DSA4212 Group Project: Training Dynamics of a Character-level LLM

Chin Zhe Ning
A0255895J
E0960565@u.nus.edu

Hee Wei Ting
A0251951E
E0957539@u.nus.edu

Abner Philip Then Yi Hao
A0265863R
E1070545@u.nus.edu

I. Introduction

In this report, we explore different deep learning architectures and their effectiveness in a character-level language modeling task. The primary objective is to predict the next character in a sequence, focusing on the accuracy of predicting the last character.

Through systematic hyperparameter tuning, architecture modifications and optimisation strategies, we aim to enhance model performance while analysing the trade-offs between computational efficiency and predictive accuracy. We compare various transformer configurations including different positional encoding schemes and regularisation techniques, while also trying out the LSTM model to provide insights into the effectiveness of different approaches for the sequence prediction task.

To evaluate these different models, we use the text8 dataset, consisting of the first 100 million characters of a cleaned and preprocessed English Wikipedia dump, containing only lowercase letters & spaces (27 unique characters). Our training set is 90MB, while our test set is 5MB, providing a 90 : 5 split of the dataset for our model training. Our goal is next-character prediction: given a sequence of L characters, predicting the next character's probability distribution. For this, the standard choice is to use the cross-entropy loss, and our optimisation objective is to use adaptive methods like Adam to minimise the cross-entropy loss, allocating a budget of 30 minutes of TPU v5e1 compute time on Google Colab for the final training run.

II. Transformer

The transformer architecture was proposed by eight scientists at Google in the paper "Attention is All You Need" [VSP+17], and it relies on attention mechanisms rather than recurrence. The stacked encoder and decoder layers process entire sequences in parallel rather than sequentially.

The encoder consists of a multi-head self-attention mechanism, which allows the model to process an entire sequence of characters in parallel, computing attention weights simultaneously with the following equation:

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

where Q, K & V are matrices that projects each character's input embedding into the query, key & value vector space respectively, while d_k is the dimension of the keys. Then, a position-wise Feed-Forward Network is applied independently to each position using the same parameters.

The decoder contains a Masked Multi-Head Self-Attention which works similarly to the

self-attention in the encoder, however it prevents positions from attending to subsequent positions, preserving the auto-regressive property of the generator. Next, Encoder-Decoder Attention enables the decoder to focus on relevant output from the encoder, and a Position-wise Feed-Forward Network works similarly to that in the encoder.

The supporting elements of the transformer model include Residual Connection & Layer Normalisation which enable gradient flow through deep networks and stabilises the training dynamics, while Positional Encodings inject information about the token positions since the model lacks inherent sequence processing. Scaled Dot-Product Attention is the attention mechanism that computes the probability scores between queries & keys.

III. Experiments and Tuning

We use a v5e1 TPU provided by Google Colab for all training and evaluation. The complete source code for this project is hosted on GitHub

https://github.com/chinzhening/char_transformer

Preliminary experiments were conducted to see how the base transformer model performs on the TEXT8 dataset up to 100,000 iterations and the effect of various modifications to the architecture and training procedure. For fair comparison across architectures, we maintain a fixed training budget of 300 seconds for each experiment training run.

A. Base Model

The base model has the following hyperparameters:

- internal model dimensions of 128,
- 8 attention heads,
- 3 Transformer layers,
- and a maximum sequence length of 128

with a total parameter count of 616,832. Internally, the model used learned positional embeddings (LPE) and GeLU activations in the MLP with a hidden dimension expansion ratio of 4. Training was performed for 100,000 iterations (218.5s or 3 minutes and 38.5 seconds) with a batch size of 128, sequence length of 32 and constant learning rate of 0.001 with the Adam optimizer.

The training and test loss history is shown in Figure 1 and the final training and test loss achieved at 100,000 iterations is 1.3155 and 1.3141 respectively. The final train and test accuracy achieved is 58.8% and 59.0% respectively, and the final last character test accuracy is 58.8%.

B. SwiGLU

The base model uses GeLU activations in the hidden layers. Following [Sha20], we use SwiGLU activation functions instead. SwiGLU is a variant of the Gated Linear Unit (GLU) and has been shown to improve the performance of transformer models in most sequence modeling tasks. For the modification, the MLP hidden dimension expansion ratio was adjusted to 8/3 to keep the parameter count similar to the base model. All other hyperparameters and training procedure were kept the same as the base model, except increasing the batch size to 256 and learning rate to 0.00025 to reduce noise in the loss. The SwiGLU model has a total parameter count of 616,958, which is roughly the same as the base model.

