Efficient and effective training of language and graph neural network models

Vassilis N. Ioannidis, Xiang Song, Da Zheng, George Karypis
Amazon Web Services AI, USA
Houyu Zhang, Jun Ma, Yi Xu, Belinda Zeng, Trishul Chilimbi
Amazon Search AI, USA

Abstract

Can we combine heterogenous graph structure with text to learn high-quality semantic and behavioural representations? Current GNN approaches are challenged by textual features, which typically need to be encoded to a numerical vector before provided to the GNN that may incur some information loss. In this paper, we put forth an efficient and effective framework termed language model GNN (LM-GNN) to jointly train large-scale language models and graph neural networks. The effectiveness in our framework is achieved by applying stage-wise fine-tuning of the BERT model first with heterogenous graph information and then with a GNN model. Several system and design optimizations are proposed to enable scalable and efficient training. We evaluate the LM-GNN framework in different datasets and tasks and showcase the effectiveness of the proposed approach.

Introduction

GNNs have shown remarkable success in a variety of graph machine learning tasks both in supervised and unsupervised learning settings (Hamilton, Ying, and Leskovec 2017b). Typically, the graphs used for profiling GNN models have node features as numerical attributes. These numerical attributes may be the output of network that encodes a much richer original information that is in the form of text or picture. One could apply such a general pre-trained network to extract the represenations and use them as feature vectors in a GNN. However, as we detail in this work such an approach is not optimal. This raises the question of how can we train better GNN models with rich text features. This work presents a stage-wise fine-tuning framework termed LM-GNN for encoding text data with transformers and GNN models. Our stage-wise fine-tuning, besides achieving good performance, significantly reduces training time compared to end-to-end training. Further, LM-GNN is a distributed framework which can scale to hundreds of millions nodes.

Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.
Accepted to Workshop on Knowledge Augmented Methods for Natural Language Processing, in conjunction with AAAI 2023.

Related work

GNNs achieve state-of-the-art performance in node classification by utilizing regular graph convolution (Kipf and Welling 2017) or graph attention (Veličković et al. 2018), while these models have later been extended in the heterogeneous graph setting (Schlichtkrull et al. 2018; Fu et al. 2020; Wang et al. 2019). Similarly, GNNs excel in performance for link prediction with numerous applications in recommendation systems (Wang et al. 2017) and drug discovery (Zhou et al. 2020; Ioannidis, Zheng, and Karypis 2020). Knowledge-graph (KG) embedding models for link prediction rely on mapping the nodes and edges of the KG to a vector space by maximizing a score function for existing KG edges (Wang et al. 2017; Yang et al. 2014; Zheng et al. 2020).

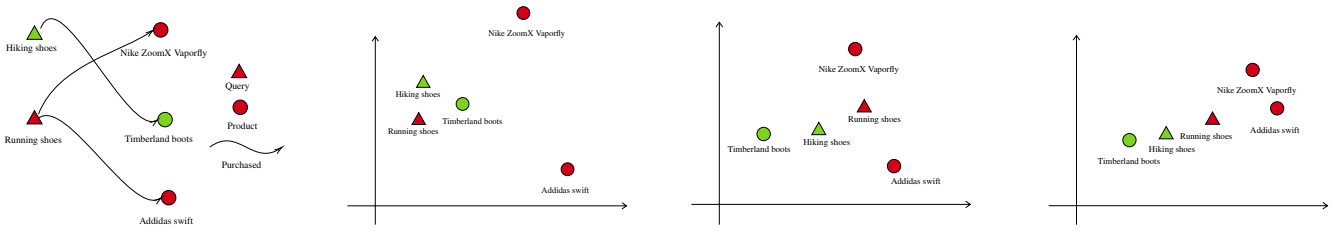
Language models (LM)s are powerful in modeling text data (Devlin et al. 2018). Harnessing the power of LMs with graph data is under-explored. This work details a framework for training large-scale LMs jointly with GNNs. (Wu et al. 2021) details recent works for learning graph structure from text, which also explores GNNs for modeling text data. However, in this work our focus is in improving the prediction performance given a known graph structure. Recent work (Chien et al. 2021) also identifies that pre-training BERT models in graph data can be beneficial and exploits a neighborhood prediction objective to enrich the BERT model with graph information. However, this work (Chien et al. 2021) did not explore to fine-tune the BERT and GNN model together. Another prominent work in (Li et al. 2021; Zhu et al. 2021) trains GNN models for improving the search results in sponsored search. The work there can be seen as a special case of this framework, although (Li et al. 2021; Zhu et al. 2021) did not explore the stage-wise fine-tuning that we introduce in this work.

