Received 23 May 2023, accepted 30 June 2023, date of publication 5 July 2023, date of current version 17 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3292314

RESEARCH ARTICLE

Embedding Imputation With Self-Supervised Graph Neural Networks

URAS VAROLGUNES, SHIBO YAO, YAO MA, AND DANTONG YU
New Jersey Institute of Technology, Newark, NJ 07102, USA

Corresponding author: Uras Varolgunes (uv27@njit.edu)

This work was supported by the Department of Energy under Grant DE-SC0022346.

ABSTRACT Embedding learning is essential in various research areas, especially in natural language processing (NLP). However, given the nature of unstructured data and word frequency distribution, general pre-trained embeddings, such as word2vec and GloVe, are often inferior in language tasks for specific domains because of missing or unreliable embedding. In many domain-specific language tasks, pre-existing side information can often be converted to a graph to depict the pair-wise relationship between words. Previous methods use kernel tricks to pre-compute a fixed graph for propagating information across different words and imputing missing representations. These methods require predefining the optimal graph construction strategy before any model training, resulting in an inflexible two-step process. In this paper, we leverage the recent advances in graph neural networks and self-supervision strategy to simultaneously learn a similarity graph and impute missing embeddings in an end-to-end fashion with the overall time complexity well controlled. We undertake extensive experiments to show that the integrated approach performs better than several baseline methods.

INDEX TERMS Embedding imputation, graph neural networks, natural language processing.

I. INTRODUCTION

Embedding techniques [1], [2], [3] have attracted numerous attention, especially in the domain of natural language processing, because high-quality word representations are indispensable for most language learning tasks. Specifically, many NLP tasks employ transfer learning by creating an embedding lookup layer using a set of pre-trained word embedding vectors trained on a large corpus. Transfer learning is preferred over training a language model from scratch to learn completely new embeddings because most real-world datasets contain a large volume of rare words, making it difficult to find the right representation for them. Additionally, when embeddings are trained from scratch, the number of trainable parameters increases, and the training process slows down significantly.

Despite all these advantages of pre-trained embedding vectors, some words in the dataset of a specific task may not have a pretrained embedding vector. For example, thousands or even millions of terminologies and abbreviations in the medical field do not have pretrained embedding vectors because they are often not contained in a general corpus. When employing a set of pretrained embeddings for an NLP task, such words without any pretrained embedding are usually assigned a randomly sampled embedding vector. This phenomenon hinders the downstream NLP tasks from benefiting from these words' embeddings. Fortunately, it is possible to leverage some side information to mitigate the problem of missing word embedding. For example, a knowledge graph in the form of a medical taxonomy and ontology, or even a collection of chemical or physiological attributes of the terminologies, can serve as side information and be fused into the semantic space to impute missing word embeddings.

In this work, we aim to design a semi-supervised learning model that leverages side information to define pair-wise similarities between entities (words) and propagates information between entities with known targets (embeddings) and the ones with unknown targets based on their similarities. Graphs can naturally be used to model such relationships. Most of the existing graph-based methods require a given/pre-constructed graph to start with; however, such a graph may

The associate editor coordinating the review of this manuscript and approving it for publication was Chuan Li.

not always be readily available. In this work, we aim to learn and optimize a graph and use it for propagating node attributes.

In [4] and [5], the authors first build a fixed graph between words based on side information and then use it to propagate information between nodes. This approach is depicted in Fig. 1. First, the side information matrix X is used to learn pair-wise similarities between words and construct the fixed similarity graph G. Then, the parameterized mapping function $f(G, X)$ learns to map the side information vector for a word w_i to its corresponding pre-trained embedding by updating its parameters.

Hence, for these methods: 1) the graph has to be pre-constructed and fixed, 2) the optimal parameters for the mapping function have to be computed for the fixed graph. If the pre-constructed graph is not suitable for the imputation task, i.e., it does not accurately capture the pairwise relationships between words, it may significantly limit the capacity and hurt the performance of the model. Optimizing the graph structure is costly because it requires repeating the two-step process every time. To address these challenges caused by the two-step approach, we propose to infer a graph from the data, similar to latent graph learning approaches, such as [6], [7], and [8] instead of calculating a fixed graph. These methods are more flexible compared to fixed graphs due to their ability to adjust graph structure to minimize the embedding learning loss function. For embedding imputation using side information, latent graph learning is a promising approach, because it does not impose any constraint on graph structure and allows joint learning of the graph and the neural network parameters in an end-to-end manner.

One challenge for latent graph learning approaches is that they heavily rely on limited labels and suffer from poor generalization because of over-fitting. This happens particularly when labeled training data are scarce [9]. In [8], the authors identify a supervision starvation problem in latent graph learning approaches in which the edges between pairs of unlabeled nodes that are far from labeled nodes receive insufficient supervision, leading to unreliable graphs during test time. Because our embedding imputation is concerned with imputing rare words, such as terminologies or proper nouns, it is likely that the domain for side information also consists of only some related rare words, resulting in a limited amount of labeled training data. Any embedding imputation on these rare words suffers from over-fitting and supervision starvation issues. In this paper, we tackle the embedding imputation problem by leveraging the recent advances in self-supervised learning with latent graph structure.

Additionally, graph learning often incurs quadratic costs concerning the number of graph nodes, and this may cause some challenges for large-scale data. To reduce the complexity of the method, we leverage the anchor-graph idea [10] to build an approximate graph where the prior knowledge is translated into the word-word pairwise relationship, with guaranteed linear time complexity and provable algebraic properties. Overall, our method generates superior

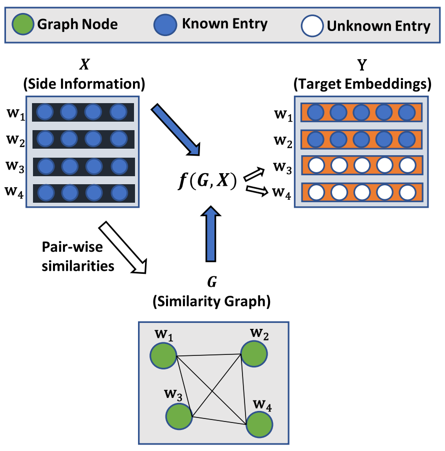


FIGURE 1. Side information matrix X and target embedding matrix Y for words w_1, w_2, w_3 and w_4. The i-th rows in X and Y correspond to the side information vector and pre-trained embedding of word w_i, respectively. For all words, the side information vectors are available, but target embeddings for w_3 and w_4 are missing.

experimental results compared to previous works and is scalable for building large graphs. The contributions of this work are as follows:

- We design a powerful embedding method built on top of the recent advances in latent graph learning to address the critical problem of word embedding imputation in natural language processing.
- We learn a dynamic graph from prior knowledge that best fits the embedding learning problem and use it to propagate and transform the prior knowledge into effective embeddings.
- We customize the graph construction using an anchor sampling process to reduce the complexity from quadratic to linear. Consequently, the overall approach is scalable to large datasets.
- We demonstrate the effectiveness of the approach with thorough experiments.

II. RELATED WORKS

The embedding imputation problem was formulated and studied in Latent Semantic Imputation [4], where the authors first use a piece of side information to construct a weighted graph based on the non-negative least squares approach defined in Locally Linear Embedding [11] and convert it into a minimum-spanning-tree-k-nearest-neighbor-graph. The missing embedding vectors are then imputed via a matrix power iteration process that theoretically guarantees deterministic convergence. KG2Vec [5] also uses prior knowledge to build a similarity graph where each node corresponds to a word. Similar to [4], the authors formulate the imputation problem as graph-based semi-supervised learning and apply graph convolutional networks (GCN) [12] to learn missing embedding vectors. The advantages of using GCN in the solution include the incorporation of graph topology into embedding and enlarged model capacity provided by the neural network parameters. Another line of work addressing the problem of missing word embedding vectors includes federated learning based on character-level information [13] and the robust backed-off approach [14] based on sub-word information. These two approaches do not incorporate side information or prior knowledge. We follow the same problem setting as the first line of work that integrates prior knowledge into the solution. Differently from these works, we learn graph topology and embeddings within the same network architecture in an end-to-end fashion as shown in Fig. 2, rather than using a non-flexible, fixed graph to propagate information. The method in this paper is closely related to the graph-based semi-supervised learning, including the early endeavors [15], [16], where the modeling process takes into consideration the feature vectors of all samples and uses the pair-wise relationships in a graph to enforce locality and smoothness. Graph neural networks [12], [17], [18] further enhance the solution with the rich capacity of neural network.

