StoryWeaverGPT

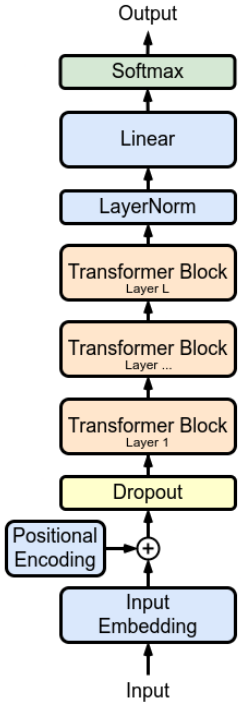
Final Model Evaluation, Results, Limitations and Potential Fixes

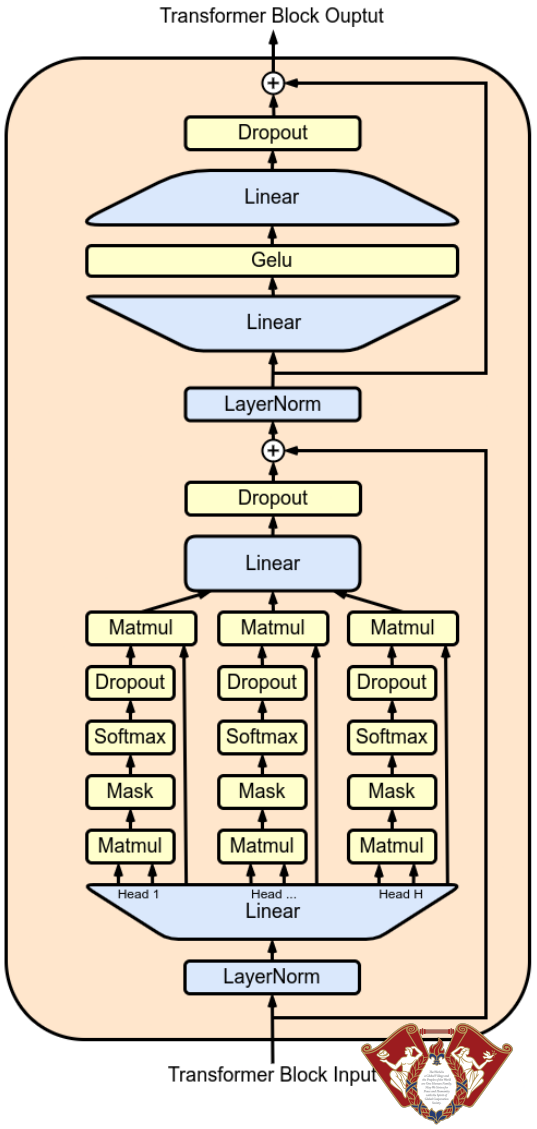
Group1

December 18, 2024

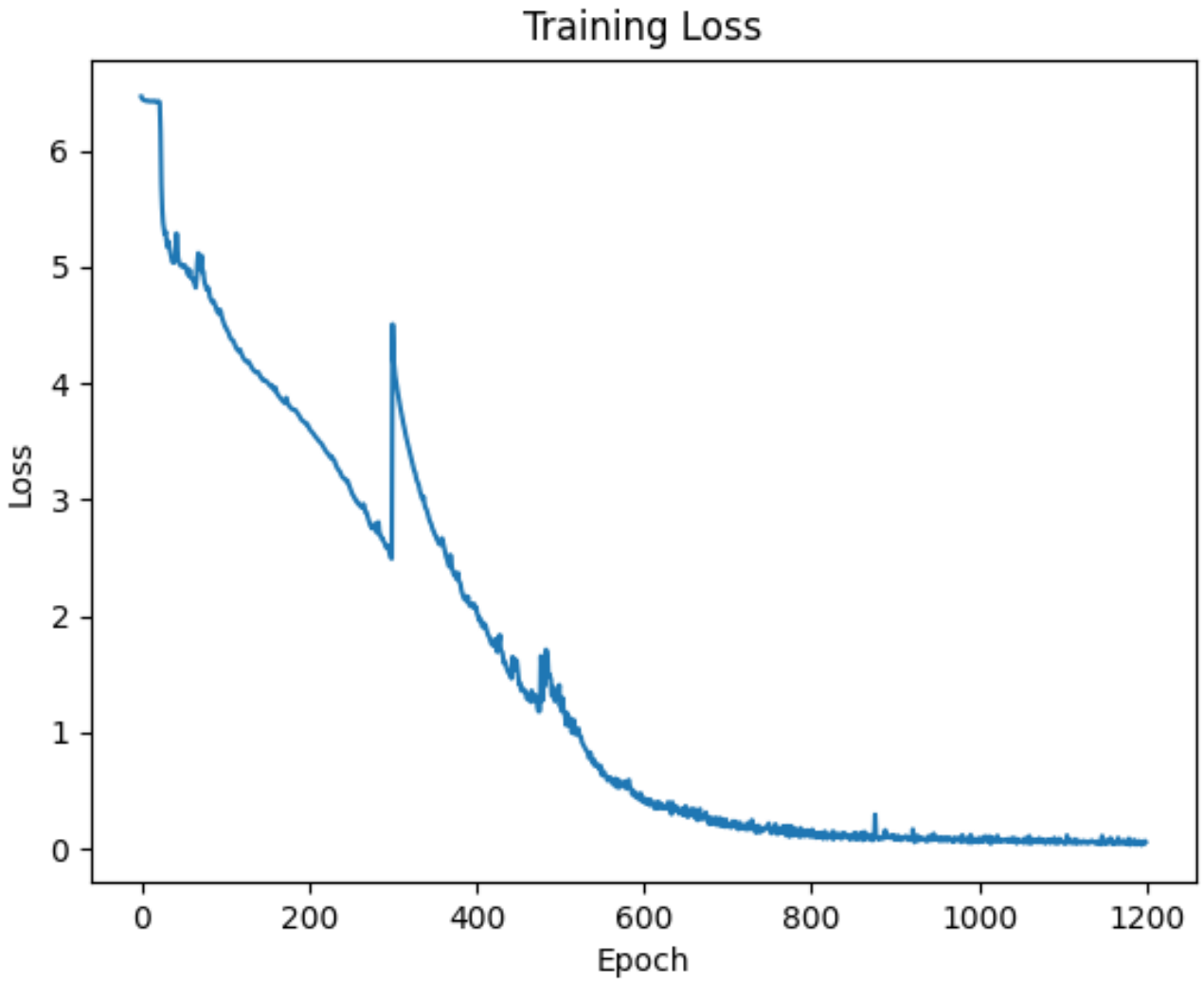
Review

- Trained on Shakesphere Dataset
- 1200 Epoch, learning rate 0.0001, no batch implemented
- Total Elapsed time: 28 hours





Loss



Sample Generation

```
Before we proceed any further, hear me speak.

All:
Speak, speak.

First Citizen:
You are all resolved rather to die than to famish?

All:
Resolved. resolved.

First Citizen:
First, you know Caius Marcius is chief enemy to the people.

All:
Say, let the p Cight she yours,
Paintsh:
Stant vauned.
He'll monounstet-gehroiaienciouancry 'gands
My E meananifedough the piADY, a pUM b!

P no pin and peemounsing hath su be tort not for the to my huse, toshed hard for dismone;
For li plue-ebalienirea
As to blet, plaaiall?
Did'd allons you waydoughteasone! I keep yewes, I are up the swif.

JULIET: have tocation now,p th
Finished splitting text into words in 2.5510787963867188e-05 seconds
```

Model Evaluation

- Main limitation were due to lack of diversity in dataset, and no batch implemented which lead to training inefficiency and overhead.
- While the model was able to generate text, it was not able to generate coherent text.
- Also, by performing most arithmetic and algorithms on python, it lacked efficiency.
- Attention mechanism implemented is not SOTA, and could be improved.
- Critically, the vocab dimension was way too big given the dataset size, which paired with lack of dataset gave weak results.