After 245.7 seconds of training, the SwiGLU model achieved a final training and test loss of 1.2848 and 1.3263 respectively. The final test accuracy is 58.2%, about the same as base model. The final last-character test accuracy is 60.1%, higher than the base model. From the loss history in Figure 1, the loss convergence of the SwiGLU model is no better than the base model.

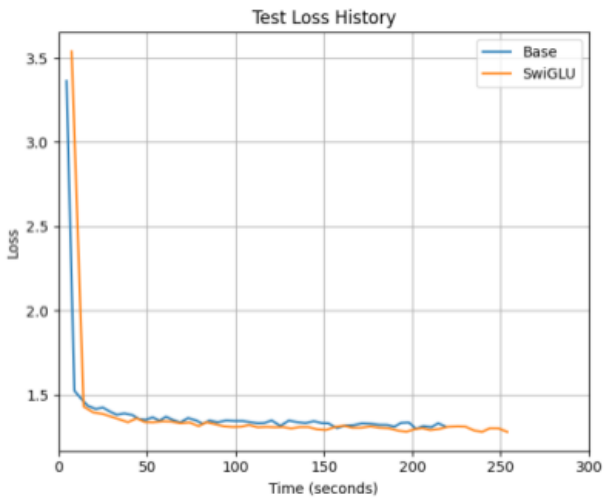

Fig. 1. Training and test loss history of base model with SwiGLU activations

C. Postitional Embeddings

The base model uses learned positional embeddings (LPE) [RNSS18] to encode the position of each character in the input sequence. Several LLMs have demonstrated that alternative positional embeddings such as sinusoidal positional embeddings (SPE) [VSP+17] and rotary positional embeddings (RoPE) [SLP+23] can improve model performance. Both SPE and RoPE do not have learned parameters, which reduces the overall parameter count of the model. We compare the training and performance of SPE and RoPE against LPE used in the SwiGLU model.

1) Sinusoidal Positional Embeddings (SPE): Since sinusoidal positional embeddings are not learned, the SwiGLU + SPE model has a lower parameter count of 600,574 compared to the SwiGLU + LPE model. After training for 100,000 iterations and 243.5 seconds, the SPE model achieved a test loss of 1.2992 and accuracy of 58.9%, comparable to the SwiGLU + LPE model. The training and test loss history is shown in Figure 2. The last character accuracy is 61.7%, similar to the SwiGLU + LPE model.

2) Rotary Postional Embeddings (RoPE): The SwiGLU + RoPE model has a parameter count of 600,574, for the same reason as SPE. A smaller learning rate of 0.001 was used to stabilise the training. Unlike SPE and LPE, the RoPE model showed significant improvement in loss convergence. We show the training and test loss history for the SwiGLU + RoPE model after 100,000 iterations and 237.7 seconds in Figure 2. The RoPE model achieved a final test loss of 0.0480 and accuracy of 98.5%, outperforming both the SPE and LPE models significantly.

Interestingly, we observe that the last character test accuracy is only 52.1%, significantly lower than the SwiGLU + LPE and SwiGLU + SPE models. This indicates that the RoPE model might generalize poorly on the last character prediction task compared to the other models.

A possible explanation for this is that the loss function used during training focuses on minimizing the average loss across all characters in the sequence, rather than encouraging the model to focus on the last character prediction. This could lead to suboptimal last character accuracy even though the overall test loss and accuracy are significantly better. To improve the generalization of the model, we experimented with using different loss functions to shift the training focus.

D. Loss Function

Since the loss function treats all positions equally, we tried a weighted cross-entropy loss to give higher importance to the last characters in the sequence. A linear weighting scheme is used where the first character has weight 0.2 and the last character has weight 1.0. We train with a batch size of 256 and a learning rate of 0.000025 for 100,000 iterations. A smaller learning rate was used to account for smaller loss scale. With the weighted loss, the SwiGLU + RoPE model still exhibits swift loss convergence in the training and test loss and

