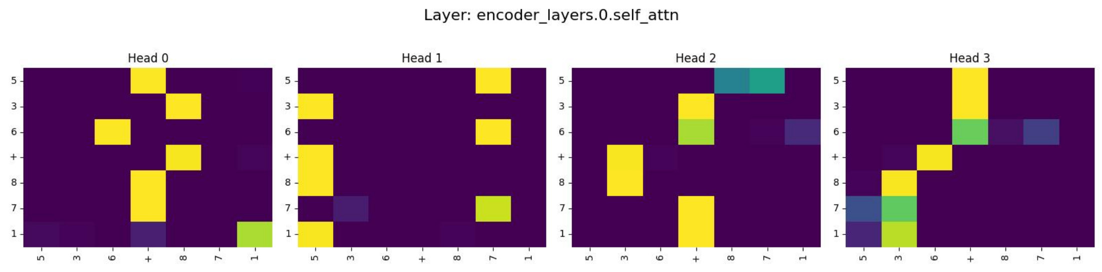
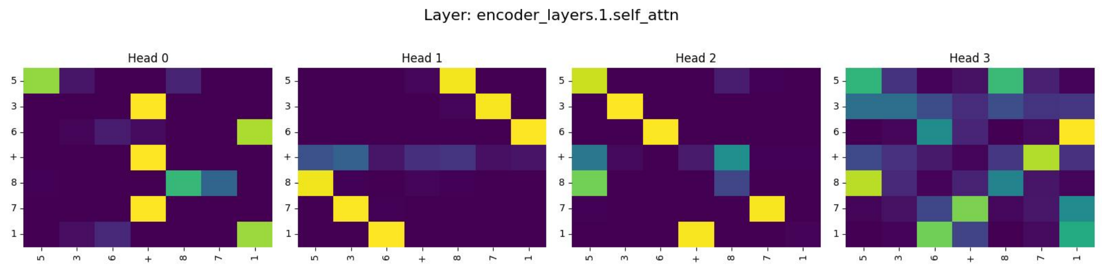
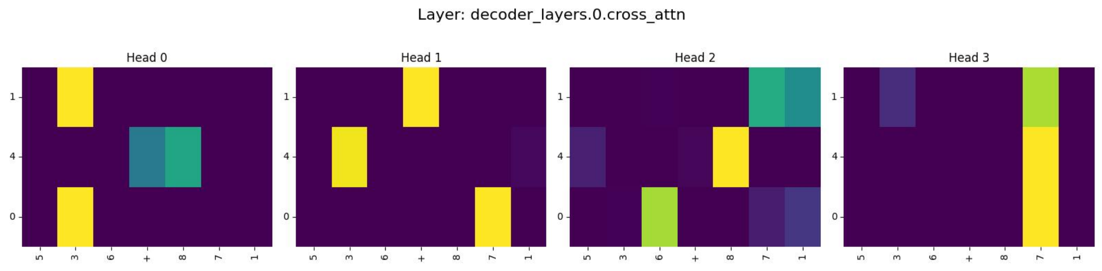
Problem 1: Multi-Head Attention - Multi-Digit Addition

