Reproducibility and Improvement of LPFormer via Shortest-Path Encodings

Selin Ceydeli
s.ceydeli@student.tudelft.nl

Lemon He
j.he-19@student.tudelft.nl

I. Introduction

Graph representation learning has become a fundamental tool in machine learning, enabling models to reason over complex relational structures. In particular, link prediction (LP) plays a key role in many applications, including recommender systems, social networks, and drug discovery. LPFormer [4] introduces a transformer-based architecture designed to integrate a broader range of pairwise features for the link prediction task. It proposes a general mechanism for pairwise encodings and leverages the Personalized PageRank (PPR) [1] scores for relative positional encodings. Extensive experiments demonstrate that LPFormer can achieve SOTA performance on a wide variety of benchmark datasets while retaining efficiency.

Given LPFormer's promising results and the growing interest in attention-based graph models, we selected this paper for reproduction. In this report, we reproduce LPFormer's training pipeline and evaluate its performance on the Citeseer and PubMed datasets. Furthermore, we propose an alternative approach to node selection using pairwise shortest-path distance instead of PPR, motivated by its deterministic nature and potential efficiency. We conducted experiments using this alternative node selection method to assess the effectiveness of our proposal [1].

II. Problem Description

The authors of the LPFormer article propose modeling pairwise encodings between the nodes using an attention mechanism. However, using full attention is computationally expensive and does not scale well to large graphs, since the number of nodes can become huge. Hence, it is important to perform node selection so that the attention mechanism can be effectively applied to larger graphs.

In the original LPFormer paper, the important set of nodes is mathematically modeled to be the nodes that have an importance $I(a, b, u)$ that is higher than a certain threshold. The authors remark that for ease of optimization and efficiency, they choose to proxy this importance factor with the PPR score. To our understanding, this choice of approximating the importance factor with the PPR score is a choice made for keeping the model simpler, and hence, efficient.

PPR score, also known as the Personalized PageRank score, is a measure of how likely a random walk starting from node a will reach the node under investigation, node u [4]. It uses a parameter α to control the probability of a restart. The recursion in the PPR score represents the following:

- With probability α, it restarts at the source node a.
- With probability $1 - \alpha$, it follows a random edge.

We saw a potential for improvement in the selection of the nodes based on their importance. In our project, we propose to use pairwise shortest-path distance between nodes as a proxy for node importance for the multi-hop nodes. This way, we aim to improve the node selection aspect and enhance the model accuracy.

We propose to use this pairwise shortest-path distance for node selection for multi-hop nodes and exclude it for common neighbors (CN) and 1-hop nodes because the shortest path distance for CN and 1-hop nodes is trivial. In the former case, the distance is always zero. In the latter case, it is always one. On the contrary, for the multi-hop nodes, the structural proximity of a given node u to the source and target nodes (a, b) is important to be captured to understand the influence of a node u when deciding on the encoding of the edge between nodes (a, b).

We believe that using pairwise shortest-path distance instead of PPR score for the selection of important multi-hop nodes has the following benefits:

- Shortest-path distance directly reflects the structural proximity of a given node to both the source and target nodes, and hence, is potentially an important metric to measure the importance of a node u to the target link (a, b).
- The shortest-path distance matrix can be precomputed using a shortest path algorithm, such as Dijkstra, and hence, does not increase the complexity of the training loop.
- Unlike PPR, which is a probabilistic approach, the shortest-path distance calculation offers clean and deterministic measurements of pairwise node positions and thus is ensured to capture the concrete distance information of two nodes.

In conclusion, we propose to use pairwise shortest-path distances between nodes as a proxy for node importance for the multi-hop nodes in order to improve node selection.

III. Related Work

Devin et al. [2] investigate the power of graph Transformers by introducing a learned positional encoding (LPE) designed

[1]Link to our GitHub repository: https://github.com/selinceydeli/LPFormer

to capture the structural roles of nodes within a graph. Through extensive experiments on graph classification and distinguishing structurally similar graphs, they demonstrate that such structural encodings play a crucial role in enhancing the expressiveness of graph neural networks.

Specifically, utilizing shortest-path-based positional encodings for identifying important nodes for link prediction in graph representation was found to be important by Ying et al. [5]. Through the use of these encodings, they aimed to make the Transformer architecture suitable for modeling graphs and graph representation learning. The authors introduce spatial encoding to capture the structural relations between nodes. These spatial encodings are computed based on the shortest-path distance between node pairs as a way to inject graph structural information into the Transformer's attention mechanism. These papers motivated us to utilize pairwise distances for node selection.

