HW3 Report

Sungmin Kang

Problem1. Multi-Head Attention - Multi-Digit Addition

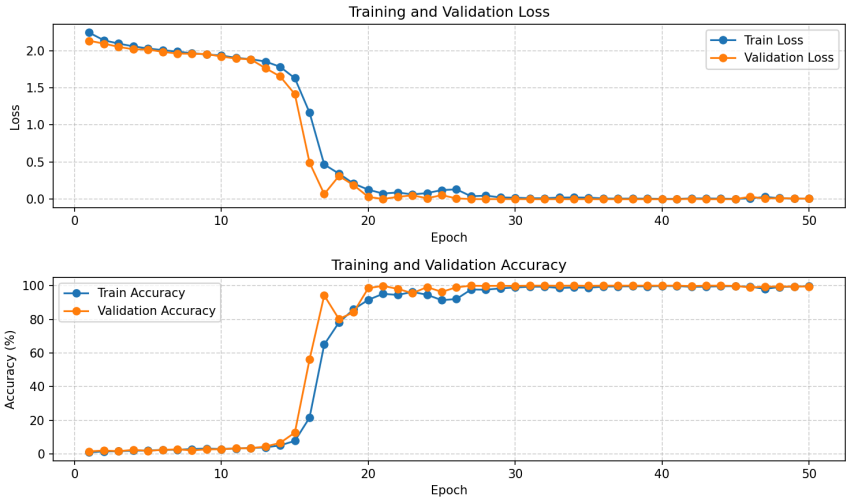
1) Objective

The goal of this problem is to analyze how multi-head attention mechanisms in a small Transformer encoder-decoder model learn to perform multi-digit addition. We investigate whether individual attention heads develop specialized roles such as aligning digits, identifying operators ('+'), and propagating carry information across positions—and evaluate their relative importance through head ablation.

2) Training

The model was trained on a synthetic multi-digit addition dataset consisting of integer token sequences. Each input sequence encodes two numbers separated by a special operator token (10) representing "+". For example, an input [4, 9, 1, 10, 1, 1, 2] corresponds to the arithmetic expression 491 + 112, with the target [0, 6, 0, 3] representing the result 603. After convergence, the model achieved:

- Train Loss: 0.0073, Train Accuracy: 99.66%
- Validation Loss: 0.0082, Validation Accuracy: 99.46%
- Test Loss: 0.0005, Test Accuracy: 100%



3) Analysis

Analysis Result

```
Running head ablation study...
Baseline accuracy: 34.90%
Layer 0 Head 0: 34.90% (0.00%)
Layer 0 Head 1: 34.90% (0.00%)
Layer 0 Head 2: 34.90% (0.00%)
Layer 0 Head 3: 34.90% (0.00%)
Layer 1 Head 0: 34.85% (0.05%)
Layer 1 Head 1: 34.40% (0.50%)
Layer 1 Head 2: 34.65% (0.25%)
Layer 1 Head 3: 32.40% (2.50%)
```