Another line of work considers solving classification problems with GNNs when a graph structure is unavailable. LDS-GNN [6], jointly learns the graph structure and the parameters of a GCN by approximately solving a bilevel program that learns a discrete probability distribution on the edges of the graph. The method allows applying GCNs in scenarios where the given graph is not available, incomplete, or corrupted. IDGL [19] uses an iterative approach and alternates over projecting the nodes to a latent space and constructing an adjacency matrix from the latent representations multiple times. In [8], the authors propose to simultaneously learn the adjacency and GNN parameters with self-supervision for inferring a robust graph structure. Our method is closely related to this line of work.

Most GNN application scenarios assume that the graph topology is given. However, graph topology is often unknown in our problem setting and may have to be constructed from prior knowledge multiple times during training. Graph construction is based on some distance metric among node vectors and incurs quadratic time complexity in the number of nodes. A brute-force approach does not scale to large datasets. To address this issue, we look into near-linear-time geometric graph construction. The recent theoretical work [20] systematically investigates the problem and presents a solution based on well-separated-pair-decomposition, coupled with Johnson-Lindenstrauss lemma if the node feature vector is high-dimensional. Fast approximate k-NN graph construction is also tackled in [21] using locality-sensitive hashing. In our work, we employ a relatively simple yet effective approach that samples anchors and is similar to the idea in [10]. Our experiment results show that the straightforward anchor sampling achieves efficacy and efficiency simultaneously. Moreover, when combined with self-supervised GNNs, the final model performance is robust against graph variation and randomness due to the anchor sampling.

III. PROBLEM DEFINITION

Given a set of words $\{w_i|1 \leq i \leq n\} = \{w_l\} \bigcup \{w_u\}$, $\{w_l\}$ is the set of words with known embedding vectors $\{\mathbf{y_l}\}$, where $\mathbf{y_i} \in \mathbb{R}^d$, and $\{w_u\}$ is the set of words without embedding vectors (for notational convenience the missing embedding vectors are denoted as $\{\mathbf{y_u}\}$). Given $\{\mathbf{y_l}\}$ and some prior knowledge about the words from an external knowledge base represented by feature vectors $\{\mathbf{x_i}|1 \leq i \leq n\} = \{\mathbf{x_l}\} \bigcup \{\mathbf{x_u}\}$, where $\mathbf{x_i} \in \mathbb{R}^f$, the objective is to infer $\{\mathbf{y_u}\}$. To apply graph-based semi-supervised learning methods, we need to build a graph based on prior knowledge $\{\mathbf{x_i}\}$, where each word w_i is a node. Note that, $|\{w_i\}| = |\{\mathbf{x_i}\}| = n$, $|\{w_l\}| = |\{\mathbf{x_l}\}| = |\{\mathbf{y_l}\}| = p$ and $|\{w_u\}| = |\{\mathbf{x_u}\}| = |\{\mathbf{y_u}\}| = q$, where $n = p + q$. n, p and q represent the total number of words, the number of words which have pretrained embedding vectors and the number of words without pretrained a embedding vector, respectively.

IV. PRELIMINARIES

We define a weighted, attributed graph as $\mathcal{G} = \{\mathcal{V}, \tilde{\mathbf{A}}, \mathbf{X}\}$, where $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is the node set, $\tilde{\mathbf{A}} \in \mathbb{R}^{n \times n}$ is the adjacency matrix with $\tilde{\mathbf{A}}_{ij}$ (the element at i-th row, j-th column) indicating the edge weight from node i to node j, ($\tilde{\mathbf{A}}_{ij} = 0$ implies there is no edge) and $\mathbf{X} \in \mathbb{R}^{n \times f}$ is the feature matrix with f representing the dimensionality of the feature vector of each node.

The graph convolutional neural network (GCN) [12] aims to improve the quality of the representations by aggregating information from neighbor nodes and applying transformations to the representations in each layer. For a graph $\mathcal{G} = \{\mathcal{V}, \tilde{\mathbf{A}}, \mathbf{X}\}$, the output of the l-th layer of a GCN is defined as:

$$\mathbf{H}^{(l)} = \sigma(\mathbf{A}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}) \tag{1}$$

$\mathbf{H}^{(l)} \in \mathbb{R}^{n \times d_l}$ and $\mathbf{H}^{(l-1)} \in \mathbb{R}^{n \times d_{l-1}}$ are the node representations of the current and the previous layers, respectively and the representations in the first layer are initialized as $\mathbf{H}^{(0)} = \mathbf{X}$. $\mathbf{A} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}} + I$ is the normalized adjacency matrix with the added self-loop, where I represents the $n \times n$ identity matrix. $\mathbf{W}^{(l)} \in \mathbb{R}^{d_{l-1} \times d_l}$ is a trainable weight matrix and σ is an activation function, such as *tanh*.

V. METHODOLOGY

In this section, we introduce the framework of our self-supervised imputer (SSI). Our goal is to define a mapping from the side information data space to the embedding space, i.e., for each word, the feature vector (side information) will be mapped to its embedding vector. We propose to use a GCN to incorporate the affinity information between words and thereby improve the quality of the learned representations. However, most GCN based architectures assume a pre-defined graph structure. Our problem setting does not have a pre-defined graph structure and requires graph construction from available side information. Existing approaches often adopt kernel-tricks and the nearest neighbors to construct a fixed similarity graph based on side information. However, these fixed graphs may not be appropriate for our imputation objective due to the difference between the two spaces and the lack of flexibility in domain adaptation. Hence, we propose to learn the graph structure

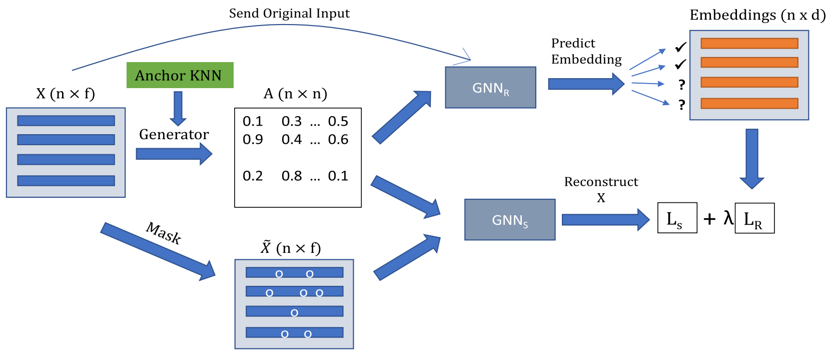


FIGURE 2. Network architecture. Side information matrix **X** is input into the generator to obtain the processed adjacency matrix **A**. **A** is then used for both tasks. GNN_S tries to reconstruct the original feature matrix **X** and the self-supervision loss L_S is computed based on its output. GNN_R predicts the embedding for each word and the regression loss L_R is computed based on known embeddings. During test time, the unknown embeddings are replaced with the predictions made by GNN_R.

and optimize the model weights simultaneously in an end-to-end fashion similar to the graph construction algorithm in [8]. Fig. 2 shows the overall architecture of the proposed embedding imputation framework, consisting of the following components:

1) Graph generator module: transforms the original node features using an MLP and computes a kNN-graph based on the similarity of the transformed features. The generated graph is then used in both the Regression module and the Self-supervision modules to increase the robustness of the learned graphs, 2) Regression module (GNN_R): this module consists of a GCN that learns to map the original node features to the target embeddings based on the kNN-graph generated by the previous module, and 3) Self-supervision module (GNN_S): it randomly masks (converts to zero) a subset of the entries in the original feature matrix and then uses the Graph AutoEncoder and the kNN-graph generated from the prior steps as input graph to reconstruct the original features from the masked (corrupted) version of the original node features.

A. GRAPH GENERATOR

The graph generator uses side information (node features) in order to construct the affinity graph. Formally, the graph generator $\mathcal{G} : \mathbb{R}^{n\times f} \rightarrow \mathbb{R}^{n\times n}$ is a function that takes the side information matrix $\mathbf{X} \in \mathbb{R}^{n\times f}$ as input and produces the affinity matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{n\times n}$ as output. First, $\mathbf{X}$ is passed to a multi-layer perceptron, $MLP : \mathbb{R}^{n\times f} \rightarrow \mathbb{R}^{n\times f'}$, which outputs the transformed node representations $\mathbf{X}' \in \mathbb{R}^{n\times f'}$. Based on $\mathbf{X}'$, the k-nearest-neighbors function kNN: $\mathbb{R}^{n\times f'} \rightarrow \mathbb{R}^{n\times n}$, selects the top k neighbors for each node and generates the sparse k-nearest-neighbor graph.