LM-GNN Models : Adapt and fine-tune

The LM-GNN framework learns informative representations by stage-wise fine-tuning that gradually fuses the transformer with graph information.

Semantic encoder

We employ the BERT model (Devlin et al. 2018) as the transformer in the LM-GNN framework to encode the nodes textual semantics. Given a node's text BERT encodes the



(a) Query to products graph. (b) Transformer embeddings. (c) Graph-aware transformer embeddings. (d) LM-GNN embedding.

Figure 1: (a) The underlying graph among products and queries where an edge signifies that a query leads to the purchase of a product. (b-d) The 2-D projected embeddings as generated by different pipelines. (b) The transformer maps entities solely on text and fails to capture semantic similarity, besides language based e.g., shoes are close to boots. (c) The graph-aware transformer maps connected entities close, however fails at capturing higher order relations and embeds the two running shoes in different regions. (d)The proposed LM-GNN captures the connectivity, higher order structure, text semantics and provides a refined representations useful for retrieval tasks.

textual information to a $F \times 1$ embedding vector $\mathbf{x}_{n_t}$. This embedding vector corresponds to the [CLS] token embedding of the BERT model and the mapping from the text to the embedding is defined as $\mathbf{x}_{n_t} := g_{\text{BERT}}(n_t; \mathbf{W}_{\text{BERT}})$. The mapping is controlled by the learnable parameters $\mathbf{W}_{\text{BERT}}$. These parameters can be instantiated by any language model pre-training approach, e.g., masked language modeling (MLM). Pre-training BERT models on large unlabeled text data has shown significant benefit in different LM applications. However, transferring this benefit for graph ML applications is not straightforward. We employ the technique in Sec. to pre-train BERT models with graph data. For different node-types $t \in \{1, \ldots, T\}$ in the graph consider different semantic encoders; e.g. queries and products.

Graph Encoder

Although the LM-GNN framework can utilize any GNN model as an encoder (Wu et al. 2020), in this paper LM-GNN uses a modified RGCN encoder (Schlichtkrull et al. 2018). RGCNs extend the graph convolution operation (Kipf and Welling 2017) to heterogeneous graphs. The lth self-RGCN layer computes the nth node representation $\mathbf{h}_n^{(l+1)}$ as follows

$$\mathbf{h}_n^{(l+1)} := \sigma\left(\mathbf{W}_{\text{self}}^{(l)}\mathbf{h}_n^{(l)} + \sum_{r=1}^{R}\sum_{n' \in \mathcal{N}_n^r} \mathbf{W}_r^{(l)}\mathbf{h}_{n'}^{(l)}\right)$$

where $\mathcal{N}_n^r$ is the neighborhood of node n under relation r, σ the rectified linear unit non linear function, $\mathbf{W}_r^{(l)}$ is a learnable matrix associated with the rth relation, and $\mathbf{W}_{\text{self}}^{(l)}$ is a projection matrix for the nodes embedding in layer l.

Structure prediction task

Structure prediction models utilize a contrastive loss function scores positive and negative examples (Zheng et al. 2020). Positive examples are the set of existing links in the graph. Negative examples are typically sampled from the missing links in the graph. For each *positive triplet* $q = (n_t, r, n'_{t'})$ a number of negative links is generated by corrupting the head and tail entities at random $(n_t, r, \nu'_{t'})$ and $(\nu_t, r, n'_{t'})$. The minimization function for link prediction can be defined as follows

$$\sum_{(n_t, r, n'_{t'}) \in \mathbb{D}^+ \cup \mathbb{D}^-} \log(1 + \exp(-y \times c(n_t, r, n'_{t'})), \quad (1)$$

where c is a scoring function that return as scalar given the head, and tail nodes and the relation such as the DistMult model (Yang et al. 2014), $\mathbb{D}^+$ and $\mathbb{D}^-$ are the positive and negative sets of triplets and $y = 1$ if the triplet corresponds to a positive example and -1 otherwise.

LM-GNN: Training at scale

A straightforward approach would directly use the LM encoder as a semantic encoder that feeds representations to the GNN encoder, and train such an architecture in an end-to-end fashion. However, training large scale language models and graph neural networks involves challenges relating to efficiency and effectiveness.

