Homework 6 COMPSCI 689

Hritwik Goklaney (hgoklaney@umass.edu)

1 Introduction

Large language models are incredibly capable, but their massive power comes with a price tag. They are simply too expensive for most hardware. This report shifts the focus onto smaller language models and asks: how can we make the most of them when computing resources are limited?

To answer that, we look at a variety of model designs. We start with very simple approaches like a Linear Predictor and a basic Multi-layer Perceptron (MLP). From there, we move on to more advanced ideas such as Multi-head Self-attention models and full Multi-layer Transformers.

By testing all of these on both character-level and word-level datasets, we can see which models give us the best performance for their size and thereby, helping us find the most efficient way forward when working with limited compute.

2 Experimental Setup

2.1 Datasets and Tokenization

Deliverable 1 - Character-level dataset In this deliverable, Tiny Shakespeare with character-level tokenization has been used to predict the next character. The dataset was preprocessed by deriving a unique character vocabulary from the raw text in the dataset and mapping each character to an integer index as follows:

$$V = \{c \mid c \in \text{Dataset}\} \tag{1}$$

where V represents the set of all unique characters. To facilitate numerical processing, we defined the following mapping functions:

$$\text{string to integer mapping (stoi)} : \text{char} \to \{0, \ldots, |V| - 1\} \tag{2}$$

$$\text{integer to string (itos)} : \{0, \ldots, |V| - 1\} \to \text{char} \tag{3}$$

The entire corpus is encoded as a single tensor of token IDs.

Deliverable 2 - Word-level datasets In this deliverable, PTB and WikiText-2 word-level datasets are used with tokenization logic as follows:

1. Read the dataset files as raw strings.
2. Split each string on whitespace to obtain token lists.
3. Build a word vocabulary from the union of train, valid and test tokens.

$$\text{vocab} = \{\text{unique tokens (words split on whitespace)}\}.$$

4. Encode each split as a 1D tensor of word IDs and build maps to convert from integer to string and vice versa.

2.2 Model Architectures

In this project, the following four architectures have been built to predict the next token.

- **Linear Predictor** – a single linear layer (using softmax) as predictor
- **Multi-layer Perceptron (MLP)** – a multi-layer perceptron with at least three layers, with nonlinear activations.
- **Multi-head self-attention model** – multi-head self-attention model with one or more layers with configurable number of heads
- **Multi-layer Transformer** – a small transformer model with multiple blocks.

All the above four architectures, use an embedding layer to map token IDs to vectors and train with a next-token objective. For Deliverable 1, I experimented with following hyperparameters:

- **LinearPredictorLM**: context length (16, 32 and 64), learning rate (0.001, 0.005, 0.01) , batch size (64, 128 and 256).
- **MLP_LM**: context length (16, 32 and 64)
- **SelfAttentionLM**: number of heads (2, 4 and 8)
- **TransformerLM**: number of heads (2, 4 and 8)

The best hyperparam setting on the character-level task for each architecture is as follows:

Linear Predictor

$$\text{Context Length (block size)} = 16, \quad \text{batch size} = 64,$$
$$\text{embedding dim} = \text{"65"}, \quad \text{learning rate} = 10^{-3}, \quad \text{optimizer} = \textit{Adam},$$

MLP

$$\text{Context Length (block size)} = 16, \quad \text{batch size} = 128, \quad \text{optimizer} = \textit{Adam},$$
$$\text{embedding dim} = 128, \quad \text{hidden dimension} = [64, 64], \quad \text{learning rate} = 10^{-3},$$

Multi-head self-attention model

$$\text{Context Length (block size)} = 16, \quad \text{batch size} = 64, \quad \text{embedding dim} = 64,$$
$$\text{number of heads} = 8, \quad \text{number of layers} = 2, \quad \text{learning rate} = 10^{-3}, \quad \text{optimizer} = \textit{Adam},$$

Transformer

$$\text{Context Length (block size)} = 16, \quad \text{batch size} = 64, \quad \text{embedding dim} = 64, \quad \text{optimizer} = \textit{Adam},$$
$$\text{hidden dim} = 128, \quad \text{number of heads} = 8, \quad \text{number of layers} = 2, \quad \text{learning rate} = 10^{-3},$$

