**Instituto Politécnico Nacional**
**Centro de Investigación en Computación**
**Cómputo Conexionista (B25)**
**Prof.: Dr. Ricardo Menchaca Méndez**

# Mini-GPT

Luis Angel Rojo Chavez
lrojoc2024@cic.ipn.mx
arojocz@gmail.com
**https://github.com/arojocz/mini-gpt**

December 15, 2025

## Abstract

This report is the implementation of a Decoder-Only Transformer architecture, trained on the Tiny Shakespeare dataset for character-level text generation. We construct the architecture from scratch using PyTorch, focusing on the manual implementation of Self-Attention, Sinusoidal Positional Encodings, and Layer Normalization. Furthermore, we analyze the generation quality using three different decoding strategies: Multinomial Sampling, Beam Search, and Greedy Decoding.

## 1 Introduction

The Transformer architecture (Vaswani et al., 2017) has become standard for Natural Language Processing, replacing Recurrent Neural Networks (RNNs) due to its ability to parallelize training and capture long-range dependencies.

We focus on implementing the *Scaled Dot-Product Attention* and *Sinusoidal Positional Encodings* , ensuring a deep understanding of the gradient flow and architectural decisions that enable these models to learn.

## 2 Methodology and Implementation

### 2.1 Dataset and Tokenization

We utilized the "Tiny Shakespeare" dataset. Preprocessing involved a character-level tokenizer mapping 65 unique characters to integers. The data was split into training (90%) and validation (10%) sets. It is implemented a data loader that serves batches of context blocks using random sampling.

### 2.2 Positional Embeddings

Unlike Recurrent networks, the Transformer has no inherent sense of sequence order. So we implemented **Sinusoidal Positional Encodings** as in Attention is all you ned for a position *pos* and dimension $i$:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}}) \tag{1}$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}}) \tag{2}$$

These encodings are added to the token embeddings, allowing the model to extrapolate to sequence lengths not seen during training.

### 2.3 Scaled Dot-Product Attention

The core mechanism is the Dictionary or memory lookup. We implemented the attention function receiving Query ($Q$), Key ($K$), and Value ($V$) tensors:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3}$$

Since this is a decoder-only model for generation, we are setting the upper triangular matrix to $-\infty$ before the softmax, preventing positions from knowing the the future.

### 2.4 Multi-Head Attention

We implemented Multi-Head Attention (MHA) to allow the model to attend to information from different representation subspaces jointly. The implementation involved:

1. Linear projections for $Q, K, V$.
2. Reshaping tensors to split heads: $(B, T, C) \rightarrow (B, T, n_{head}, head\_size)$.
3. Transposing for batch matrix multiplication.
4. Concatenation and final linear.

## 3 Experimental Setup

The model was trained using the AdamW optimizer on a GPU environment.

- **Batch Size:** 32
- **Block Size (Context):** 128
- **Embedding Dimension ($d_{model}$):** 384
- **Heads:** 6
- **blocks:** 4
- **Learning Rate:** $3e^{-4}$
- **Dropout:** 0.2
- **Training Steps:** 5000

## 4 Results

### 4.1 Pos encoding

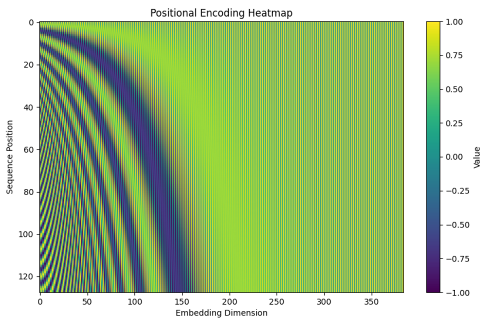
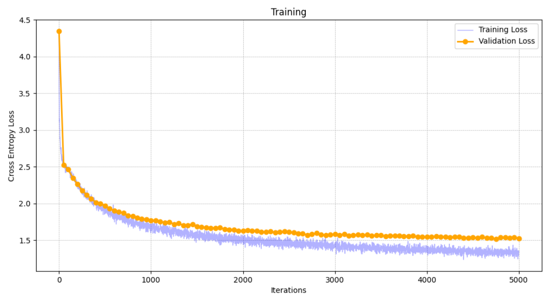


Figure 1: Heatmap of positional encoding

### 4.2 Loss Convergence

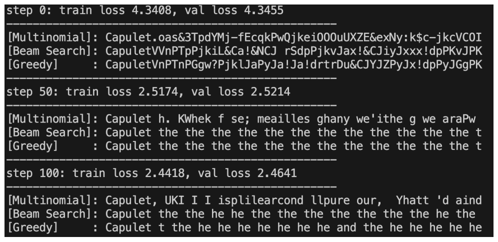
We monitored the Cross-Entropy Loss at every iteration for training and at regular intervals for validation. The training process showed good performance before overfitting.



