Text Generation by implementing RNN, LSTM, and Transformer models using PyTorch

Abstract: This project implements three sequential deep learning models- RNN, LSTM, and Transformer for a text generation task using PyTorch. Each model was trained on a dataset of prompt-completion text pairs, tokenized using a SentencePiece BPE tokenizer. Each model performance was evaluated using perplexity and BLEU score metrics. The Transformer model outperformed the others, achieving the lowest test perplexity (51.10) and better generative diversity under temperature-controlled sampling. This report summarizes the methodology, model architectures, and evaluation outcomes.

Methodology: I designed, trained and evaluated three different architecture models. Before designing models, I preprocessed the dataset following below steps.

1. **Dataset and preprocessing:**

   The dataset consists of short text sequences derived from public domain literature available on Project Gutenberg. To prepare the dataset for training generative language models, I performed the following preprocessing steps:
   - I trained a Byte Pair Encoding (BPE) tokenizer using the SentencePiece library with a vocabulary size of 10,000.
   - The raw text corpus was tokenized into subword units to reduce out-of-vocabulary issues and improve model generalization.
   - Tokens for start (<bos>), end of sequence (</eos>), and padding (<pad>) were added to ensure consistent formatting during training and generation.
   - All text was segmented into fixed-length sequences with appropriate padding to enable batch processing.

2. **Training and Evaluation:**

To ensure all models were evaluated fairly, a consistent training approach was used throughout. Each model was optimized with the CrossEntropyLoss function, which is ideal for tasks involving multiple classification categories, such as language modeling. The training also utilized the AdamW optimizer along with a learning rate scheduler, which helped adjust the learning rate dynamically during the training process. A batch size of 128, and training for up to 30 epochs varies with each model. However, if the validation loss didn't improve over five consecutive epochs, early stopping was triggered to avoid overfitting depending on each model.

I evaluated model performance using two widely recognized metrics: Perplexity (PPL) and BLEU Score. Perplexity evaluates how accurately a model predicts word sequences; lower scores indicate better performance. The BLEU score compares the model-generated text with reference sentences using n-gram precision, offering a way to measure how close the model output is to human-written content. Higher BLEU scores suggest better alignment with the reference data. All models were tested on a separate test set that wasn't used during training to ensure models performance on unseen data. During this phase, each model generated text from predefined prompts using autoregressive decoding. The resulting sequences were then compared to actual reference texts to calculate the BLEU score.

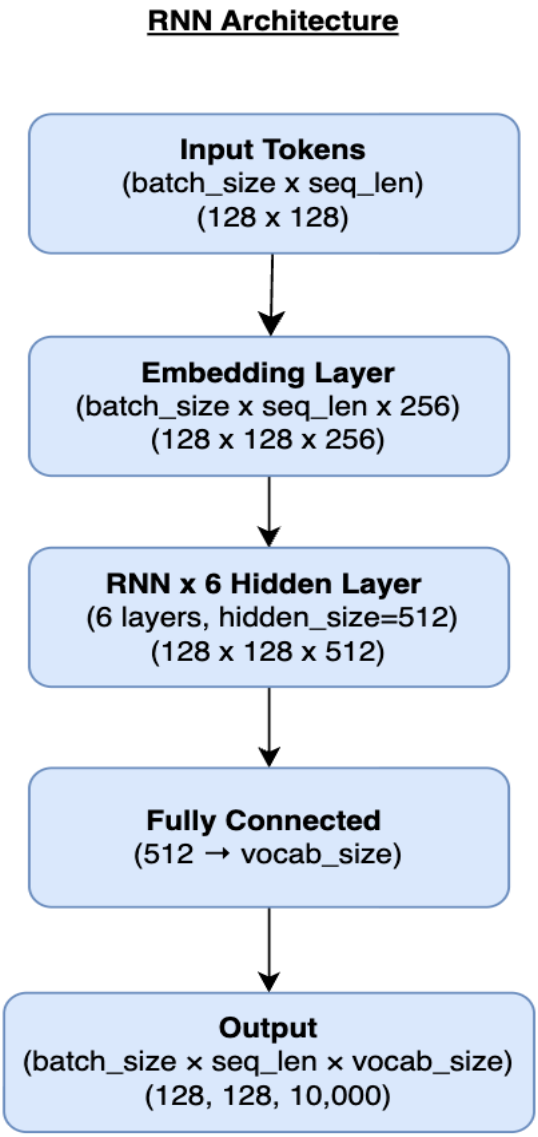
3. Model Architecture:

3.1 RNN

The Recurrent Neural Network (RNN) architecture handles sequences and predicting the next word in natural language text. The model takes in batches of tokenized text sequences shaped as (batch_size × sequence_length). In this RNN architecture, each batch has 128 sequences, and every sequence contains 128 tokens.

These token IDs first go through an embedding layer, which converts each token into a 256-dimensional dense vector. This transformation results in an embedded tensor with the shape (128 × 128 × 256). Next, the embedded sequences are passed through a RNN with six hidden layers, each with a hidden state size of 512. This part of the model is responsible for capturing the relationships between words across the sequence. It outputs a hidden state at every time step, producing a tensor shaped (128 × 128 × 512).

Finally, a fully connected linear layer maps each 512-dimensional hidden state to the model's vocabulary, which consists of 10,000 possible tokens. This gives a final output tensor of shape (128 × 128 × 10,000). These output values, called logits, are then used to calculate the training loss or to generate text one token at a time during inference.

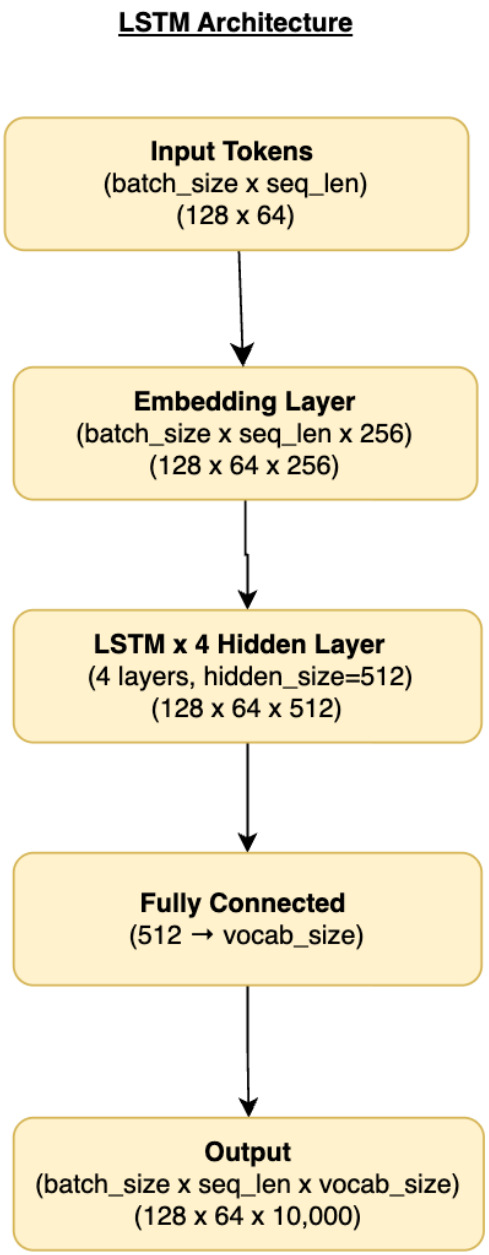


3.2. LSTM

The Long Short-Term Memory (LSTM) model is built to effectively learn patterns and relationships over time in sequential text data, making it especially useful for tasks like language modeling and text generation. The model takes in sequences of tokenized text shaped as (128 × 64), where 128 is the batch size and each sequence contains 64 tokens.

