UCSD CSE256 WI26 PA2

Yuandong Zhang

Overview

This report includes (1) the implementation and experimental analysis of transformer encoder, (2) the implementation and experimental analysis of transformer decoder, and (3) architectural exploration of transformer decoder with Rotary Position Embedding (RoPE), Grouped Query Attention (GQA), and Swish-Gated Linear Unit (SwiGLU).

1 Transformer Encoder

1.1 Implementation

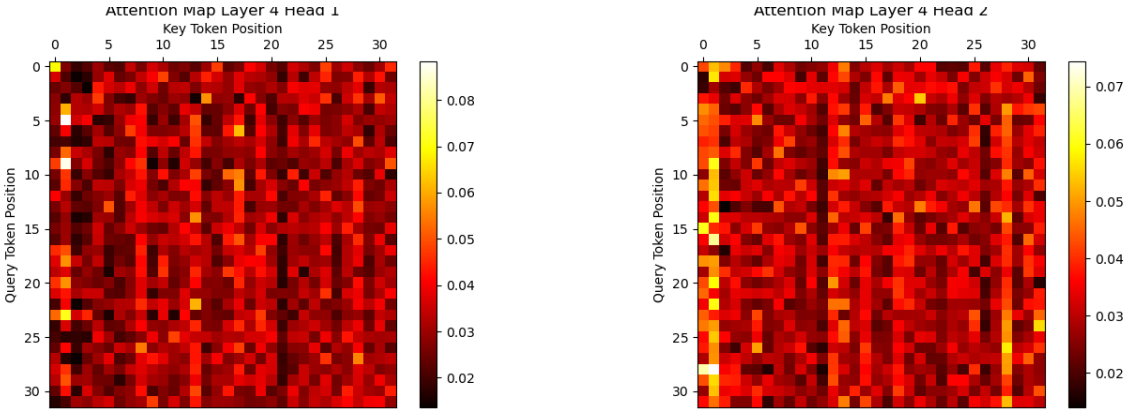
- **Encoder (1.1):** The transformer encoder follows Karpathy's tutorial and nanoGPT-style Pre-Norm design; hyperparameters match `main.py` and dropout is applied in the model.
- **Classifier (1.2):** A single hidden-layer feedforward network: mean-pooled encoder output $\rightarrow$ Linear(n_{input}, n_{hidden}) $\rightarrow$ GELU $\rightarrow$ Linear(n_{hidden}, n_{output}), with dimensions from `main.py`.
- **Joint training (1.3):** Encoder and classifier are trained end-to-end with CrossEntropyLoss and AdamW; the same loss is backpropagated through both so the encoder learns representations suited to the classification task.

1.2 Sanity Check

The encoder sanity check passes. The visualizations in Figure 1 for Layer 4, heads 1 and 2 (1a and 1b) show the expected bidirectional attention pattern with no causal mask. The softmax also normalized the distribution as each row of the passed attention map sum to 1.

1.3 Evaluation

The encoder has 569,664 parameters (the classifier is small and omitted from this count); within the encoder, the token embedding table and the feed-forward blocks in each layer account for most



(a) Layer 4, head 1.

(b) Layer 4, head 2.

Figure 1: Transformer encoder attention maps (sanity check).

of the parameters. Figure 2 shows training and test accuracy over 15 epochs. The model reaches **84.93%** test accuracy; training accuracy continues to rise while test accuracy flattens from around epoch 8 onward, indicating overfitting. Fifteen epochs are sufficient for this setup; further gains would likely require better data quality or regularization rather than more training.

2 Transformer Decoder

2.1 Implementation

The implementation of the transformer decoder follows the transformer decoder with an extra step to map the upper triangle area of the weight matrix for preventing future tokens to communicate the current token and ensure causal attention.

- **Decoder (2.1):** Same Pre-Norm stack as the encoder but with causal self-attention: the attention logits matrix is masked so that position i cannot attend to positions $j > i$ (upper triangle set to $-\infty$ before softmax), ensuring autoregressive behavior. The FFN uses hidden size 100 and ReLU as specified.
- **Pretraining (2.2):** The decoder is trained to minimize cross-entropy on next-token prediction over the LM training set for 500 iterations (batch size 16, block size 32); no separate pretraining script—training is done in `main.py` with AdamW.