Overall best architecture - Transformer

2.3 Training Procedure

All models are trained with cross-entropy loss on the last token in the context:

$$\mathcal{L} = -\frac{1}{B}\sum_{b=1}^{B} \log p_\theta(y_b \mid x_b)$$

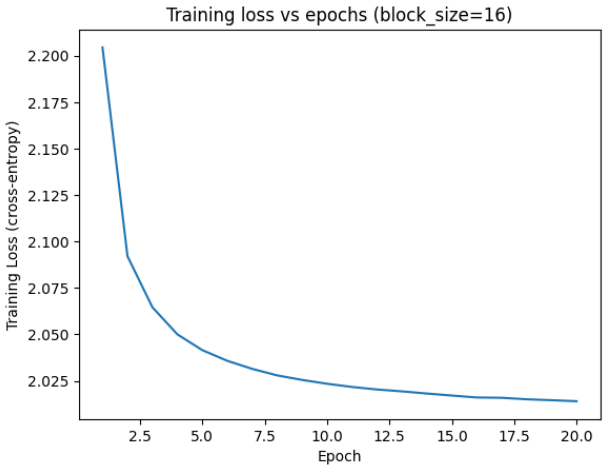
2.4 FLOP Estimation

To relate performance to compute, I use PyTorch's built-in flop counter: FlopCounterMode (using get_total_flops()) to get the flop estimation for forward and backward calls during training.

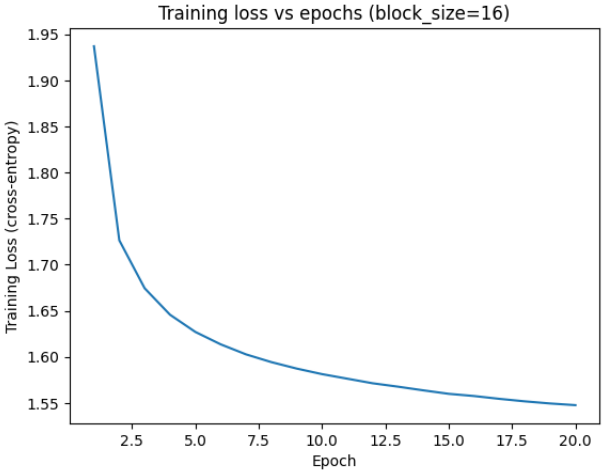
3 Results: (Deliverable 1)

