Cisse, Yelene

Dr. Spencer W. Luo

COMS 4995W32

17 November 2025

Tiny Transformer Model for Next Token Prediction: Hyperparameter Analysis

Introduction

This report analyzes a tiny transformer model built for next token prediction on Shakespeare text, consisting of 581,009 tokens tokenized using Byte-Level BPE. The primary learning objective is to understand how key hyperparameters—number of epochs, learning rate, batch size, sequence length, vocabulary size, and model architecture dimensions—affect transformer performance through systematic experimentation.

The analysis follows a structured approach:

(1) Data preprocessing with BPE tokenization and sequence formatting
(2) Baseline model development (v1) with 60 training epochs
(3) Systematic hyperparameter variation across six model versions (v2-v7) to evaluate their impact on perplexity, accuracy, and loss.

Training was conducted on Tesla T4 GPU with average runtime of ~13 minutes per model, with evaluation metrics focusing on perplexity as the primary measure of model quality.

Data preprocessing

The Shakespeare dataset provided was tokenized using ByteLevelBPETokenizer with vocab_size=500 and min_frequency=2, producing 581,009 tokens. A SeqFormat function created overlapping sequences of length 50 (baseline) with stride of 1, generating 580,985 sequences. Data splitting preceded all transformations to prevent leakage, with an 80-20 train-test split (random_state=62, shuffle=True) yielding 464,787 training and 116,197 test sequences.

The model was built to get next sequence of tokens Y = [32, 98, 101, 11, 111, 114] for an input X = [71, 32, 98, 101, 11, 111].

Model Architecture

The tiny transformer implements standard transformer components: nn.Embedding() for token embeddings, learnable positional encoding via nn.Parameter(), and two transformer blocks. Each block contains RMSNorm layers applied before self-attention and feed-forward network (FFN) calls, single-head self-attention with causal masking, two-layer MLP with ReLU activation, and residual connections after both attention and FFN sub-layers. The model uses cross-entropy loss, Adam optimizer with weight_decay=1e-5, gradient clipping (max_norm=1.0), and ReduceLROnPlateau scheduler (patience=50, factor=0.5). Total parameters: 493,940.

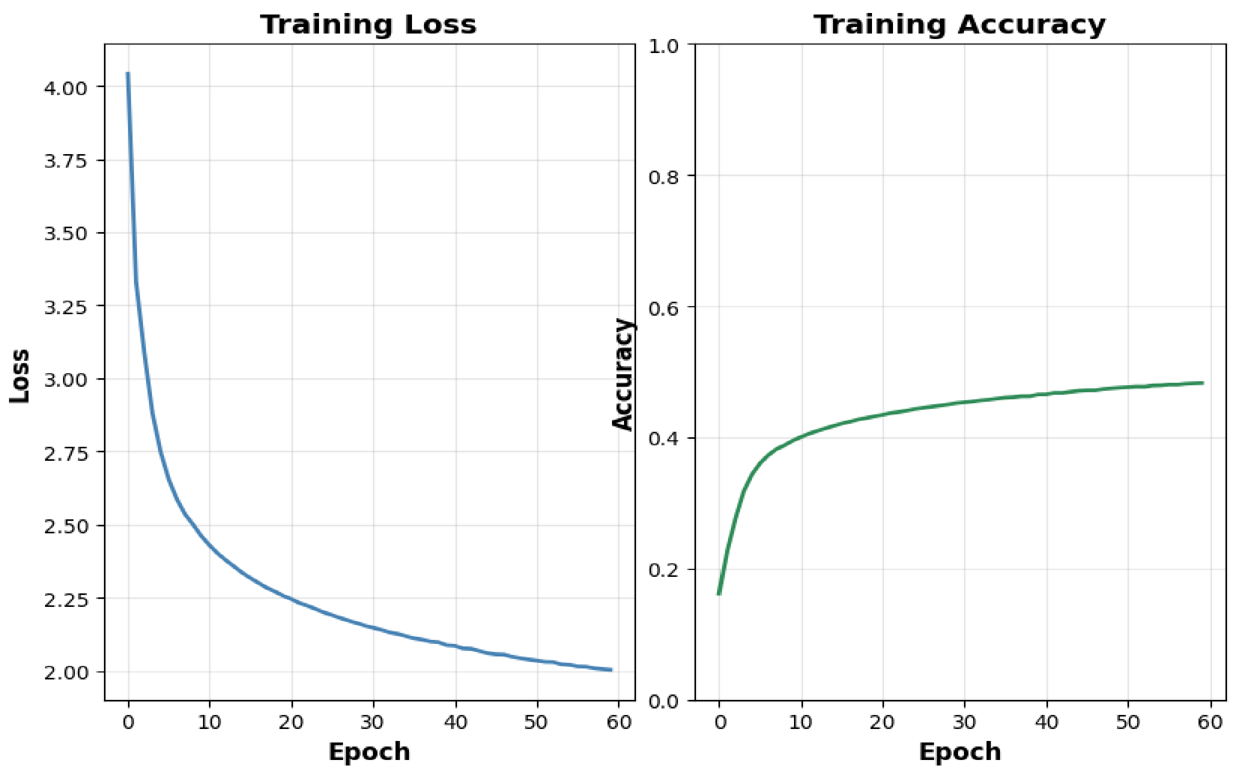
Baseline Parameters (v1):

