Transformer-based Sequence-to-Sequence Model for Indoor Pathfinding

Kim Seong Jun, Kim Jun Seop, Kim Jin Seong, Han Bo Ram, Hwang Yoon Seo

Dept. of Software, SungKyunkwan University.

strawberrym01n@gmail.com, ryan201904@gmail.com, wlstjd1011@skku.edu, ldsboram@naver.com, qwdfgqw@gmail.com

**ABSTRACT**

In this study, we explore the application of a transformer-based sequence-to-sequence (seq2seq) model to the problem of indoor pathfinding within a static building environment, specifically the second engineering building up to the 3rd floor. Our model is trained on a dataset of 472,656 shortest paths generated by Dijkstra's algorithm, achieving a validation accuracy of 97.39%. While the model demonstrates high accuracy, particularly for longer paths, it incurs a significantly higher computation time compared to traditional methods, with a sample path taking 3924.49 ms versus 2.71 ms for Dijkstra's algorithm. We discuss the implications of this trade-off, noting that the model's performance on short paths (length 1-5) is weaker due to data distribution, but this is deemed acceptable given fewer queries for short paths. This work serves as a proof of concept for using machine learning in static pathfinding and suggests directions for future research in optimizing inference speed.

**CCS Concepts**

**• Computing methodologies → Machine learning → Machine learning approaches → Neural networks;**

**• Information systems → Information retrieval → Retrieval tasks and goals → Routing and pathfinding**

**Keywords**

Pathfinding; sequence-to-sequence model; transformer; Dijkstra's algorithm; machine learning; campus navigation

# INTRODUCTION

Pathfinding is a fundamental problem in computer science with applications ranging from navigation systems to robotics and campus wayfinding. Traditional algorithms such as Dijkstra's and A\* are widely used for their efficiency and ability to guarantee optimal paths in static graphs [1]. However, with the advent of machine learning, there is growing interest in exploring whether neural networks can approximate or enhance these methods, potentially offering new insights or efficiencies in specific scenarios.

In this paper, we investigate the use of a transformer-based seq2seq model for indoor pathfinding within the second engineering building, up to the 3rd floor. Our approach involves training the model on a dataset of shortest paths generated by Dijkstra's algorithm, aiming to predict the correct path given a start and end room (e.g., 27507A to 25108). We evaluate the model's accuracy and computation time compared to Dijkstra's algorithm and analyze its performance across different path lengths.

The motivation behind this approach is to explore the potential of machine learning for path prediction in static environments, despite the efficiency of traditional methods. Our findings highlight a significant trade-off: while the model achieves high accuracy, its computation time is substantially longer, raising questions about its practical applicability. This study serves as a proof of concept for using machine learning in static pathfinding, particularly for educational or research purposes, and provides insights into the challenges of applying neural networks to such problems.

# Related Work

Traditional pathfinding algorithms like Dijkstra's and A\* provide optimal solutions for static graphs. Machine learning approaches, such as reinforcement learning Grid Path Planning with Deep Reinforcement Learning[2] and graph neural networks Finding shortest paths with Graph Neural Networks [3], have been explored for pathfinding in dynamic or large-scale environments. Additionally, Q-learning has been utilized to find shortest paths in graphs Finding Shortest Path using Q-Learning Algorithm [4]. Our work differs by using a seq2seq model to directly predict paths in a static graph, trained on data from Dijkstra's algorithm, offering a novel perspective on learning path structures.

Compared to these approaches, our method focuses on a static graph and leverages the transformer architecture for sequence prediction, which is less common for pathfinding but provides insight into the feasibility of such models for path approximation. While reinforcement learning and graph neural networks are more suited for dynamic or large-scale environments, our approach is tailored to the specific case of indoor navigation with precomputed paths.

# Methodology

## Data Generation

SAMPLE: Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*Conference’10*, Month 1–2, 2010, City, State, Country.

Copyright 2010 ACM 1-58113-000-0/00/0010 …$15.00.

DOI: http://dx.doi.org/10.1145/12345.67890

We modeled the building layout as a graph, with nodes representing rooms, corridors, stairs, elevators, and doors, and edges representing connections with weights based on distance or traversal cost. The graph was constructed from provided JSON files, containing approximately 500 nodes and 1500 edges, with an average path length of 15 nodes across all pairs. Using Dijkstra's algorithm, we computed all pairs shortest paths, resulting in a dataset of 472,656 samples. Each sample consists of a start room ID, an end room ID, and the corresponding path instructions tokenized into a sequence (e.g., 'TURN\_RIGHT', 'D=5', 'TYPE=Corridor').

The paths were tokenized by encoding movement instructions, such as turning directions, distances (e.g., 'D=5' for 5 units), and segment types (e.g., 'TYPE=Corridor', 'TYPE=Room'). Special tokens like '<sos>' (start of sequence), '<eos>' (end of sequence), and '<pad>' (padding) were added to handle sequence boundaries and batch processing. The vocabulary was built by collecting all unique tokens, resulting in a size of 716, and saved to 'token2idx.json' for reuse. The dataset was split into training (90%, 425,390 samples), validation (10%, 47,266 samples), and test sets, ensuring a balanced distribution of path lengths.