3.1 Training loss vs epochs plots

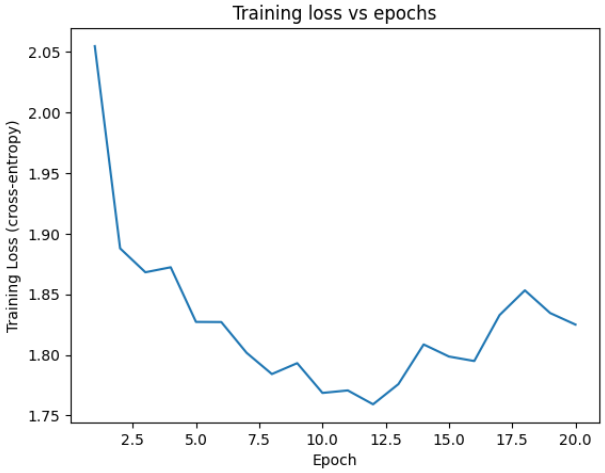
Linear Predictor



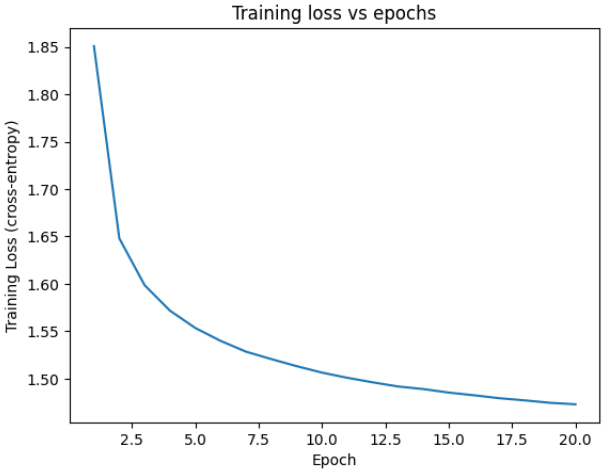
MLP



Multi-head Self-attention



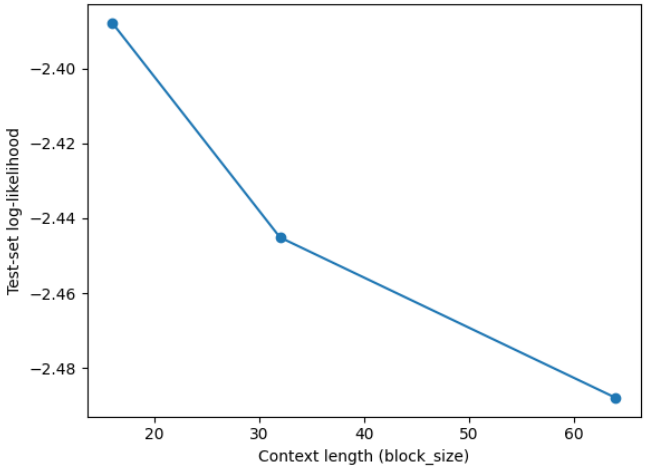
Transformer



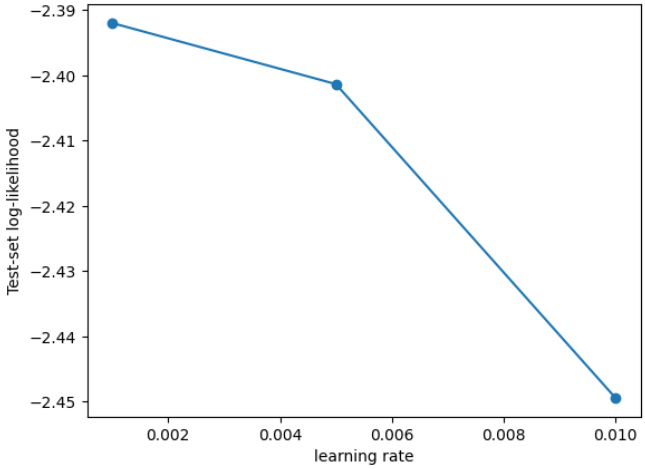
3.2 Test Log-Likelihood vs. Hyperparameters

For each architecture, I experimented with following hyperparameters and plot test-set log-likelihood vs. that hyperparameter.

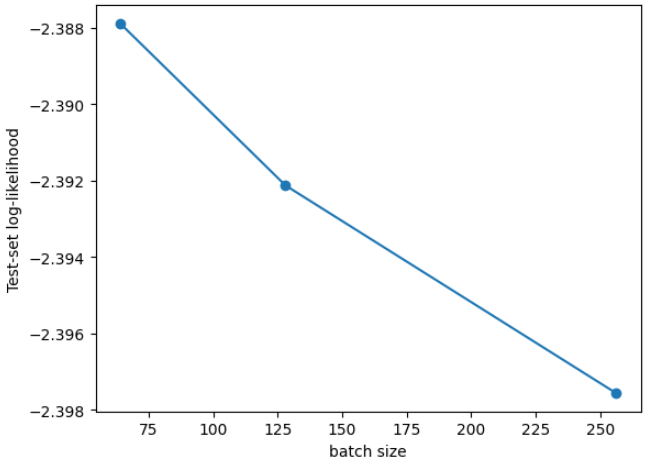
Linear Predictor (Context Length)



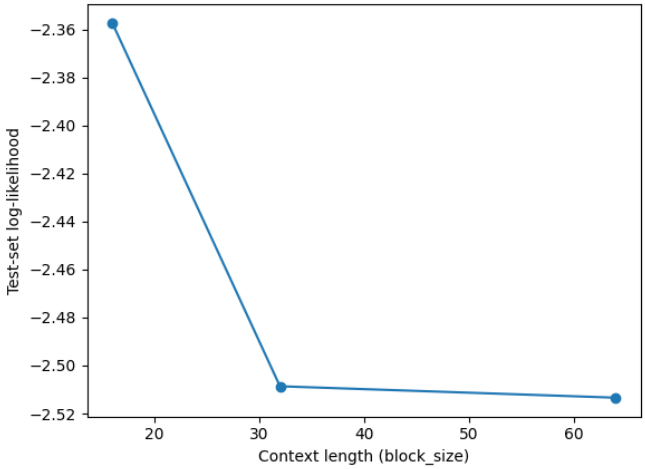
Linear Predictor (Learning Rate)