IV. Methodology

A. Training Architecture

Unlike conventional single-seed training, LPFormer employs a two-stage training strategy, as illustrated in Algorithm 1. In stage 1, the model is trained multiple times, each with a different random seed. The final output of stage 1 is the distribution of the final output from each run (represented by mean ± standard deviation). The goal is to identify the run with the best validation performance, as results may vary significantly due to randomness in initialization or sampling. In stage 2, the model is trained for the full number of epochs, following a standard single-run training procedure.

As discussed in section II, the probabilistic nature of the PPR score introduces randomness to the training process. Thus, it is important to repeat the result several times to mitigate the effect of randomness. By explicitly setting the random seed in each run, the outcomes become reproducible, thereby supporting consistent reporting and experimental integrity. Finally, decoupling the seed selection from final training improves clarity and structure compared to simply selecting the best test score post hoc.

Additionally, we identified three areas where the original implementation could be improved to enhance reproducibility and transparency. First, the original experiment pipeline computed the final test score at the end of training without saving intermediate model checkpoints corresponding to the highest validation performance. While this may save some communication overhead, it results in a lack of reproducibility. To address this, we incorporated an explicit checkpoint-saving logic in the training process, enabling recovery and evaluation of the best model directly from local storage.

Secondly, we observed that the logging mechanism was ambiguous. Specifically, the misuse of `tqdm` for progress display caused logs to be flushed before they could be reliably captured, making it difficult to trace training dynamics. We made modifications to the logging setup to guarantee a trackable and clear log.

Algorithm 1 train_data

```
procedure TRAIN_DATA
    for each seed in seeds do {— Stage 1—}
        best_model ← None
        best_score ← −∞
        for epoch = 1 to N_epochs do {— Stage 2—}
            TRAIN_EPOCH
            if epoch mod eval_steps == 0 then
                TEST
                if current_score > best_score then
                    best_score ← current_score
                    best_model ← current_model
                end if
            end if
            if EARLY_STOPPING_TRIGGERED then
                break
            end if
        end for
        SAVE(best_model)
    end for
    AGGREGATE_RESULTS
end procedure=0
```

Lastly, the authors set up the logging such that the best validation and test scores of the training are printed only once, after the training is completed entirely. We updated the logging architecture to log the best validation score obtained so far in the training loop, logging it after each epoch. Again, with this update, we aimed for a more trackable and clear log.

B. Dense Matrix Computation for Pairwise Distance

The pseudo-code for computing the dense matrix representation for storing pairwise distances between the nodes of the input graph is described in Algorithm 2. This implementation creates a full NumPy array of size num_nodes x num_nodes, filling in the matrix with:

- a finite distance if a path exists between node i and j
- an infinity value otherwise

This initial implementation of storing pairwise distances has its limitations because we are allocating space for every possible node pair, even though some entries store infinity values, which are not meaningful for the analysis. Hence, this dense matrix representation results in performance bottlenecks because the operations become slow and inefficient, especially for larger graphs.

An improvement we introduced to overcome the performance bottleneck was to compute the distance matrix once and store it for later uses. Hence, the actual call to the shortest path algorithm for computing the matrix is done only once.

Nevertheless, for each call to our node selection algorithm, the matrix is traversed entirely to find the node pairs that are below a certain threshold, resulting in a square computational complexity. This is reflected in the runtime results obtained from this implementation, as discussed in subsection VI-B.

Algorithm 2 Compute Pairwise Distance Matrix

procedure COMPUTEDISTMATRIX(data_name, save_dir)
  Create directory `save_dir` if it doesn't exist
  Set `dist_path` ← `save_dir` + "/{`data_name`}_dist_matrix.npy"
  if `dist_path` exists and `force_recompute` is False **then**
    Load and return distance matrix from `dist_path`
  end if
  `num_nodes` ← data['num_nodes']
  `edge_index` ← data['edge_index']
  Initialize empty undirected graph G
  Add `num_nodes` nodes to G
  Add edges from `edge_index` to G
  `lengths` ← shortest path lengths between all node pairs in G using NetworkX
  Initialize `dist_matrix` with shape $[\texttt{num\_nodes}, \texttt{num\_nodes}]$ filled with ∞
  for $i = 0$ to `num_nodes` $- 1$ **do**
    for all (j, d) in `lengths`$[i]$ **do**
      `dist_matrix`$[i, j] \leftarrow d$
    end for
  end for
  Save `dist_matrix` to `dist_path`
  return `dist_matrix`
end procedure=0