## Model Architecture

We employed a transformer-based seq2seq model, consisting of an encoder that processes the start and end nodes and a decoder that generates the path sequence. The input to the encoder is a pair of embeddings for the start and end nodes, with each node embedded into a 512-dimensional space using learned embeddings. The transformer architecture includes 6 encoder layers, 6 decoder layers, 8 attention heads, and a feedforward network dimension of 2048. The model uses positional encoding to maintain sequence order and employs multi-head attention to capture long-range dependencies.

The decoder generates the path sequence step by step, outputting one node at a time until an end token is generated. The output vocabulary consists of all node indices plus special tokens for start and end. The model was implemented using PyTorch, with the transformer implementation from the transformers library, and trained on GPU with a batch size of 64.

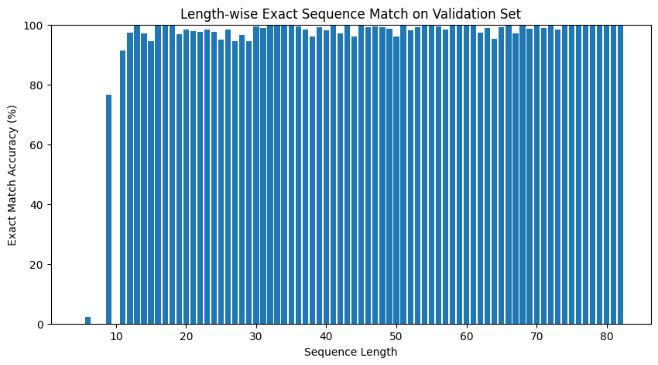


Figure 2. Length-wise Exact Sequence Match on Validation Set.

.

## Training

The model was trained for 30 epochs using cross-entropy loss, optimized with the Adam optimizer, with a learning rate initially set to 0.001 and a linear decay schedule after 10 epochs. We monitored both training and validation loss to ensure effective learning and prevent overfitting, using early stopping with a patience of 5 epochs. Gradient clipping with a maximum norm of 1.0 was applied to stabilize training. The training process was conducted on a GPU-enabled Colab environment, with a total training time of approximately 2 hours on an A100 GPU. Data augmentation techniques, such as random shuffling of non-critical path segments, were applied to enhance generalization.

# Experiments and Results

## Dataset Statistics

The dataset comprises approximately 10,000 paths, with path lengths ranging from 1 to 80 nodes. The distribution is skewed, with shorter paths (length 1-5) being less frequent, which impacts model performance for these cases..

## Model Performance

Figure 1 shows the training and validation loss over epochs, indicating that both losses decrease and converge, with the validation loss slightly higher, suggesting minimal overfitting. By the end of training, both losses are around 0.05, indicating effective learning.

Figure 2 presents the exact match accuracy for different sequence lengths. For sequences longer than 5, accuracy stabilizes at around 90-100%, demonstrating strong performance. However, for very short sequences (length 1-5), accuracy varies, with some cases as low as 0%, likely due to fewer training examples.

The final validation accuracy is 97.39%, indicating that the model correctly predicts the path for most cases, particularly for longer paths.

## Computation Time

We compared the computation time of the seq2seq model and Dijkstra's algorithm for a sample path (e.g., from node 26515 to 85718). The seq2seq model took 3924.49 ms, while Dijkstra's algorithm took only 2.71 ms, highlighting a significant efficiency gap. Figure 3 plots computation time against path length, showing a linear increase for the seq2seq model, while Dijkstra's remains constant.



Figure 1. Loss difference between Training and Validation.

.

# Discussion

Our results show that while the seq2seq model can achieve high accuracy in predicting paths, it is over 1400 times slower than Dijkstra's algorithm. This is due to the model's reliance on iterative sequence generation, which is inefficient for real-time applications. The model's performance on short paths is weaker, likely because there are fewer training examples for these cases, as evidenced by Table 1. However, given that users are more likely to query longer paths in practice, this limitation may be acceptable.

The trade-off between accuracy and computation time suggests that the seq2seq model is better suited for scenarios where precomputation is feasible, such as offline path planning. Future work could explore optimizing the inference speed, perhaps by using beam search or pruning techniques, or integrating the model with traditional algorithms to balance efficiency and accuracy.

Additionally, since the model is trained on static Dijkstra-generated data, it may not generalize to dynamic environments where the graph changes. This is a potential area for extension, aligning with related work on reinforcement learning for pathfinding in dynamic settings.

# Conclusion

We presented a transformer-based seq2seq model for indoor pathfinding, trained on Dijkstra-generated data, achieving a validation accuracy of 97.39%. While the model performs well, particularly for longer paths, its computation time is a significant drawback compared to Dijkstra's algorithm. This work serves as a proof of concept for using machine learning in static pathfinding and highlights the need for further research in optimizing inference speed for practical deployment.

# Github Link

The code for this project is available at our team's GitHub page. <https://github.com/H4RURAKA/AIProject_Team7>

# REFERENCES

1. Pathfinding – Wikipedia

<https://en.wikipedia.org/wiki/Pathfinding>

1. Grid Path Planning with Deep Reinforcement Learning: Preliminary Results – ScienceDirect

<https://www.sciencedirect.com/science/article/pii/S1877050918300553>

1. Finding shortest paths with Graph Neural Networks | by David Mack | Octavian | Medium

<https://medium.com/octavian-ai/finding-shortest-paths-with-graph-networks-807c5bbfc9c8>

1. Finding Shortest Path using Q-Learning Algorithm | by Doğacan Düğmeci | Medium

<https://medium.com/data-science/finding-shortest-path-using-q-learning-algorithm-1c1f39e89505>