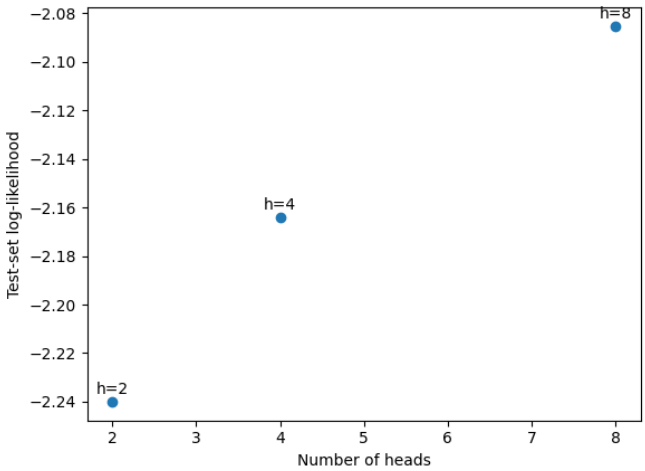
Linear Predictor (Batch size)



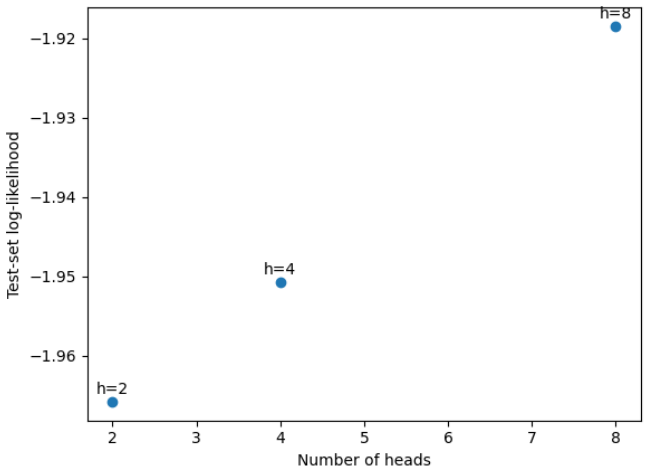
MLP(Context Length)



Multi-head Self Attention (No. of heads)

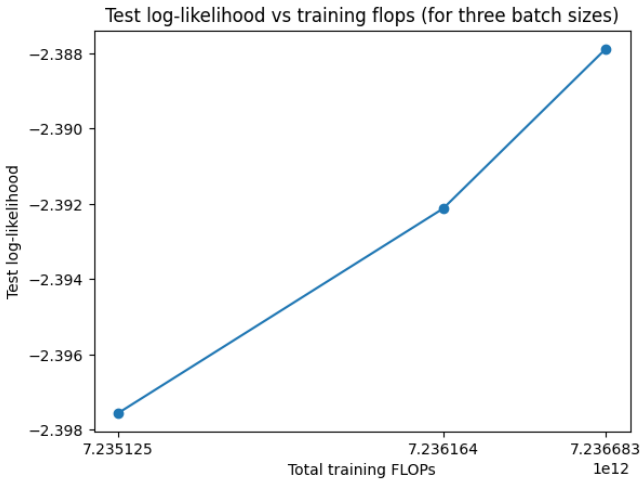


Transformer (No. of heads)

