Improving Link Prediction on Citation Graphs Using LLMs

Chenyiteng Han
Rice University
ch181@rice.edu

Yiyi Tang
Rice University
yt61@rice.edu

Christina Zhou
Rice University
cz82@rice.edu

Abstract

With the rapid growth of academic publications, automatically identifying missing or potential citations is becoming increasingly important for literature retrieval and research evaluation. This project focuses on citation link prediction by comparing two types of node text features–BoW and SciBERT embeddings–and two decoders (dot product and MLP) under a unified GCN framework. We construct a directed citation graph of 9,000 computer science papers and evaluate model performance on future citation prediction using a time-based split. Results show that the SciBERT + MLP combination achieves around 0.95 in both AUC and Average Precision, significantly outperforming other baselines. We also use t-SNE to visualize latent space structure. Our findings demonstrate the benefits of incorporating LLM-based semantics into graph learning and provide a foundation for future improvements.

1 Introduction

In recent years, the explosive growth of academic publications has shifted researchers' primary challenge from "finding a relevant paper" to "identifying which papers are truly worth reading among countless candidates." Against this backdrop, citation link prediction plays a vital role–not only in automatically recovering missing citations caused by database delays or author oversight, but also in supporting recommendation systems, scientific impact evaluation, and the discovery of potential collaborations.

Unlike social networks or e-commerce platforms, each node in a citation network naturally carries rich textual context such as titles and abstracts, providing additional semantic cues for models. However, how to effectively integrate both textual semantics and graph structure remains an open problem: traditional bag-of-words features offer limited expressiveness, while deep graph models, when lacking high-quality node representations, also struggle to capture structural patterns.

Therefore, this project aims to investigate whether incorporating LLM-generated scientific text embeddings into a graph neural network framework can significantly enhance the accuracy of citation link prediction.

2 Related Work

Link prediction on citation graphs has been studied extensively as a means to infer missing or future citations. Early approaches often relied on graph structural heuristics such as Common Neighbors or Adamic–Adar [5], but these lack the ability to incorporate rich node attributes. With the rise of representation learning, embedding-based methods became popular: node2vec encodes structural contexts via random walks [2], while Variational Graph Auto-Encoders (VGAE) leverage a probabilistic encoder–decoder framework to learn latent node representations for link prediction

[4]. More recently, SEAL extracts enclosing subgraphs around each node pair and applies a labeled graph neural network to capture localized structural patterns [8].

Textual features of papers play a critical role in citation prediction. Traditional Bag-of-Words (BoW) or TF-IDF representations provide sparse, high-dimensional vectors that capture surface-level term frequency information [7]. In contrast, pretrained language models such as SciBERT produce dense contextual embeddings that more effectively encode semantic nuances in scientific text [1]. Empirically, SciBERT embeddings have been shown to improve downstream tasks like classification and retrieval over BoW baselines, motivating our comparison.

Once node embeddings are obtained, a decoder computes link scores. The simplest approach is the dot product decoder, which measures similarity in the embedding space and has been widely used due to its efficiency and interpretability [4]. However, its linearity limits modeling of complex interactions. To address this, an MLP decoder concatenates the two embeddings and passes them through non-linear layers, enabling richer modeling at the cost of more parameters and potential overfitting [3]. Our work examines both decoders under different embedding regimes.

Beyond embedding–decoder pipelines, end-to-end GNN architectures have been applied directly for link prediction. For example, Zhang et al.'s SEAL framework learns from subgraph patterns [8], while in Rossi's Temporal Graph Networks (TGN), temporal extensions account for evolving citation dynamics [6]. These methods provide valuable context but often assume homogeneous feature types; in contrast, our study focuses on disentangling the effects of textual embeddings and decoder architectures on prediction performance.

3 Methodology

3.1 Dataset Construction

We built a citation graph using data from the OpenAlex API. OpenAlex gives open information about academic papers. We selected papers in the field of computer science using the concept ID `C41008148`. We also required each paper to have an abstract.