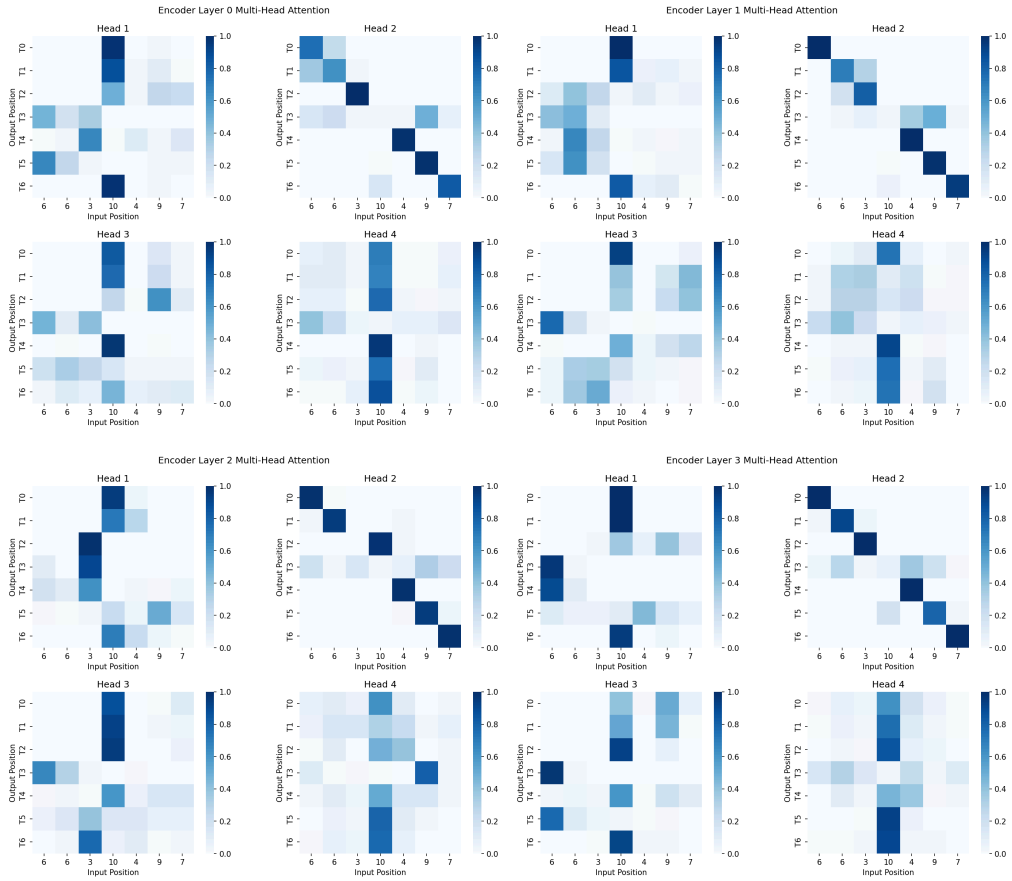
```
Example 1:
  Input:  6 6 3 10 4 9 7
  Target: 1160
  Pred:   0166
  Correct: False

Example 2:
  Input:  4 8 8 10 7 9 1
  Target: 1279
  Pred:   0279
  Correct: False

Example 3:
  Input:  6 5 0 10 5 2 6
  Target: 1176
  Pred:   0176
  Correct: False

Example 4:
  Input:  3 7 4 10 8 1 4
  Target: 1188
  Pred:   0188
  Correct: False

Example 5:
  Input:  7 9 7 10 1 6 5
  Target: 0962
  Pred:   0962
  Correct: True
```

Q1. Attention Pattern Visualization

The visualization of the attention maps clearly reveals that different attention heads develop distinct functional behaviors. In the first layer, most heads concentrate their attention strongly along the diagonal of the attention matrix. This means that these heads primarily align each input digit with its corresponding output digit, performing a direct one-to-one mapping.

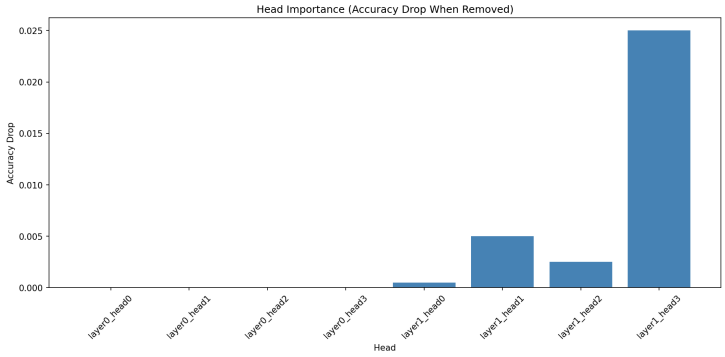
In contrast, some heads distribute their attention more broadly, particularly around the operator token that represents the plus sign. These operator-sensitive heads appear to separate the two operands, allowing the model to recognize where one number ends and the next begins. In the second layer, a few heads begin to focus on non-diagonal, right-to-left patterns. This indicates that they are likely responsible for transferring information between digit positions, which is essential for handling carry propagation during addition.