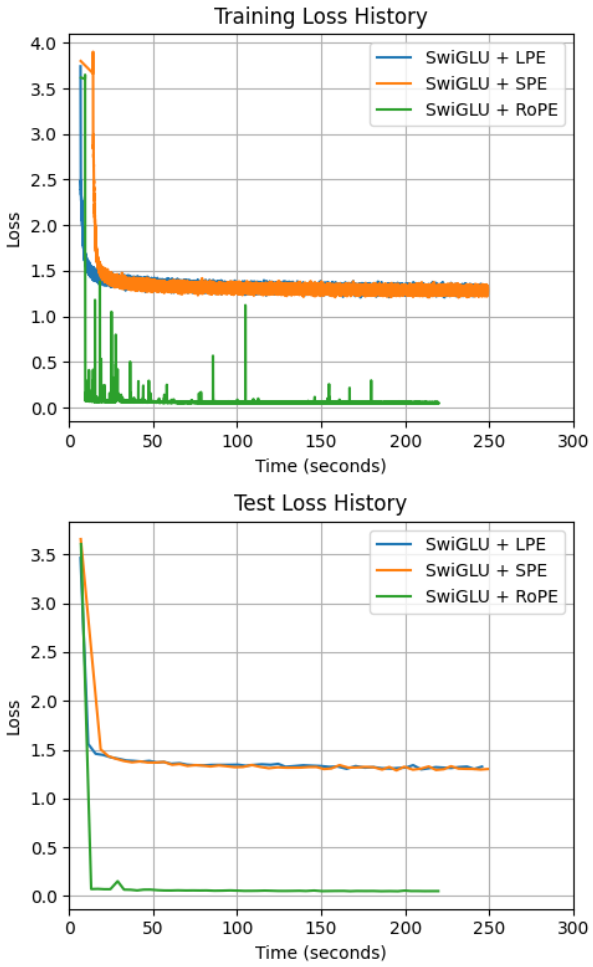

Fig. 2. Training and test loss history of SwiGLU model with LPE, SPE, and RoPE

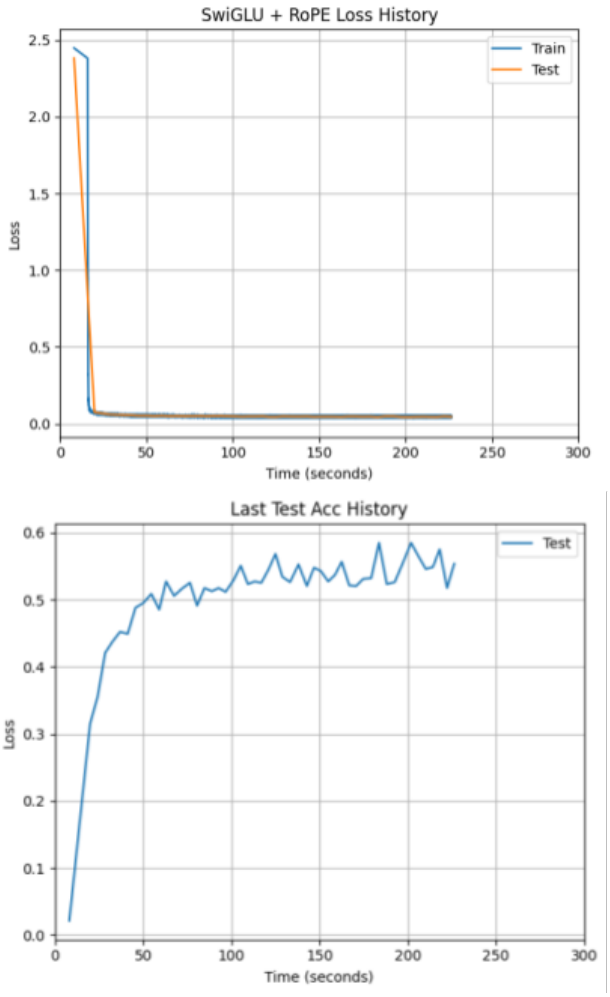

Fig. 3. SwiGLU RoPE + Weighted Loss: loss history (top), and last-character test accuracy (bottom).

the final last character test accuracy achieved is 55.4% compared to 52.1% previously. However, it is hard to discern whether this is due to noise, or if it is an actual improvement in shifting the training focus. From Figure 3, we observe that the last-character test accuracy is very noisy, and the loss bottoms out. This indicates that the model has reached its capacity limit, and further tuning and will not yield significant improvements.

Due to this, we return our attention to LPE based models.

E. Capacity

Taking the SwiGLU model, we increase the model's capacity by doubling the internal model dimensions from 128 to 256. The new model has 2,412,796 parameters, but the learning rate and batch size are unchanged from the SwiGLU + LPE model. From Figure 4, we see that the loss converges faster than the base model. In addition, the final last-character test accuracy appears to be less noisy, and gradually increasing.