Effectiveness challenges stem from the fact that the pre-trained language model is well optimized in language tasks but has not trained before in graph tasks, which surfaces three main issues. (1) Using such pre-trained transfomers may not be the most appropriate initialization and may trap the GNN to a sub-optimal local minimum. (2) Further, the well optimized transformer for the text tasks, may be more resistant in parameter updates. (3) Another hurdle stems from the random initialization of the GNN weights relative to the well-attuned transformer model, which may challenge the optimization of such an end-to-end framework.

Efficiency challenges relate to the large number of neighbors required by message passing in GNNs. In mini-batch training of a k-layer GNN the k-hop ego-network of every target node is created and the target node embedding is computed as a function of all the node in the expanded ego-network (also known as source nodes). The number of source nodes in an ego-network may be very large even for shallow GNNs. Alleviating this issue, recent GNN approaches apply random sampling (Hamilton, Ying, and Leskovec 2017a)

to reduce the number of neighbors. However, even with a shallow GNN (2 layers) and modest sampling (20 neighbors per layer), there are up to 400 source nodes. This remains a serious challenge in our unique setting where the transformer model needs to make 400 forward passes to calculate the embedding of a single target node. As a consequence the size of the required GPU is quite large even for small mini-batch sizes, which is a unique challenge in our framework.

Addressing effectiveness

Consider the search graph among products and queries depicted in Fig. 1a. Such a graph is typically encountered in catalog systems for query-product datasets. One could attempt to directly use the embedding generated by a transformer as an input to a GNN model for further fine-tuning. However, the transfomer embedding of such a model will only take into account the text information and may introduce noise at message passing. Indeed, Fig. 1b shows that embeddings that are connected in the graph may be located in different regions of the embedding space. The poor performance of such a scheme is also detailed in the experiments; see Section . Our contribution in this context is to pre-fine-tune the transformer with graph information, which will endow the text embeddings with relational semantics and boost the performance when used as a semantic encoder.

Graph-aware pre-fine-tuning. We consider the structure prediction decoder that directly uses the scoring function c instantiated in Section . The graph-aware transformer model directly uses the structure prediction decoder as a supervision. Specifically, the transformer generates the CLS token embeddings for the text associated with the nodes and the vectors are contributing to the loss (1). The resulting graph-aware transformer embeddings respect both the semantics introduced by the language as well as the relations imposed by the graph; see also Fig. 1c. The graph-aware pre-fine-tuning also results LM that is more suitable for the end-to-end training with the GNN, which is also supported in Section . The top part of Fig. 2 showcases the graph-aware pre-fine-tuning pipeline.

Our proposed framework LM-GNN employs the graph-aware transformer as a semantic encoder that first embeds the text and then is fed to the GNN encoder. However, since the GNN model is typically initialized at random this may challenge the end-to-end fine-tuning method and get trapped in not desirable local minima. Hence, we warm-start the GNN weights by keeping fixed the transformer weights for a few iterations and optimize only the GNN encoder. Finally, we fine-tune end-to-end the semantic encoders and GNN models for the downstream task. Such a scheme will provide a good initial point for the GNN model. The resulting embeddings abide by the text semantics, graph relations and the multi-hop graph structure; see Fig. 1d. Our overall stage-wise fine-tuning pipeline is depicted in the bottom part of Fig. 2.

Addressing efficiency

The high computation overhead and memory consumption required by the LM-GNN framework limits the wide applicability of the approach. Addressing these issues, we adopt several optimizations to efficiently train LM-GNN.

Back-propagate on samples. Instead of back-propagating gradients to the transformer models on all nodes, we subsample a fixed-size number of nodes (train nodes) in a mini-batch where we back-propagate gradients to the BERT model. For the rest of the nodes (inference nodes), we just run BERT forward computation to generate BERT embeddings. To further reduce memory consumption and allow training in limited GPU machines, we split the inference nodes into multiple sub-batches of fixed size.

Cache BERT embeddings. To further reduce transformer computations, we cache text embeddings of some nodes in a mini-batch. During the training, whenever we compute new text embeddings, we save them in the cache. Whenever we need node BERT embeddings, we fetch them from the cache. Some cached text embeddings may be out-of-date in a large graph, which may lower the overall model accuracy.