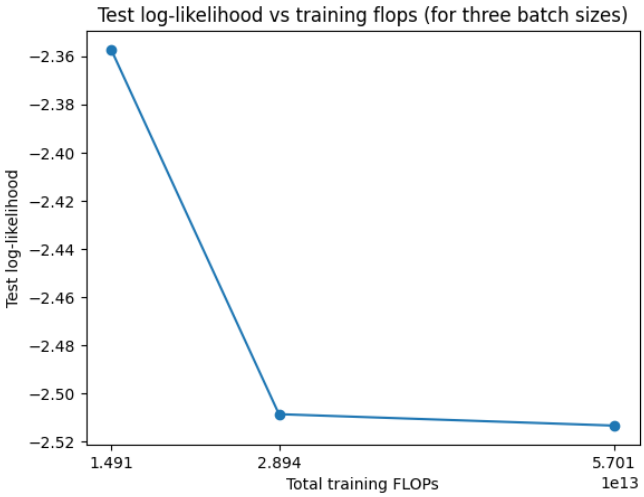


3.3 Test Log-Likelihood vs. Training FLOPs

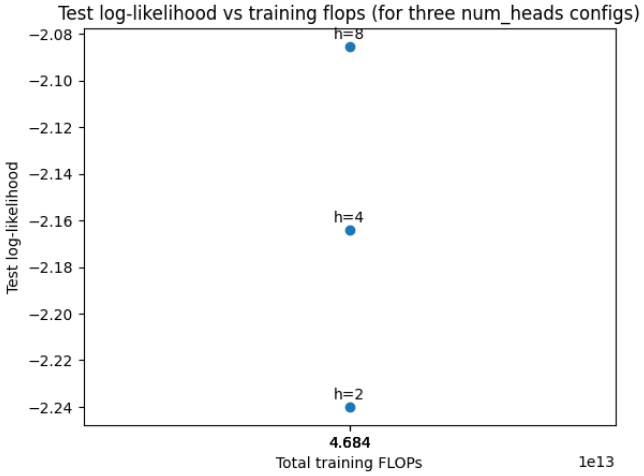
Linear Predictor



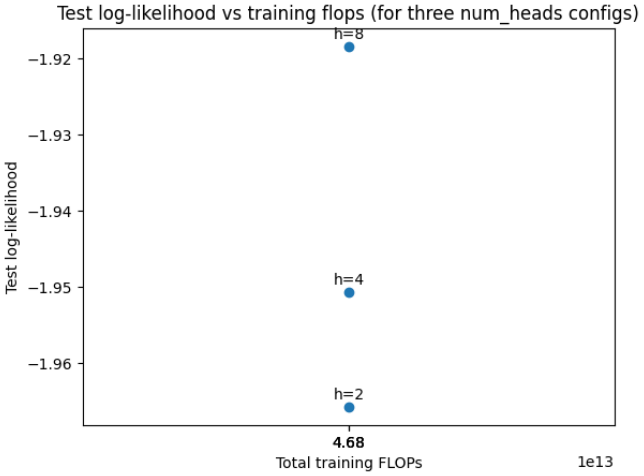
MLP



Multi-head Self Attention



Transformer



3.4 Generated Samples

Linear Predictor

```
HAMLET:ar hit loull
And dois pentom:
Bof oalcemby
me ine by Yeak.
You wht be shall, As Liok say hresten nou
```

MLP

```
HAMLET:AY:
He plant thousand uncurrland.

CAMILLO:
Nevel?

WARWICK:
It one joints wine
my eneedmen take,
Or
```

Multi-layer self attention

```
HAMLET:
He it hall, our speins?

FRITER; I with buls, anting, wease sheom's is I'll pear so, givessectent.
```

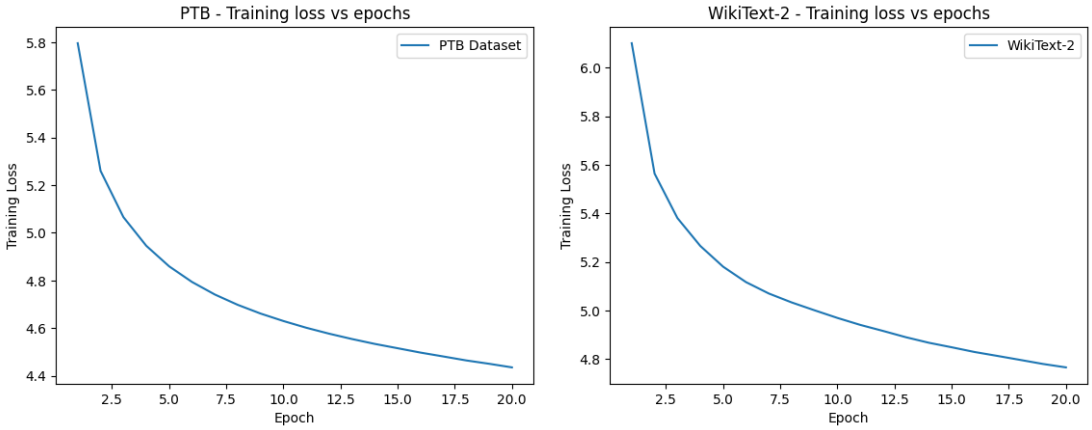
Transformer

```
HAMLET:
Anl incheea face.
Do not nece the baid to please me. They who stays of thinning: net pardown to, yo
```