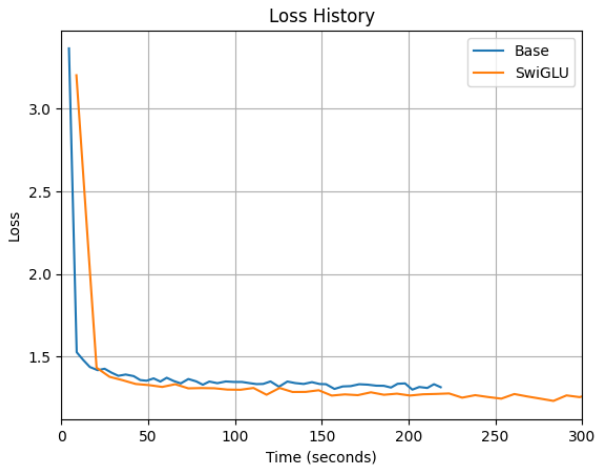

Fig. 4. SwiGLU with hidden dimension of 256

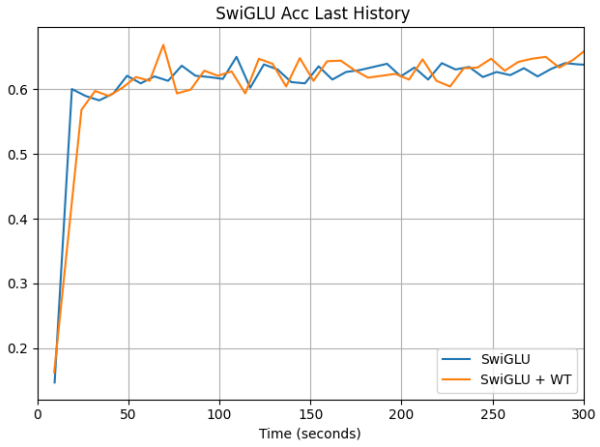

Fig. 5. Last Character Accuracy for model with SwiGLU (blue) & SwiGLU + weight tying (orange)

F. Weight Tying

We also experimented with weight tying to improve generalization but it did not make much difference, as seen in Figure 5. The parameter count was reduced to 2,405,884.

G. Optimizer

The optimizer combines gradient clipping with AdamW, a variant of the Adam optimiser that incorporates weight decay, and a warmup-cosine learning rate schedule. Gradients are clipped to prevent instability, while AdamW adaptively updates parameters with decoupled weight decay to stabilize training. Warmup and cosine decay avoid large initial updates, promoting smoother convergence [LGP+25], better generalization, and lower test loss. Training to 100,000 iterations, we use peak learning rate of 1e-3, end learning rate of 1e-5, and 5,000 warm up steps. The SwiGLU model + WT with this warmup-cosine scheduler demonstrates a faster drop in loss than the Adam optimizer with a constant learning rate of 2.5e-5 in the first 10 seconds of training as seen in Figure 6. The losses are about the same up until 150s after which the scheduled learning rate improves on the loss marginally.

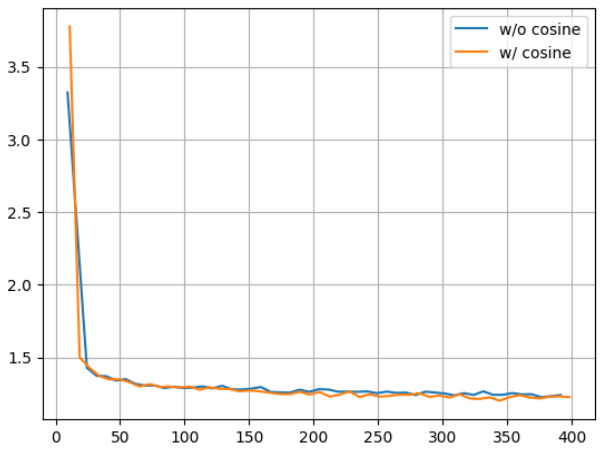

Fig. 6. AdamW optimizer with weight decay 0.01, warmup-cosine learning rate schedule.