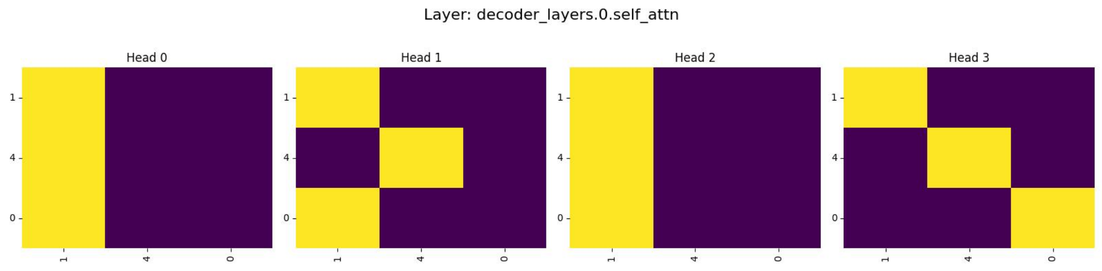
Haomiao Wang

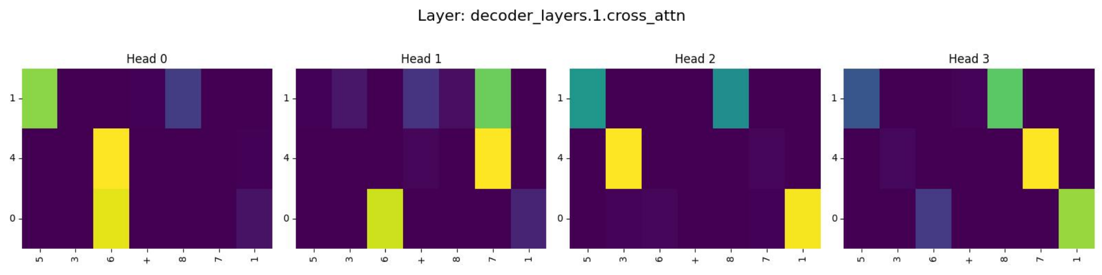
1. Attention pattern visualizations from at least 4 different heads

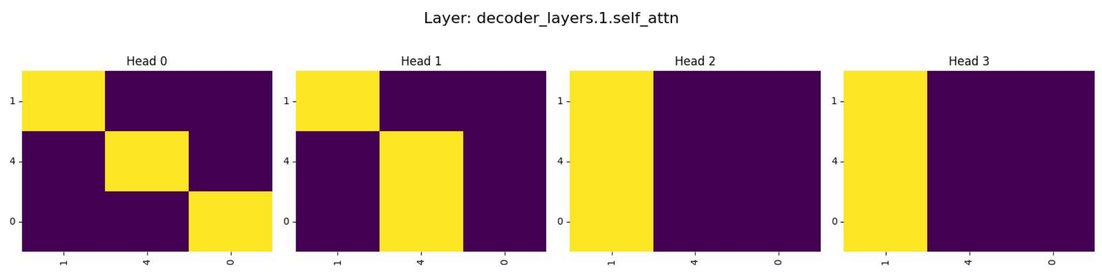












Some analyses:

The attention heatmaps visually confirm a clear division of labor among the heads.

Head 2 in encoder layers 1 shows a diagonal pattern. This preserves a token's information as it moves up the layers. Heads identified as redundant, like those in encoder layers 0, show sparse, unstructured, or mostly dark patterns. This indicates they have not learned a meaningful or unique function.

Head 0 in decoder layers 0.self attn shows a vertical line. Its function may be to constantly remind the model of the sequence's starting point. Head 3 in decoder layers 0.self attn shows a diagonal pattern. It stably passes each token's own representation forward through the layers, preventing information loss.

2. Head ablation study: which heads are critical vs redundant?

The result of the head ablation study was shown in folder head_analysis. This was done by systematically setting the weights of each head to zero and measuring the resulting drop in sequence accuracy on the test set. The baseline model achieved a sequence accuracy of 99.51%.

Critical Heads:
① encoder_layers.1.self_attn.head_1 is by far the most critical head. Its removal caused the accuracy to plummet by 67.58%.
②decoder_layers.1.self_attn.head_1 is the second most critical, causing an accuracy drop of 37.06%.
③decoder_layers.1.self_attn.head_0 is the third most critical, with a 35.89% accuracy drop.
④ decoder_layers.1.cross_attn.head_1 is the fourth most critical, causing a 22.46% drop.