Algorithm 3 Convert Dense Matrix to Sparse Format

procedure CONVERTTOSPARSEMATRIX(dist_matrix) {Convert precomputed dense matrix to sparse representation}
  Replace ∞ values in `dist_matrix` with 0
  (`row_idxs`, `col_idxs`) ← indices where `dist_matrix` $\neq 0$
  `values` ← `dist_matrix[row_idxs, col_idxs]`
  `sparse_matrix` ← sparse COO tensor constructed from (`row_idxs`, `col_idxs`, `values`)
  Save `sparse_matrix` to disk (e.g., NumPy file)
  return `sparse_matrix`
end procedure=0

Algorithm 4 Use Sparse Distance Matrix During Training

procedure LOADANDUSESPARSEMATRIX(sparse_path)
  Load `sparse_matrix` from disk
  for all entries (i, j, d) in `sparse_matrix` **do**
    $d \leftarrow d + 1$ {Apply transformation to ensure all data is loaded to dense matrix}
  end for
  Convert `sparse_matrix` back to dense format
  Filter node pairs using threshold as in dense implementation
  Use filtered results for downstream tasks
end procedure=0

C. Sparse Matrix Computation for Pairwise Distance

To address the inefficiencies arising from using the pairwise distances in dense matrices, we adopt a sparse pairwise distance representation inspired by the PPR score storage strategy used in the original LPFormer paper. The key idea if that for many real-world graphs, the number of nodes with meaningful(finite and small) distances is significantly smaller than the total number of pairs. Therefore, instead of allocating memory for all entries, we only store distances that are meaningful. The pseudocode in Algorithms 3 and 4 outlines our improved approach for handling pairwise distance matrices more efficiently:

1) During the one-time computation phase at data loading time, the full dense pairwise distance matrix is first calculated. We then convert it into a sparse format by removing infinite entries and storing only nonzero values (i.e., finite distances) as a sparse COO tensor. This sparse representation is saved to disk as a tensor for future use (Algorithm 3).
2) During training, since direct access to individual elements in sparse format is limited, we convert the sparse matrix back to dense form before filtering node pairs based on a distance threshold, as done in the original dense version (Algorithm 4). The plus-one operation after matrix loading ensures a correct conversion.

This approach drastically reduces both the memory and time required for downstream computations. For instance, the node selection algorithm now traverses only the non-infinite entries of the sparse distance matrix, leading to sub-quadratic complexity in practice, which can be further proved in subsection VI-B.

V. EXPERIMENTAL SETTINGS

The link prediction experiments we aim to reproduce focus on evaluating the model's ability to infer missing or future connections between nodes in a graph. To ensure reproducibility, we strictly followed the initial setup from the paper, which will be described below.

A. Dataset Selection for Reproducing the Results

The original study validated the effectiveness of LPFormer through link-prediction experiments, benchmarking its performance against baseline models. As outlined in the results table in the original paper (see Appendix C for this table), the experiments spanned six datasets, with Cora, Citeseer, and PubMed being the smaller-scale datasets, which are computationally easier to handle. Among these, the results for Cora showed minimal performance variation across models, making it less insightful for comparative analysis. Consequently, we exclude Cora from our reproduction efforts. Instead, we focus on Citeseer and PubMed due to their distinct experimental outcomes. For these datasets, we experiment under a single fixed split, as was also done in the original paper.

B. Evaluation Metrics

To evaluate the reproduction and improvement results on the Citeseer and PubMed datasets, we adopt the same evaluation method used in the original study. Specifically, each positive target link is ranked against a set of negative samples, and the position of the positive link within this ranked list is used to assess performance. Following the original paper, we report the Mean Reciprocal Rank (MRR) as the primary evaluation metric for the two datasets.

MRR is a metric used to evaluate the effectiveness of retrieval systems by measuring how well they rank the first relevant document for a set of queries [3]. Using an MRR score is preferred over an accuracy score because link prediction often involves ranking potential links (edges) between nodes, making MRR a good candidate for evaluating the performance of the model.