Let $\mathbf{x}'_\mathbf{i}$ denote the transformed representation of node v_i. To select the nearest neighbors for v_i, we compute the dot product between $\mathbf{x}'_\mathbf{i}$ and $\mathbf{x}'_\mathbf{j}$ for all $j = \{1, 2, \ldots, n\}$ and select the top k nodes with the largest dot product. Finally, for all selected nodes v_j, we set $\tilde{\mathbf{A}}_{ij} = \mathbf{x}'_\mathbf{i} \cdot \mathbf{x}'_\mathbf{j}$, where $\cdot$ represents the dot product operation.

The output matrix $\tilde{\mathbf{A}}$ may contain negative and positive values, may be asymmetrical and needs to be normalized, so we further process $\tilde{\mathbf{A}}$ to obtain the affinity matrix of the learned graph: $\mathbf{A} = \frac{1}{2}\tilde{\mathbf{D}}^{-1/2}(\mathbf{P}(\tilde{\mathbf{A}})+\mathbf{P}(\tilde{\mathbf{A}})^\top)\tilde{\mathbf{D}}^{-1/2}$ where $\tilde{D}$ is the matrix of node degrees and $\mathbf{P}$ is an element-wise function with non-negative range, such as $ReLU$.

B. REGRESSION MODULE

The GNN-based regression module $GCN_R : \mathbb{R}^{n\times f} \times \mathbb{R}^{n\times n} \rightarrow \mathbb{R}^{n\times d}$, takes the original node features $\mathbf{X}$ and the generated adjacency matrix $\mathbf{A}$ as input and outputs the predicted embedding, where d is the dimensionality of the embedding vector to be imputed. The GNN module parameterized by two matrices $\mathbf{W}_R^{(1)}$ and $\mathbf{W}_R^{(2)}$ of trainable weights generates output as follows:

$$GCN_R(\mathbf{A}, \mathbf{X}; \mathbf{W}_R^{(1)}, \mathbf{W}_R^{(2)}) = \mathbf{A}ReLU(\mathbf{A}\mathbf{X}\mathbf{W}_R^{(1)})\mathbf{W}_R^{(2)} \quad (2)$$

The regression loss $\mathcal{L}_R$ for training is defined as the mean squared difference between the original embedding and the predicted embedding, which is computed for all nodes with available target embedding. Here, target embeddings serve the same role as labels in supervised learning. We generalize the definition of label to be the learning target regardless of whether the model is for classification or regression.

C. SELF-SUPERVISION MODULE

The graph generator learns to generate graphs mainly based on the regression loss $\mathcal{L}_R$ computed on only labeled nodes. In [8], the authors identify the problem of starved edges and argue that only relying on the supervision from labels may not be sufficient while learning a dynamic graph. A starved edge is defined as an edge generated between two nodes that receives no supervision from the labels because it is more than k-hops away from any labeled node when using a k-layer GCN. They show that a graph suitable for predicting node features can also be useful in predicting node labels and such a graph can be used to regularize the starved edges. We use this idea and introduce the self-supervision module which is useful for learning a more robust graph.

First, we augment the node features by masking out some node attributes with a binary mask matrix $\mathbf{M} \in \{0, 1\}^{n\times f}$ and obtain a randomly sampled version $\tilde{\mathbf{X}}$ of $\mathbf{X}$. A different $\mathbf{M}$ is created randomly at each epoch and contains a fixed number of zero entries. $\tilde{\mathbf{X}} = \mathbf{X} \odot \mathbf{M}$, where $\odot$ represents the Hadamard product.

We define an Autoencoder $GCN_S : \mathbb{R}^{n\times f} \times \mathbb{R}^{n\times n} \rightarrow \mathbb{R}^{n\times f}$ with parameters $\mathbf{W}_S^{(1)}$ and $\mathbf{W}_S^{(2)}$ that takes the node features $\tilde{\mathbf{X}}$ and the generated adjacency matrix $\mathbf{A}$ and tries to reconstruct $\tilde{\mathbf{X}}$. The self-supervision loss is defined as:

$$\mathcal{L}_S = MSE(\mathbf{X}_{idx}, GCN_S(\mathbf{A}, \tilde{\mathbf{X}}; \mathbf{W}_S^{(1)}, \mathbf{W}_S^{(2)})_{idx}) \quad (3)$$

where $idx = \{(i, j)|M_{ij} = 0\}$ is a set of indices selected uniformly at random in each epoch. MSE is the mean-squared-error loss. The final model is trained to minimize $\mathcal{L} = \mathcal{L}_R + \lambda\mathcal{L}_S$. λ is a hyperparameter that controls the relative importance between the embedding regression loss $\mathcal{L}_R$ and the self-supervision loss $\mathcal{L}_S$.

D. ANCHOR-kNN GRAPH CONSTRUCTION

The graph generator computes a new *kNN* graph at each epoch. This may result in scalability issues for large datasets because conventional *kNN* graph construction requires quadratic time with respect to the number of nodes. To mitigate the scalability issue, we adopt the idea of anchor-graph [10].

The Anchor-*kNN* process is outlined in Algorithm 1. *choice*() is the random sampling process on a uniform distribution. $choice(\mathbf{X}, m)$ samples m unique rows from $\mathbf{X}$ and constructs $\mathbf{X}_m$. We take the inner product of the feature matrix $\mathbf{X}$ and $\mathbf{X}_m$ to construct the similarity matrix $\mathbf{C}$, where $\mathbf{C}_{ij}$ denotes the similarity between nodes v_i and v_j. The graph adjacency $\tilde{\mathbf{A}}$ is initialized as a matrix of zeros. *NN_index* takes in $\mathbf{C}_i$ (the i-th row of $\mathbf{C}$, i.e., the list of similarities for v_i) and an integer k (the node degree) as inputs, and returns a list of indices, γ, corresponding to the largest k elements in $\mathbf{C}_i$. Then, for all indices $j \in \gamma$, the corresponding entry in $\tilde{\mathbf{A}}_{ij}$ is set to be the similarity score $\mathbf{C}_{ij}$. To use anchor graph, we replace the *kNN* function in the Graph Generator module with the Anchor-*kNN* function.

1) COMPLEXITY ANALYSES

In addition to the model effectiveness, we need to evaluate whether the end-to-end approach for embedding imputation is scalable to large datasets. During graph construction, the anchor sampling process takes $O(1)$ time, the similarity computation between n nodes and m anchors takes $O(fmn)$ time where f denotes the dimension of the original feature vector. Overall time complexity for k nearest neighbor search is $O(mn)$. The time complexity of the GNN model evaluation is also linear in n given that the graph is sparse. Therefore, the end-to-end approach with a learnable graph has a time complexity $O(n)$ with $m \ll n$.

Algorithm 1 Anchor-*kNN*

```
Input  : $(\mathbf{X}, k, m)$ ;   // $\mathbf{X} \in \mathbb{R}^{n\times f}$:feature
         matrix; $k$:desired node degree;
         $m$:number of anchors
Output: Affinity matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{n\times n}$
1 $\mathbf{X}_m = choice(\mathbf{X}, m)$
2 $\mathbf{C} = \mathbf{X} \cdot \mathbf{X}_m^T$ where $\mathbf{C} \in \mathbb{R}^{n\times m}$
3 $\tilde{\mathbf{A}} \leftarrow 0$ ;   // initialize as zero matrix
4 for $i$ in $1, 2, \ldots, n$ do
5     $\gamma = NN\_index(C_i, k)$;
6     for $j$ in $\gamma$ do
7         $\tilde{\mathbf{A}}_{ij} = C_{ij}$;
8     end
9 end
```

VI. EXPERIMENTS

We carry out comprehensive experiments on various real-world datasets, including the finance industry and online application markets, to demonstrate the effectiveness, scalability and robustness of the proposed approach. Specifically, we try to answer the following questions:

- Is SSI more effective in mapping the side information to the semantic space and facilitate downstream tasks than the baseline methods?
- Can the anchor sampling process reduce graph construction complexity while retaining model performance?
- What is the impact of self-supervision on model performance?

We use LSI [4], GCN [12], IDGL [19] and LDS-GNN [6] as the baseline methods for comparison. These methods cover a variety of machine learning models with different architectures. LSI is based on the standard (linear) matrix power method while GCN takes advantage of the rich capacity of the multi-layer-perceptron (MLP) and the graph neural networks. IDGL and LDS-GNN jointly learn the graph structure and the parameters of a GCN and SSI further improves upon all the aforementioned methods by deriving robust graphs from self supervision. Implementation and hyperparameter tuning details are provided in Appendix A.