Joint negative sampling. Link prediction task requires positive and negative samples; see Sec. . By default, we sample k negative edges for each positive edge independently, which requires us to sample $k \times n$ new nodes with n is the number of positive links. To maximize efficiency we sample n nodes and use them to construct k negative edges jointly. Specifically, we reuse these nodes and randomly pair them with nodes in our positive set to generate negative pairs. Hence, we reduce the number of nodes in a mini-batch and accelerate training. The default method generates $2\times n+k\times n$ end-point nodes and their neighbor nodes, while our negative sampling generates $3 \times n$ end-point nodes.

Distributed GNN training. We exploit and extend the distributed GNN training framework (Zheng et al. 2021) to scale to billion node graphs and accomodate our end-to-end fine-tuning setting. To increase the training efficiency, we apply hierarchical graph partitioning in DGL's distributed training. When using this method, the target nodes/edges are sampled from the same partition. Therefore, when we sample their neighbors, it's more likely that different target nodes may sample the same neighbor nodes and thus, reduce the number of nodes in a mini-batch.

Experimental setting

Datasets.

Our unique setting requires graph datasets where the nodes are associated with text. We employ the arxiv, and products datesets from the OGB benchmark (Hu et al. 2020) with N=169,343 and E =1,166,243 and N=2,449,029 E=61,859,140 respectively with the standard split ratios from (Hu et al. 2020). We further augment the data with the original text features for each node; the data are collected in (Chien et al. 2021). In the arxiv dataset the original title and abstract is used as text feature for the node. On the other hand the product dataset represents Amazon products and the product title was crawled from the web and used as the text feature. The task here is to predict the labels on the nodes in a standard semi-supervised setting. For these datasets we also formulate a link prediction problem with splits 80% training, 10% validation and 10% testing and the tasks are predicting paper citation and product co-purchase links. Further we also construct the Yelp dataset

augmented with the text using sources provided from (Yel). The following node types are included with corresponding number of nodes business N_1=160,585, category N_2=1330, city N_3=836, review N_4=8,635,403, user N_5=2,189,457. The following edges are considered (user, friendship, user) E_1=17,971,548, (business, in, city) E_2=160585, (business, in category, category) E_3=708968, (review, on, business) E_4=8,635,403, (user, write, review) E_5=8,635,403. The task here is to predict the stars for a business and is formulated as a node prediction task

Additionally, we consider the dataset provided by the recent Amazon KDD22 challenge (Ama). The graph structure is indeed similar to the one depicted in Fig. 1a. There are $N_1 = 646,640$ product and $N_2 = 33,804$ query nodes in the graph and $E = 781,744$ edges that represent a match among the query and the product. Each edge in this dataset is associated with a label which corresponds to whether a match between the query and the product is an exact, substitute, complement or irrelevant. This problem is known as ESCI and is solved as an edge classification task. We create a custom split for this task by splitting the set of edges to 60% for training 10% for validation and 30% for testing.

Baseline setting

Next we explain the different parameters that define the various approaches considered in this work.

Encoders. We consider the following possible semantic encoders in this work. BERT is the pre-trained BERT model from (Wolf et al. 2019). GRAPH-BERT is the pre-trained BERT model (Wolf et al. 2019) that we further fine-tune it for graph structure prediction as in equation (1). BERT-PR is a BERT model that is pre-trained using the MLM objective in the proprietary data of the company.

Task encoders. MLP is a single-layer MLP that projects the text embeding to an appropriate dimension for classification and allows us to directly compare with the language model. GNN is the GraphSAGE model presented in (Hamilton, Ying, and Leskovec 2017a). We use the RGCN (Schlichtkrull et al. 2018) for the ESCI.

Fine-tune(FT). This parameter defines whether we will back-propagate the loss to the semantic encoder during learning or not. The training is orders of magnitude faster when the loss is not back-propagated.

Warm-start(WS). This parameter defines whether we will warm-start the GNN model by keeping the semantic encoder parameters fixed for some iterations before end-to-end fine-tuning.

Model configuration. We optimize the parameters such that the validation set performance is optimized. We select the number of GNN layers from $1, 2, 3$, GNN hidden dimension from $128, 256, 512$ and learning rate from $10^{-3}, 10^{-4}, 10^{-5}$.