C. Experimental Setup

1) Reproduction Setup: All experiments are conducted on DelftBlue. For the Citeseer dataset, we trained our model for 100 epochs per run, as was done by the authors originally to achieve the results described in the results table of the original paper (see Appendix C for this table).

To reduce the effect of random initialization and stochastic training dynamics, we repeated the experiment 10 times (ran the same experiment for 10 runs), the same as the original paper did. During our runs, the exact training configurations were used with the authors.

For the PubMed dataset, we again trained our model for 100 epochs per run, as was done by the authors. However, due to the difficulties with batch scripting (see section VIII) and the higher computational needs of the PubMed dataset, we could only repeat the experiment 2 times, each with only one run. Again, during our runs, the exact training configurations were used with the authors.

2) Improvement Setup: Since the computation of the pairwise distances, as well as accessing them during the training loop, is much more expensive than the PPR scores (see subsection VI-B for runtime results), the improvement results are tested under different settings.

For the Citeseer dataset, we again trained our model for 100 epochs per run. We averaged the results across 2 runs. For the PubMed dataset, we were limited to 20 epochs per run due to resource constraints, as detailed in section VIII. Again, the results are averaged across 2 runs. Besides the changes in the number of epochs and the number of runs, the remaining training configurations were kept the same as the original configurations.

VI. Results

A. Reproduction

We evaluate the reproducibility of LPFormer on Citeseer and PubMed, focusing on both test accuracy and training stability across seeds. The experimental settings are described in section V.

TABLE I
COMPARISON OF TEST MRR SCORES ON CITESEER AND PUBMED DATASETS.

Setting	Citeseer (MRR)	PubMed (MRR)
Trial 1 (ours)	52.95 ± 13.53	32.22 ± nan
Trial 2 (ours)	52.51 ± 13.04	34.92 ± nan
Average across trials (ours)	52.73 ± 13.29	33.57 ± 1.35
Original (from paper)	65.42 ± 4.65	40.17 ± 1.92

TABLE II
VALIDATION AND TEST METRIC COMPARISON ON CITESEER AND PUBMED

Trial	Citeseer	PubMed
Trial 1 (Valid)	67.62 ± 2.89	42.97 ± nan
Trial 1 (Test)	52.95 ± 13.53	32.22 ± nan
Trial 2 (Valid)	68.21 ± 3.17	44.30 ± nan
Trial 2 (Test)	52.51 ± 13.04	34.92 ± nan
Average across trials (Valid)	67.92 ± 3.03	43.64 ± 0.67
Average across trials (Test)	52.73 ± 13.29	33.57 ± 1.35

Table I summarizes the test scores of LPFormer on the Citeseer and PubMed dataset. For each dataset, we conducted two trials to mitigate the randomness and compare our results with the one from the original paper. Each cell represents the distribution of the aggregated output metric in the form of $mean \pm standard\ deviation$. As shown, our test scores for both datasets are lower than the original results reported by the authors. On Citeseer, the average of the reproduced test scores showcases a drop from the original score of 65.42 to 52.73, with a high variance of 13.29 across trials. PubMed shows a similar trend, with test metrics decreasing from 40.17 to approximately 33.57.

To better understand this discrepancy, Table II presents the highest validation score and the final test score of each trial for our reproduction. Notably, both trials achieved metrics above 67 on Citeseer and above 42 on PubMed, which aligns with the test score from the original results. However, the discrepancies between the test and validation results remain greater than 10 points. This indicates a generalization gap, potentially caused by overfitting or sensitivity to random seed initializations.

It is also worth noting that, since the PubMed experiment was limited to one run per trial due to computational resource limitations (as detailed in section VIII), the standard deviation per run was not calculated, thus, it is indicated by nan. For the average results described in Table I and Table II, the standard deviation was computed using the two trial results.

B. Accelerating the Training

In our initial custom implementation, pairwise distances between nodes are stored in a dense matrix (see subsection IV-B. While this approach is straightforward, it results in a significant computational and memory overhead, particularly for larger graphs.