Parameter	Value
vocab_size	500
fixed_len_seq	50
embedding_dim	128
n_layers	2
d_ff	512
learning_rate	0.001
n_epochs	60
batch_size	4096

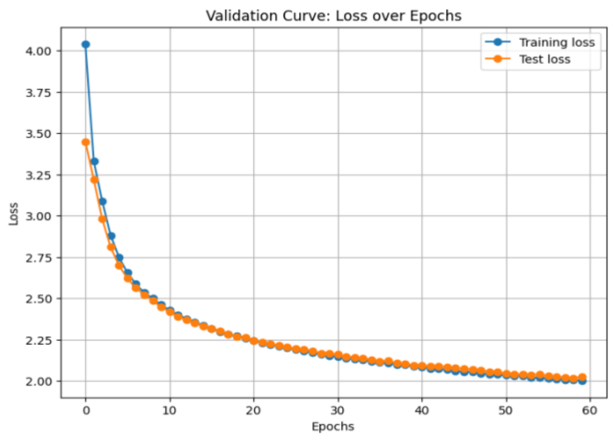
Key Observations

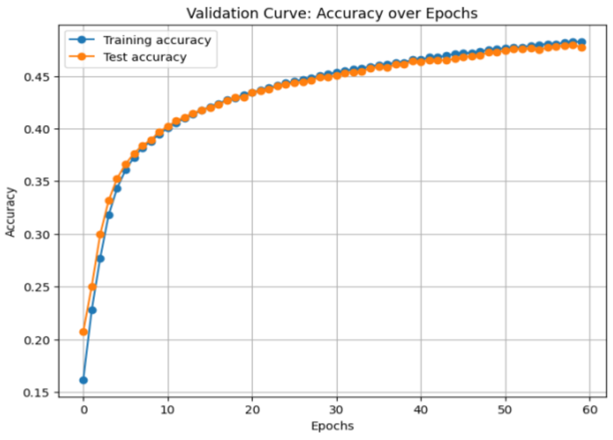
1. **Baseline Model (v1)**

The baseline model (v1) was trained for 60 epochs with the parameters specified above. The model's best performance was the 59th epoch with perplexity=7.527, loss=2.019, and accuracy=0.480. Training loss and accuracy improved as the number of epochs increased, demonstrating that a higher number of trainings leads to improvement in metrics, as you would expect the model to become better at predicting next token as it learns more over training runs.

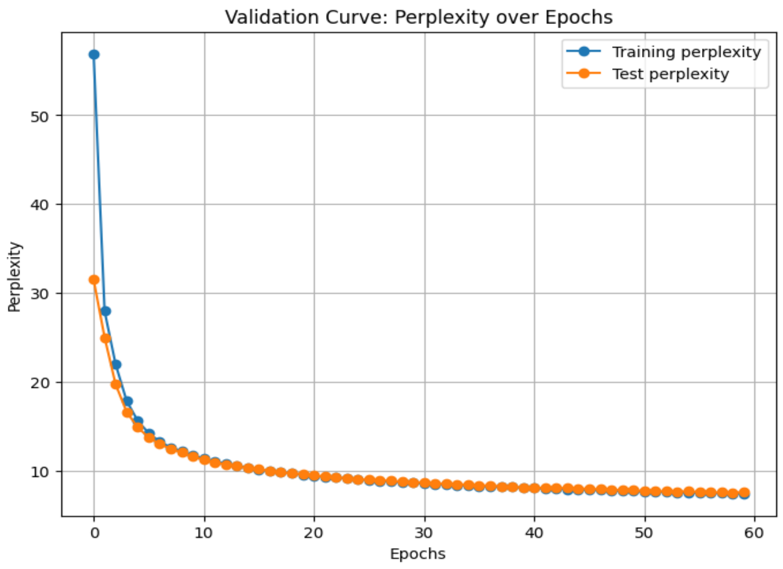


The model also performed well on unseen data, as we can see from the validation curves showing little to no gap between train and test results as epochs increase.