Node classification

Table 1 collects the results for the public datasets for different training configurations and encoder models in node classification. Notice that the first two rows apply the node prediction loss directly on the node representation of the text embeddings after it is appropriately mapped by a single layer MLP. The target node for the Yelp dataset does not have any text (rows 1,2). By comparing lines 2 and 3 we observe that fine-tuning the BERT directly for the downstream tasks and disregarding the graph structure achieves on-par performance as the one of keeping the BERT model fixed and using these representations as input to the GNN model. This suggests that the initial BERT embeddings are indeed not the most appropriate semantic embeddings. By comparing lines 3 and 4 we see a performance benefit of fine-tuning the BERT model through the GNN, since the multi-hop information is captured by the GNN. By comparing lines 3 and 5 we observe the clear advantage of the graph-BERT. The graph-aware pre-fine-tuning fuses the transformer with graph information and is the most suitable semantic encoder. Finally, line 6 coincides with the proposed LM-GNN framework. We observe that this configuration achieves the best overall performance and includes the proposed stage-wise fine-tuning approach. Hence, fine-tuning the BERT model for link prediction provides good performance in the node classification tasks. This result is very important since it allows to train a BERT model on link prediction and transfer the knowledge on different downstream tasks which may speed up the overall training.

Table 1: Node classification results for the public datasets. Results measured in accuracy.

	Semantic enc	Graph enc	FT	arxiv	products	Yelp
1	BERT	MLP	No	62.91	61.83	-
2	BERT	MLP	Yes	72.98	77.64	-
3	BERT	GNN	No	71.39	79.10	65.81
4	BERT	GNN	Yes	73.42	81.24	73.06
5	graph-BERT	GNN	No	73.79	80.53	66.88
6	graph-BERT	GNN	Yes	**74.97**	**82.35**	**76.47**

Table 2: Link prediction results for the public datasets. The performance is measured in MRR scores.

	Semantic enc.	Graph enc.	WS	FT	arxiv	products
1	graph-BERT	MLP	No	Yes	59.32	82.29
2	BERT	GNN	No	No	12.43	74.50
3	BERT	GNN	No	Yes	10.11	72.13
4	BERT	GNN	Yes	Yes	15.23	77.42
5	graph-BERT	GNN	No	No	58.13	84.34
6	graph-BERT	GNN	No	Yes	55.32	78.31
7	graph-BERT	GNN	Yes	Yes	**63.21**	**87.23**

Link prediction

Table 2 collects the link prediction performance of the various baselines measured using the MRR score. The first row applies the link prediction supervision in (1) directly on the node representation of the text encoding after mapped by a single layer MLP to an embedding and the whole architecture is fine-tuned end-to-end. This model corresponds to the graph-aware pre-fine-tuned model for link prediction and is the same as the used as a semantic encoder in lines 5, 6, 7.

By comparing lines 1 versus 2, 3, and 4 we observe that the original BERT model is indeed not appropriate as a semantic

Table 3: Edge classification results for the public ESCI dataset. The performance is measured in F1-score, all classes reported. The converged model in row 2 is the graph-BERT used in rows 6,7,8.

	Semantic encoder	Graph encoder	Warm-start	Fine-tune	EvSvCvI	E	S	C	I
1	BERT-PR	MLP	No	No	32.62	57.42	38.12	14.21	20.56
2	graph-BERT	MLP	No	Yes	37.36	59.25	36.13	21.23	32.34
3	BERT-PR	GNN	No	No	35.12	53.34	38.21	25.23	25.42
4	BERT-PR	GNN	No	Yes	**40.56**	**61.90**	42.21	26.30	30.21
5	BERT-PR	GNN	Yes	Yes	40.23	60.21	**45.21**	30.02	30.11
6	graph-BERT	GNN	No	No	37.43	55.62	43.22	**32.60**	**35.24**
7	graph-BERT	GNN	No	Yes	39.51	60.81	42.45	21.56	28.80
8	graph-BERT	GNN	Yes	Yes	40.13	60.90	43.82	18.20	31.52

encoder used with the GNN. On the other hand, fine-tuning the BERT model for link prediction in row 1 achieves a very good MRR performance. Lines 5, 6, 7 relative to 2, 3, 4 showcase the advantage of using the graph-aware pre-fine-tuning as an essential step in our LM-GNN framework, where the former leads to a large performance boost. Furthermore, by comparing 5 with 6 and 7 we observe the necessity of warm-starting in certain cases of the GNN encoder to avoid non-desirable local minima. Since the GNN model is initialized at random and the graph-BERT is well-trained optimizing this model without warm-start is challenging. By comparing lines 3 and 4 we see a performance benefit of fine-tuning the BERT model through the GNN, since the multi-hop information is captured by the GNN.

