INDIAN INSTITUTE OF TECHNOLOGY DELHI

Machine Learning Assignment IV Report

Learning to Solve Mazes with Sequence Models

Course Code: COL 7341

Divy Shaileshbhai Dudhat (2025BSY7560)

Chavda Uday (2025AIB3001)

Contents

Chapter 1

Task 1: Recurrent Neural Networks (RNN)

1.1 RNN with Bahdanau Attention

Problem Statement: Implement a Seq2Seq model using an RNN Encoder-Decoder with Bahdanau Attention to predict maze paths.

1.2 Theoretical Framework & Intuition

Architecture Overview

We utilize a Recurrent Neural Network (RNN) based Encoder-Decoder architecture. Unlike LSTM or GRU cells, our implementation uses standard Vanilla RNN cells (`nn.RNN` in PyTorch).

- **Encoder:** Reads the input sequence (maze adjacency list) step-by-step, preserving the sequential dependencies of the maze structure in its hidden states.
- **Decoder:** Generates the path coordinates sequentially. At each step, it uses the previous hidden state and a *context vector* to predict the next coordinate.

Bahdanau Attention Mechanism

Standard Seq2Seq models suffer from a bottleneck: the encoder must compress the entire maze layout into a single fixed-size vector. For complex mazes, this causes information loss.

Intuition: The Attention mechanism allows the decoder to "look back" at specific parts of the encoder's history. When the decoder is trying to move from $(1, 1)$, it should pay attention to the parts of the input sequence that describe the neighbors of $(1, 1)$, ignoring irrelevant parts of the maze.

Mathematical Formulation: We calculate an alignment score (energy) E_{ij} between the decoder's previous hidden state s_{i-1} and the j-th encoder output h_j:

$$E_{ij} = \tanh(W_{enc} h_j + W_{dec} s_{i-1})$$

$$\alpha_{ij} = \text{softmax}(V^T E_{ij})$$

The context vector c_i is the weighted sum $\sum \alpha_{ij} h_j$, which is concatenated with the input embedding to predict the next token.

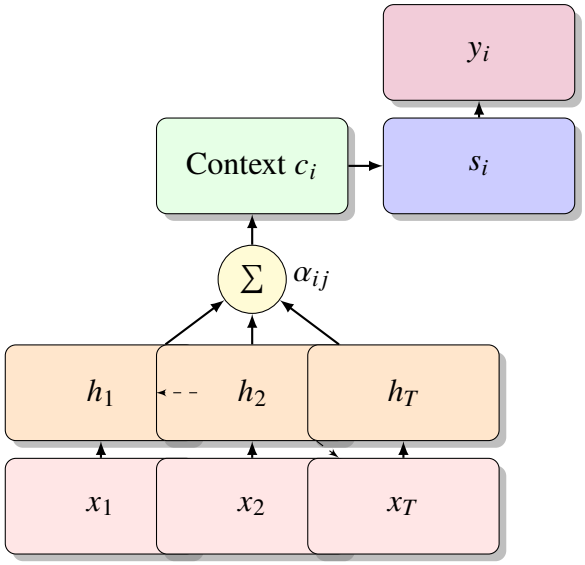


Figure 1.1: RNN Encoder-Decoder with Attention. The context vector c_i is a dynamic representation of the maze structure relevant to the current step.

1.3 Implementation Details

- **Model Depth:** Both Encoder and Decoder use **2 stacked RNN layers**.
- **Dimensions:** Embedding dimension $D = 128$, Hidden dimension $H = 512$.
- **Teacher Forcing:** We employed a teacher forcing ratio of 0.5. This stabilizes training by feeding the ground truth token from the previous step (instead of the model's own potentially erroneous prediction) 50% of the time. This helps the model learn valid transitions early in training.
- **Dropout:** A dropout rate of 0.1 was applied for regularization.