This close alignment between training and validation performance is crucial - it indicates strong generalization with minimal overfitting, meaning the model is learning genuine patterns rather than memorizing training data. The same is observed for perplexity: it decreases as the model learns from the data over more training rounds.



Note that the 60th epoch performed slightly worse than the 59th, perhaps the start of an indicator of overfitting and reminding the importance of calibrating the number of epochs as to not overfit the data and include high variance in the model.

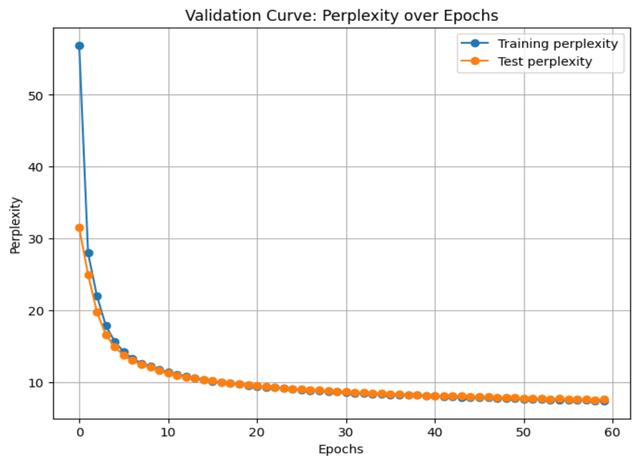
Overall, this shows that the number of trainings help to get better next token predictions, but we still need to be careful of overfitting as illustrated by epochs 59 and 60. Additionally, the learning rate scheduler (patience=50) never triggered throughout training, indicating the model continued making consistent progress with lr=0.001, suggesting potential for even longer training runs.

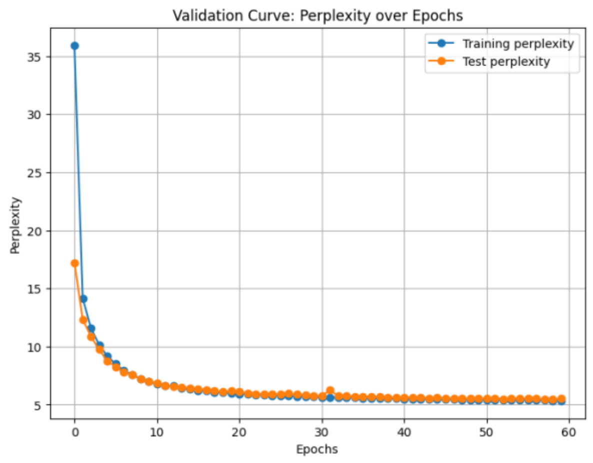
We will use this v1 configuration (epoch=60, along with other parameters) as baseline for the rest of this analysis.

2. The Effect of Learning Rate

The second version of the model used a higher learning rate of 0.01 compared to baseline model (=0.001). From the results below, we can see that the v2 model finds optimal metrics faster than v1, and achieves best perplexity of 5.460.

Model	Best_epoch	Best_perplexity	Best_loss	Best_accuracy
v1	59	7.527	2.019	0.480
v2	59	5.460	1.698	0.550





Train and test perplexity v2 (left plot) vs v3 (right plot)

These results illustrate the importance of choosing the right learning rate, as too slow means we stop training before we can find a better result (as seen here with v1) and too fast could lead to missing optimal results altogether.

We can try to avoid stagnation at a given rate by using a scheduler with a set patience that would update the learning rate if no improvement is shown for a given number of epochs. This is especially useful when running a great number of epochs for a model. In our analysis, this approach was tested but the learning rate remained the same over the 60 epochs as we could see continual improvements over training rounds.

3. The Effect of Batch Size

The 3rd version of the model evaluated the impact of batch_size=128 on model performance, compared to the baseline batch_size=4096. While training the model, perplexity and other metrics for the first 3 epochs were immediately significantly better for version 3 than for version 1.