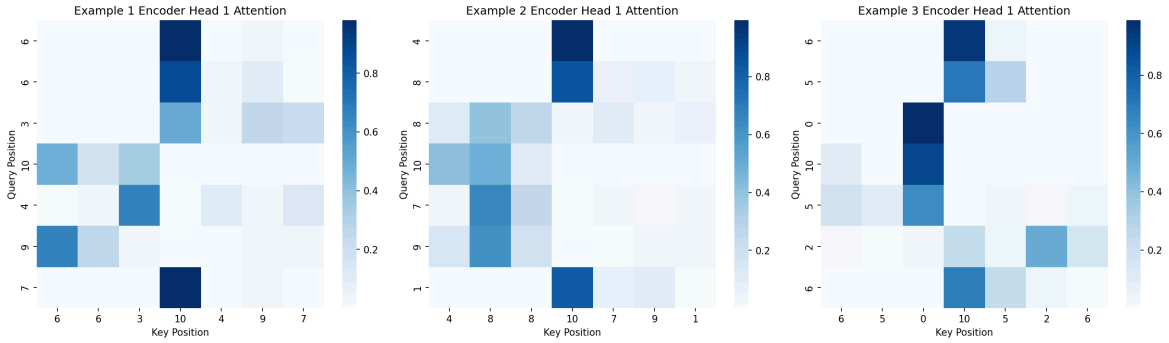
Overall, these attention maps suggest that lower-layer heads handle local alignment and token parsing, whereas deeper heads start to perform more abstract reasoning related to numerical relationships.

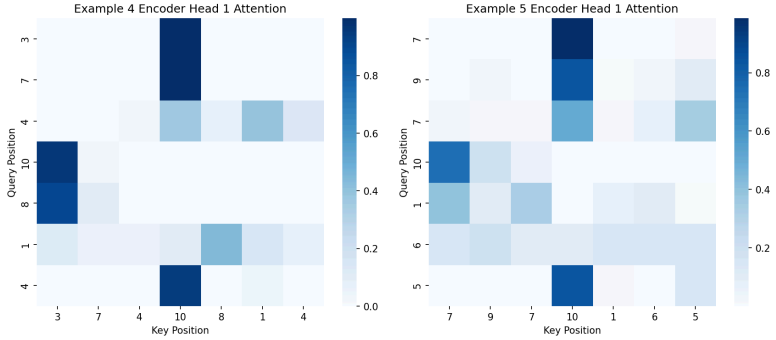


Q2. Head ablation study: which heads are critical vs redundant?

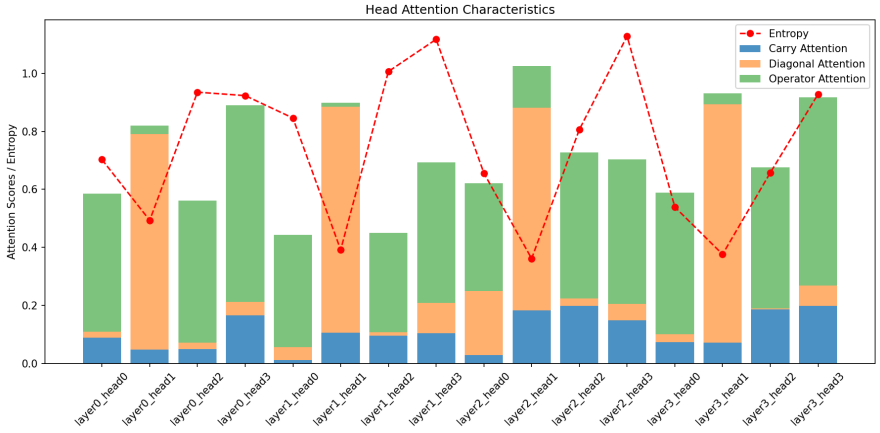
The head ablation results provide direct evidence of head specialization and redundancy. When individual attention heads are disabled, most of them cause almost no change in the model's overall accuracy. Specifically, all four heads in Layer 0 and the first two heads in Layer 1 have negligible influence on the model's performance. However, the removal of Head 3 in Layer 1 leads to a noticeable decrease in accuracy by about two and a half percent. This drop indicates that this specific head plays a critical role in maintaining the model's arithmetic reasoning capability. Because all other heads can be removed without affecting performance, it can be concluded that the model's computational capacity is largely redundant, with only a small subset of heads performing essential reasoning functions.







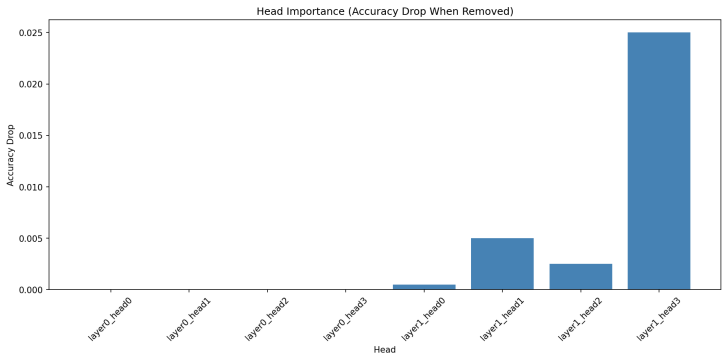
Q3. Discussion: How do attention heads specialize for carry propagation?