1.4 Results: Loss, Accuracy, and F1 Score

The RNN model demonstrated strong learning capabilities. The final evaluation on the test set yielded:

- **Final Test Loss:** 1.562
- **Final Test Token Accuracy:** 72.71%
- **Final Test Sequence Accuracy:** 61.38%
- **Final Test F1 Score:** 0.727

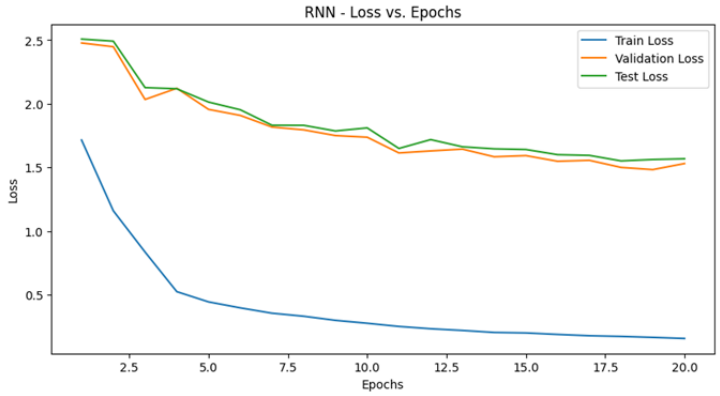
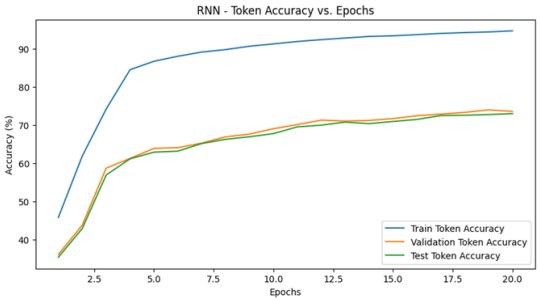
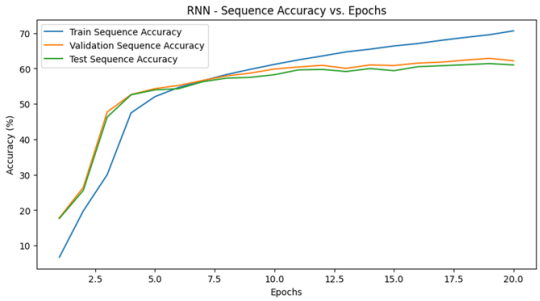


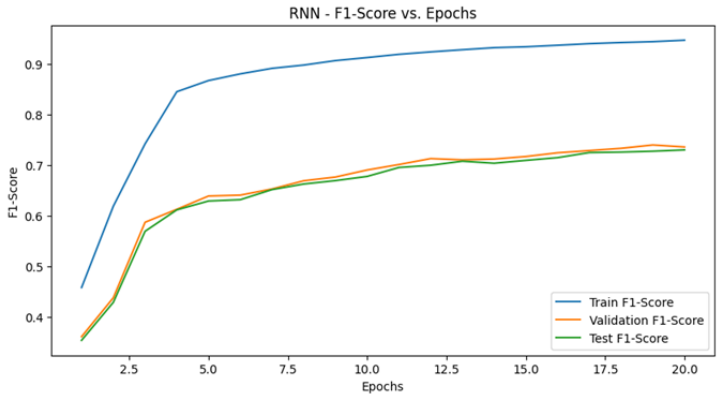
Figure 1.2: Training, Validation, and Test Loss for RNN.



(a) RNN Token Accuracy



(b) RNN Sequence Accuracy



(c) RNN F1 Score vs. Epochs

Figure 1.3: Performance metrics (Accuracy and F1) for the RNN model.