As illustrated in Table III, each epoch in the dense matrix implementation of the pairwise distance takes, on average, 4.08 minutes to complete for the Citeseer dataset, resulting in a 100-times longer runtime. Similarly, the per-epoch runtime

TABLE III
PER-EPOCH RUNTIME COMPARISON ON CITESEER AND PUBMED DATASETS

Method	Citeseer	PubMed
PPR Score (Original)	0.04 min/epoch	3.05 min/epoch
Dense Matrix (Pairwise Distance)	4.08 min/epoch	71.31 min/epoch
Sparse Matrix (Pairwise Distance)	1.15 min/epoch	42.25 min/epoch

for PubMed increases by a factor of 25 over the original implementation, which utilizes a sparse matrix for PPR computation.

It should be noted that the impact of using a dense matrix is more pronounced on smaller graphs, such as the Citeseer dataset, compared to larger graphs, such as the PubMed dataset. Nevertheless, for both datasets, the increase in computational time is very significant. This inefficiency in our initial custom implementation reduces the scalability of our approach.

To address this inefficiency in computation time, we replaced the dense matrix with a sparse representation when computing the pairwise distances. With this update, the computation time is reduced from 4.08 min/epoch to 1.15 min/epoch on the Citeseer dataset, achieving a reduction by a factor of 4. Similarly, the computation time is reduced by 40% for the PubMed dataset.

Although with the sparse matrix implementation we achieved improvements over our dense matrix method, the original implementation by the authors still outperforms our custom approach in terms of runtime.

In the remaining sections of this report, the results of our improvement will be presented for the sparse matrix implementation used for encoding the pairwise distances between the nodes.

C. Improvement

We tested our improvement on the Citeseer and PubMed datasets, again focusing on both test accuracy and training stability across seeds. Table IV summarizes the test and validation scores of the implementation that utilizes sparse matrix pairwise distances, comparing them against the reproduction results we achieved. Table V demonstrates the relative improvement of our custom pairwise distance implementation for node selection over the reproduction results. The reproduction results described in Table IV are taken from the average validation and test results depicted in Table II.

The results for the Citeseer dataset show a remarkable increase in both the validation and test scores as a result of using pairwise distance information for node selection. On Citeseer, the validation score improved from 67.92 to 75.55, and similarly, test score improved from 52.73 to 59.90, showing an 11.23% improvement in the validation score and a 13.60% improvement in the test score. These results demonstrate better generalization of the transformer model when node selection is conducted using shortest-path information.

Despite the significant improvements achieved on the Citeseer dataset, we failed to improve on the PubMed dataset. The custom results show a 69.41% reduction in the validation score and a 75.22% reduction in the test score when compared against the reproduction results.

TABLE IV
VALIDATION AND TEST SCORES (REPRODUCTION — CUSTOM) FOR CITESEER AND PUBMED DATASETS.

Dataset	Metric	Reproduction	Custom
Citeseer	Validation Score	67.92 ± 3.03	75.55 ± 0.53
	Test Score	52.73 ± 13.29	59.90 ± 1.08
PubMed	Validation Score	43.64 ± 0.67	13.35 ± 1.05
	Test Score	33.57 ± 1.35	8.32 ± 0.98

TABLE V
RELATIVE IMPROVEMENT (%) OF THE CUSTOM IMPLEMENTATION OVER THE REPRODUCTION RESULTS.

Dataset	Metric	Improvement (%)
Citeseer	Validation Score	+11.23
	Test Score	+13.60
PubMed	Validation Score	-69.41
	Test Score	-75.22

VII. DISCUSSION

A. Reproduction

As discussed in subsection VI-A, the final test MRR scores achieved during reproduction remain almost 13 points lower for the Citeseer dataset than the original results communicated in the paper. Similarly, the difference is almost 8 points for the PubMed dataset. The best validation scores we achieved with our reproduction are close to the original MRR scores described in the paper. Although it is not stated explicitly what type of score the authors communicate in their results table, we assume that the results they describe are the test scores they achieved during the trainings.

Although we used the exact same training configuration as the authors described for their training setup, as well as ran the training for 100 epochs the same as the authors did, and finally averaged the results across 10 runs to account for the stochasticity of the training, achieving significantly lower training results might have several reasons.

A big issue with Citeseer, among other datasets, is the high performance variance. This high variance is visible not only in the results we replicated but also in the original results (see Table I and Table II as well as Appendix C for the reported results in the original paper). This by itself makes replicating the results difficult.

Another possible issue is with random seeds. The authors have explicitly set them to be the same for everyone in each run. While technically, they should control for randomness in various aspects, in reality it does not work that way with PyTorch due to various reasons (e.g., non-deterministic CUDA algorithms).