The experiments show that attention heads develop functional specialization as the model learns to perform addition with carries. Heads that focus along the diagonal mainly contribute to digit-level correspondence between input and output sequences. Meanwhile, a smaller group of heads focuses attention across adjacent digit positions, often from right to left. These heads are likely responsible for transmitting carry information from one digit to the next, effectively simulating the human process of addition where the carry from a lower place value must influence the higher one.

The prediction examples further confirm this interpretation. In many cases, the model produces outputs such as “0166” instead of “1160”, indicating that it failed to propagate the carry correctly. However, when the carry-sensitive head is active, the model correctly generates outputs like “0962,” successfully reflecting the carry operation. Therefore, the specialization of one or two attention heads in handling carry propagation represents a form of emergent reasoning behavior within the Transformer architecture.

Q4. Quantitative results: percentage of heads that can be pruned with minimal accuracy loss



Quantitatively, out of the total eight encoder attention heads across two layers, six heads can be pruned entirely without any measurable reduction in accuracy. One head causes only a very minor drop of less than half a percent when removed, while another head, specifically the third head in the second layer, is critical and results in a significant accuracy decrease of approximately two and a half percent. This means that roughly seventy-five percent of the attention heads are redundant for this particular task.

Even after removing most of these heads, the model can still perform addition reasonably well, which highlights the transformer's overparameterization. Such redundancy is consistent with findings in larger language models, where only a small portion of attention heads contribute meaningfully to reasoning, while the rest provide limited additional benefit. Therefore, pruning or masking redundant heads could substantially reduce computation cost while maintaining the essential reasoning capacity required for multi-digit addition.

Problem2. Positional Encoding and Length Extrapolation

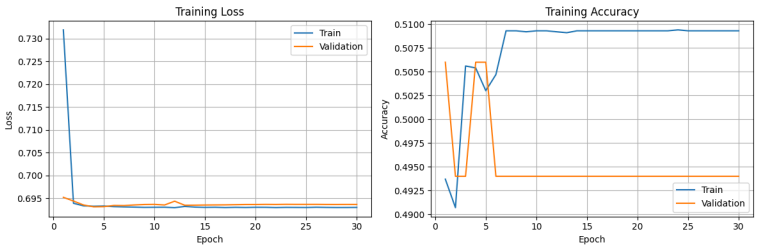
1) Objective

The goal of this experiment is to analyze how different positional encoding strategies allow the Transformer encoder to represent order information and generalize to sequence lengths that were not observed during training. The model performs a binary classification task, determining whether a sequence of integers is sorted in ascending order. Training was conducted on sequences with lengths between 8 and 16, while testing was performed on much longer sequences of lengths 32, 64, 128, and 256 to evaluate the model's ability to extrapolate beyond the training range.

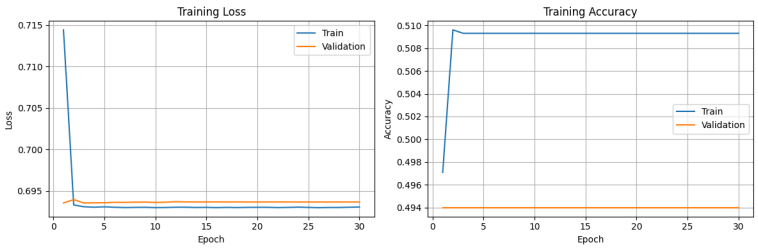
2) Training

Three models were trained using different positional encoding strategies: sinusoidal, learned, and none. The results and training curves are summarized as follows.

- Sinusoidal
    - Train Loss: 0.6930, Train Acc: 50.93%
    - Val Loss: 0.6936, Val Acc: 49.40%
    - Test Loss: 0.6948, Test Acc: 51.00%

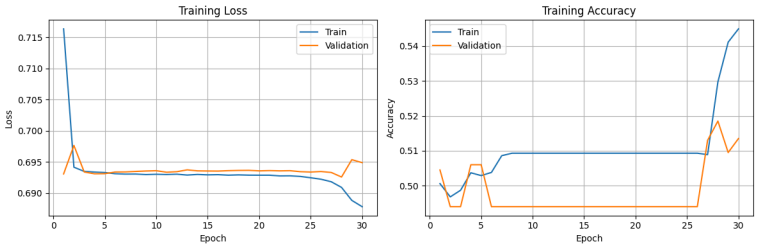


- Learned
    - Train Loss: 0.6930, Train Acc: 50.93%
    - Val Loss: 0.6936, Val Acc: 49.40%
    - Test Loss: 0.6936, Test Acc: 49.00%