A. DOWNSTREAM TASKS

When we want to avoid building a language model from scratch, we typically utilize a pretrained embedding set such as GloVe [3]. However, some word embedding in the dataset corpus may not be available in this pretrained embedding set. To impute those missing words, we can utilize a side information source and apply an imputation model. Note that, for data imputation, we do not need to have side information for all the words in the corpus. It is enough to have side information only for a subset of the words which includes the missing words and some selected words from the corpus having the pretrained embedding available. For example, in a finance sentiment analysis task, the corpus may consist of a collection of financial articles, which include some company names along with some other more commonly used words. After picking a pretrained embedding set for the task, we may observe that some of the company names in the corpus do not have a pretrained embedding. Suppose that we identify a side information source (e.g., daily stock returns) that contains information about n companies in the corpus: q of those n companies do not have a pretrained embedding, while the remaining $p = n - q$ companies have a pretrained embedding. Then, we apply the designed imputation algorithm to predict the embedding vectors for those q companies using the guidance from the other p companies. Below, we conduct experiments on two real-world datasets and demonstrate the effectiveness of our imputation model.

1) IMPUTING FINANCE COMPANY EMBEDDINGS

For this task, we adopt the datasets from [4]. There are two datasets of different sizes. The small dataset has a word set of 488 company names retrieved from S&P500 index. The large dataset has a word set of 4092 company names covering almost all publicly listed stocks in US market retrieved from NYSE and NASDAQ. The goal is to successfully impute the missing embeddings that are not available in the pretrained embeddings using the available side information (historical return data). This is done separately for all three pretrained embedding sets. A more detailed description of the dataset is provided in the Appendix B. As LSI and GCN require a fixed graph, we follow the steps in [4] to construct an MST-kNN graph using the side information matrix X as the domain matrix, solve for the optimal edge weights using Non-Negative-Least-Squares and normalize the weights to obtain the graph. LDS-GNN, IDGL and SSI do not require this as they learn the graph during training. For GCN, LDS-GNN, IDGL and SSI, side information matrix X is used as the feature matrix.

2) IMPUTING MOBILE APPLICATION EMBEDDINGS

Mobile App Statistics dataset contains more than 7000 Apple iOS mobile application details extracted from the iTunes Search API at the Apple Inc website. Each app has a name and a primary genre such as Games, Sports, or Business. There are 23 possible genres. Each app also has categorical (e.g., content maturity rating) and numerical features (e.g., price) and a textual description. For each app, we process and merge the features and obtain a feature vector of size f. Details of the dataset and the processing steps are provided in the Appendix B. Side information matrix $X \in \mathbb{R}^{n \times f}$ is obtained by stacking the individual feature vectors of all apps. Again, X is used to construct the fixed graph for LSI and GCN and also as the feature matrix for GCN, LDS-GNN, IDGL and SSI.

3) EVALUATION

After the imputation, we obtain a set of predicted embedding vectors $\{\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots, \hat{\mathbf{y}}_n\}$ for all words. Any word w_i which previously had a pretrained embedding retains its original embedding $\mathbf{y_i}$, while we set $\mathbf{y_j} = \hat{\mathbf{y}}_j$ for all the remaining words w_j (the words without any pretrained embedding). After this step, we refer to $\{\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_n}\}$ as the completed set of embeddings. To evaluate and compare the quality of the learned embeddings using different methods, we perform the k-Nearest-Neighbors (*kNN*) evaluation described in Algorithm 2 in the Appendix C and provide the reasoning for this evaluation. Essentially, we iteratively leave one company/app out of the set and try to predict its label (industry for finance, app category for app statistics dataset) using its k-Nearest-Neighbors.

TABLE 1. kNN accuracies (%) for the small finance dataset. *E* denotes the name of the embedding set.

E \ k	5	10	20
(w2v)			
LSI	77.41 ± 0.14	74.78 ± 0.17	68.45 ± 0.11
GCN	67.18 ± 0.17	64.51 ± 0.22	60.24 ± 0.18
LDS-GNN	65.00 ± 0.53	63.28 ± 0.46	61.55 ± 0.94
IDGL	68.03 ± 0.31	65.91 ± 0.89	61.05 ± 1.23
SSI	$\mathbf{79.05} \pm 0.35$	$\mathbf{76.59} \pm 0.32$	$\mathbf{75.11} \pm 0.17$
(GloVe)			
LSI	79.09 ± 0.17	78.85 ± 0.21	69.89 ± 0.18
GCN	67.06 ± 0.11	62.79 ± 0.17	58.76 ± 0.17
LDS-GNN	73.63 ± 0.45	71.94 ± 0.69	65.90 ± 0.79
IDGL	68.78 ± 0.61	65.50 ± 0.54	55.23 ± 0.35
SSI	$\mathbf{80.65} \pm 0.17$	$\mathbf{80.69} \pm 0.25$	$\mathbf{77.82} \pm 0.25$
(FastText)			
LSI	72.93 ± 0.22	70.39 ± 0.39	63.32 ± 0.11
GCN	65.79 ± 0.11	63.36 ± 0.18	59.13 ± 0.29
LDS-GNN	67.55 ± 1.14	64.47 ± 0.68	59.66 ± 1.14
IDGL	66.94 ± 0.79	67.76 ± 0.52	62.83 ± 0.68
SSI	$\mathbf{76.46} \pm 0.18$	$\mathbf{75.15} \pm 0.48$	$\mathbf{73.51} \pm 0.70$

The results are shown in Table 1, Table 2, Table 3 for the small finance, large finance and Mobile App Statistics datasets, respectively. We run every experiment five times and compute the mean and standard deviation. We perform the *kNN* classification and gradually change the k to examine the robustness of different methods under different choices of k. From Table 1, Table 2 and Table 3 (see Appendix D for more detailed tables) we make the following observations: (1) SSI, LDS-GNN and IDGL often outperform GCN, demonstrating the effectiveness of using latent graph learning compared to using fixed graphs. This result highlights the potential of these approaches in improving the mapping function from the side information space to the embedding space and consequently, their ability to improve overall embedding quality; (2) LSI outperforms GCN, LDS-GNN and IDGL. This result shows that neither the capacity improvement brought by the MLP parameters nor the latent graph learning approach can bring enough improvement to outperform LSI. A possible explanation for this is LSI's deterministic convergence, which guarantees the model to reach the optimal solution once the hyperparameters are set, which is not the case for the other models. (3) SSI outperforms LDS-GNN and IDGL, indicating that latent graph learning guided by self-supervision is able to learn more robust graphs for domain adaptation compared to the alternative approaches offered by LDS-GNN and IDGL. (4) SSI outperforms all baselines, proving that GCN based methods combined with a robust latent graph learning strategy is the most effective way to perform embedding imputation.

TABLE 2. kNN accuracies (%) for the large finance dataset.

E \ k	5	10	20
(w2v)			
LSI	45.09 ± 0.12	46.10 ± 0.16	48.10 ± 0.05
GCN	35.28 ± 0.11	36.75 ± 0.13	37.09 ± 0.05
LDS-GNN	45.12 ± 0.59	46.67 ± 0.83	46.57 ± 0.67
IDGL	42.11 ± 0.72	43.04 ± 0.64	44.60 ± 0.83
SSI	**50.43** ± 0.61	**51.36** ± 0.28	**51.94** ± 0.49
(GloVe)			
LSI	46.31 ± 0.05	48.64 ± 0.16	48.94 ± 0.13
GCN	36.26 ± 0.04	37.21 ± 0.02	37.78 ± 0.07
LDS-GNN	42.61 ± 0.32	43.99 ± 0.52	44.20 ± 0.52
IDGL	39.39 ± 0.19	41.57 ± 0.32	41.76 ± 0.36
SSI	**51.84** ± 0.48	**52.67** ± 0.47	**53.64** ± 0.42
(FastText)			
LSI	47.91 ± 0.13	49.02 ± 0.08	48.87 ± 0.13
GCN	38.04 ± 0.10	38.61 ± 0.13	38.44 ± 0.10
LDS-GNN	44.21 ± 0.35	45.27 ± 0.72	44.95 ± 0.49
IDGL	40.00 ± 0.57	40.47 ± 0.43	39.66 ± 0.12
SSI	**52.43** ± 0.25	**53.60** ± 0.20	**53.88** ± 0.27

TABLE 3. kNN accuracies (%) for the mobile app statistics dataset.