4 Results: Word-Level PTB and WikiText-2 (Deliverable 2)

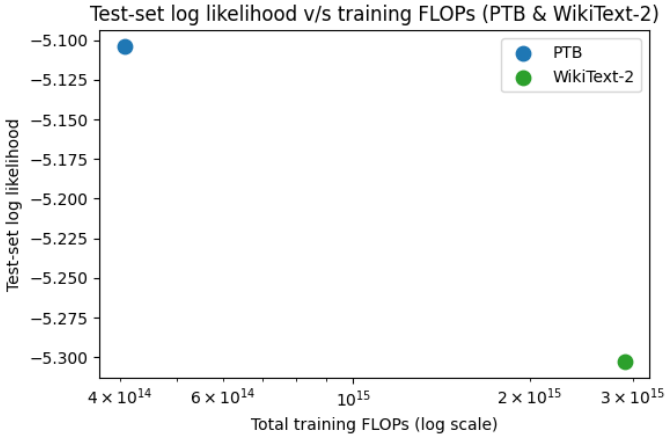
For Deliverable 2, I implemented the best transformer architecture from the *character-level* experiments and applied it to PTB and WikiText-2, now at *word* level.

4.1 Training loss vs epochs plots



4.2 Test Log-Likelihood vs. Training FLOPs

At the end of training, plotted the test-set log likelihood v/s total training flops used for both the dataset. To visualize both, I used a simple scatter plot:



4.3 Word-Level Generated Samples

PTB sample

> the school announced that analysts were with the week ended friday 's dealers said the offer added from about N billion lire $ N this year and were up N N from the $ N million loss <unk> on its carpet quality schedule are high instead of a truck maker the steel and engines products lost N to N the steel hit a <unk> call television <unk> aircraft n.v. had net income of $ N billion up N N from $ N million or N cents a share or about $ N million or $ N a share changed from $ N million in

WikiText-2 sample

> The history of machine learning begins with the children being done . In 1935 , Valve began working during the first aided by Madrid biographer Benoît Assessment . He claimed that Patterson had a site , and through the corner was short until a failed east . She was asked to Commander Blaine with a New York by Arsenal to play the ship of <unk> (" sink " seat " All Good Things Hale and an area work in the New York corruption , which included a mentioned by Richardson as an Nobel Prize for George Quiney 's alleged pattern of ; in his 2009

5 Discussion

5.1 Compute vs. Performance

Across all experiments, I observed a clear trade-off between training compute (FLOPs) and test-set log-likelihood:

- Moving from the linear model to the MLP significantly improves test log-likelihood with a moderate increase in FLOPs.
- Adding self-attention and full transformer blocks improves performance further, but with a much larger compute cost.

5.2 Effect of Context Length and Architecture

For the linear and MLP models, increasing the context length (`block_size`) generally improves performance up to a point, but the benefit is limited by the models' inability to effectively model long-range dependencies and lead to poor performance on test dataset due to overfitting. Additionally, longer context also increases compute for attention, so choosing a modest `block_size` (e.g., 16) was a good compromise for this small-scale setting.

5.3 Scaling from Characters to Words

When scaling the best transformer architecture from characters to words, several observations were made:

- Vocabulary size increases dramatically from characters to words, which affects both embedding and output layers.
- The learned models capture some dataset-specific patterns (for instance, financial information from PTB dataset) but text is still far from human quality, reflecting that the models are small and trained for a limited number of epochs.

5.4 Creative/Unsuccessful Ideas Explored

During experimentation I tried:

- Larger MLP hidden dimensions beyond 128, which increased FLOPs substantially but provided limited gains.
- More transformer layers did lead to slight improvements in performance however, due to limited training time and compute, kept the number of layers to 2.
- Higher learning rates, which sometimes sped up early training but made validation loss unstable.

Even when these ideas did not improve final test log-likelihood, they were useful for understanding the sensitivity of small LMs to hyperparameters and capacity.

6 Summary

Key takeaways from this project include:

- **Architectural Influence on Performance:** Our experiments show that architectural configurations significantly dictate model efficacy even under strict resource constraints. Transformer-based architectures consistently achieve superior test log-likelihood compared to Linear and MLP baselines, provided the computational budget is sufficient to leverage their attention mechanisms.
- **Optimal Resource Allocation:** Choosing the right context length (how much text/tokens the model looks at once) and the right model depth (how many layers it has) makes a big difference in balancing prediction quality against computational cost (measured in FLOPs).
- **Tokenization and Text Quality:** Smaller Transformer models that use word-level tokenization can produce basic, coherent text on datasets like PTB and WikiText-2. But to generate high-quality, fluent text at scale still requires much larger models and more computation.

Overall, this project illustrates the core design trade-offs behind efficient small language models and connects them to the broader compute vs. performance trends seen in larger-scale LMs.