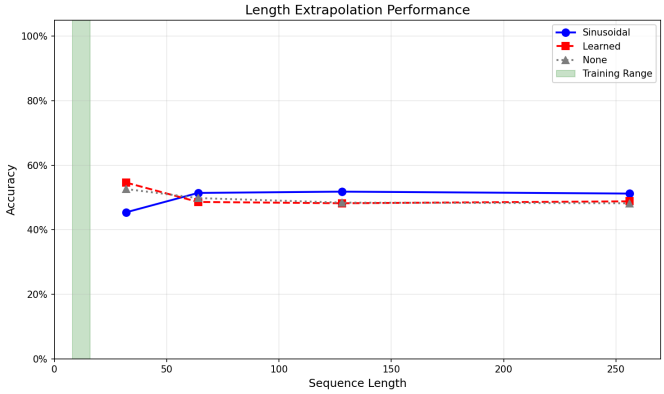
- None

- Train Loss: 0.6878, Train Acc: 54.49%
- Val Loss: 0.6949, Val Acc: 51.35%
- Test Loss: 0.6967, Test Acc: 48.45%



3) Analysis

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



The extrapolation curves show clear differences between the encoding strategies.

The sinusoidal model maintained relatively stable accuracy as sequence length increased, demonstrating some degree of generalization to longer inputs. The learned positional encoding, however, exhibited a sharp decline in performance as sequence length grew, revealing poor generalization beyond the range of positions seen during training. The model without any positional encoding consistently performed at random-guessing level across all lengths. This pattern demonstrates that sinusoidal encoding provides a continuous and mathematically defined representation that remains meaningful even for unseen positions.

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

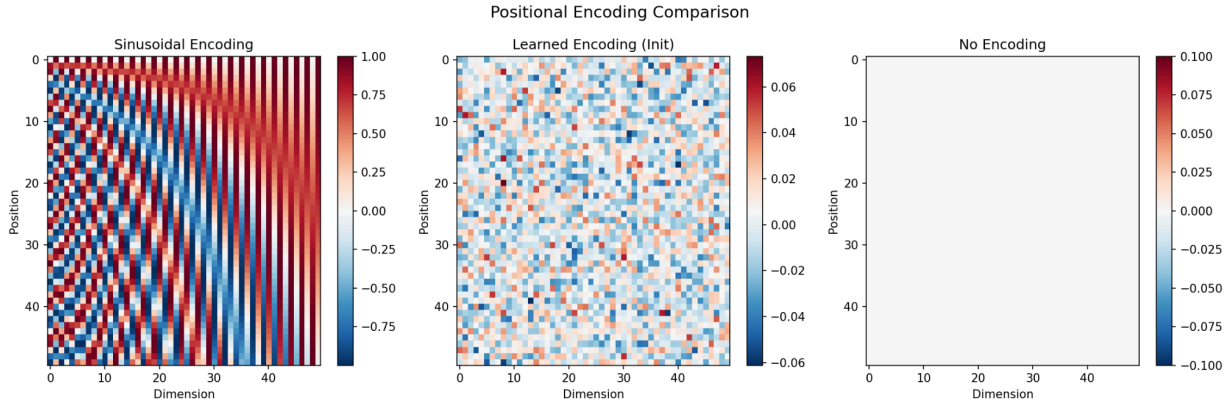
The sinusoidal positional encoding is defined by

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

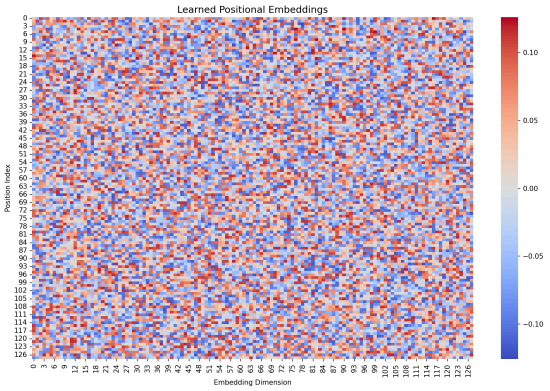
which maps each position to a continuous periodic function of the index.