We collected 9000 papers. For each paper, we kept:

- **Title**: the paper's title;
- **Abstract**: the abstract text, converted from the inverted index format;
- **References**: a list of other papers it cites;
- **Publication Year**: used later to split the data by time.

We made a directed graph. Each node is a paper. If paper i cites paper j, we add a directed edge from node i to node j.

To make the prediction task more realistic, we split the edges using the publication year. We used older edges for training. Newer edges were used for testing. This helps us test how well the model can predict future citations.

3.2 Node Embedding Construction

To represent each paper as a vector, we used its title and abstract. Our goal was to test whether large language model (LLM) embeddings can outperform traditional text features. For comparison, we used two types of embeddings: Bag-of-Words (BoW), as a simple baseline, and SciBERT, a scientific language model.

For both methods, we combined the title and abstract using a special separator: `title [SEP] abstract`.

BoW Embedding We applied `TfidfVectorizer` from scikit-learn to create sparse vectors. We removed English stopwords and used only words with at least two letters. Each paper was encoded as a 10,000-dimensional feature vector.

SciBERT Embedding We used the pre-trained model `allenai/scibert_scivocab_uncased` from HuggingFace. The input text was tokenized and passed into SciBERT. We used the embedding of the `[CLS]` token as the final 768-dimensional vector for each paper. This LLM-based method is expected to capture deeper semantic information than BoW.

3.3 Graph Neural Network Encoder

We employed a two-layer Graph Convolutional Network (GCN) as the encoder to learn structural node representations from the citation graph. Formally, given input node features $X \in \mathbb{R}^{N \times d}$ and graph adjacency A, the l-th GCN layer updates node embeddings as:

$$H^{(l+1)} = \text{ReLU}(\hat{A} H^{(l)} W^{(l)}), \quad \hat{A} = D^{-\frac{1}{2}}(A + I)D^{-\frac{1}{2}}$$

where $H^{(0)} = X$, $W^{(l)}$ is the weight matrix, and $\hat{A}$ is the normalized adjacency with self-loops.

We used two GCN layers with hidden size 512. A dropout layer (rate = 0.2) was applied after each non-linear activation. The output embedding for each node was a 512-dimensional vector.

We experimented with deeper architectures, residual connections, batch normalization, and larger hidden sizes, but found that these did not consistently improve performance across all decoder and embedding combinations. A detailed comparison is presented in Section 5.

3.4 Decoder

We used two types of decoders to compute link probabilities between node pairs: a simple dot product and a two-layer multilayer perceptron (MLP).

Dot Product. Given two node embeddings z_i, z_j, the decoder computes the score as their inner product:

$$\text{score}(i, j) = z_i^\top z_j$$

This method is simple, efficient, and often performs well when embedding space is well-structured.

MLP Decoder. We also implemented a two-layer MLP decoder. For each edge (i, j), we concatenate the two node embeddings $[z_i \| z_j]$ and feed them into a neural network:

$$\text{score}(i, j) = \text{MLP}([z_i \| z_j]) = W_2 \cdot \text{ReLU}(W_1 [z_i \| z_j] + b_1) + b_2$$

The MLP decoder can model more complex relations between nodes, and may better capture the semantic interactions in LLM-based embeddings like SciBERT.

We selected these two decoders to investigate how well simple geometric operations (dot product) perform compared to trainable neural decoders (MLP), especially under different embedding methods. Detailed results and comparisons are presented in Section 4.2.

3.5 Training Setup

We split the graph edges into training, validation, and test sets based on publication year. Edges between papers published in or before 2018 were used for training. Later citations were used for evaluation, simulating a realistic future link prediction task.

We used the Adam optimizer with a learning rate of 0.001 and weight decay of 1e-4. The loss function was binary cross-entropy with logits. We trained for up to 200 epochs with early stopping (patience = 10) based on validation AUC.