H. Final Model

Our final SwiGLU + LPE model, trained with weight tying and the warmup–cosine AdamW optimizer, was run for 500,000 iterations (1965.2s or 33 minutes), after which it appeared to reach its performance ceiling. The model achieved a final training and test loss of 1.1435 and 1.2153 respectively, with final test accuracy of 61.8% and last-character accuracy of 66.7%. The loss and accuracy history is

plotted in Figure 7. To verify whether further improvements were possible, training was resumed for an additional 100,000 iterations using the saved checkpoint; however, the results after this extension showed negligible gains, with the test loss essentially unchanged at 1.2063 and last-character accuracy fluctuating randomly, see Figure 8. The accuracy oscillates within a narrow band rather than improving, indicating the model has fully saturated its representational capacity. Together, these results confirm that the model has effectively bottomed out, and additional training is unlikely to yield further performance gains.

IV. Discussion

A. Interpretation of Results

B. Hardware efficiency

Our modifications to the transformer components, such as SwiGLU activations, weight tying, and larger hidden dimensions, increased the model's expressivity and helped it converge more effectively, but they also raised the compute cost of each training step. Replacing GeLU with SwiGLU changes the structure of the feed-forward block by adding extra projections and element-wise operations. This leads to an increased number of small kernels and additional memory traffic, which GPUs and TPUs handle less efficiently. As the hardware spends more time on memory movement, kernel launches, and synchronization rather than sustained computation, overall utilisation drops. Consequently, while the model learned better, the actual training throughput decreased due to reduced hardware efficiency.

Furthermore, we tried to apply Bayesian Optimisation to tune the hypeparameters of the LSTM model (which is discussed in IV-C). However, we were not able complete this tuning process due to the insufficient compute resources available to us through Colab. We encountered memory overflow issues before all

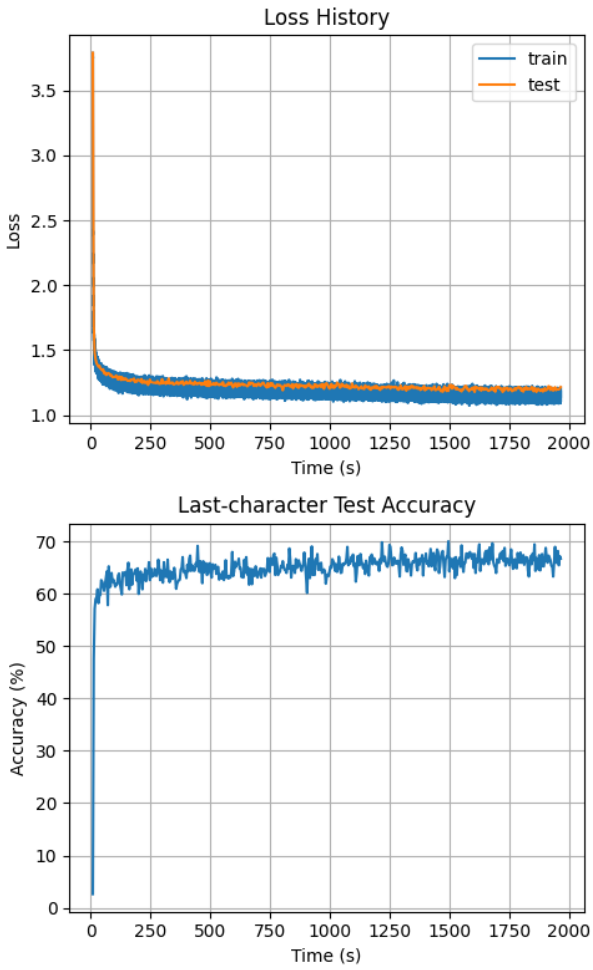

Fig. 7. Final Model Loss History & Last-character Test Accuracy

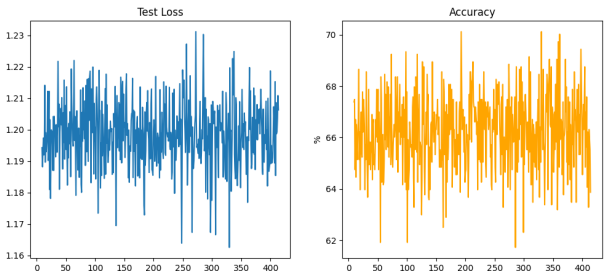

Fig. 8. Test Loss & Accuracy for iterations 500,000 - 600,000

of the sampled hyperparameters meaningfully converged.

C. Comparisons with Long Short-Term Memory (LSTM)

Since LSTM models were first proposed [HS96], LSTM models were traditionally used for next in-sequence prediction tasks. We performed testing on a LSTM model with a similar amount of parameters (up to 2 million) and compared their results with our transformer model. Our inputs are pre-processed by a token embedding layer, and then they are subsequently fed into LSTM cells, with forget gates and hidden states that allow information to propagate throughout the whole model, fixing the vanishing gradient problem that plagued previous recurrent models.