E \ k	5	10	20
(w2v)			
LSI	79.25 ± 0.08	78.24 ± 0.09	77.14 ± 0.04
GCN	76.48 ± 0.05	75.94 ± 0.21	74.64 ± 0.31
LDS-GNN	78.73 ± 0.25	78.38 ± 0.24	76.56 ± 0.24
IDGL	72.83 ± 0.34	72.37 ± 0.21	70.06 ± 0.42
SSI	**81.00** ± 0.22	**80.68** ± 0.16	**79.01** ± 0.22
(GloVe)			
LSI	79.58 ± 0.05	78.35 ± 0.14	77.60 ± 0.11
GCN	76.97 ± 0.35	75.94 ± 0.35	75.17 ± 0.44
LDS-GNN	78.94 ± 0.06	78.49 ± 0.13	77.31 ± 0.11
IDGL	70.48 ± 0.56	69.62 ± 0.41	67.70 ± 0.13
SSI	**80.24** ± 0.13	**79.47** ± 0.15	**77.91** ± 0.19

B. SENSITIVITY ANALYSES

1) NUMBER OF ANCHORS (m)

In this section, we evaluate the effectiveness of the anchor sampling process in reducing computation complexity while retaining the model's accuracy. For the large finance dataset and fastText, fixing the found optimal hyperparameters and only varying the number of anchors, we train SSI multiple times and observe the *kNN* accuracies (for k = 30) and runtimes for building a single graph. The reported results are the averages of five different runs for each configuration. Figure 4 confirms that the Anchor-*kNN* achieves a comparable accuracy based on an approximate graph that only considers the node affinity concerning a small subset of nodes (anchors) in comparison with the exact solution that has quadratic time complexity and requires comprehensive pair-wise calculations among all nodes. With 500 and 1000 anchors, Anchor-*kNN* loses around 1%-2% accuracy, but manages to decrease construction time significantly, by 77.6% and 63.3%, respectively. Using 2000 anchors (approximately half of all the nodes) roughly preserves original performance while reducing the construction time by 43.4%. Recall that the complexity of anchor graph construction is linear to the number of anchors and the number of all graph nodes. Our algorithm uses a constant number of anchors, achieves comparable performance to those full-scale graph-based algorithms, and has the advantage of the overall linear complexity in terms of the number of nodes.

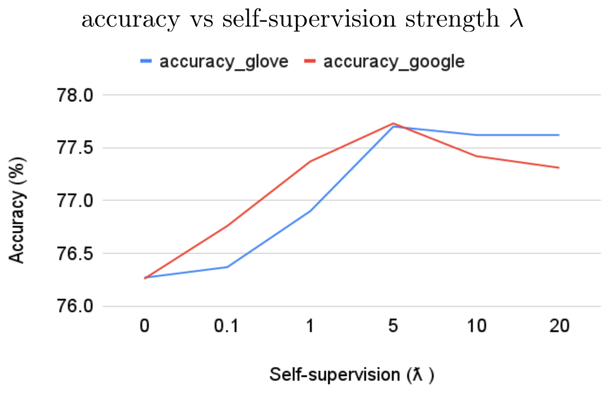


FIGURE 3. Effect of λ. Missing word2vec and GloVe embeddings are imputed. *kNN* results (k = 30) for different λ are shown. We report the mean of 5 different runs.

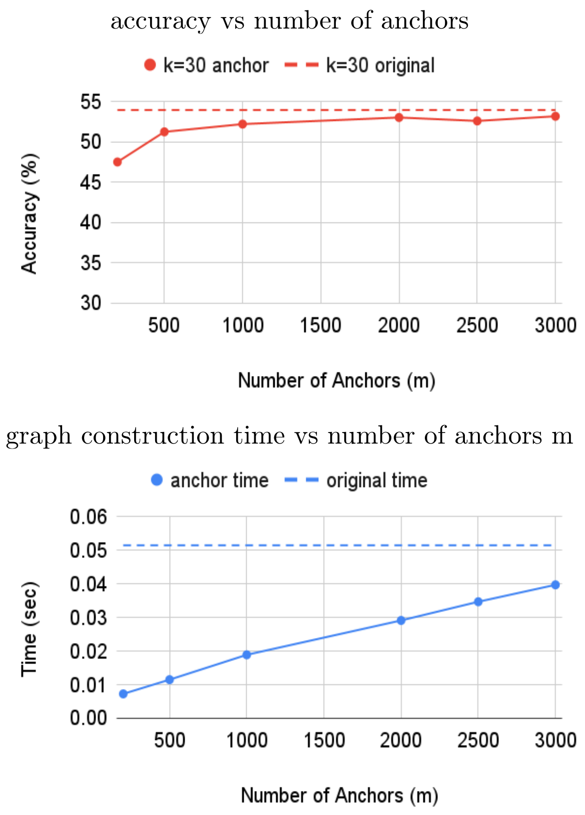


FIGURE 4. Effect of Anchor-*kNN* on model performance and runtime. kNN classification results (k = 30) and Anchor-*kNN* graph construction times (for building a single graph) are shown versus different number of anchors. Dashed lines indicate the original accuracy/runtime without Anchor-*kNN*. We observe that the graph construction time can be reduced by 43.4% while preserving the original accuracy.

2) SELF-SUPERVISION STRENGTH (λ)

Figure 3 shows the results for the Mobile App Statistics dataset for different values of λ. We train multiple models with varying λ values using the optimal hyperparameters. Note that $\lambda = 0$ corresponds to no self-supervision. For both word2vec and glove embeddings, we observe that when λ increases up to a particular value, the model

TABLE 4. Detailed kNN accuracies (%) for the small finance dataset. Mean and standard deviation of 5 different runs are reported. k denotes the number of neighbors in a kNN classifier and E denotes the name of the embedding set.

k / E	2	5	8	10	15	20	30
(w2v)							
LSI	**79.05** $\pm$ 0.01	77.41 $\pm$ 0.14	75.64 $\pm$ 0.11	74.78 $\pm$ 0.17	71.99 $\pm$ 0.11	68.45 $\pm$ 0.11	64.68 $\pm$ 0.01
GCN	70.67 $\pm$ 0.17	67.18 $\pm$ 0.17	65.62 $\pm$ 0.31	64.51 $\pm$ 0.22	62.21 $\pm$ 0.41	60.24 $\pm$ 0.18	56.13 $\pm$ 0.37
LDS-GNN	62.33 $\pm$ 0.92	65.00 $\pm$ 0.53	63.77 $\pm$ 0.54	63.28 $\pm$ 0.46	62.78 $\pm$ 0.95	61.55 $\pm$ 0.94	58.39 $\pm$ 0.34
IDGL	69.67 $\pm$ 1.01	68.03 $\pm$ 0.31	66.18 $\pm$ 0.66	65.91 $\pm$ 0.89	64.47 $\pm$ 0.94	61.05 $\pm$ 1.23	55.30 $\pm$ 0.66
SSI	78.76 $\pm$ 0.11	**79.05** $\pm$ 0.35	**78.02** $\pm$ 0.20	**76.59** $\pm$ 0.32	**74.25** $\pm$ 0.18	**75.11** $\pm$ 0.17	**72.27** $\pm$ 0.20
(GloVe)							
LSI	**78.85** $\pm$ 0.01	79.09 $\pm$ 0.17	78.85 $\pm$ 0.20	78.85 $\pm$ 0.21	73.55 $\pm$ 0.17	69.89 $\pm$ 0.18	65.74 $\pm$ 0.60
GCN	69.32 $\pm$ 0.11	67.06 $\pm$ 0.11	63.36 $\pm$ 0.26	62.79 $\pm$ 0.17	61.76 $\pm$ 0.09	58.76 $\pm$ 0.17	55.23 $\pm$ 0.32
LDS-GNN	72.23 $\pm$ 0.58	73.63 $\pm$ 0.45	72.52 $\pm$ 0.55	71.94 $\pm$ 0.69	68.62 $\pm$ 0.62	65.90 $\pm$ 0.79	62.82 $\pm$ 0.75
IDGL	69.06 $\pm$ 0.23	68.78 $\pm$ 0.61	66.80 $\pm$ 0.31	65.50 $\pm$ 0.54	60.23 $\pm$ 1.36	55.23 $\pm$ 0.35	50.85 $\pm$ 0.83
SSI	77.86 $\pm$ 0.22	**80.65** $\pm$ 0.17	**79.83** $\pm$ 0.17	**80.69** $\pm$ 0.25	**78.76** $\pm$ 0.18	**77.82** $\pm$ 0.25	**74.98** $\pm$ 0.33
(FastText)							
LSI	75.40 $\pm$ 0.09	72.93 $\pm$ 0.22	71.62 $\pm$ 0.33	70.39 $\pm$ 0.39	65.53 $\pm$ 0.22	63.32 $\pm$ 0.11	54.98 $\pm$ 0.17
GCN	67.18 $\pm$ 0.22	65.79 $\pm$ 0.11	63.32 $\pm$ 0.23	63.36 $\pm$ 0.18	61.31 $\pm$ 0.11	59.13 $\pm$ 0.29	58.56 $\pm$ 0.09
LDS-GNN	68.33 $\pm$ 0.72	67.55 $\pm$ 1.14	65.62 $\pm$ 0.97	64.47 $\pm$ 0.68	63.44 $\pm$ 1.49	59.66 $\pm$ 1.14	56.62 $\pm$ 0.94
IDGL	65.50 $\pm$ 0.69	66.94 $\pm$ 0.79	67.97 $\pm$ 0.41	67.76 $\pm$ 0.52	65.71 $\pm$ 0.93	62.83 $\pm$ 0.68	57.29 $\pm$ 0.73
SSI	**76.34** $\pm$ 0.17	**76.46** $\pm$ 0.18	**76.26** $\pm$ 0.23	**75.15** $\pm$ 0.48	**74.62** $\pm$ 0.27	**73.51** $\pm$ 0.70	**70.96** $\pm$ 0.23