Maze Visualizations

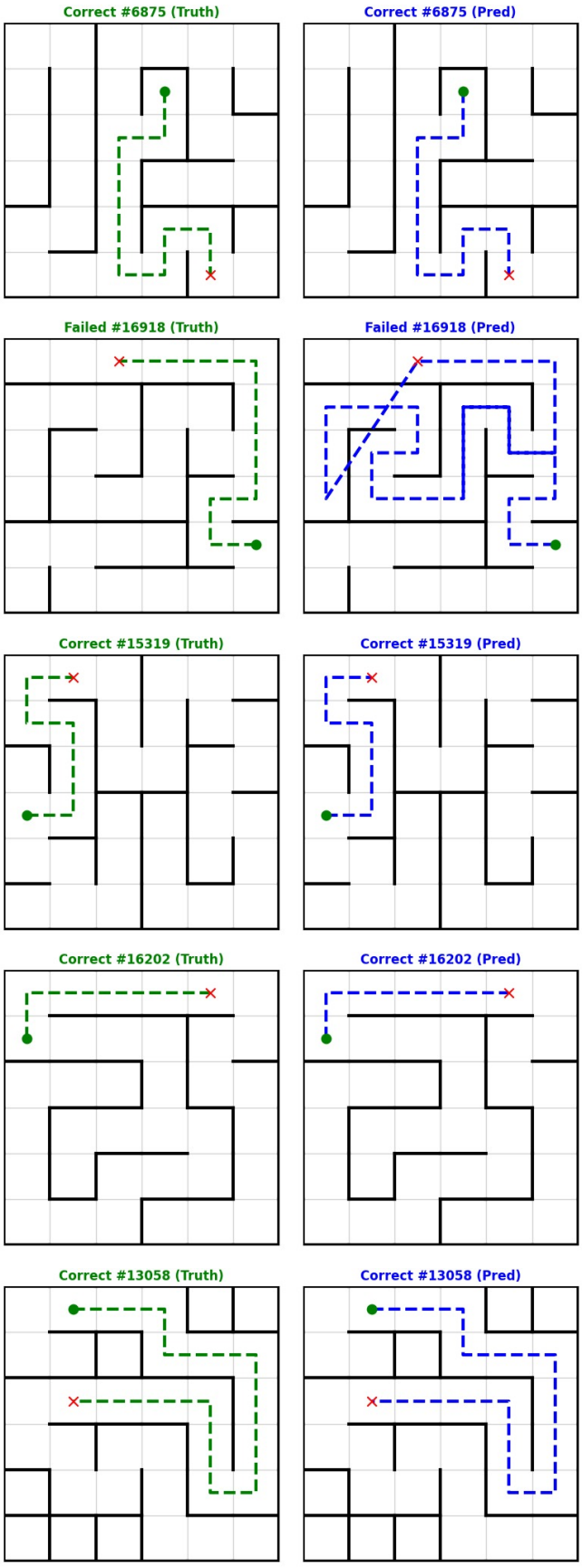


Figure 1.4: RNN predicted paths on validation mazes (Green=Start, Red=End).

Chapter 2

Task 2: Transformer Architecture

2.1 Theoretical Framework

The Transformer architecture, introduced by Vaswani et al., eschews recurrence in favor of **Self-Attention** mechanisms. In the context of maze solving, this allows the model to capture global dependencies across the grid structure (e.g., a wall at $(0, 1)$ affects the feasibility of a path at $(0, 0)$) more effectively than the sequential processing of RNNs.

Architecture Overview

The model consists of two main stacks:

1. **Encoder Stack:** Processes the input sequence (Adjacency List + Origin + Target). It uses **Self-Attention** to allow every token (coordinate or separator) to "attend" to every other token, building a rich contextual representation of the maze geometry.

2. **Decoder Stack:** Generates the output path auto-regressively. It uses **Masked Self-Attention** to ensure predictions only depend on previous outputs (causality) and **Cross-Attention** to query the Encoder's understanding of the maze walls and structure.