Our LSTM model with internal model dimensions of 256 & 4 LSTM cells managed to get a peak last character test accuracy of around 65%, as seen in 9, performing slightly worse than our final model. It is also interesting to note from our results that increasing the number of layers without changing the model dimensions may not necessarily improve the performance of the model.

We also tested on a bidirectional variant as first proposed in [GS05]. It delivered very promising results, with a very high overall accuracy (more than 90%), but a last character accuracy roughly similar to that of the regular LSTM model. However, after further observing the architecture, we realised that the bidirectional LSTM, with forward & backward LSTMs, has information about future characters, which violates the causality required for autoregressive generation. Transformers adhere to this causality with the implementation of a causal mask.

D. Future improvements

A natural extension from a character-level LLM would be exploring larger, more expressive token units. Character level models offer simplicity & an extremely small vocabulary, but they force the network to model long-range dependencies over very long sequences, and to infer word-level structures purely from characters. Using subword-level tokens or byte-level units can shorten effective sequence lengths, allowing attention layers to allocate more capacity to semantic & syntactic relationships, rather than spelling reconstruction.

Empirically, larger token units yield lower perplexity, faster convergence & more coherent long-range generation, since each token carries richer information, and sequences become

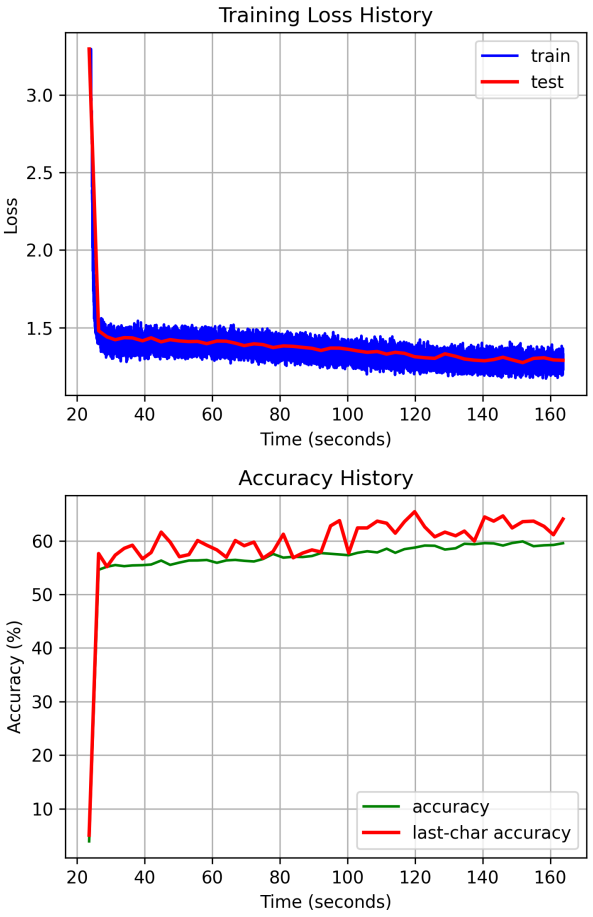

Fig. 9. LSTM training & test loss (left) and overall & last character accuracy (right)

shorter & easier to model. Future work can compare character-level training with hybrid or adaptive tokenization strategies to assess whether these approaches improve efficiency & output quality, while retaining the open-vocabulary advantages of character-level models.

References

[GS05] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm networks. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 4, pages 2047–2052 vol. 4, 2005.

[HS96] Sepp Hochreiter and Jürgen Schmidhuber. Lstm can solve hard long time lag problems. In *Proceedings of the 10th International Conference on Neural Information Processing Systems*, NIPS'96, page 473–479, Cambridge, MA, USA, 1996. MIT Press.

[LGP+25] Yuxing Liu, Yuze Ge, Rui Pan, An Kang, and Tong Zhang. Theoretical analysis on how learning rate warmup accelerates convergence, 2025.

[RNSS18] Alec Radford, Karthik Narasimhan, Tim Saliman, and Ilya Sutskever. Improving language understanding by generative pre-training. In *OpenAI Technical Report*, 2018.

[Sha20] Noam Shazeer. Glu variants improve transformer, 2020.

[SLP+23] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.