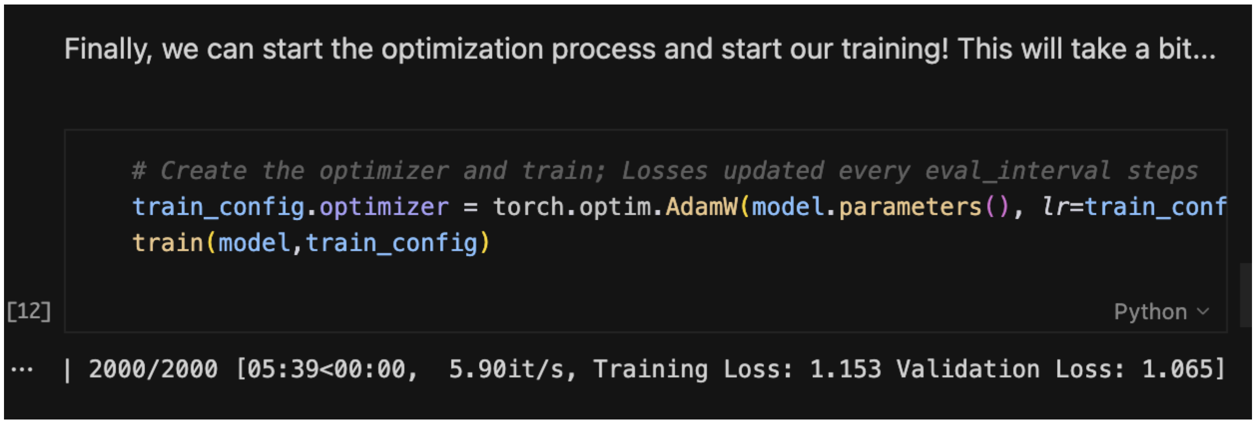
Part 3B: Post-Lab Write-Up

(a) Model description and parameter justification:
The model is a decoder-only Transformer (GPT-style) for autoregressive next-token prediction on the Shakespeare dataset. It consists of token embeddings, sinusoidal positional encoding, six transformer blocks (each with multi-head self-attention with causal masking and a feed-forward network with LayerNorm and residual connections), and a final linear layer that maps to the vocabulary size for logits. I used the default configuration from the notebook: context length 256 (so the model sees up to 256 tokens at a time when predicting the next token), 6 layers and 6 attention heads with embedding dimension 384, giving an attention dimension of 64 per head. These choices follow a small "nano" GPT setup that is manageable to train in a few minutes while still illustrating the full architecture. I kept dropout at 0.2 to reduce overfitting on the limited data, batch size 64 and 2000 training iterations with learning rate 1e-3 and gradient clipping at 1.0 to keep training stable without tuning extensively.

(b) Model evaluation and evidence:
Training and validation loss decreased over the 2000 iterations, indicating that the model learned to predict tokens better than random. The generated samples show that the model captured the format of the dataset: it outputs speaker names such as ROMEO, followed by lines of dialogue, and uses archaic or Shakespeare-like vocabulary, such as "thou," "say'st," and "garland," "councils". The actual sentences are often incoherent or ungrammatical, like "Say thou say'st me? These is the with me.". This is expected for a small model trained for a short time: it learns local patterns and structure, like format and word style, but not long-range coherence or grammar. The observed behavior is consistent with a correctly implemented transformer trained under these constraints.

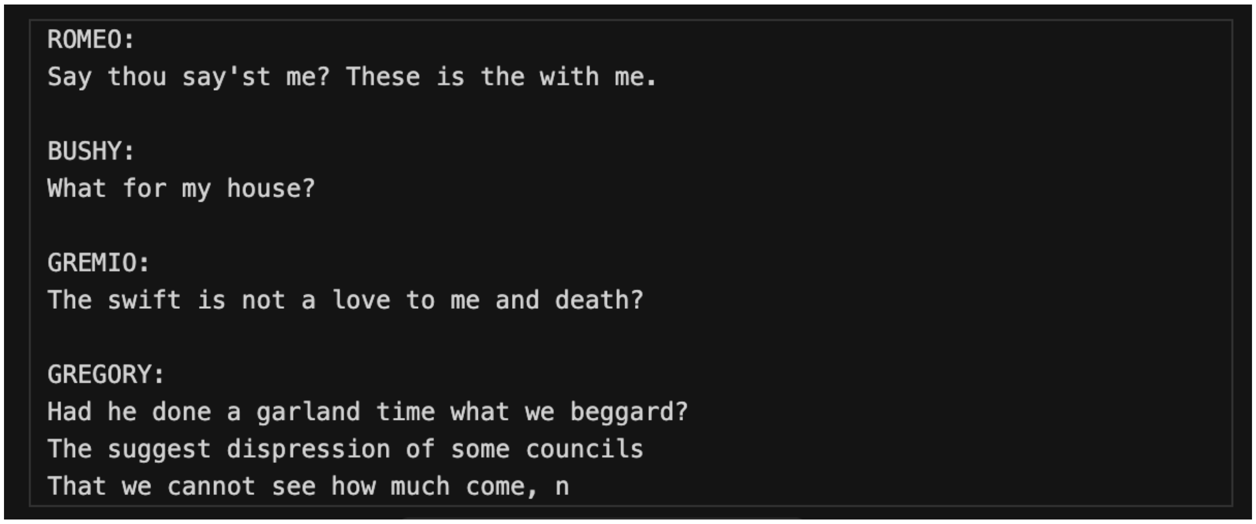
Training Error Log:


Finally, we can start the optimization process and start our training! This will take a bit...

```
# Create the optimizer and train; Losses updated every eval_interval steps
train_config.optimizer = torch.optim.AdamW(model.parameters(), lr=train_conf
train(model,train_config)
```

```
… | 2000/2000 [05:39<00:00,  5.90it/s, Training Loss: 1.153 Validation Loss: 1.065]
```

Output:



```
ROMEO:
Say thou say'st me? These is the with me.

BUSHY:
What for my house?

GREMIO:
The swift is not a love to me and death?

GREGORY:
Had he done a garland time what we beggard?
The suggest dispression of some councils
That we cannot see how much come, n
```

(c) Reflection:
Implementing the Transformer from scratch clarified how scaled dot-product attention, multi-head attention, and causal masking work together for autoregressive generation, and why positional encoding is necessary given that attention alone is permutation-invariant. Seeing the model produce formatted but mostly incoherent Shakespeare-style text reinforced the idea that architecture and implementation correctness come first, and that better text quality would require a larger model and more training.