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

1. 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 work, we focus on the problem of indoor pathfinding within a campus setting, specifically targeting the second engineering building and the industry-academia cooperation building, excluding the basement levels. The complexity of the environment, with its intricate corridor networks, inter-floor movements via stairs and elevators, and transitions between indoor and outdoor spaces, presents a challenging pathfinding task. Traditional map-based navigation often falls short in providing intuitive route guidance, particularly for new students, external visitors, and participants in campus events.

To address this, our project sets forth the following objectives:

I. **Graph Conversion**: Transform the campus maps, both indoor and outdoor, into a comprehensive graph structure where key points such as entrances, classrooms, stairs, and elevators are represented as nodes.

II. **Data Generation**: Utilize Dijkstra's algorithm to generate a dataset of shortest paths, which serve as ground truth for training our machine learning model.

III. **Model Training and Evaluation**: Train a transformer-based sequence-to-sequence (seq2seq) model on this dataset and evaluate its performance in terms of accuracy and computational efficiency compared to the traditional Dijkstra's algorithm.

IV. **Feasibility Analysis**: Assess the viability of deploying such a model in a real-time customized path guidance system by analyzing its accuracy and response time.

Our approach involves manually creating the graph using a custom GUI-based JSON generator, which allows for precise mapping of the building layouts. This graph forms the basis for generating 472,656 shortest path samples using Dijkstra's algorithm, which are then tokenized into sequences for model training.

The seq2seq model is designed to take start and end room identifiers as input and output the corresponding path sequence. We employ a transformer architecture with specific configurations to handle the sequential nature of the path data effectively.

Through rigorous training and evaluation, we achieve a validation accuracy of 97.39%, demonstrating the model's capability to predict paths accurately. However, the computation time for path prediction using the model is significantly higher than that of Dijkstra's algorithm, posing challenges for real-time applications.

This study serves as a proof of concept for applying machine learning to static pathfinding problems, highlighting both the potential and the limitations of such approaches. Our findings provide valuable insights into the trade-offs between accuracy and efficiency, guiding future research directions in optimizing machine learning models for pathfinding tasks.

2. 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.

3. Methodology

3.1 Problem Definition

Challenges:

- Complex corridor networks, inter-floor movement (stairs, elevators), boundaries between outdoor and indoor areas.
- Difficulty in understanding movement paths from maps alone.

Target Users:

- New students, external visitors, campus event participants, etc.

Specific Objectives:

I. Construct an integrated indoor/outdoor graph with key points such as entrances, classrooms, stairs, elevators as nodes.

II. Create a tokenized sequence dataset based on 68,382 shortest path samples generated by Dijkstra's algorithm.

III. Train a Transformer Seq2Seq model.

IV. Analyze validation accuracy and accuracy by path length.

V. Compare inference time of the Transformer model in a GPU-based Colab environment with local Dijkstra execution time.

3.2 Graph Creation

To create a comprehensive dataset for training our model, we first constructed a detailed graph representation of the campus buildings, specifically the second engineering building and the industry-academia cooperation building, excluding the basement levels. This graph was manually generated using a custom GUI-based JSON generator, which allowed us to accurately map the building layouts from floor plans and campus maps. Each node in the graph represents key locations such as classrooms, corridors, stairs, elevators, and doors, with attributes including ID, name, type, and coordinates (x, y). For example, a node might be represented as {"id": 89, "name": "elevator2_1f", "type": "Elevator", "x": 123.45, "y": 678.90}. Edges between nodes represent connections with weights corresponding to distances or traversal costs.