Model	Epoch	Best_perplexity	Best_loss	Best_accuracy
v1	1	31.504	3.450	0.207
v1	2	24.993	3.219	0.250
v1	3	19.712	2.981	0.299

Metrics Model v1

Model	Epoch	Best_perplexity	Best_loss	Best_accuracy
v3	1	10.517	2.353	0.414
v3	2	8.792	2.174	0.447
v3	3	7.956	2.074	0.467

Metrics Model v3

This is as expected, as the smaller batch size means the model will make more gradient updates per epoch (464,787/128 ≈ 3,631 updates vs. 464,787/4096 ≈ 113 updates for v1).

While each update uses a noisier gradient estimate based on fewer samples, the more frequent updates allow the model to explore the loss landscape more thoroughly and make faster progress per epoch. Additionally, gradient noise from small batches acts as implicit regularization, often improving generalization.

However, the running time for version 3 was also significantly greater: approximately 30 minutes compared to v1 training which took roughly 13 minutes. This highlights a trade-off between computational running time and optimal performance.

Model	Best_epoch	Best_perplexity	Best_loss	Best_accuracy
v1	59	7.527	2.019	0.480
v3	60	5.366	1.680	0.552

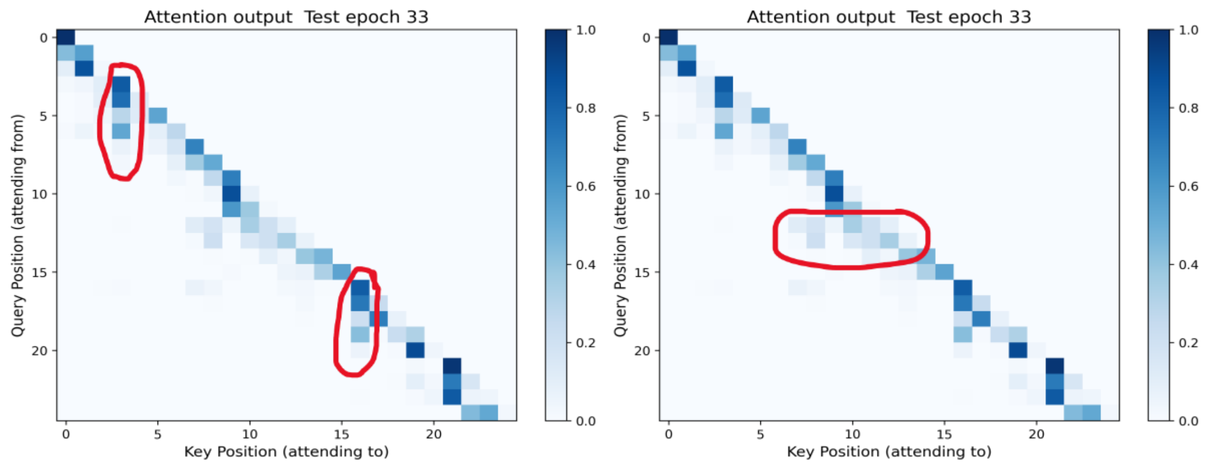
Overall, while the performance is better for v3, the computational cost is not worth the trouble. Similar results can be achieved in much lesser time with better learning rate (v2 model for reference).

4. Sequence Length Variation

For the 4th run, sequence_length=25 was tested instead of 50 from baseline. The attention matrix was also created since smaller sequence made the plot more readable for interpretation. We can see from the table below that the smaller sequence length led to suboptimal results compared to v1.

Model	Best_epoch	Best_perplexity	Best_loss	Best_accuracy
v1	59	7.527	2.019	0.480
v4	60	8.339	2.121	0.456

This is in accordance with what we would expect as a smaller sequence length gives the model less material to learn from at a time. It suggests less overlap from one sequence to another, leading to worse generalization compared to a higher sequence length. This intuition is confirmed when looking at the self-attention matrix.



Vertical (left) and horizontal (right) patterns

From the plots, we noticed some important tokens at key positions highlighted on the left plot, as they are attended from multiple tokens surrounding them. We also see a cluster of two tokens attending to multiple tokens (right plot), indicating that they gather a broad context for model interpretation for next token prediction. We also see a strong diagonal pattern, which means tokens attend to themselves as expected. The blank upper right triangle confirms the causal mask is enforced, ensuring tokens can only learn from previous ones and cannot "peek" into the future, which is important for next token prediction.