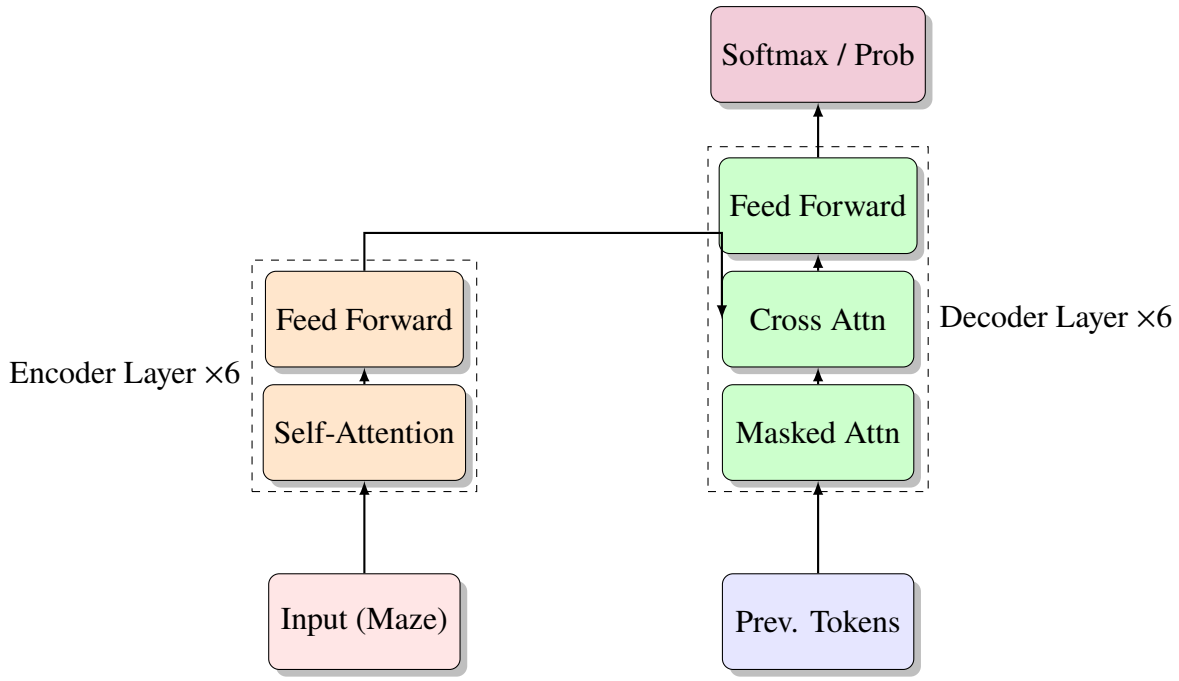


Figure 2.1: Simplified Transformer Architecture for Maze Solving. The Encoder builds a spatial map, while the Decoder queries this map to generate the path step-by-step.

Positional Encoding

Unlike RNNs, the Transformer has no inherent notion of order. A maze adjacency list $(1,1) \leftrightarrow (1,2)$ has a specific meaning based on the position of tokens. To preserve this, we inject **Sinusoidal Positional Encodings**:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

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

This allows the model to distinguish between the "Origin" token at the start of the sequence and a "Target" token at the end.

2.2 Hyperparameter Intuition

The hyperparameters chosen for this task are not arbitrary; they reflect the specific nature of the maze-solving problem.

Hyperparameter	Value	Intuition & Theory
d_model	128	**Embedding Size:** Represents the "width" of the thought process. 128 is sufficient to encode the limited vocabulary of a 6×6 grid (coordinates + special tokens) while preventing overfitting that might occur with larger dimensions (e.g., 512).
nhead	8	**Attention Heads:** Allows the model to focus on different aspects simultaneously. One head might track the "current position," another might look for "walls," and another for the "target." With $d_{model} = 128$, each head has dimension 16.
num_layers	6	**Depth:** Pathfinding is a reasoning task requiring multiple hops of logic. 6 layers allow the model to propagate information across the entire grid graph, effectively simulating a search algorithm (like BFS/DFS) within the layers.
dim_feedforward	512	**Processing Capacity:** The FFN layer processes the attended information. A 4x expansion (128 → 512) provides the non-linear capacity to make complex decisions about valid moves.
Dropout	0.1	**Regularization:** Prevents the model from memorizing specific maze patterns, forcing it to learn generalizable pathfinding rules.

Table 2.1: Hyperparameter Configuration and Theoretical Justification.