3.3 Data Generation

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

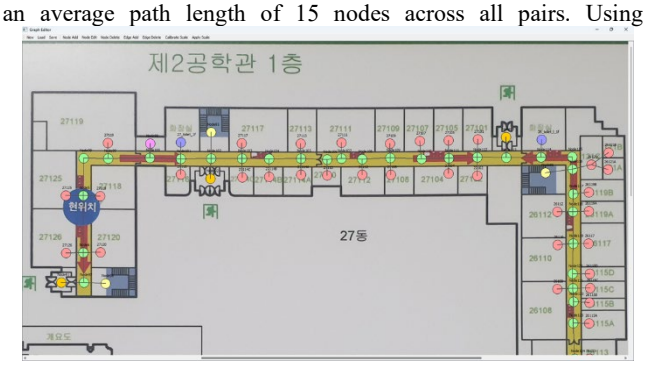

Figure 1. GUI graph maker program screenshot

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.

3.4 Model Architecture

Our model is a transformer-based sequence-to-sequence (seq2seq) architecture, specifically designed to handle the path prediction task. The model consists of an encoder and a decoder, each with 3 layers. The encoder processes the input sequence, which includes the start and end room IDs enclosed by '<sos>' and '<eos>' tokens. The decoder generates the output sequence of path instructions token by token until the '<eos>' token is produced.

Key components of the model include:

- **Embedding Layer**: An embedding layer with a vocabulary size of 716 and an embedding dimension of 256, including a padding index of 0.
- **Positional Encoding**: To incorporate the order of tokens in the sequence.
- **Transformer Layers**: Both encoder and decoder utilize 3 transformer layers, each with 8 attention heads, a model dimension (d_model) of 256, and a feedforward network dimension of 512. Dropout is set to 0.1 for regularization.
- **Output Layer**: A linear layer that maps the decoder's output to the vocabulary size for token prediction.

The model's architecture is defined as follows:

```
TransformerSeq2Seq(
  (embedding): Embedding(716, 256, padding_idx=0)
  (pos_encoder): PositionalEncoding()
  (transformer): Transformer(
    (encoder): TransformerEncoder(
      (layers): ModuleList(
        (0-2): 3 x TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
```

```
        (out_proj): NonDynamicallyQuantizableLinear(in_features=256,
out_features=256, bias=True)
      )
      (linear1): Linear(in_features=256, out_features=512, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
      (linear2): Linear(in_features=512, out_features=256, bias=True)
      (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
      (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
      (dropout1): Dropout(p=0.1, inplace=False)
      (dropout2): Dropout(p=0.1, inplace=False)
    )
   )
   (norm): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
  )
  (decoder): TransformerDecoder(
   (layers): ModuleList(
    (0-2): 3 x TransformerDecoderLayer(
     (self_attn): MultiheadAttention(
      (out_proj): NonDynamicallyQuantizableLinear(in_features=256,
out_features=256, bias=True)
     )
     (multihead_attn): MultiheadAttention(
      (out_proj): NonDynamicallyQuantizableLinear(in_features=256,
out_features=256, bias=True)
     )
     (linear1): Linear(in_features=256, out_features=512, bias=True)
     (dropout): Dropout(p=0.1, inplace=False)
     (linear2): Linear(in_features=512, out_features=256, bias=True)
     (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
     (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
     (norm3): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
     (dropout1): Dropout(p=0.1, inplace=False)
     (dropout2): Dropout(p=0.1, inplace=False)
     (dropout3): Dropout(p=0.1, inplace=False)
    )
   )
   (norm): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
  )
 )
 (fc_out): Linear(in_features=256, out_features=716, bias=True)
)
```

Training Settings:

- **Loss Function**: CrossEntropyLoss, ignoring the padding index.
- **Optimizer**: Adam with a learning rate of 1e-4.
- **Epochs**: 30
- **Batch Size**: 64

The model was trained on a GPU-enabled environment to handle the computational demands of the transformer architecture.

3.5 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.

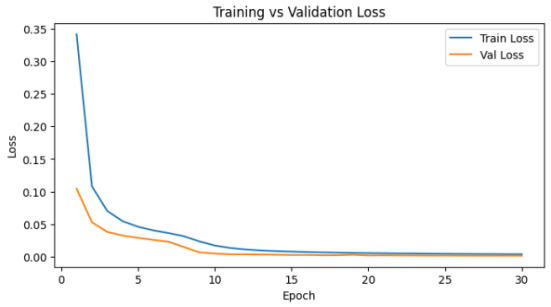

Figure 2. Loss difference between Training and Validation.