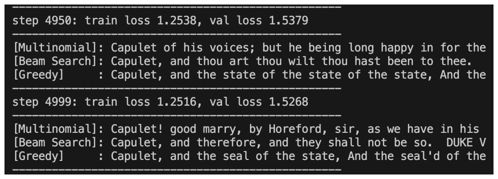
### 4.3 Decoding Strategies Analysis

Three decoding strategies.

- **Multinomial Sampling:** Generates from different probabilities, choosing that probability as the probability of predicted output.
- **Greedy Decoding:** Selects the most probable token at each step.
- **Beam Search ($k = 3$):** Explores multiple paths in a tree to find the most likely sequence of k paths.



```
step 0: train loss 4.3408, val loss 4.3455
--------------------------------------------------
[Multinomial]: Capulet.oas&3TpdYMj-fEcqkPwQjkeiOOOuUXZE&exNy:k$c-jkcVCOI
[Beam Search]: CapuletVVnPTpPjkiL&Ca!&NCJ rSdpPjkvJax!&CJiyJxxx!dpPKvJPK
[Greedy]     : CapuletVnPTnPGgw?PjklJaPyJa!Ja!drtrDu&CJYJZPyJx!dpPyJGgPK
--------------------------------------------------
step 50: train loss 2.5174, val loss 2.5214
--------------------------------------------------
[Multinomial]: Capulet h. KWhek f se; meailles ghany we'ithe g we araPw
[Beam Search]: Capulet the the the the the the the the the the the the t
[Greedy]     : Capulet the the the the the the the the the the the the t
--------------------------------------------------
step 100: train loss 2.4418, val loss 2.4641
--------------------------------------------------
[Multinomial]: Capulet, UKI I I isplilearcond llpure our,  Yhatt 'd aind
[Beam Search]: Capulet the the he he the the the the the the the he the
[Greedy]     : Capulet t the he he he he he he he and the he he he he he
```



```
--------------------------------------------------
step 4950: train loss 1.2538, val loss 1.5379
--------------------------------------------------
[Multinomial]: Capulet of his voices; but he being long happy in for the
[Beam Search]: Capulet, and thou art thou wilt thou hast been to thee.
[Greedy]     : Capulet, and the state of the state of the state, And the
--------------------------------------------------
step 4999: train loss 1.2516, val loss 1.5268
--------------------------------------------------
[Multinomial]: Capulet! good marry, by Horeford, sir, as we have in his
[Beam Search]: Capulet, and therefore, and they shall not be so.  DUKE V
[Greedy]     : Capulet, and the seal of the state, And the seal'd of the
--------------------------------------------------
```

## 5 Discussion

### 5.1 The Importance of Scaling ($\sqrt{d_k}$)

**Why divide by $\sqrt{d_k}$?** The dot product $q \cdot k$ tends to grow in magnitude as the dimension $d_k$ increases. If the components are independent random variables with variance 1, the dot product has variance $d_k$. Without scaling, the logits entering the Softmax function would be large. The Softmax function is extremely sensitive to large inputs; it pushes probabilities towards 0 or 1, landing in regions where the gradient is effectively zero (vanishing gradients). The term $\frac{1}{\sqrt{d_k}}$ normalizes the variance to 1, ensuring stable gradients during backpropagation.

### 5.2 Additive vs. Concatenated Positional Encodings

**Why add instead of concatenate?** We add the positional encodings to the token embeddings ($x = x_{tok} + x_{pos}$) to preserve the dimensionality of the model. Concatenation would increase the input dimension, requiring larger weight matrices in subsequent layers, thus increasing parameter count and computational cost. Mathematically, in high-dimensional spaces ($d_{model} = 384$), the model can easily learn to separate the semantic information (token) from the syntactic information (position) within the same vector space, effectively treating them as orthogonal signals.

### 5.3 Computational Complexity Analysis

**The cost of Self-Attention.** The time complexity of the Self-Attention mechanism is $O(T^2 \cdot d)$, where $T$ is the sequence length and $d$ is the embedding dimension. This quadratic cost arises from the matrix multiplication $QK^T$, which produces a $(T \times T)$ attention matrix.

**Implication:** This presents a significant bottleneck for very long texts. Doubling the context length quadruples the memory and compute requirements, making standard Transformers inefficient for tasks requiring processing entire books or long DNA sequences without modifications.

## 6 Conclusion

We successfully implemented a fully functional Mini-GPT from scratch. The experiments confirm that the Transformer's power lies in its ability to route information via attention. the implementation of Beam Search proved crucial for generating coherent text, overcoming the limitations of greedy decoding. The analysis highlights that while the architecture is powerful, it requires careful stabilization and suffers from quadratic complexity with respect to sequence length.

## References

1. Vaswani, A., et al. (2017). Attention is All You Need.
2. Karpathy, A. (2023). Let's build GPT: from scratch, in code, spelled out. *YouTube*.