Convergence improvement. The warm-starting option behinds performance gains in Table 2 it provides significant training speed up. Specifically, for the ogbn-products dataset it takes 168 hours for the option without warm-start (row 6) to reach the maximum performance reported, whereas for the option with warm-start (row 7) it takes only 13 hours to reach the same MRR. Thus warm-start provides a 13x speed up in training speed.

Public ESCI: edge classification

The Table 3 contains the edge classification results for the various baselines in ESCI. Note that here we also report the performance for each individual class since we are interested in predicting well also for the rare classes in our application.

By comparing rows 2 and 4 that both fine-tune the BERT-PR model we observe a strong boost of 320 bps in performance when the GNN is used. This indicates that the GNN can indeed help boosting the performance probably for the rare classes (S-C-I) by a large extend. By comparing rows 3 and 4 we observe that it is very important to fine-tune the BERT-PR model during GNN training. By comparing rows 3 and 6 we observe that the graph-aware pre-fine-tuning is giving a significant boost when the BERT embeddings are fixed. This benefit diminishes when the BERT model is fine-tuned. Finally, the performance in lines 5-8 is quite similar, but interesting the performance in the rare classes is maximized in rows 5 and 6. We plan to dive deeper into these results and analyze the performance for different sample sizes besides the current split.

Conclusion

In this paper we develop a framework termed LM-GNN that achieves high-quality representations for graph data with rich textual features. Our framework employs stage-wise fine-tuning steps that allow for the BERT model to gradually adapt to the graph domain data. We verify with experiments in four public datasets the power of the LM-GNN framework.

References

???? Amazon KDD 2022 challenge. [Online]. Available: https://www.aicrowd.com/challenges/esci-challenge-for-improving-product-search.

???? Yelp dataset. [Online]. Available: https://www.yelp.com/dataset.

Chien, E.; Chang, W.-C.; Hsieh, C.-J.; Yu, H.-F.; Zhang, J.; Milenkovic, O.; and Dhillon, I. S. 2021. Node Feature Extraction by Self-Supervised Multi-scale Neighborhood Prediction. *arXiv preprint arXiv:2111.00064*.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Fu, X.; Zhang, J.; Meng, Z.; and King, I. 2020. MAGNN: Metapath Aggregated Graph Neural Network for Heterogeneous Graph Embedding. In *Proceedings of The Web Conference 2020*, 2331–2341.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017a. Inductive Representation Learning on Large Graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, 1025–1035.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017b. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 22118–22133. Curran Associates, Inc.

Ioannidis, V. N.; Zheng, D.; and Karypis, G. 2020. Few-shot link prediction via graph neural networks for covid-19 drug-repurposing. In *ICML 2020; Graph Representation Learning and Beyond workshop*.

Kipf, T. N.; and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *Proc. Int. Conf. on Learn. Represantions*. Toulon, France.

Li, C.; Pang, B.; Liu, Y.; Sun, H.; Liu, Z.; Xie, X.; Yang, T.; Cui, Y.; Zhang, L.; and Zhang, Q. 2021. Adsgnn: Behavior-graph augmented relevance modeling in sponsored search. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 223–232.

Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; Van Den Berg, R.; Titov, I.; and Welling, M. 2018. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, 593–607. Springer.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2018. Graph attention networks. In *Proc. Int. Conf. on Learn. Represantions*.

Wang, Q.; Mao, Z.; Wang, B.; and Guo, L. 2017. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12): 2724–2743.

Wang, X.; Ji, H.; Shi, C.; Wang, B.; Ye, Y.; Cui, P.; and Yu, P. S. 2019. Heterogeneous graph attention network. In *The World Wide Web Conference*, 2022–2032.

Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Wu, L.; Chen, Y.; Shen, K.; Guo, X.; Gao, H.; Li, S.; Pei, J.; and Long, B. 2021. Graph neural networks for natural language processing: A survey. *arXiv preprint arXiv:2106.06090*.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Philip, S. Y. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*.

Yang, B.; Yih, W.-t.; He, X.; Gao, J.; and Deng, L. 2014. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*.