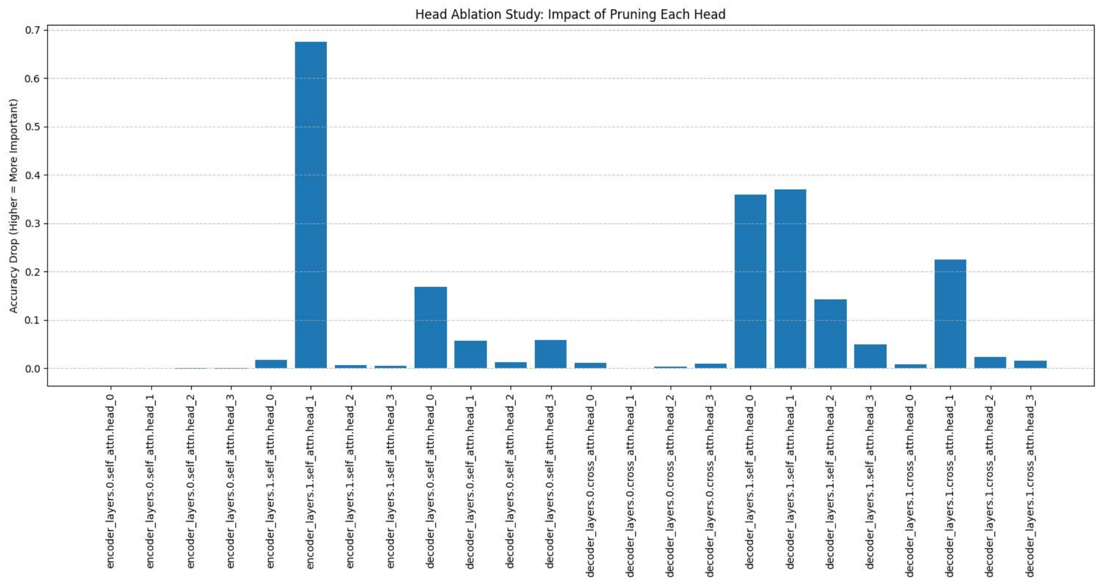
Heads in the final layer (Layer 1) are significantly more important than heads in the initial layer (Layer 0).

Redundant Heads:
① encoder_layers.0.self_attn.head_3 and head_2 were the most redundant. Their removal slightly increased model accuracy, with accuracy drops of -0.0015% and

-0.0005%, respectively.
② decoder_layers.0.cross_attn.head_1 and encoder_layers.0.self_attn.head_1 are entirely redundant, showing a 0.0000% accuracy drop when ablated.

In fact, all four heads in encoder_layers.0.self_attn are non-essential, with accuracy drops near zero.

The visual image is as follows：



3. Discussion: How do attention heads specialize for carry propagation?

The model may handle carry propagation through a three-step process involving different, specialized components:

①Lookup (Cross-Attention)
The model first uses "alignment" heads in cross-attention to perform a parallel lookup. These heads identify and gather the two corresponding input digits for the current calculation step. For example, head 3 in decoder_layers_1_cross_attn.png, for a given output step, attends to the two corresponding input digits that need to be summed.

②Calculation (FFN)
This gathered information is then passed to the Feed-Forward Network (FFN). The FFN performs the mathematical computation (e.g., 3 + 7 = 10) and encodes the result, including both the sum (0) and the carry (1), into that time's token vector.

③Propagation (Self-Attention)
The self_attn heads are then responsible for propagating this hidden carry information. Diagonal patterns (e.g., Head 0 in decoder_layers_1_self_attn.png) perform state preservation, protecting the token's vector from being lost. Vertical patterns (e.g.,

Head 2 in decoder_layers_1_self_attn.png) may allow all future timesteps to look back at a specific past token's vector to access the stored carry bit.

4. Quantitative results: percentage of heads that can be pruned with minimal accuracy loss

A prunable head is defined as one that can be removed with minimal accuracy loss, specified as an accuracy drop of less than 1.0% (0.01).

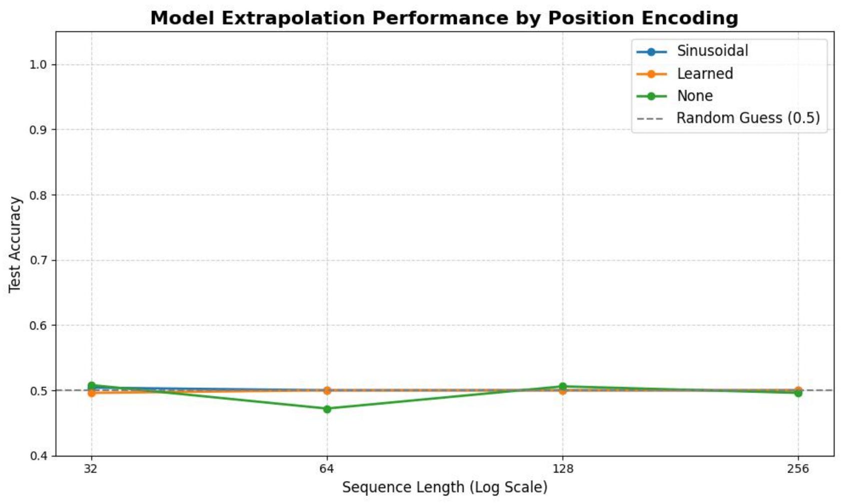
Based on the ranking, 10 of the 24 heads meet this criterion (ranks 15-24). (result is shown in head_importance_ranking.txt)
Percentage of Heads that can be Pruned: (10 / 24) = 41.67%