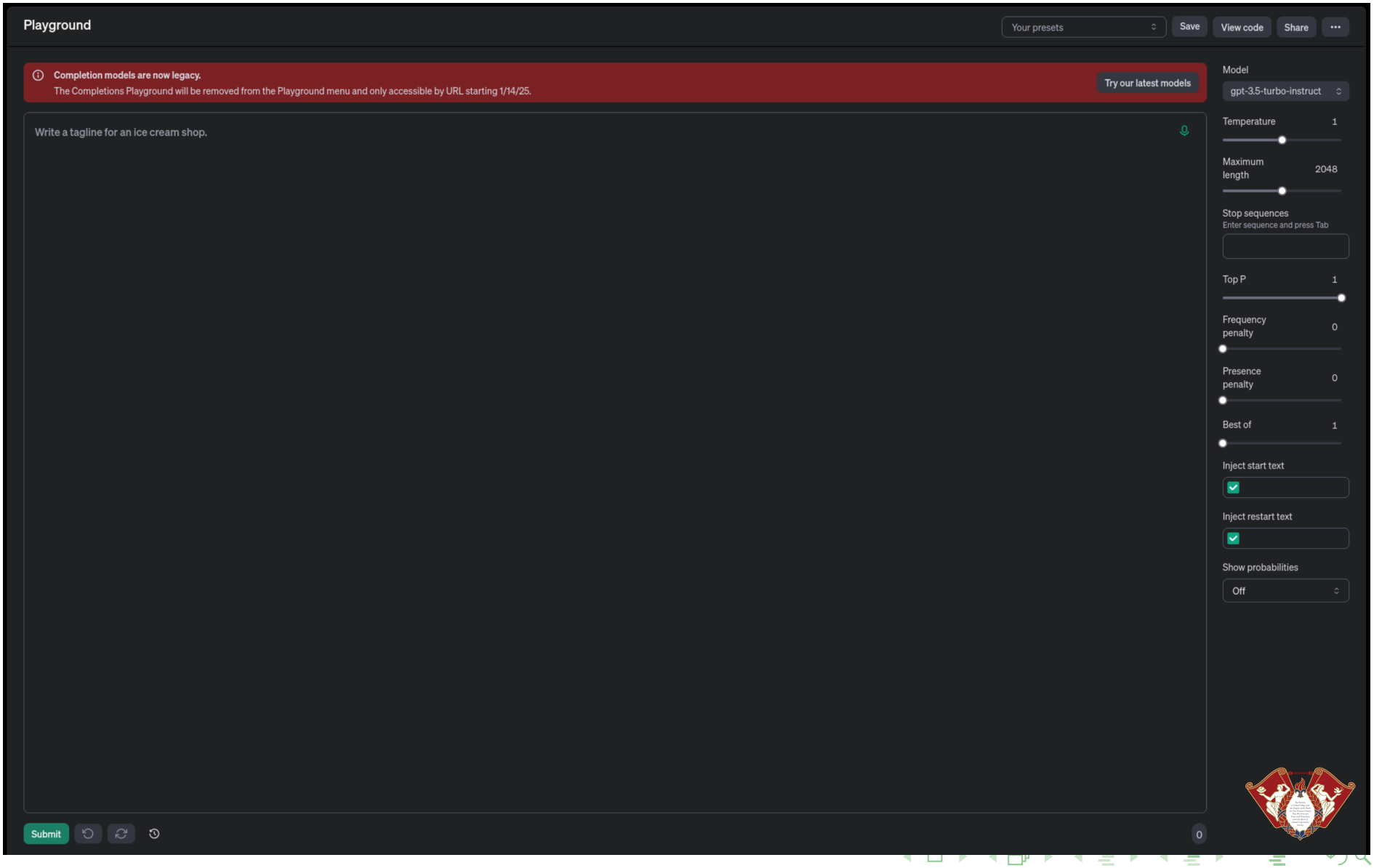
Potential Fixes

- Implement batch training
- Use more diverse dataset
- Implement SOTA attention mechanism
- ...And maybe not make everything from scratch, especially complex model such as transformer.

Hyperparameters

- Hyperparameters: Parameters that adjusts model generation.
- Currently implemented: Temperature, repetition penalty, stop sequence and max tokens.
- Temperature: Controls randomness of the model, implemented by dividing logits by temperature.
- Repetition penalty: Controls how much the model avoids repeating itself, implemented by $logit = \frac{logit}{1+repetationpenalty*frequencyvec}$.
- Max tokens: Controls the maximum number of tokens generated.
- Stop Sequence: Define a sequence of token that will stop the generation.