Zheng, D.; Song, X.; Ma, C.; Tan, Z.; Ye, Z.; Dong, J.; Xiong, H.; Zhang, Z.; and Karypis, G. 2020. DGL-KE: Training Knowledge Graph Embeddings at Scale. *arXiv preprint arXiv:2004.08532*.

Zheng, D.; Song, X.; Yang, C.; LaSalle, D.; Su, Q.; Wang, M.; Ma, C.; and Karypis, G. 2021. Distributed Hybrid CPU and GPU training for Graph Neural Networks on Billion-Scale Graphs. *arXiv preprint arXiv:2112.15345*.

Zhou, Y.; Hou, Y.; Shen, J.; Huang, Y.; Martin, W.; and Cheng, F. 2020. Network-based drug repurposing for novel coronavirus 2019-nCoV/SARS-CoV-2. *Cell discovery*, 6(1): 1–18.

Zhu, J.; Cui, Y.; Liu, Y.; Sun, H.; Li, X.; Pelger, M.; Yang, T.; Zhang, L.; Zhang, R.; and Zhao, H. 2021. Textgnn: Improving text encoder via graph neural network in sponsored search. In *Proceedings of the Web Conference 2021*, 2848–2857.

Model depiction

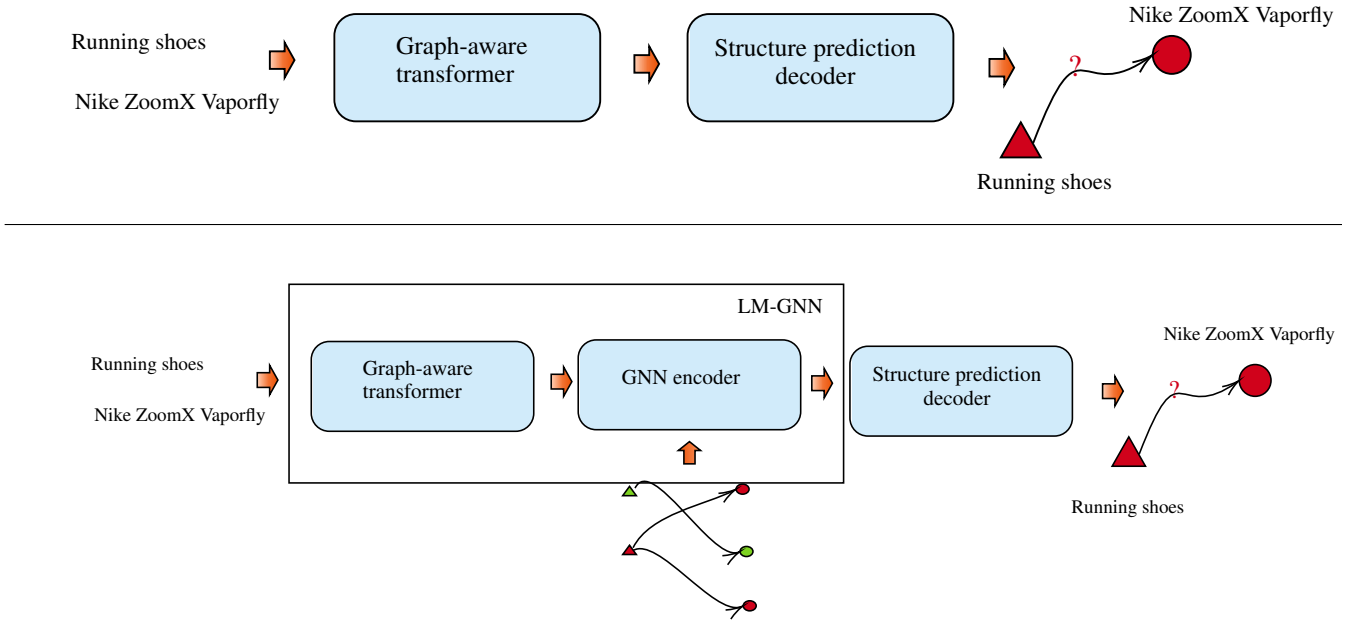


Figure 2: (Top) The graph-aware transformer framework relies on the input text to predict whether two entities are connected in the heterogenous graph. (Bottom) The LM-GNN framework employs the graph-aware transformer as a semantic encoder that is further fine-tuned using the GNN encoder, for predicting links in the heterogeneous graph. Different than the graph-aware transformer framework the LM-GNN can access nodes in multi-hop neighborhood.