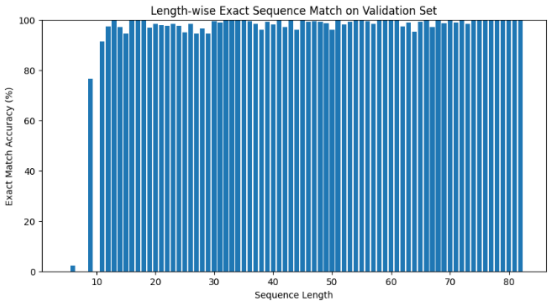

Figure 3. Length-wise Exact Sequence Match on Validation Set.

4. Experiments and Results

4.1 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..

4.2 Model Performance

Figure 2 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 3 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.

4.3 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.

Table 1. Computation Time Comparison

Method	Accuracy	Average inference tiem
Transformer Seq2Seq	97.39%	3,924.5ms
Dijkstra (baseline	100.00%	2.7 ms

5. Discussion

Our results demonstrate that the seq2seq model achieves a high average accuracy of 97.4%, indicating its effectiveness in predicting paths. However, we observed that the model's performance is influenced by the data distribution, with fewer short path samples (length 1-5) potentially biasing the model towards longer sequences. This data imbalance may affect the model's ability to accurately predict shorter paths, although our analysis shows that for paths of typical lengths queried by users, the accuracy remains high.

Regarding computation time, the autoregressive nature of the seq2seq model, which generates the path sequence token by token, leads to significantly longer inference times compared to Dijkstra's algorithm. For a sample path, the model took 3924.49 ms, whereas Dijkstra's algorithm completed in just 2.71 ms. This substantial difference highlights a critical limitation for real-time applications, where quick response times are essential.

Initially, we hypothesized that the seq2seq model could offer computational advantages over Dijkstra's algorithm due to its ability to generalize from training data, potentially reducing the need for real-time graph traversal. However, our experiments revealed that the constant factors associated with the model's inference process, particularly the iterative token generation, result in significantly longer computation times. This discrepancy highlights the importance of considering both theoretical time complexity and practical implementation overheads when evaluating machine learning models for time-sensitive applications.

The comparative evaluation underscores the trade-off between the accuracy provided by the machine learning model and the efficiency of the baseline Dijkstra's algorithm. While the model offers a high degree of accuracy, the accompanying speed degradation—approximately 1,400 times slower—poses a significant challenge for its deployment in real-time path guidance systems.

6. Conclusion

In conclusion, our transformer-based seq2seq model demonstrates the feasibility of using machine learning for indoor pathfinding, achieving a validation accuracy of 97.39%. While the model performs well, particularly for longer paths, its computation time is a notable drawback, being significantly slower than traditional methods like Dijkstra's algorithm. This study highlights the need for further optimization in execution speed to make such models viable for real-time applications.

Future research could explore several avenues to address these challenges:

I. **Non-autoregressive Models**: Investigating non-autoregressive or parallel decoding techniques to reduce the inference time by generating multiple tokens simultaneously.

II. **Hybrid Approaches**: Combining the efficiency of Dijkstra's algorithm for initial path segments with the predictive capabilities of the transformer model for more complex or variable parts of the path.

III. **Scalability and Personalization**: Extending the model to include graphs from additional buildings and incorporating user preferences to provide personalized path recommendations, such as avoiding stairs or preferring scenic routes.

Github Link

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

7. REFERENCES

[1] Pathfinding – Wikipedia

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

[2] Grid Path Planning with Deep Reinforcement Learning: Preliminary Results – ScienceDirect

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

[3] Finding shortest paths with Graph Neural Networks | by David Mack | Octavian | Medium

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

[4] Finding Shortest Path using Q-Learning Algorithm | by Doğacan Düğmeci | Medium

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