TABLE 5. Detailed kNN accuracies (%) for the large finance dataset. Mean and standard deviation of 5 different runs are reported.

k / E	2	5	8	10	15	20	30
(w2v)							
LSI	40.49 $\pm$ 0.10	45.09 $\pm$ 0.12	45.66 $\pm$ 0.17	46.10 $\pm$ 0.16	47.16 $\pm$ 0.14	48.10 $\pm$ 0.05	47.77 $\pm$ 0.05
GCN	32.57 $\pm$ 0.07	35.28 $\pm$ 0.11	36.55 $\pm$ 0.09	36.75 $\pm$ 0.13	36.87 $\pm$ 0.08	37.09 $\pm$ 0.05	36.55 $\pm$ 0.09
LDS-GNN	40.04 $\pm$ 0.38	45.12 $\pm$ 0.59	46.28 $\pm$ 0.55	46.67 $\pm$ 0.83	46.89 $\pm$ 0.76	46.57 $\pm$ 0.67	46.17 $\pm$ 0.68
IDGL	36.27 $\pm$ 0.91	42.11 $\pm$ 0.72	42.52 $\pm$ 0.43	43.04 $\pm$ 0.64	43.70 $\pm$ 0.59	44.60 $\pm$ 0.83	43.43 $\pm$ 0.54
SSI	**45.49** $\pm$ 0.38	**50.43** $\pm$ 0.61	**51.17** $\pm$ 0.49	**51.36** $\pm$ 0.28	**51.94** $\pm$ 0.41	**51.94** $\pm$ 0.49	**51.82** $\pm$ 0.51
(GloVe)							
LSI	41.86 $\pm$ 0.09	46.31 $\pm$ 0.05	48.67 $\pm$ 0.21	48.64 $\pm$ 0.16	49.03 $\pm$ 0.11	48.94 $\pm$ 0.13	48.17 $\pm$ 0.15
GCN	34.33 $\pm$ 0.07	36.26 $\pm$ 0.04	36.97 $\pm$ 0.08	37.21 $\pm$ 0.02	38.04 $\pm$ 0.06	37.78 $\pm$ 0.07	37.31 $\pm$ 0.09
LDS-GNN	37.70 $\pm$ 0.48	42.61 $\pm$ 0.32	43.75 $\pm$ 0.53	43.99 $\pm$ 0.52	43.70 $\pm$ 0.33	44.20 $\pm$ 0.52	44.40 $\pm$ 0.26
IDGL	35.65 $\pm$ 0.57	39.39 $\pm$ 0.19	40.66 $\pm$ 0.61	41.57 $\pm$ 0.32	41.30 $\pm$ 0.21	41.76 $\pm$ 0.36	41.50 $\pm$ 0.17
SSI	**46.49** $\pm$ 0.47	**51.84** $\pm$ 0.48	**52.15** $\pm$ 0.28	**52.67** $\pm$ 0.47	**53.34** $\pm$ 0.14	**53.64** $\pm$ 0.42	**53.83** $\pm$ 0.21
(FastText)							
LSI	42.49 $\pm$ 0.05	47.91 $\pm$ 0.13	47.91 $\pm$ 0.13	49.02 $\pm$ 0.08	49.15 $\pm$ 0.13	48.87 $\pm$ 0.13	49.32 $\pm$ 0.10
GCN	34.01 $\pm$ 0.11	38.04 $\pm$ 0.10	39.61 $\pm$ 0.08	38.61 $\pm$ 0.13	38.88 $\pm$ 0.06	38.44 $\pm$ 0.10	37.75 $\pm$ 0.11
LDS-GNN	38.73 $\pm$ 0.75	44.21 $\pm$ 0.35	44.83 $\pm$ 0.56	45.27 $\pm$ 0.72	44.97 $\pm$ 0.51	44.95 $\pm$ 0.49	44.33 $\pm$ 0.31
IDGL	35.97 $\pm$ 0.25	40.00 $\pm$ 0.57	40.10 $\pm$ 0.16	40.47 $\pm$ 0.43	39.91 $\pm$ 0.49	39.66 $\pm$ 0.12	38.83 $\pm$ 0.24
SSI	**48.19** $\pm$ 0.29	**52.43** $\pm$ 0.25	**53.31** $\pm$ 0.19	**53.60** $\pm$ 0.20	**53.93** $\pm$ 0.53	**53.88** $\pm$ 0.27	**53.71** $\pm$ 0.39

TABLE 6. Detailed kNN accuracies (%) for the Mobile App Statistics dataset. Mean and standard deviation of five runs are reported.

k / E	2	5	8	10	15	20	30
(w2v)							
LSI	78.50 $\pm$ 0.08	79.25 $\pm$ 0.08	78.26 $\pm$ 0.06	78.24 $\pm$ 0.09	77.57 $\pm$ 0.10	77.14 $\pm$ 0.04	75.73 $\pm$ 0.12
GCN	76.37 $\pm$ 0.32	76.48 $\pm$ 0.05	76.12 $\pm$ 0.11	75.94 $\pm$ 0.21	75.10 $\pm$ 0.43	74.64 $\pm$ 0.31	73.59 $\pm$ 0.14
LDS-GNN	77.36 $\pm$ 0.09	78.73 $\pm$ 0.25	78.41 $\pm$ 0.18	78.38 $\pm$ 0.24	77.65 $\pm$ 0.05	76.56 $\pm$ 0.24	75.93 $\pm$ 0.17
IDGL	73.06 $\pm$ 0.16	72.83 $\pm$ 0.34	72.54 $\pm$ 0.32	72.37 $\pm$ 0.21	71.23 $\pm$ 0.35	70.06 $\pm$ 0.42	68.58 $\pm$ 0.08
SSI	**80.28** $\pm$ 0.09	**81.00** $\pm$ 0.22	**80.94** $\pm$ 0.09	**80.68** $\pm$ 0.16	**79.84** $\pm$ 0.12	**79.01** $\pm$ 0.22	**77.44** $\pm$ 0.18
(GloVe)							
LSI	79.25 $\pm$ 0.06	79.58 $\pm$ 0.05	78.56 $\pm$ 0.12	78.35 $\pm$ 0.14	77.95 $\pm$ 0.05	77.60 $\pm$ 0.11	76.07 $\pm$ 0.09
GCN	76.62 $\pm$ 0.24	76.97 $\pm$ 0.35	76.56 $\pm$ 0.15	75.94 $\pm$ 0.35	75.69 $\pm$ 0.23	75.17 $\pm$ 0.44	74.47 $\pm$ 0.09
LDS-GNN	77.41 $\pm$ 0.12	78.94 $\pm$ 0.06	78.55 $\pm$ 0.21	78.49 $\pm$ 0.13	78.13 $\pm$ 0.34	77.31 $\pm$ 0.11	76.30 $\pm$ 0.27
IDGL	70.58 $\pm$ 0.34	70.48 $\pm$ 0.56	70.33 $\pm$ 0.38	69.62 $\pm$ 0.41	68.67 $\pm$ 0.26	67.70 $\pm$ 0.13	66.42 $\pm$ 0.47
SSI	**79.67** $\pm$ 0.26	**80.24** $\pm$ 0.13	**79.75** $\pm$ 0.13	**79.47** $\pm$ 0.15	**78.38** $\pm$ 0.24	**77.91** $\pm$ 0.19	**76.87** $\pm$ 0.04

accuracy improves for both glove and google, confirming that self-supervision indeed enhances the model robustness and performance.