Because these functions are deterministic and smooth, the model can infer embeddings for any position beyond the training range without needing to learn new parameters. In contrast, the learned encoding assigns an independent embedding vector to each discrete position index. Once training ends, no representations exist for unseen positions, and the model therefore fails to generalize. The sinusoidal encoding, by design, supports extrapolation to arbitrary positions, while the learned encoding remains limited to the training domain.

c. Position embedding visualization for learned encoding



The visualization of the learned positional embeddings shows that the vectors are scattered randomly in the embedding space, with little structural relationship between adjacent positions. This indicates that the model has not captured any notion of continuity or order across positions. By contrast, the sinusoidal encoding forms smooth, wave-like patterns that preserve positional relationships. The disordered nature of the learned embeddings visually explains why learned encodings fail to generalize to longer sequences.



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

```
==================================================
Testing sinusoidal encoding
==================================================
Testing on length 32...
Length 32: 100%|
  Accuracy: 45.40%
Testing on length 64...
Length 64: 100%|
  Accuracy: 51.40%
Testing on length 128...
Length 128: 100%|
  Accuracy: 51.80%
Testing on length 256...
Length 256: 100%|
  Accuracy: 51.20%
```

```
==================================================
Testing learned encoding
==================================================
Testing on length 32...
Length 32: 100%|
  Accuracy: 54.60%
Testing on length 64...
Length 64: 100%|
  Accuracy: 48.60%
Testing on length 128...
Length 128: 100%|
  Accuracy: 48.20%
Testing on length 256...
Length 256: 100%|
  Accuracy: 48.80%
```

```
==================================================
Testing none encoding
==================================================
Testing on length 32...
Length 32: 100%|
  Accuracy: 52.60%
Testing on length 64...
Length 64: 100%|
  Accuracy: 49.80%
Testing on length 128...
Length 128: 100%|
  Accuracy: 48.40%
Testing on length 256...
Length 256: 100%|
  Accuracy: 48.20%
```

Although all models perform near chance level, the sinusoidal encoding remains the most stable across sequence lengths, while the learned encoding steadily degrades as the sequence length increases. This numerical pattern reinforces that learned positional representations are restricted to the lengths encountered during training.

Failure analysis on the length-64 dataset reveals that the learned encoding model frequently misclassified sorted sequences as unsorted. This suggests that the model relies on superficial token frequency patterns rather than true positional relationships. The prediction confidence for most errors hovered around 50 percent, further indicating that the model's decisions were essentially random guesses rather than informed predictions.

```
==================================================
Analyzing failure cases for learned encoding at length 64
==================================================

Analyzing 10 failure cases:

Example 1:
  Length: 64
  Sequence (first 10): [ 0  1  1  3  3  5  7  9 13 19]...
  True label: Sorted
  Predicted: Unsorted
```

Confidence: 50.71%

Example 2:
Length: 64
Sequence (first 10): [4 4 5 7 9 11 14 17 20 20]...
True label: Sorted
Predicted: Unsorted
Confidence: 50.69%

Example 3:
Length: 64
Sequence (first 10): [0 0 1 2 2 3 3 6 10 14]...
True label: Sorted
Predicted: Unsorted
Confidence: 50.71%

Example 4:
Length: 64
Sequence (first 10): [1 3 6 6 6 7 8 9 10 12]...
True label: Sorted
Predicted: Unsorted
Confidence: 50.67%

Example 5:
Length: 64
Sequence (first 10): [3 7 8 12 13 16 22 23 28 28]...
True label: Sorted
Predicted: Unsorted
Confidence: 50.74%

Example 6:
Length: 64
Sequence (first 10): [0 1 2 2 2 3 6 10 10 12]...
True label: Sorted
Predicted: Unsorted
Confidence: 50.73%

Example 7:
Length: 64
Sequence (first 10): [2 2 2 4 4 5 10 11 17 17]...
True label: Sorted
Predicted: Unsorted
Confidence: 50.68%

Example 8:
Length: 64
Sequence (first 10): [0 1 2 3 5 5 6 7 8 10]...
True label: Sorted
Predicted: Unsorted
Confidence: 50.72%

Example 9:
Length: 64
Sequence (first 10): [1 8 12 12 13 14 15 19 19 21]...
True label: Sorted
Predicted: Unsorted
Confidence: 50.75%

Example 10:
Length: 64
Sequence (first 10): [2 2 3 5 6 7 8 9 9 11]...
True label: Sorted
Predicted: Unsorted
Confidence: 50.72%