B. Improvement

As compared in subsection VI-C, the improved method achieved an approximately 8-point increase over the original method in both the validation score and test score on the

Citeseer dataset. Although the final test score remains lower than the one reported in the original paper, the gain represents a significant advancement. Moreover, the standard deviation is drastically reduced by nearly a factor of 10, suggesting a much greater stability across each seed. These results are in accord with our hypothesis that incorporating pairwise distance can improve both the test accuracy and training stability.

In contrast, the outcome on the PubMed dataset, illustrated in Table IV, deviates notably from the Citeseer results. The improved method yields significantly lower scores than the original reproduction. However, this is expected due to a key experimental constraint: we could only train the PubMed for 20 epochs on the improved setup, while the number of epochs used in the original method is 100 as discussed in section VIII. This discrepancy makes a direct comparison invalid. For this reason, we limit our interpretation to the Citeseer dataset, where fair conditions are maintained.

There are several possible explanations for the observed improvement on Citeseer:

1) Firstly, we included all kinds of nodes (CN, 1-hop, non-1-hop) to reconstruct the nodes in Citeseer when reconstructing node embeddings. In contrast, the original method considered only 1-hop neighbors. This broader context likely enhanced the model's capacity to capture structural relationships between nodes.
2) We replaced the probabilistic PPR scores with deterministic pairwise shortest-path distances for selecting relevant neighbors. While PPR is computationally efficient, its inherent randomness may introduce variance during training. In contrast, the shortest-path distance yields a fixed neighborhood structure, contributing to the improved stability across seeds.

That being said, a notable side effect of introducing pairwise distance is the significant increase in per-epoch training time, as shown in Table III. Initially, we attributed this overhead to the use of dense matrices for storing distance values. After switching to sparse matrix representations, the training time was greatly reduced. However, even with sparse matrices, the method remains nearly 100 times slower than the PPR-based baseline.

A possible explanation for this residual gap lies in the differing sparsity patterns and neighborhood sizes of the PPR and pairwise distance matrices. While both representations undergo similar storage and loading procedures, PPR scores tend to yield more sparse matrices due to the rapid decay and probabilistic formulation, which means only a small number of nodes receive meaningful scores, thus shrinking the size of the sparse matrix representation. In contrast, pairwise matrices usually produce larger neighborhoods, even when thresholded. This leads to more extensive computations during node filtering and selection. Future work could explore adaptive thresholding to combine the determinism of the distance-based method with the sparsity advantage of PPR.

VIII. Challenges Encountered During Experiments

Before reproducing the results, we had to configure a working environment for our experiments. The requirements file that the authors prepared only contained half of the required libraries for training their models. We had to manually install six libraries into our conda environment, which are outlined in Appendix A.

Furthermore, the link to the datasets they provided on their GitHub repository was out of date. We had to contact the authors via GitHub issues to gain access to the datasets to be able to reproduce the results.

To train our models, we utilized the DelftBlue Computing Cluster, as the project specifies a maximum storage requirement of 32GB, which is difficult to meet on local machines. According to DelftBlue's documentation, there are two main modes for executing experiments: the interactive session (typically used for quick testing and debugging) and the batch session (recommended for full training runs with large epochs).

While our code ran smoothly in the interactive session, we encountered different issues when attempting to use the batch session, as outlined in the script attached in Appendix B. The primary challenge stemmed from incomplete documentation for DelftBlue usage, making debugging difficult. Additionally, the queue times for batch jobs were extremely long, typically requiring ≈ 10 hours of waiting before a job would begin. After more than seven unsuccessful attempts, we decided to abandon the batch mode altogether.

Instead, we proceeded by running our training through interactive sessions via SSH, which, although functional, significantly limited the number of training runs we could execute due to the time constraints of 18 hours per connection. As a result, and as described in subsection V-C, we opted to run the PubMed dataset experiment twice separately, and with fewer epochs, rather than as part of a single automated batch workflow.

IX. Conclusion

In this project, we explored an alternative node selection strategy for the LPFormer by replacing the PPR-based selection with the pairwise shortest distance approach. Our aim was to improve the model accuracy of the link prediction experiments and reduce randomness to improve training stability.