Once λ reaches a specific value, the model performance peaks. After this point, increasing λ deteriorates the performance. This behavior is similar to cases where applying too much regularization starts to introduce a lot of bias. In fact, the self-supervision loss term indeed serves as a regularization mechanism to ensure that the model trades off between the supervised learning objective and graph robustness in preserving node features.

VII. DISCUSSION AND CONCLUSION

In this paper, we tackle the problem of embedding imputation with the recent advances in graph neural networks. Instead of using a pre-computed graph, we use the idea of self-supervision to learn and evolve graph structure during training to address the challenge of converting the side information into a suitable network structure for imputation. Combining the reconstruction loss of the original node features and the actual prediction task has a close connection to regularization and is highly effective. We also integrate the idea of anchor sampling into our framework to reduce the complexity of graph construction for scalability. The approach yields performance superiority and robustness on multiple tasks and numerous datasets compared to previous works. We anticipate our embedding imputation technology will be especially useful in domain NLP tasks.

APPENDIX A IMPLEMENTATION DETAILS

The hyperparameters for all models are tuned using the losses on the validation set (mean squared error between predicted and original embedding). After finding the optimal hyperparameters, we train and test each model using 5 random seeds and report the average results and standard deviations on the test sets. We use the Adam [22] for all models except LSI unless stated otherwise, and tune the weight decay parameter from (0.0, 1e-7, 1e-6, 1e-5).

For LSI and GCN, we tune the number of neighbors δ for constructing the MST-kNN graph from (10, 20, 30). We tune the learning rate from (1e-5, 1e-4, 1e-3) and apply dropout on the adjacency matrix and the weight matrices with the keep probabilities selected from (1.0, 0.75, 0.5), applied at each layer. We train for 400 epochs. For LSI, stopping criterion η is set to 1e-4.

For LDS-GNN, the inner objective function is set as the regularized mean-squared error and optimized with Adam. The outer objective is set as the unregularized mean-squared error and optimized using Stochastic Gradient Descent. We further split the validation set evenly to form the validation and early stopping (patience = 20) sets. We use the kNN-LDS version and tune: outer optimization learning rate from (1e-5, 1e-4, 1e-3, 1e-2), inner optimization learning rate from (1e-5, 1e-4, 1e-3, 1e-2), hyper batch size from (5, 10, 15), distance metric from (cosine, minkowski) and k from (10, 20, 30) for building the initial kNN graph, keep probability for dropout on the GCN weight matrices from (1.0, 0.75, 0.5), L2 regularization on the weights from (0.0, 1e-7, 1e-6, 1e-5, 1e-4).

For IDGL, input graph knn size is tuned from (10, 20, 30). Weight dropout and adjacency dropout rate is tuned from (0.0, 0.25, 0.5). A two layer GCN is used for the GNN module. Weighted cosine is used for the graph metric type. λ is to 0.9, η is set to 0.2, δ is set to 8.5e-5. Number of perspectives is set to 4. Learning rate is tuned from (1e-5, 1e-4, 1e-3, 1e-2), early stopping is applied with a patience of 20 epochs.

For SSI, we apply dropout on the weight matrices for GNC_R with keep probability selected from (1.0, 0.75, 0.5), applied at each layer. We tune the learning rates for GCN_R and GCN_S from (1e-5, 1e-4, 1e-3), the mask out ratio, i.e., what portion of entries to mask out (convert to zero) on the feature matrix from (0.1, 0.2, 0.5, 0.9), k for the kNN function in the Graph Generator from (10, 20, 30), self-supervision strength parameter λ from (0.1, 1, 2, 3, 4, 5, 10), all MLP activations from (tanh, ReLU). Graph Generator uses two 300×300 diagonal weight matrices. We train for 400 epochs and the first 20% of the epochs are used to train only the self-supervision module.

Finally, for a fair comparison, we keep the capacity of the regression modules the same, i.e., GCN, inner objective GCN of LDS-GNN, GNN module of IDGL and GCN_R of SSI all have two layers of 600 hidden units.

APPENDIX B DATASET DETAILS

A. FINANCE DATASET

For the small dataset, GloVe [3] pretrained embedding contains 207 out of 488 company names, fastText [1] contains 263, and Word2vec [2] contains 119 respectively. For the large dataset, GloVe, fastText and Word2vec pretrained embedding contain 192, 399, 764 of those 4092 company names, respectively.

Each company has an industry category label, e.g., Google belongs to the IT industry, while Blackrock belongs to the financial industry. There are eleven different category labels representing eleven industry sectors. Every company also has a historical daily trading return vector $\vec{r} = [r_{t_1}, r_{t_2}, \ldots, r_{t_f}]$ available as side information. For the small dataset, this vector contains the daily stock returns from 2016-08-24 to 2018-08-27. For the large dataset, this vector contains the daily returns for 400 trading days ending on 2018-11-01. For each dataset, we obtain the matrix of returns $X \in \mathbb{R}^{n \times f}$ by stacking the individual return vectors of all companies.

For all three pretrained embedding sets, the companies that are not contained in the embedding set form the test set. We perform a stratified split on the remaining companies by forcing the distribution of the industry category labels to be same across training and validation sets. We designate 80% of the data as training set and the remaining 20% as the validation set. Training and hyperparameter tuning details are provided in A.

B. MOBILE APP STATISTICS DATASET

The Mobile App Statistics dataset from Kaggle[1] contains more than 7000 Apple iOS mobile application details extracted from the iTunes Search API at the Apple Inc website. Each app has a name and a primary genre such as Games, Sports, or Business. There are 23 possible genres in total.

Each app also has numerical features including: "price": Price amount, "size_bytes": Size (in Bytes), "ratingcounttot": User Rating counts (for all version), "ratingcountver": User Rating counts (for current version), “user_rating”: Average User Rating value (for all version), "userratingver": Average User Rating value (for current version), "ver": Latest version code, "sup_devices_num": Number of supporting devices, "ipadSc_urls.num": Number of screenshots showed for display, "lang.num": Number of supported languages. There are also categorical features including: "cont_rating": Content (Maturity) Rating (e.g., 7+, 13+), "vpp_lic": Vpp Device Based Licensing Enabled (True or False). "currency": Currency Type. We drop the ver and currency (only one currency) columns because they to do not provide any useful

[1]https://www.kaggle.com/ramamet4/app-store-apple-data-set-10k-apps

information. We also drop the ratingcountver and userratingver columns because they are only valid for the current version of the app. Instead, we keep ratingcounttot and user_rating.

We convert the categorical features into one-hot vectors and normalize the numerical features using a min-max normalization. We use word embeddings to transform textual descriptions of each app into an average representation vector $\vec{v} = \frac{1}{n}\sum_{i=1}^{n}\vec{w}_i$, where n is the number of words in the description and $\vec{w}_i$ is the pre-trained embedding from Word2vec or Glove (the same one as the task) for the i-th word. If there are any words in the textual description which is not contained in the pretrained embedding, it is excluded from the computation. We merge the textual representation $\vec{v}$, the numerical features and the one hot vectors for the categorical features to obtain a final feature vector, $\vec{x} \in \mathbb{R}^f$, for each app. We obtain the side information matrix $X \in \mathbb{R}^{n\times f}$ by stacking the individual feature vectors of all app names. Same with the finance task, X is used as the domain matrix to construct the fixed graph for LSI and GCN and also as the feature matrix for GCN, LDS-GNN and SSI.

Word2vec, GloVe, and fastText embeddings already contain 190, 260, and 10 app names, respectively while the remaining app names are missing from these embedding sets. We only work with Word2vec and GloVe (fastText contains only a few apps) and try to impute the missing embeddings using the app features as side information.

For both Word2vec and GloVe, the app names which are not contained in the embedding set form the test set. We perform a stratified split on the remaining companies by forcing the distribution of the primary genre labels to be same across training and validation sets. We designate 80% of the data as training set and the remaining 20% as the validation set. Training and hyperparameter tuning details are provided in A.