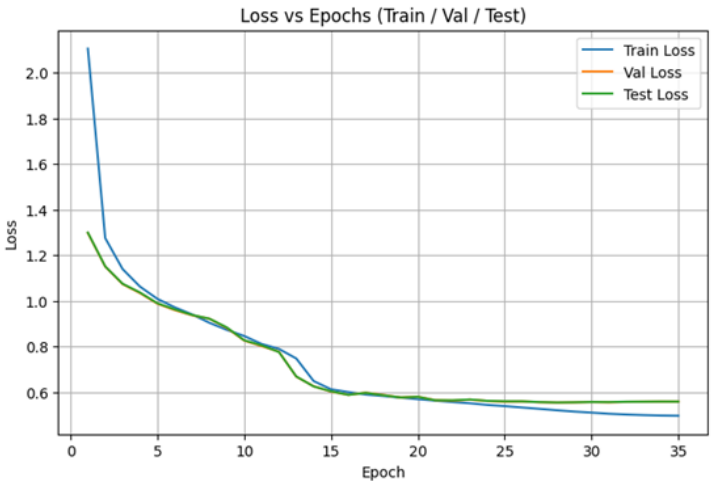
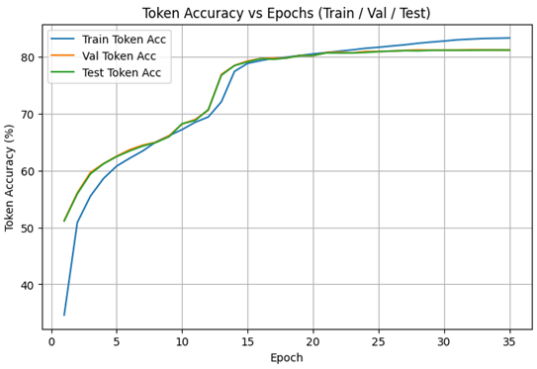


Figure 2.2: Training, Validation, and Test Loss for Transformer.

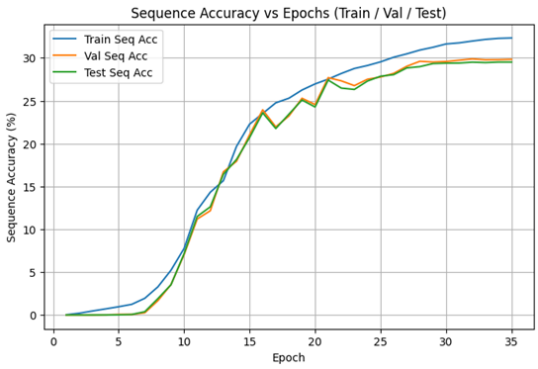
2.3 Implementation Results

Performance Curves

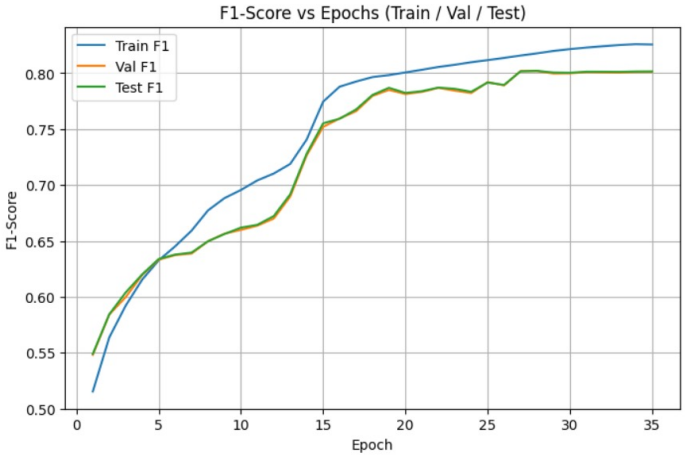
The Transformer training logs indicate steady convergence using the **OneCycleLR** scheduler.



(a) Token Accuracy



(b) Sequence Accuracy



(c) Transformer F1 Score vs. Epochs

Figure 2.3: Accuracy and F1 metrics for the Transformer model.