2.2 Sanity Check

The decoder sanity check passes. The visualizations in Figure 3 for Layer 4, heads 1 and 2 (3a and 3b) show the expected causal attention pattern: the upper triangle is zero (no attention to

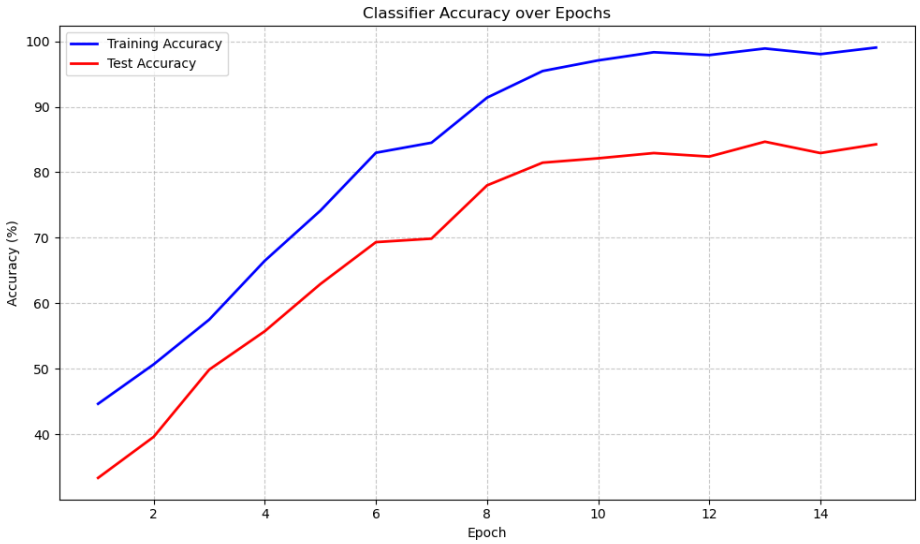


Figure 2: Classifier accuracy during training/evaluation for the encoder-based classifier.

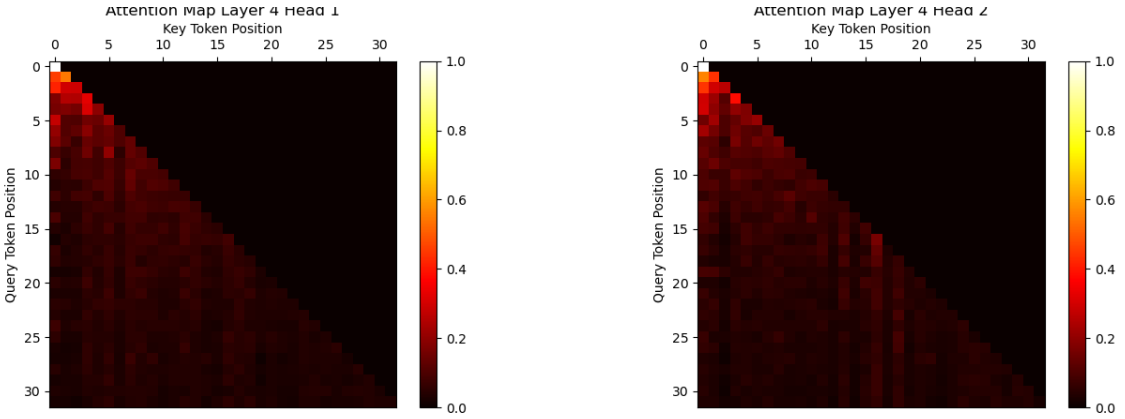
Table 1: Decoder language modeling results.

Metric	Value
Parameters	857 488.00
Train Perplexity	181.65
Test Perplexity (obama)	392.34
Test Perplexity (wbush)	496.65
Test Perplexity (hbush)	436.06

future tokens) and the lower triangle contains the valid weights. The softmax also normalizes the distribution so that each row of the passed attention map sums to 1.

2.3 Evaluation

The decoder has 857,488 parameters. Figure 4 shows train and per-president test perplexity over 500 training steps; Table 1 reports the final numbers. Training perplexity drops sharply (to 181.65) while the three test curves stay higher and flatten, with a clear train–test gap consistent with overfitting. The ranking across presidents is stable throughout: obama (392.34) best, then hbush (436.06), then wbush (496.65), suggesting the difficulty of each test set is largely data-driven. Differences may reflect how well each president's style or vocabulary matches the training distribution (e.g., obama test text may be closer to the training mix than wbush).