These tokens are first passed through an embedding layer that turns each token index into a dense vector with 256 dimensions, giving an output shape of (128 × 64 × 256). The resulting vectors are then processed by a 4-layer LSTM with a hidden size of 512. This setup helps the model learn more complex patterns and context across time steps.

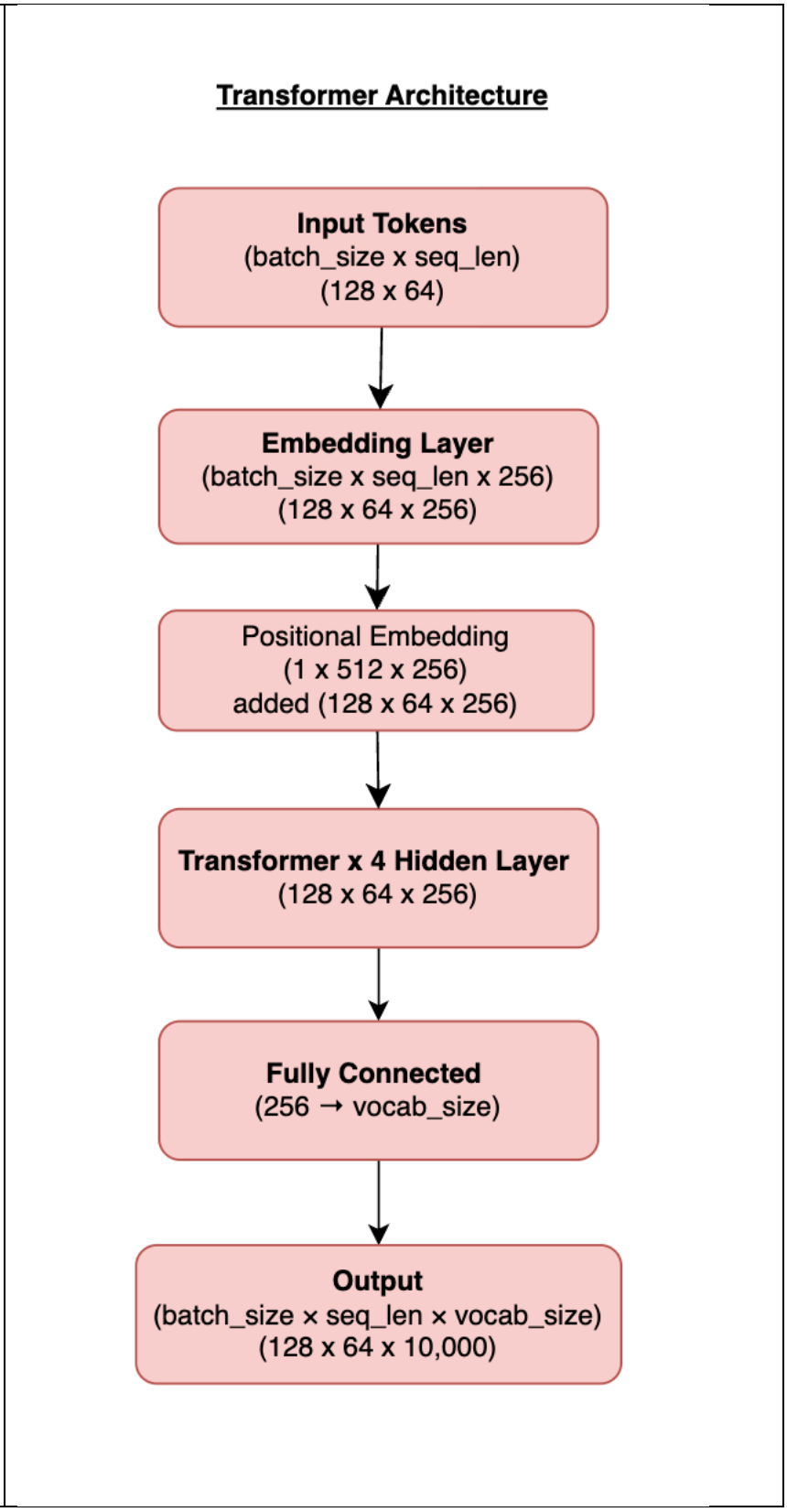
When generating text, the model uses temperature-controlled sampling to add variety and creativity to the output. After processing, the LSTM produces hidden states with the shape (128 × 64 × 512). These are then passed through a fully connected layer that maps each vector to the model's vocabulary, resulting in an output tensor shaped (128 × 64 × 10,000), which represents the probability distribution over possible next tokens.