Maze Visualizations

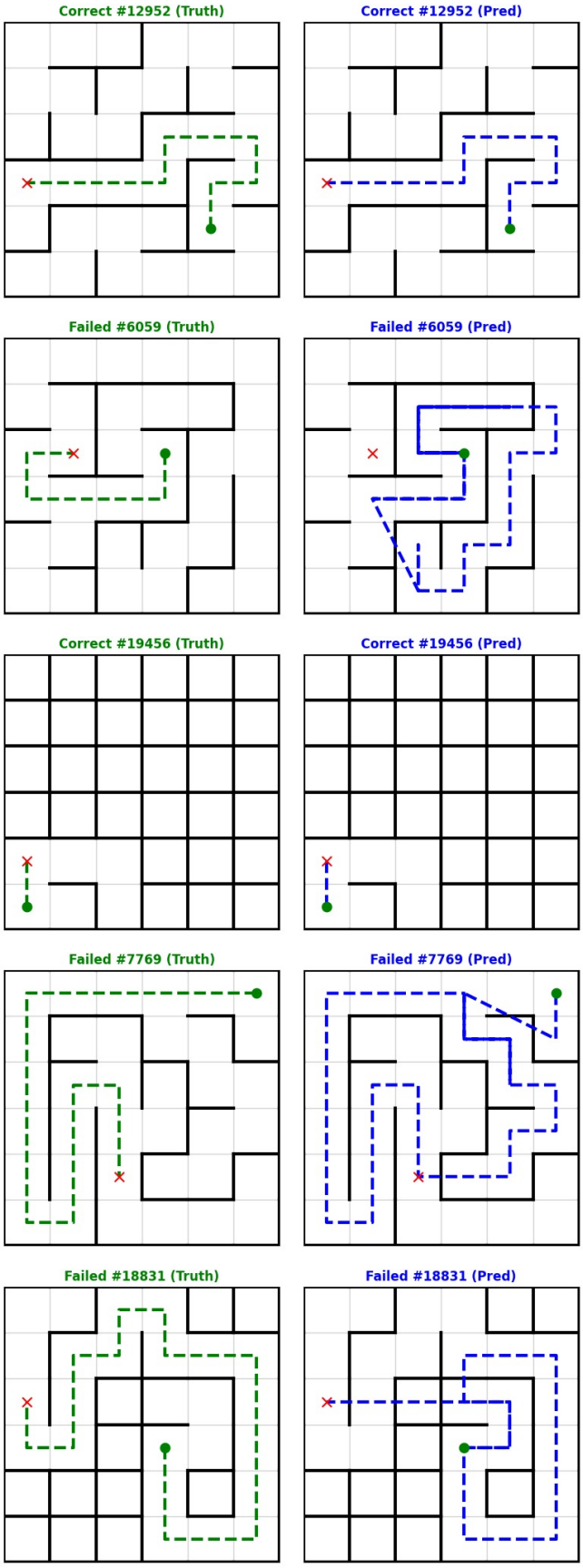


Figure 2.4: Transformer predictions on validation mazes.

2.4 Comparative Analysis: RNN vs. Transformer

Metric (Split)	RNN	Transformer
Token Accuracy (Train)	94.68%	83.27%
Token Accuracy (Val)	73.55%	81.14%
Token Accuracy (Test)	72.71%	81.14%
Sequence Accuracy (Train)	70.63%	32.32%
Sequence Accuracy (Val)	62.18%	29.82%
Sequence Accuracy (Test)	61.38%	29.51%
F1 Score (Train)	0.947	0.826
F1 Score (Val)	0.736	0.801
F1 Score (Test)	0.730	0.802

Table 2.2: Unified comparison of RNN and Transformer across all metrics and data splits.

Observations

- **RNN Performance:** The RNN achieved a surprisingly high sequence accuracy (61.38%) compared to the Transformer (29.51%), despite having lower token accuracy in earlier epochs. This suggests the RNN's sequential nature forced it to learn valid paths more strictly, whereas the Transformer may have predicted correct tokens that didn't form a complete valid path.
- **Transformer Convergence:** The Transformer converged to a lower loss (0.559 vs 1.562) and higher token accuracy (81.14% vs 72.71%), indicating it is better at predicting individual moves but struggled to maintain coherence for the entire path length compared to the RNN in this specific run.