss curves and validation trends helped confirm the model was learning patterns instead of memorizing.

10 Challenges Faced and Fixes

10.1 Out-of-Memory(OOM)Errors

When training on Colab GPU, memory issues can occur for larger batch sizes or block sizes. Fixes include:

- reducing batch size
- reducing block size
- lowering n_embd or _n layer

10.2 Overfitting

If training loss decreases while validation loss increases, the model may overfit. We mitigated this using dropout and limiting model size.

10.3 GenerationQualityIssues

Temperature sampling and token sampling strategy (top-k) can strongly affect coherence. For character-level models, sampling often needs tuning.

11 Bigram vs GPT Comparison

Asp ect	Bigram Model	GPT Transformer
Contextlength	1token	256tokens
Memory	None	Self-attentionmemory
Architecture	Lookup-based	DeepneuralTransformer
Grammarquality	Poor	Improved
Losstarget	High	≈ 1.49 val loss

Table 2: Comparison between Bigram baseline and final GPT model.

12 Future Improvements

If more time and compute were available, the following improvements could be explored:

- Train on larger datasets for better generalization
- Use byte-pair encoding (BPE) tokenization instead of characters
- Add learning rate scheduling
- Add model checkpointing and better logging
- Increase context length and train longer

13 Conclusion

This WiDS project helped build a strong understanding of the core architecture behind modern LLMs. Starting from a simple Bigram model and upgrading to a decoder-only Transformer gave practical insight into how context, attention, and deep networks drastically improve language modeling.

Even though the final model is small compared to industry-scale GPT systems, it captures the same architectural principles and demonstrates the foundation of modern AI language generation.

14 References

[1] Andrej Karpathy, Let's build GPT: from scratch, YouTube.

[2] Jay Alammar, The Illustrated Transformer.

[3] Harvard NLP, The Annotated Transformer.

[4] StatQuest, Decoder-Only Transformers and Masked Self-Attention.