3.3. Transformer

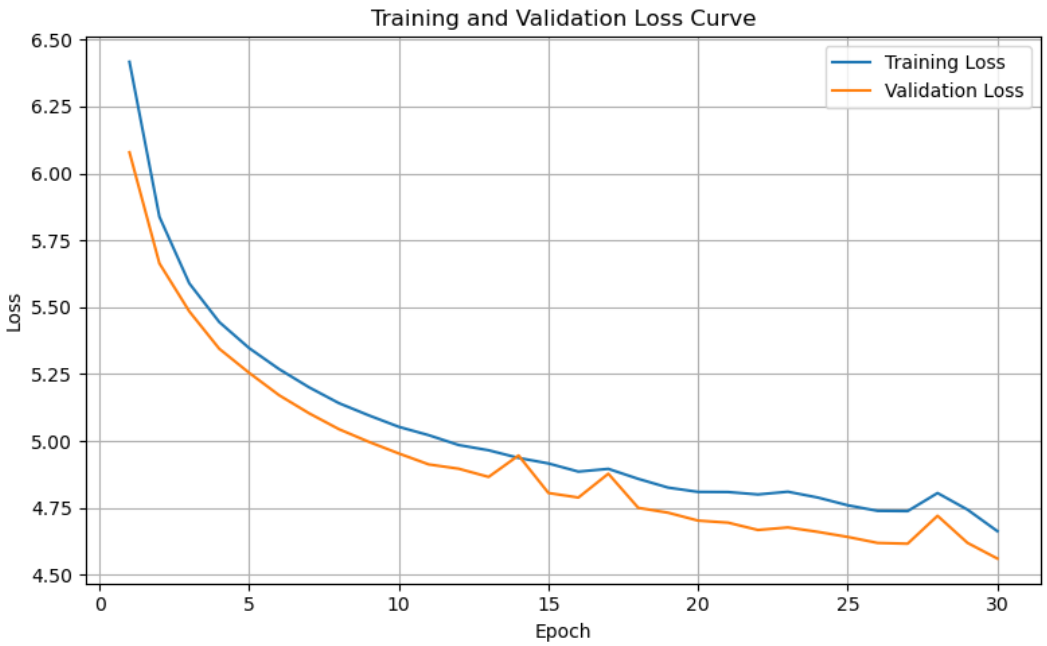
The Transformer architecture works efficiently for sequence modeling, which makes it a great fit for natural language generation tasks. It starts by taking input tokens shaped as (128 × 64), where 128 is the batch size and 64 is the length of each sequence. These tokens go through an embedding layer that converts them into dense vectors with 256 dimensions, resulting in a tensor shaped (128 × 64 × 256). Since Transformers don't process input sequentially like RNNs, they need a way to understand the order of tokens. To handle this, a learnable positional embedding of shape (1 × 512 × 256) is added to the token embeddings to preserve the sequence order. The combined embeddings are then passed through a stack of 4 Transformer encoder layers. Results are then passed through a fully connected layer that projects each vector onto the vocabulary of 10,000 tokens, producing the final logits with shape (128 × 64 × 10,000).

For text generation, the Transformer implements temperature-controlled sampling. This flexibility allows users to fine-tune the balance between coherence and creativity which makes Transformer both a powerful and adaptable tool for generating text.