5. Vocabulary Size

The 5th version of the model used vocab_size=260 instead of baseline 500. The results for this run were surprising, as we could see significantly better performance for this model:

Model	Best_epoch	Best_perplexity	Best_loss	Best_accuracy
v1	59	7.527	2.019	0.480
v5	60	3.690	1.306	0.584

Upon reflection, several factors explain this substantial improvement (51% reduction in perplexity). One possible explanation is that the smaller vocabulary reduces the output layer parameters by approximately 30,700 (~6% of total model), providing better regularization for our dataset size. With vocab_size=260, each token appears more frequently in the training data, allowing the model to learn more robust token embeddings rather than spreading capacity across many rare tokens that appear infrequently, as could be the case for vocab size 500. Another possibility is that the reduced output space (260 vs 500 classes) simplifies the softmax prediction task, making it easier for the model to learn accurate probability distributions over the vocabulary.

6. Model Size (Feed Forward Network and Embedding Dimension)

For the 6th and 7th run of the model, two modifications were made to reduce model capacity compared to baseline: v6 reduced the FFN hidden dimension from 512 to 256 while maintaining embedding dimension at 128, and v7 reduced the embedding dimension from 128 to 64 while maintaining FFN at 512.

Model	Feed forward Network (FFN)	Embedding dimension (ED)
Baseline v2	512	128
V6	256	128
V8	512	64

Overall, the reduced model size led to suboptimal results as illustrated in the table below:

Model	Best_epoch	Best_perplexity	Best_loss	Best_accuracy
v2	59	7.527	2.019	0.480
v6	60	9.278	2.228	0.437
v8	59	10.851	2.384	0.410

This is in accordance with our expectation as a larger FFN dimension increases model capacity, allowing it to learn more complex non-linear feature transformations. The v6 model's 50% reduction in FFN resulted in 23% worse perplexity. Similarly, a greater embedding dimension gives the model more representational space to encode token information - v7's reduction to 64 dimensions made it the worst performer with 44% higher perplexity, as the compressed space cannot effectively distinguish between 500 vocabulary tokens.

It is important to keep in mind the running time while tuning these parameters as similar to batch size, a model size too grand could cause unnecessary computational cost, as well as memory usage, while a smaller model could achieve close results to optimal solution in a fraction of the time. For this dataset size, the baseline architecture appears appropriately sized - further reductions degrade performance significantly.

Conclusion

This comparative analysis demonstrates systematic effects of hyperparameter choices on transformer model performance for next token prediction task. The baseline model v1 (60 epochs, perplexity=7.527) established strong performance with excellent generalization (minimal train-test gap). Surprisingly, v5 (vocab_size=260, perplexity=3.690) emerged as a better performer, achieving 51% improvement over baseline by providing better token frequency distribution and parameter efficiency for the dataset size.

Key findings reveal that learning rate optimization is critical - v1's lr=0.001 was too slow, while v2's lr=0.01 achieved 27% better perplexity (5.460) in the same training time.

Batch size presented a clear trade-off: v3 (batch_size=128) achieved strong results (perplexity=5.366) but required 2.3× longer training time (30 vs 13 minutes), making larger batch sizes more practical.

Sequence length of 50 outperformed 25, providing sufficient context for pattern learning and larger model architectures (FFN=512, embedding_dim=128) consistently outperformed reduced versions, with v7's smaller embedding dimension showing 44% worse perplexity.

These findings emphasize that thoughtful hyperparameter selection can yield substantial performance gains.

Notebook:

- Colab link: https://colab.research.google.com/drive/1MUAzU1YLLEB08abaVAvfwUr9sUU6OgkS?usp=sharing
- Github link: https://github.com/ymciss0/Applied_Machine_Learning/blob/dev/AML_HW3_YC.ipynb

---

AI Tool Usage Disclosure

This analysis utilized ChatGPT and Claude AI for code debugging, hyperparameter tuning suggestions, plotting function generation, and general troubleshooting throughout the analysis workflow. Claude AI generated the polished final report based on the provided outline, draft observations, and annotated code. The final report underwent manual editing for technical accuracy and alignment with observations. Analytical conclusions, model interpretations, and comparative evaluations represent original work based on direct examination of model outputs and training results.

Works Cited (Includes code references)

Columbia University. "L8_Transformer.ipynb." Applied Machine Learning Course Materials, 2025, colab.research.google.com/drive/1lsYBX2X_MeZtJOrI_CiUNCVRlVa9DzmK.

---. "L9_Transformer++.ipynb." Applied Machine Learning Course Materials, 2025, colab.research.google.com/drive/1UMaCbu1-4giQQbKblTkWmKHSrne7KgtP.

Hugging Face. "Byte-Pair Encoding Tokenization." Hugging Face LLM Course, huggingface.co/learn/llm-course/en/chapter6/5.