APPENDIX C
kNN EVALUATION

The algorithm takes in the completed set of embeddings $\{\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_n}\}$, the list of industry category labels l with the i-th element l_i representing the industry category (or primary genre for the mobile apps dataset) label for w_i and an integer k. The classification is conducted by leaving out one word at a time and predicting its label based on the labels of its k nearest neighbors in terms of Euclidean distance in the embedding space (lines 8-10). We repeat this for all the words in the dataset and compute the overall accuracy, i.e., the ratio of the number of correct predictions to the number of all predictions (line 12).

A company is expected to be semantically more similar to a company from the same industry compared to another from a different industry. For example, Google is more similar to another tech company, Apple, than it is to Walmart. Any effective word imputation method should preserve the semantic locality in the embedding space, in other words, similar words should be embedded closely in the embedding space. Hence, if the imputation method works well, we should be able to accurately infer the industry of a company based the industry labels of the nearby companies in the embedding space, resulting in a high k-Nearest-Neighbors accuracy. A similar reasoning also applies to mobile apps and their genres.

Algorithm 2 *kNN* Evaluation

```
Input  : $(y_1, y_2, \ldots, y_n, l, k)$ ;  // each $y_i \in \mathbb{R}^d$ :
         embedding vector for $i$-th
         word, $l \in \mathbb{R}^n$: category labels,
         $k$: number of neighbors
Output: $knn\_accuracy$
1  $preds \leftarrow []$ ;    // initialize empty list
2  for $i$ in $\{1, 2, \ldots, n\}$ do
3      $distances \leftarrow []$ ;    // initialize empty
        distance list for $y_i$
4      for $j$ in $\{1, 2, \ldots, n\} \setminus \{i\}$ do
5          $dist = ||y_i - y_j||_2$
           ;          // distance of $y_j$ to $y_i$
6          $distances.append(dist)$
           ;          // add to end of list
7      end
8      $nbrs = get\_nearest\_neighbors(distances, k)$
9      $\hat{l}_i = get\_majority\_label(nbrs)$
10     $preds.append(\hat{l}_i)$
11 end
12 $knn\_accuracy = compute\_accuracy(preds, l)$
```

APPENDIX D
DETAILED RESULTS

This appendix provides detailed results for the downstream tasks.

ACKNOWLEDGMENT

The content is solely the responsibility of the authors and does not necessarily represent the official views of the DOE.

REFERENCES

[1] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Trans. Assoc. Comput. Linguistics*, vol. 5, pp. 135–146, Dec. 2017.
[2] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.
[3] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1532–1543.
[4] S. Yao, D. Yu, and K. Xiao, "Enhancing domain word embedding via latent semantic imputation," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 557–565.
[5] Z. Yang, C. Zhu, V. Sachidananda, and E. Darve, "Embedding imputation with grounded language information," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 3356–3361.
[6] L. Franceschi, M. Niepert, M. Pontil, and X. He, "Learning discrete structures for graph neural networks," in *Proc. 36th Int. Conf. Mach. Learn.*, vol. 97, PMLR, 2019, pp. 1972–1982.
[7] A. Kazi, L. Cosmo, S. Ahmadi, N. Navab, and M. M. Bronstein, "Differentiable graph module (DGM) for graph convolutional networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 2, pp. 1606–1617, Feb. 2023.

[8] B. Fatemi, L. El Asri, and S. M. Kazemi, "SLAPS: Self-supervision improves structure learning for graph neural networks," in *Advances in Neural Information Processing Systems*, vol. 34, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. W. Vaughan, Eds. Curran, 2021, pp. 22667–22681. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/bf499a12e998d178afd964adf64a60cb-Paper.pdf

[9] Y. Liu, M. Jin, S. Pan, C. Zhou, Y. Zheng, F. Xia, and P. S. Yu, "Graph self-supervised learning: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 6, pp. 5879–5900, Jun. 2023.

[10] W. Liu, J. He, and S.-F. Chang, "Large graph construction for scalable semi-supervised learning," in *Proc. 27th Int. Conf. Int. Conf. Mach. Learn. (ICML)*, Jun. 2010, pp. 679–686.

[11] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, Dec. 2000.

[12] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.

[13] M. Chen, R. Mathews, T. Ouyang, and F. Beaufays, "Federated learning of out-of-vocabulary words," 2019, *arXiv:1903.10635*.

[14] N. Fukuda, N. Yoshinaga, and M. Kitsuregawa, "Robust backed-off estimation of out-of-vocabulary embeddings," in *Proc. Findings Assoc. Comput. Linguistics, EMNLP*, 2020, pp. 4827–4838.

[15] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, CMU CALD Tech. Rep. CMU-CALD-02-107, 2002.

[16] X. J. Zhu, "Semi-supervised learning literature survey," Dept. Comput. Sci., Univ. Wisconsin-Madison, Madison, WI, USA, Tech. Rep. TR1530, 2005. [Online]. Available: https://minds.wisconsin.edu/handle/1793/60444?show=full

[17] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, Dec. 2016, pp. 3844–3852.

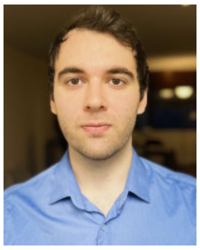
[18] J. Gasteiger, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized PageRank," 2018, *arXiv:1810.05997*.

[19] Y. Chen, L. Wu, and M. Zaki, "Iterative deep graph learning for graph neural networks: Better and robust node embeddings," in *Advances in Neural Information Processing Systems*, vol. 33, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds. Curran, 2020, pp. 19314–19326. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/e05c7ba4e087beea9410929698dc41a6-Paper.pdf

[20] J. Alman, T. Chu, A. Schild, and Z. Song, "Algorithms and hardness for linear algebra on geometric graphs," in *Proc. IEEE 61st Annu. Symp. Found. Comput. Sci. (FOCS)*, Nov. 2020, pp. 541–552.

[21] M. Slaney and M. Casey, "Locality-sensitive hashing for finding nearest neighbors," *IEEE Signal Process. Mag.*, vol. 25, no. 2, pp. 128–131, Mar. 2008.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017, *arXiv:1412.6980*.



URAS VAROLGUNES received the B.A. degree in economics from Bogazici University, in 2019. He is currently pursuing the Ph.D. degree in business data science with the New Jersey Institute of Technology. His research interests include graph neural networks and their applications in recommender systems, healthcare, and natural language processing.

SHIBO YAO received the B.S. degree in management science from the University of Science and Technology of China, in 2015, the M.S. degree in technological systems management from Stony Brook University, in 2016, and the Ph.D. degree in business data science from the New Jersey Institute of Technology, in 2022. He is currently affiliated with Meta as a Research Scientist mainly working on large-scale recommendation systems. He has authored papers in top-tier conferences, such as KDD and ACML. His research interests include graph learning and embedding and their applications.

YAO MA received the B.S. degree in mathematics and applied mathematics from Zhejiang University, in 2015, the M.S. degree in statistics, probability and operations research from the Eindhoven University of Technology, in 2016, and the Ph.D. degree from Michigan State University, in 2021, under the supervision of Dr. Jiliang Tang. He is currently an Assistant Professor with the Department of Computer Science, New Jersey Institute of Technology (NJIT). He has published innovative works in top-tier conferences, such as WSDM, ASONAM, ICDM, SDM, WWW, KDD, and IJCAI. For more information visit the link (https://web.njit.edu/ỹm329).

DANTONG YU received the B.S. degree in computer science from Beijing University, China, in 1995, and the Ph.D. degree in computer science from the State University of New York at Buffalo, USA, in 2001. He designed and implemented a novel high-dimensional indexing algorithm (termed ClusterTree) using the semantics of datasets. He has published papers in leading technical journals and conferences. Recently, he has published papers in *ACM Knowledge Discovery and Data Mining*, *ACM Transactions on Knowledge Discovery from Data* (ACM TKDD), IEEE Transactions on Knowledge and Data Engineering, *Quantitative Finance*, and *Journal of Behavior and Experimental Finance*. His research interests include FinTech, machine learning, business data science, high-performance computing, data mining, database, and data warehouse. He serves on the Organization Committee for ACM KDD 2022. He is also a PC Member of CIKM, ICDM, KDD, and SIAM Data Mining. He served in the review panels for NSF CDI and DOE Early Career Principal Investigator for networking research and DOE Small Business Innovative Research (SBIR) and the Co-Chair of several DOE Advanced Networking Workshops for Distributed Petascale Science.