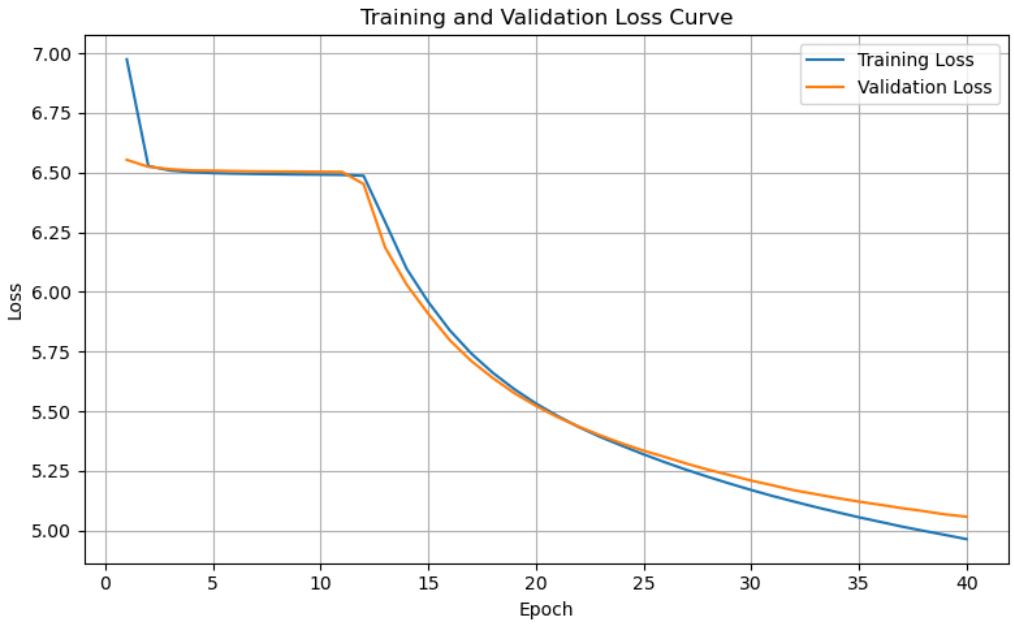


Results:

RNN training/Validation Loss Curve:

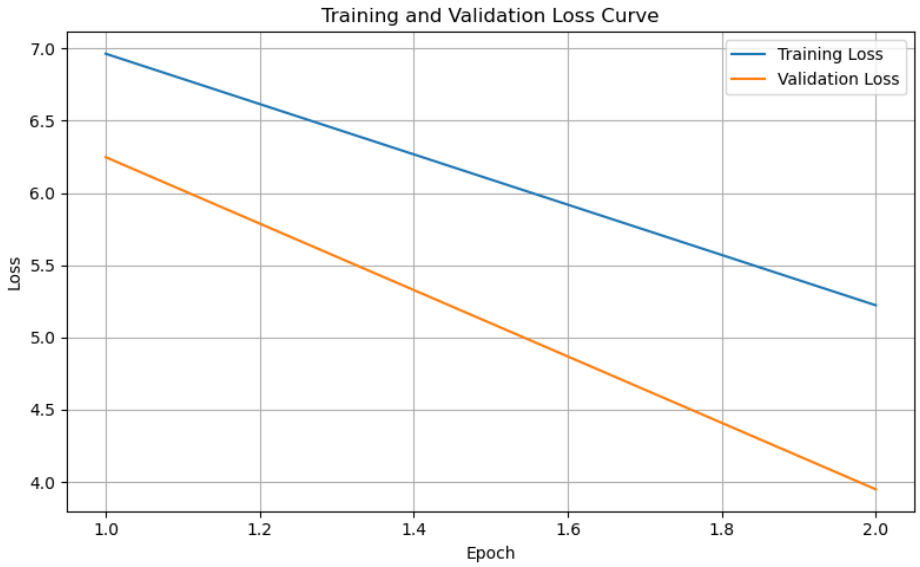


LSTM training/Validation Loss Curve:



Transformer training/Validation Loss Curve:

(I used epoch 2 for transformer model as performed well)