To achieve this, we implemented both dense and sparse matrix representations of pairwise shortest-path distances and integrated them into the original LPFormer training pipeline. Our improvements demonstrated a significant gain in both validation and test accuracy on the Citeseer dataset, alongside a substantial reduction in variance across training runs. These results validate our hypothesis that using deterministic structural information leads to better generalization and consistency.

Overall, our work provides a reproducible and interpretable extension to LPFormer, and sets the stage for future work on stable neighborhood encoding strategies in graph transformers.

X. Limitations and Future Work

The main limitation of this study stems from resource limitations. Due to technical difficulties with setting up the batch sessions on the supercomputer, we were forced to test our improvements using the interactive session. Although functional, interactive sessions significantly limited the number of training runs we could execute due to the time constraints of 18 hours per connection. This limitation had a direct impact on the number of epochs we could run on the larger dataset, PubMed.

Due to the significant increase in runtime when we introduced the pairwise distance-based node selection rather than utilizing PPR scores, we managed to train our improved model for only 20 epochs on the PubMed dataset. The original implementation, however, had a training loop of 100 epochs. This decrease in the number of training epochs impacted the model quality, possibly producing an underfitted model. We acknowledge that our improvement results for the PubMed dataset would potentially be higher if we managed to run it for longer epochs, enabling the model to better learn from data.

Another limitation of our study was the number of runs we could perform when reproducing the original results and when testing our improvement. For the Citeseer dataset, we could test the reproduction for 10 runs, whereas when testing our improvement, we could only run it for 2 runs. For the PubMed dataset, both the reproduction and improvement could be run only for 2 runs. The drawback of having fewer number of runs is that it increases the impact of randomness in our results.

The future work should focus on repeating the experiments discussed in this report by running each model training loop for 100 epochs, and averaging the results for at least 10 runs. These settings would (1) enable the transformer model to better learn from data and (2) reduce the impact of randomness in the presented results.

A further line of investigation would be conducting statistical testing on the presented results to show the statistical significance of the improvements achieved by the usage of shortest-path distance information when selecting the important nodes.

With these improvements on our current work, the comparisons that are made between the original approach using PPR scores and our introduced improvement using pairwise distances would be more reliable and statistically meaningful.

XI. Acknowledgment

This paper discusses a possible improvement on the *LPFormer: An Adaptive Graph Transformer for Link Prediction* [4] paper, specifically focusing on the important node selection approach of the original paper. We forked the GitHub repository created by the authors of the LPFormer paper, and all our improvements were conducted on this repository.

We would like to thank the instructors and teaching assistants of the Scalable Learning Systems seminar for their valuable feedback, guidance, and support throughout this project.

References

[1] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized pagerank over distributed main memory databases. *CoRR*, abs/1006.2880, 2010.

[2] Devin Kreuzer, Dominique Beaini, William L. Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *CoRR*, abs/2106.03893, 2021.

[3] Xiangkui Lu, Jun Wu, and Jianbo Yuan. Optimizing reciprocal rank with bayesian average for improved next item recommendation. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 2236–2240, New York, NY, USA, 2023. Association for Computing Machinery.

[4] Harry Shomer, Yao Ma, Haitao Mao, Juanhui Li, Bo Wu, and Jiliang Tang. Lpformer: An adaptive graph transformer for link prediction. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 2686–2698. ACM, August 2024.

[5] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform bad for graph representation? In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc.

APPENDIX A: DEPENDENCY ISSUE

To configure a working environment and resolve dependency issues, we made the following changes:

- **scikit-learn:** The package name `sklearn` was corrected to `scikit-learn` in the `requirements.txt` file.
- **OGB - PygLinkPropPredDataset:** An import error occurred due to the missing symbol `PygLinkPropPredDataset` in the module `ogb.linkproppred`. Since this class was not used anywhere in the codebase, the import statement was removed altogether.
- **torch_sparse:** A `ModuleNotFoundError` was raised for `torch_sparse`. First, we verified the environment had `torch==1.13.0+cu117` and CUDA 11.7. Then, we installed `torch_sparse` manually via pip using the command: *pip install torch_sparse -f https://data.pyg.org/whl/torch-1.13.0+cu117.html*.
- **torch_scatter:** This library was similarly installed manually using the command: *pip install torch_scatter -f https://data.pyg.org/whl/torch-1.13.0+cu117.html*.
- **NumPy:** The package `numpy` was not initially available. We resolved this by adding `numpy==1.26` (or any version strictly less than 2) to the `requirements.txt` file.
- **Numba:** The environment also lacked `numba`. We installed `numba==0.57.1` and `llvmlite==0.40.1` using pip.