Each experiment was repeated 50 times with different random splits to reduce variance. We reported the mean and standard deviation of AUC and average precision across runs.

4 Experiments

4.1 Experimental Setup

We constructed a citation graph with 9000 nodes, where each node corresponds to a paper from the OpenAlex dataset. The graph includes both content features (BoW or SciBERT embeddings) and citation-based edges.

To ensure statistical reliability, we repeated each experiment 50 times with different random seeds and data splits. We used the same random split strategy for both positive and negative edges in all combinations of embeddings and decoders.

For evaluation, we computed the average and standard deviation of two metrics:

- AUC (Area Under the ROC Curve);
- AP (Average Precision).

All models were trained and evaluated under the same hyperparameter settings and training protocols described in Section 3.5.

4.2 Main Results

We evaluated four combinations of node embeddings and decoders:

- BoW + Dot Product
- BoW + MLP Decoder
- SciBERT + Dot Product
- SciBERT + MLP Decoder

Each combination was run 50 times. Table 1 shows the average test performance and standard deviation across runs, in terms of AUC and average precision (AP).

Combination	**AUC (mean ± std)**	**AP (mean ± std)**
BoW + Dot Product	0.9291 ± 0.0065	0.9277 ± 0.0070
BoW + MLP Decoder	0.8601 ± 0.0581	0.8680 ± 0.0477
SciBERT + Dot Product	0.9085 ± 0.0073	0.8979 ± 0.0086
SciBERT + MLP Decoder	**0.9485 ± 0.0263**	**0.9475 ± 0.0232**

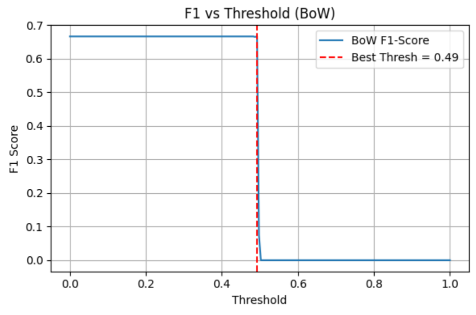
Table 1: SciBERT + MLP Decoder Does the Best

Among all combinations, SciBERT + MLP Decoder achieved the best performance in both AUC and AP. On the other hand, BoW + MLP showed the weakest and most unstable performance, possibly due to the limited expressiveness of BoW embeddings and overfitting in the MLP decoder.

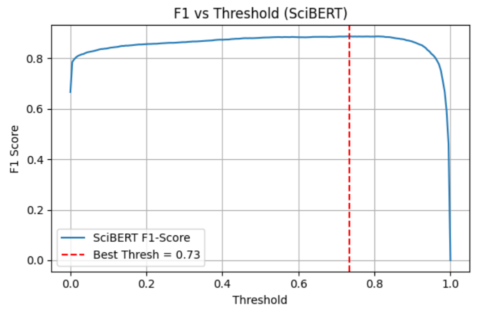
To further understand why BoW + MLP performs poorly and inconsistently, we analyzed the F1 score as a function of the classification threshold. The F1 score is the harmonic mean of precision and recall, and it depends on a cutoff threshold that determines which predicted links are considered positive.

Figure 1 compares the F1-threshold curves of BoW + MLP and SciBERT + MLP. The curve for BoW + MLP exhibits a striking binary behavior: the F1 score stays high across low thresholds but drops abruptly near 0.5. This indicates that the model outputs highly polarized predictions, and small shifts in threshold can cause dramatic performance collapse.

Although not all BoW + MLP runs exhibit this extreme behavior, such instability appears frequently. For instance, the run shown in Figure 1 had AUC = 0.7784 and AP = 0.8040, both lower than the mean values. These threshold-sensitivity patterns, combined with the high standard deviation in Table 1, confirm that BoW + MLP is both less accurate and less robust than the other combinations. To further compare the best-performing embedding-decoder pairs, we visualize the distribution of AUC and AP scores over 50 runs using box plots (Figure 2). BoW + Dot Product and SciBERT + MLP represent the strongest configurations for each embedding class.