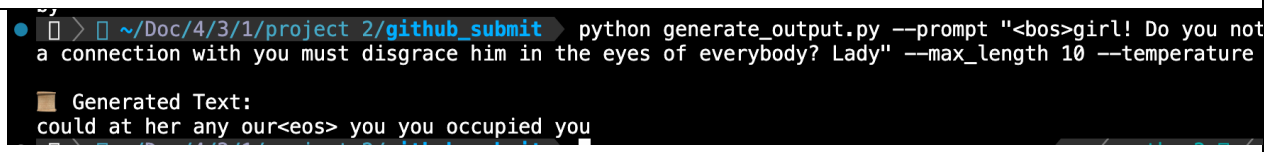
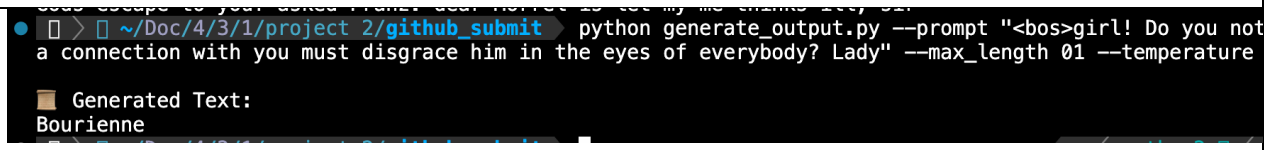
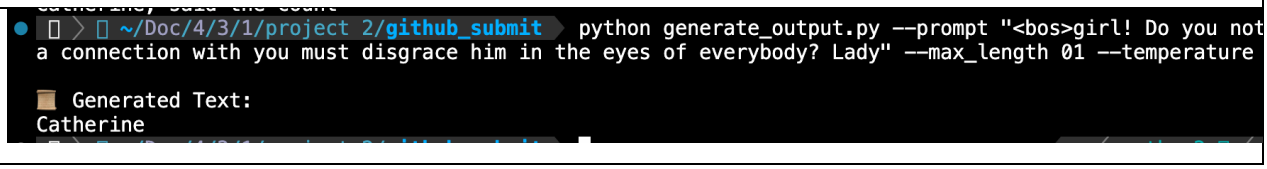
Evaluation metrics for each model on test dataset:

Model	Perplexity	BLEU score
RNN	131.88	0.0097
LSTM	156.28	0.0108
Transformer	51.10	0.0112

Each Model Response on following prompt: " Which do you prefer? Dogs or cats?"

Model	" Which do you prefer? Dogs or cats?"
RNN	~/Doc/4/3/1/project 2/github_submit python generate_output.py --prompt "Which do you prefer? --max_length 20 --temperature 1 Generated Text: house lofty3 gray skull only are have more had reserved yousties withdrawMr somethings income means
LSTM	~/Doc/4/3/1/project 2/github_submit python generate_output.py --prompt "Which do you prefer? --max_length 20 --temperature 1 Generated Text: Gods escape to you? asked Franz. dear Morrel is let my me thinks ill, sir
Transformer	~/Doc/4/3/1/project 2/github_submit python generate_output.py --prompt "Which do you prefer? --max_length 20 --temperature 1 Generated Text: I have been a man of mine. I have been a man of the world, and I will

Each Model Response on following prompt: " girl! Do you not consider that a connection with you must disgrace him in the eyes of everybody? Lady"

Model	" girl! Do you not consider that a connection with you must disgrace him in the eyes of everybody? Lady"
RNN	
LSTM	
Transformer	

Code Repository: GitHub Repo Link

Discussion & Conclusion: In this project, I implemented and evaluated RNN, LSTM, and Transformer-based language models to explore how well they learn for text generation tasks. The RNN struggled with understanding broader context and often produced repetitive and gibberish text. The LSTM performed better, showing lower perplexity as training advanced. However, the Transformer model outperformed, achieving the lowest test perplexity (~51.1). This highlighted how attention mechanisms excel at capturing long-range dependencies. Overall, the experience gave me a deeper insight into sequence modeling and the strengths and limitations of both recurrent and attention-based architectures.