APPENDIX B: BATCH SCRIPT FOR DELFTBLUE

The following algorithm describes the batch script used to run LPFormer on the Citeseer dataset using DelftBlue. The script for the PubMed dataset is nearly identical, differing only in the dataset-specific hyperparameters.

Algorithm 5 Citeseer Training Batch Script on DelftBlue

0: Set SLURM job configuration:
- `--job-name="LPFormer_citeseer"`
- `--partition=gpu-a100`
- `--time=02:00:00`
- `--ntasks=1`, `--cpus-per-task=4`, `--gpus-per-task=2`
- `--mem-per-cpu=6GB`

0: Load required modules: `2023r1`, `python`, `cuda`
0: Navigate to home directory: `cd /home/sceydeli/`
0: Initialize and activate conda environment: `conda activate myenv`
0: Verify Python environment and print timestamp
0: Change to project directory: `cd LPFormer`
0: Create checkpoints directory: `mkdir -p checkpoints/citeseer`
0: Explicitly state the correct Python bin directory: `/home/sceydeli/miniconda3/envs/myenv/bin/python`
0: Run training script with the following parameters:
- `--data_name citeseer`
- `--lr 5e-3`
- `--gnn-layers 1`
- `--dim 256`
- `--batch-size 1024`
- `--epochs 100`
- `--kill_cnt 100`
- `--eps 1e-7`
- `--dropout 0.1`
- `--feat-drop 0.1`
- `--pred-drop 0.1`
- `--att-drop 0.1`
- `--gnn-drop 0.1`
- `--num-heads 1`
- `--thresh-1hop 1e-2`
- `--thresh-non1hop 1`
- `--eval_steps 1`
- `--decay 0.95`
- `--l2 0`
- `--non-verbose`
- `--runs 10`
- `--device 0`

0: Output logs to: `citeseer_training_custom.log`
0: Display training completion message =0

APPENDIX C: RESULTS TABLE FROM THE ORIGINAL PAPER

TABLE VI
COMPARISON OF PERFORMANCE ON VARIOUS DATASETS (MRR OR H@K). BEST RESULTS PER COLUMN ARE HIGHLIGHTED.

Metric	**Cora (MRR)**	**Citeseer (MRR)**	**Pubmed (MRR)**	**ogbl-collab (H@50)**	**ogbl-ppa (H@100)**	**ogbl-citation2 (MRR)**	**Mean Rank**
CN	20.99±0.00	28.34±0.00	14.02±0.00	56.44±0.00	27.65±0.00	51.47±0.00	11.0
AA	31.87±0.00	29.37±0.00	16.66±0.00	64.35±0.00	32.45±0.00	51.89±0.00	8.5
RA	30.79±0.00	27.61±0.00	15.63±0.00	64.00±0.00	49.33±0.00	51.98±0.00	8.7
GCN	32.50±6.87	50.01±6.04	19.94±4.24	44.75±1.07	18.67±1.32	84.74±0.21	8.0
SAGE	37.83±7.75	47.84±6.39	22.74±5.47	48.10±0.81	16.55±2.40	82.60±0.36	7.7
GAE	29.98±3.21	63.33±3.14	16.67±0.19	OOM	OOM	OOM	NA
SEAL	26.69±5.89	39.36±4.99	38.06±5.18	64.74±0.43	48.80±3.16	87.67±0.32	6.2
NBFNet	37.69±3.97	38.17±3.06	44.73±2.12	OOM	OOM	OOM	NA
Neo-GNN	22.65±2.60	53.97±5.88	31.45±3.17	57.52±0.37	49.13±0.60	87.26±0.84	7.0
BUDDY	26.40±4.40	59.48±8.96	23.98±5.11	65.94±0.58	49.85±0.20	87.56±0.11	5.7
NCN	32.93±3.80	54.97±6.03	35.65±4.60	64.76±0.87	61.19±0.85	88.09±0.06	3.8
NCNC	29.01±3.83	64.03±3.67	25.70±4.48	66.61±0.71	61.42±0.73	89.12±0.40	3.8
LPFormer	**39.42±5.78**	**65.42±4.65**	**40.17±1.92**	**68.14±0.51**	**63.32±0.63**	**89.81±0.13**	**1.2**