(a) Layer 4, head 1.

(b) Layer 4, head 2.

Figure 3: Transformer decoder attention maps (sanity check).

Table 2: Architectural exploration results.

Setting	Params	Train PPL	obama	wbush	hbush
Baseline	857,488	176.38	384.39	485.63	437.05
RoPE	855,440	104.90	323.23	426.09	363.40
GQA ($n_{kv} = 2$)	841,104	176.51	388.49	492.78	422.57
SwiGLU	883,488	156.32	371.69	465.60	400.21
All three	865,056	89.84	319.13	426.79	359.71

3 Architectural Exploration

I explore Rotary Position Embedding (RoPE), Grouped Query Attention (GQA), and Swish-Gated Linear Unit (SwiGLU) in the decoder. My motivation is that all three have improved trip mode prediction accuracy in my transformer for human mobility; I therefore test them here on language modeling to see how they affect perplexity in this setting.

3.1 Rotary Position Embedding

RoPE encodes position by rotating query and key vectors so that attention scores depend on relative position rather than absolute indices. In our experiments (Table 2), RoPE alone cuts train PPL from 176 to 105 and improves all three test PPLs; with fewer parameters (no learned position embedding table), the model no longer has to learn position inside attention, which is especially helpful for small, data-limited models like ours.

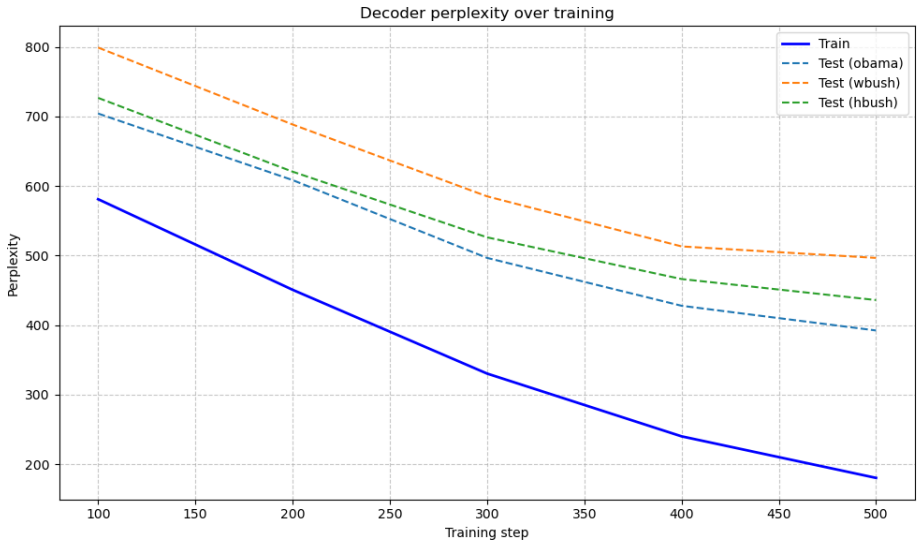


Figure 4: Decoder perplexity during language model pretraining.

3.2 Grouped Query Attention

GQA uses fewer key/value heads than query heads (e.g., $n_{\text{kv}} = 2$ with 4 query heads), reducing memory and compute. In our setting, GQA alone gives almost no gain (train PPL 176, test slightly worse on obama/wbush); the benefit is mainly in large-scale models where KV cache and capacity matter, so we do not see a clear advantage at this scale.

3.3 SwiGLU

SwiGLU replaces the standard FFN with a gated variant (gate and value branches combined by element-wise product), as in LLaMA. Here it gives a solid improvement on its own (train 156, test PPL down across all three presidents) and pairs well with RoPE: the "All three" config reaches the best train (89.84) and test numbers, with RoPE providing the largest gain and SwiGLU adding a useful boost.

AI Usage

I used LLMs for (i) Class structure scaffholding (ii) Comments (iii) Terminal output beautification

I implemented all decoder, encoder functions by hand with LLM to debug and make neccesary edit suggestions. Also, LLM was used to provide hints for me in comments.

Prism was used for grammar checks and for formatting equations, tables, and figures.