(a) F1 vs Threshold for BoW + MLP



(b) F1 vs Threshold for SciBERT+ MLP

Figure 1: The Instability of BoW + MLP

As expected, SciBERT + MLP outperforms BoW + Dot Product in terms of both mean and median values. The interquartile range is also slightly higher, indicating overall stronger predictive capability.

However, we also observe the presence of a few low outliers in the SciBERT + MLP distribution. This highlights a potential weakness in MLP decoders: occasional instability and threshold sensitivity, similar to the two-phase behavior seen earlier in BoW + MLP.

Despite these rare failures, the mean performance of SciBERT + MLP remains superior, which shows its upper limit and potential. This suggests that SciBERT is more resilient to MLP's instability, possibly due to its richer and more informative embedding space. In contrast, BoW-based inputs are more susceptible to overfitting and sharp performance drops.

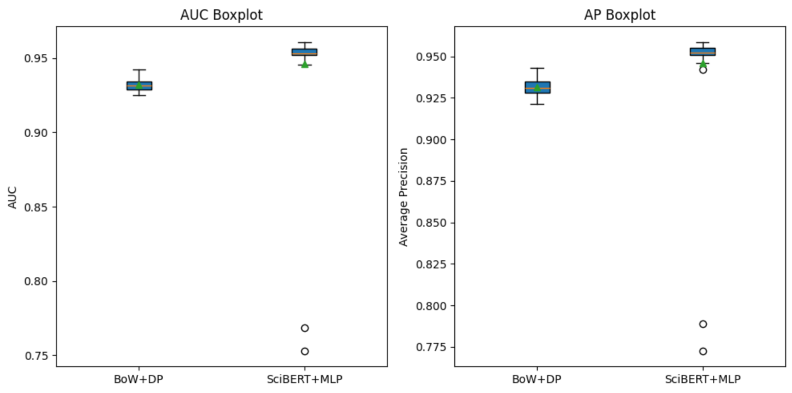


Figure 2: Boxplot Comparison (a) AUC

4.3 Visualization

Figures 3 and 4 present the node embeddings generated by four different models after GCN encoding, projected into 2D space using t-SNE. For the BoW features, the inner product decoder (BoW + DP) produces a slightly more convergent point distribution, while incorporating an MLP leads to a more diffuse, cloud-like cluster with many peripheral points. Under SciBERT features, SciBERT + DP shows no clear cluster structure, whereas SciBERT + MLP exhibits more compact groupings. Although the clustering in Figure 4a appears less distinct compared to the others, its performance is not the worst–suggesting that cluster compactness does not have a simple one-to-one correspondence with predictive performance. Based on this, we believe that t-SNE embedding plots are better suited as supplementary visualizations for examining how different feature-decoder combinations structure the latent space, rather than as quantitative evidence for evaluating model quality.