This quantitative result confirms the finding from the ablation study that a significant portion of the model’s heads are redundant, particularly in the early layers.

Problem 2: Positional Encoding and Length Extrapolation

Haomiao Wang

1. Extrapolation curves showing accuracy vs. sequence length for all three methods



Based on the extrapolation_curves.png:

①Sinusoidal
This model's validation accuracy was ~99.6% when learning the training task. Its accuracy immediately plummeted to 50.4% at sequence length 32 and remained at the 50.0% random-guess baseline for all longer sequences.

②Learned
This model's validation accuracy was ~50.2% when learning the training task. It also failed on all extrapolation tests, with accuracy hovering between 49.6% and 50.0%.

③None
This model also failed to learn the task and performed as a random guess on all test sets, with accuracy fluctuating between 47.2% and 50.8%.

2. Mathematical explanation: Why does sinusoidal encoding extrapolate while learned encoding fails?

①Sinusoidal
This encoding is not learned; it's a fixed mathematical formula:

$$PE(pos, 2i) = sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

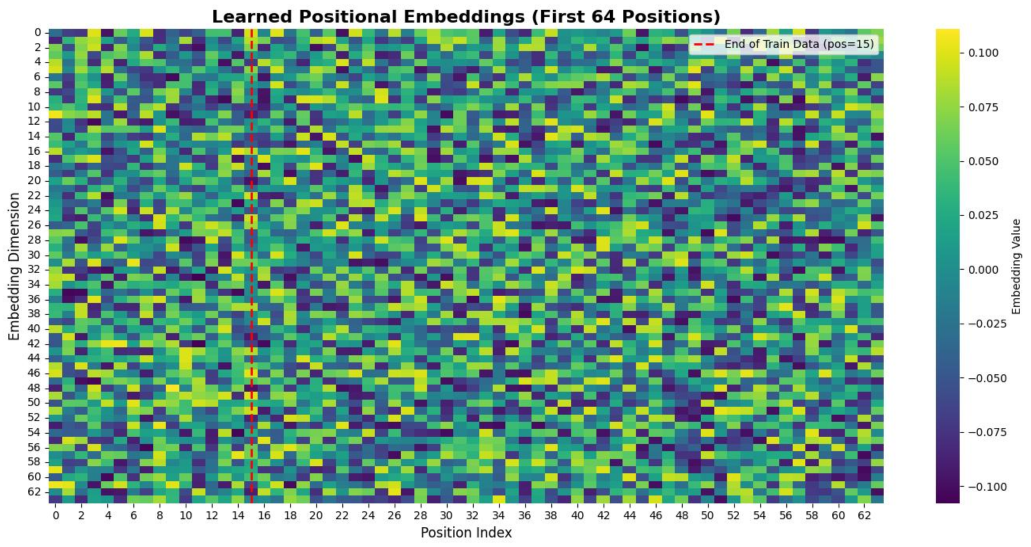
$$PE(pos, 2i+1) = cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

This formula is defined for any position pos, including those never seen in training. More importantly, for any fixed offset k, PE(pos+k) can be represented as a linear transformation of PE(pos). This means the model can learn relative position relationships, which should theoretically allow it to generalize to new lengths.

②Learned

This encoding is an nn.Embedding lookup table. It learns a unique, fixed vector for each position it sees during training. When the model encounters an unseen position, it remains a randomly initialized, meaningless noise vector (as seen in the visualization in the third part).