Hyperparameters

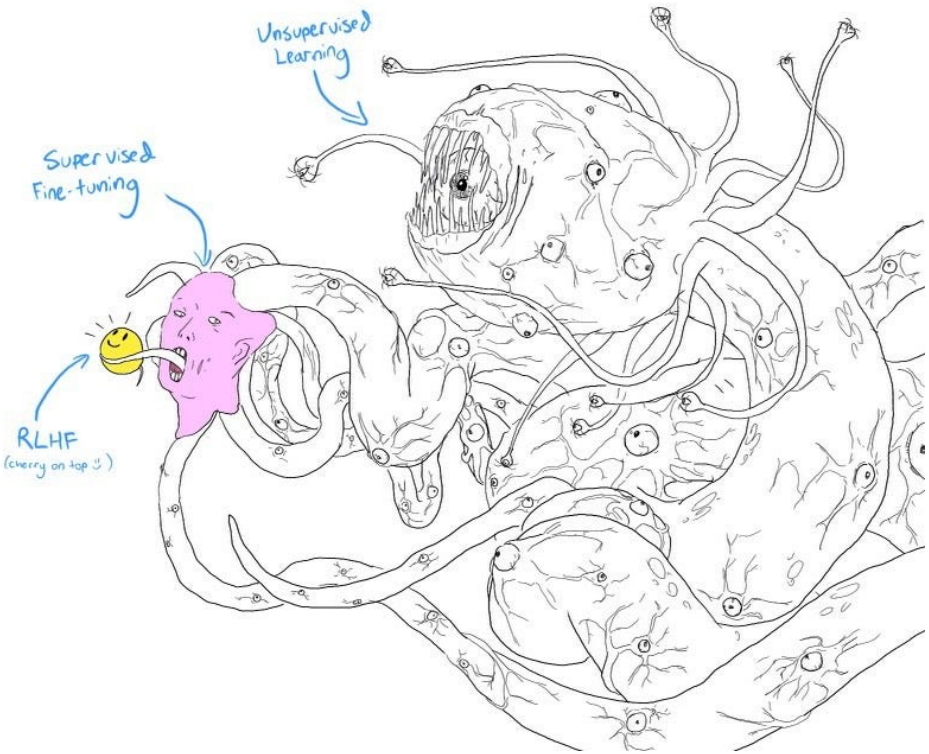


Text Generation and Hyperparameters Implementation

```python
def generate_sequence(self,
                      initial_input,
                      max_length,
                      temperature:float = 1.0,
                      frequency_penalty:float = 0,
                      stop_token:list[int] = None,
                      greedy:bool = False) -> Tensor:
    self.eval_mode()
    input_indices = initial_input.clone()
    stop_token_len = len(stop_token) if stop_token is not None else 0
    token_frequencies = torch.zeros(self.vocab_size, dtype=torch.float64)

    for _ in range(max_length - len(initial_input)):
        # Forward pass
        probs = self.forward(input_indices)
        # Get the last token's probability distribution
        next_token_probs = probs[-1] / temperature

        # Apply frequency penalty
        if frequency_penalty > 0:
            next_token_probs /= (1 + frequency_penalty * token_frequencies)

        next_token_probs = torch.softmax(next_token_probs, dim=-1)

        # Sample the next token
        if greedy:
            next_token = torch.argmax(next_token_probs)
        else:
            next_token = torch.multinomial(next_token_probs, 1).squeeze()
        # Update token frequencies
        token_frequencies[next_token] += 1

        # Append the next token to the input sequence
        input_indices = torch.cat((input_indices, next_token.unsqueeze(0)), dim=0)
        # If input_indices length exceeds max_seq_len, truncate and continue
        if len(input_indices) > self.max_seq_len:
            input_indices = input_indices[-self.max_seq_len:]
        # If stop token is generated, break
        if stop_token is not None and input_indices[-stop_token_len:] == stop_token:
            break

    return input_indices
```

Fine-Tuning

- Fine tuning is a method to teach model to generate text in a specific style.
- On GPT Models, it is done by training the model on a specific fine-tuning dataset.
- On Shakesphere, it could be implemented by two speakers conversing, one being query and the other being response.



Conclusion

- The model was able to generate text, but not coherent text.
- The model was not efficient, and lacked diversity in dataset.
- The model could be improved by implementing batch training, using more diverse dataset, implementing SOTA attention mechanism, and reducing vocab dimension.
- Hyperparameters such as temperature, repetition penalty, stop sequence and max tokens were implemented.
- Fine-tuning could be implemented to teach the model to generate text in a specific style.

Thank You!