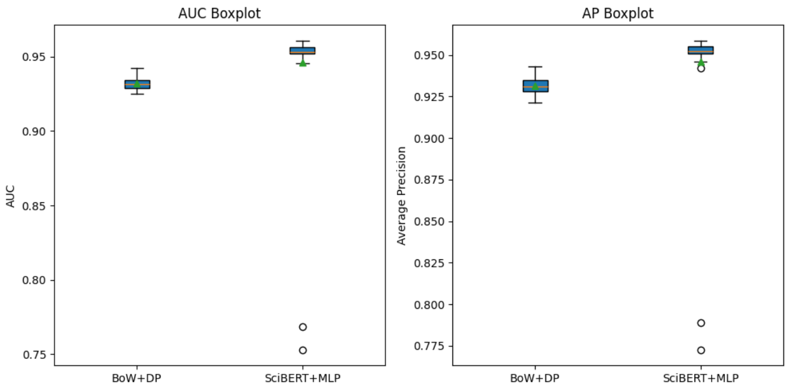
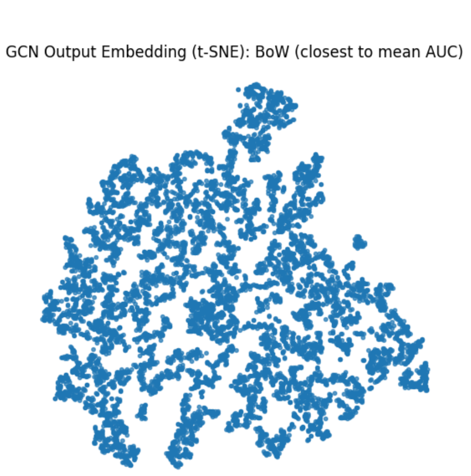
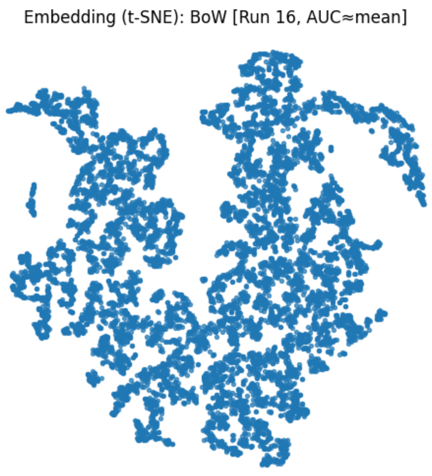


Figure 2: Boxplot Comparison (b) AP

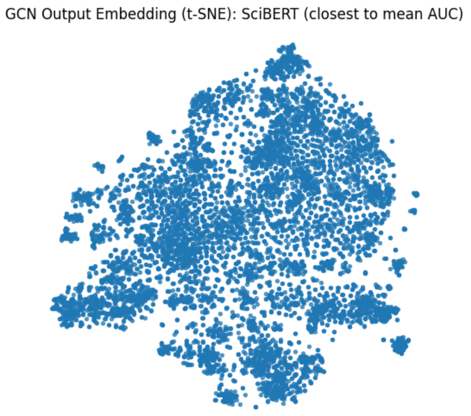


(a) Embedding Visualization for BoW + DP

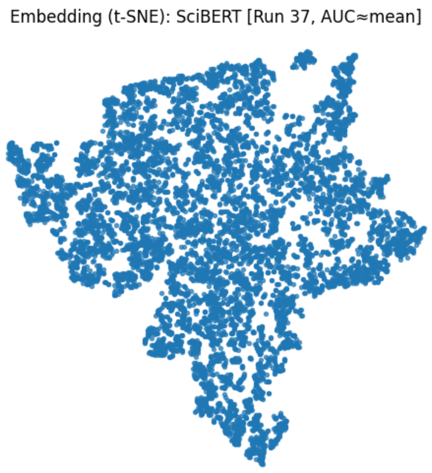


(b) Embedding Visualization for BoW+ MLP

Figure 3: Embedding Visualization for BoW



(a) Embedding Visualization for SciBERT + DP



(b) Embedding Visualization for SciBERT+ MLP

Figure 4: Embedding Visualization for SciBERT

5 Analysis and Discussion

5.1 Embedding vs. Decoder Matching

Our results suggest a strong interaction between the choice of embedding and the decoder architecture. Specifically, we find that:

- BoW embeddings work best with a simple dot product decoder;
- SciBERT embeddings benefit significantly from a trainable MLP decoder.

We attribute this to the different nature of the embeddings. BoW vectors are high-dimensional but sparse and linear. They encode term frequencies without capturing deeper semantic relationships. As a result, applying a more complex MLP decoder on top of BoW may introduce overfitting without substantial gain, especially given the limited expressiveness of the input features.