3. Position embedding visualization for learned encoding

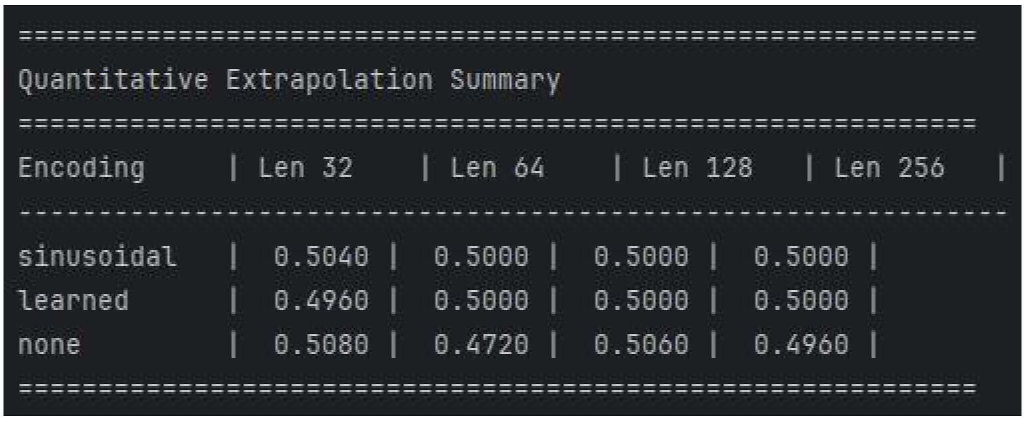


The Red Line (pos=15): This line marks the boundary of the training data. The model only ever saw positions 0-15 during training.

The Pattern (Noise): The entire plot, including the trained region (0-15) and the untrained region (16-63), appears to be random, unstructured noise.

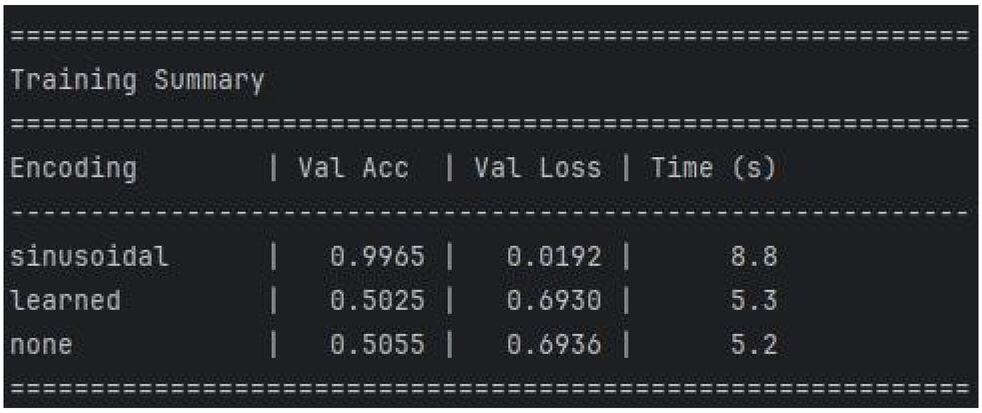
This image is the visual proof of the model's failure. Because the learned model achieved only ~50% accuracy (a random guess), it learned nothing about the sorting task. Consequently, it also learned no meaningful structure for its position vectors, leaving the trained vectors just as chaotic as the untrained ones.

4. Quantitative comparison: accuracy at lengths 32, 64, 128, 256



```
============================================================
Quantitative Extrapolation Summary
============================================================
Encoding     | Len 32    | Len 64    | Len 128   | Len 256   |
--------------------------------------------------------------
sinusoidal   |  0.5040 |  0.5000 |  0.5000 |  0.5000 |
learned      |  0.4960 |  0.5000 |  0.5000 |  0.5000 |
none         |  0.5080 |  0.4720 |  0.5060 |  0.4960 |
============================================================
```

5. Addition: Training result output



```
============================================================
Training Summary
============================================================
Encoding        | Val Acc  | Val Loss | Time (s)
------------------------------------------------------------
sinusoidal      |   0.9965 |   0.0192 |      8.8
learned         |   0.5025 |   0.6930 |      5.3
none            |   0.5055 |   0.6936 |      5.2
============================================================
```