In contrast, SciBERT embeddings are dense, contextualized representations derived from a large-scale pretrained language model. These embeddings carry rich semantic information, including syntax, context, and inter-word dependencies. A simple dot product may not fully exploit this richness. Instead, an MLP decoder can learn non-linear combinations and interactions between SciBERT features, enabling more precise link prediction.

This matching effect is also reflected in the observed stability. As shown in Table 1, BoW + MLP yields the lowest average AUC and AP, along with the highest standard deviation. On the other hand, SciBERT + MLP not only achieves the best performance, but also maintains relatively low variance, indicating robustness.

Finally, we note that SciBERT is the only embedding method that achieves an AUC over 0.94, suggesting that large language models offer a higher performance ceiling than traditional text representations like BoW.

5.2 Overfitting and Optimization Failure

We experimented with various architectural modifications to the GCN encoder to improve model performance. These included:

- Adding residual connections between GCN layers;
- Increasing the number of GCN layers (from 2 to 3 or more);
- Applying batch normalization;
- Expanding the hidden dimension size.

Surprisingly, these modifications did not consistently improve performance. In many cases, we observed a degradation in both AUC and AP, particularly when using BoW embeddings. We hypothesize two main reasons for this:

First, the graph convolutional layers may have overfitted to the sparse or low-quality input features (especially with BoW), leading to unstable or deteriorated link predictions. Additional layers and capacity can amplify such overfitting.

Second, the gain from deeper architectures may be constrained by the quality of the input embeddings. When using powerful LLM-based embeddings such as SciBERT, even a shallow GCN is sufficient to propagate useful information. In such cases, adding more layers or complexity does not necessarily translate to better generalization.

In contrast, we found that increasing the dataset size, from 10k to 90k nodes, did yield measurable improvement. This suggests that data scale, rather than model complexity, is a more promising direction for improving performance. Unfortunately, due to resource limitations, we were unable to explore larger datasets. We consider this an important avenue for future work.

6 Conclusion and Future Work

6.1 Insights

This project compares four combinations of node text features (BoW vs. SciBERT) and decoders (dot product vs. MLP), using a GCN as the shared graph encoder. Experimental results show that the SciBERT + MLP combination significantly outperforms others in both AUC and Average Precision, reaching scores as high as 0.95. This highlights the decisive role of high-dimensional semantic information from large language models in citation link prediction. Moreover, the results reveal a key insight: there exists a compatibility pattern between embedding expressiveness and decoder complexity – simple and sparse BoW embeddings align better with geometric decoders (dot product), while semantically rich SciBERT embeddings benefit more from nonlinear MLP decoders.

6.2 Potential Improvements

Future improvements may include incorporating edge features, exploring stronger generative frameworks, or applying temporal GNNs. By utilizing citation context sentences, paragraph positions, or publication venues, one could construct heterogeneous graphs or assign multi-dimensional edge weights to better capture the motivations behind citations. Adopting generative models like VGAE and its contrastive learning variants would allow for explicit modeling of uncertainty in the latent space, producing smoother and more interpretable edge probability distributions. Treating citations as a dynamic process over time, temporal GNNs such as TGAT can be used to model the temporal evolution of nodes and edges, aiming to improve early recall in realistic "future citation" prediction scenarios. By exploring these directions, the project can provide more reliable technical support for academic search, recommendation systems, and scientific impact evaluation.

Reproducibility

All code, datasets, and trained models are available at: `https://github.com/Hhnxxxxxx/LLM-Link-Prediction`

References

[1] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. *arXiv preprint arXiv:1903.10676*, 2019.

[2] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks, 2016.

[3] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering, 2017.

[4] Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016.

[5] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 556–559, 2003.

[6] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs, 2020.

[7] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[8] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in neural information processing systems*, 31, 2018.