NODEPIECE: COMPOSITIONAL AND PARAMETER-EFFICIENT REPRESENTATIONS OF LARGE KNOWLEDGE GRAPHS

Anonymous authors
Paper under double-blind review

ABSTRACT

Conventional representation learning algorithms for knowledge graphs (KG) map each entity to a unique embedding vector. Such a shallow lookup results in a linear growth of memory consumption for storing the embedding matrix and incurs high computational costs when working with real-world KGs. Drawing parallels with subword tokenization commonly used in NLP, we explore the landscape of more parameter-efficient node embedding strategies with possibly sublinear memory requirements. To this end, we propose NodePiece, an anchor-based approach to learn a fixed-size entity vocabulary. In NodePiece, a vocabulary of subword/sub-entity units is constructed from anchor nodes in a graph with known relation types. Given such a fixed-size vocabulary, it is possible to bootstrap an encoding and embedding for any entity, including those unseen during training. Experiments show that NodePiece performs competitively in node classification, link prediction, and relation prediction tasks while retaining less than 10% of explicit nodes in a graph as anchors and often having 10x fewer parameters. To this end, we show that a NodePiece-enabled model outperforms existing shallow models on a large OGB WikiKG 2 graph having ~70x fewer parameters.

1 INTRODUCTION

Representation learning tasks on knowledge graphs (KGs) often require a parameterization of each unique *atom* in the graph with a vector or matrix. Traditionally, in multi-relational KGs such *atoms* constitute a set of all nodes $n \in N$ (entities) and relations (edge types) $r \in R$ (Nickel et al., 2016). Assuming parameterization with vectors, *atoms* are mapped to d-dimensional vectors through shallow encoders $f_n : n \to \mathbb{R}^d$ and $f_r : r \to \mathbb{R}^d$ which scale linearly to the number of nodes and edge types[1], i.e., having $O(|N|)$ space complexity of the entity embedding matrix. Albeit efficient on small conventional benchmarking datasets based on Freebase (Toutanova & Chen, 2015) (~15K nodes) and WordNet (Dettmers et al., 2018) (~40K nodes), training on larger graphs (e.g., YAGO 3-10 (Mahdisoltani et al., 2015) of 120K nodes) becomes computationally challenging. Scaling it further up to larger subsets (Hu et al., 2020; Wang et al., 2021; Safavi & Koutra, 2020) of Wikidata (Vrandecic & Krötzsch, 2014) requires a top-level GPU or a CPU cluster as done in, e.g., PyTorch-BigGraph (Lerer et al., 2019) that maintains a 78M $\times$ 200d embeddings matrix in memory (we list sizes of current best performing models in Table 1).

Taking the perspective from NLP, shallow node encoding in KGs corresponds to shallow word embedding popularized with word2vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014) that learned a *vocabulary* of 400K-2M most frequent words, treating rarer ones as *out-of-vocabulary* (OOV). The OOV issue was resolved with the ability to build infinite combinations with a finite vocabulary enabled by *subword units*. Subword-powered algorithms such as fastText (Bojanowski et al., 2017), Byte-Pair Encoding (Sennrich et al., 2016), and WordPiece (Schuster & Nakajima, 2012) became a standard step in preprocessing pipelines of large language models and allowed to construct fixed-size token vocabularies, e.g., BERT (Devlin et al., 2019) contains ~30K tokens and GPT-2 (Radford et al., 2019) employs ~50K tokens. Importantly, relatively small input embedding matrices enabled investing the parameters budget into more efficient encoders (Kaplan et al., 2020).

[1]We then concentrate on nodes as usually their size is orders of magnitude larger than that of edge types.

Table 1: Node embedding sizes of state-of-the-art KG embedding models compared to BERT Large. Parameters of type *float32* take 4 bytes each. FB15k-237, WN18RR, and YAGO3-10 models as reported in Sun et al. (2019), OGB WikiKG2 as in Zhang et al. (2020c), Wikidata 5M as in Wang et al. (2021), PBG Wikidata as in Lerer et al. (2019), and BERT Large as in Devlin et al. (2019).

	FB15k-237	WN18RR	YAGO3-10	OGB WikiKG2	Wikidata 5M	PBG Wikidata	BERT Large
Vocabulary size	15k	40k	120k	2.5M	5M	78M	30k
Embedding dim	2000	1000	1000	200	512	200	1024
GPU RAM, GB	0.12	0.15	0.46	1.87	9.69	58.1	0.12

Drawing inspiration from subword embeddings in NLP, we explore how similar strategies for *tokenizing* entities in large graphs can dramatically reduce parameter complexity, increase generalization, and naturally represent new unseen entities as using the same fixed vocabulary. To do so, tokenization has to rely on *atoms* akin to subword units and not the total set of nodes.

To this end, we propose *NodePiece*, an anchor-based approach to learn a fixed-size vocabulary V ($|V| \ll |N|$) of any connected multi-relational graph. In NodePiece, the set of atoms consists of anchors and all relation types that, together, allow to construct a combinatorial number of sequences from a limited atoms vocabulary. In contrast to shallow approaches, each node n is first tokenized into a unique $\mathit{hash}(n)$ of k closest anchors and m immediate relations. A key element to build a node embedding is a proper encoder function $\mathit{enc}(n) : \mathit{hash}(n) \to \mathbb{R}^d$ which can be designed leveraging inductive biases of an underlying graph or downstream tasks. Therefore, the overall parameter budget is now defined by a small fixed-size vocabulary of atoms and the complexity of the encoder function.

Our experimental findings suggest that a fixed-size NodePiece vocabulary paired with a simple encoder still yields competitive results on a variety of tasks including link prediction, node classification, and relation prediction. Furthermore, anchor-based hashing enables conventional embedding models to work in the *out-of-sample* scenario when unseen entities arrive at inference time, which otherwise required tailored inductive learning mechanisms.

2 Related Work

Conventional KG embedding approaches. To the best of our knowledge, all contemporary embedding algorithms (Ji et al., 2020; Ali et al., 2020) for link prediction on KGs employ shallow embedding lookups mapping each entity to a unique embedding vector thus being linear $O(|N|)$ to the total number of nodes $|N|$ and size of an embedding matrix. This holds for different embedding families, e.g., translational (Sun et al., 2019), tensor factorization (Lacroix et al., 2018), convolutional (Dettmers et al., 2018), and hyperbolic (Chami et al., 2020; Balazevic et al., 2019). The same applies to relation-aware graph neural network (GNN) encoders (Schlichtkrull et al., 2018; Vashishth et al., 2020) who still initialize each node with a learned embedding or feature vector before message passing. Furthermore, shallow encoding is also used in higher-order KG structures such as hypergraphs (Fatemi et al., 2020) and hyper-relational graphs (Rosso et al., 2020; Galkin et al., 2020). NodePiece can be used as a drop-in replacement of the embedding lookup with any of those models.

Distillation and compression. Several recent techniques for reducing memory footprint of embedding matrices follow successful applications of distilling large language models in NLP (Sanh et al., 2019), i.e., distillation (Wang et al., 2020; Zhu et al., 2020) into low-dimensional counterparts, and compression of trained matrices into discrete codes (Sachan, 2020). However, all of them require a full embedding matrix as input which we aim to avoid designing NodePiece.

Vocabulary reduction in recommender systems. Commonly, recommender systems operate on thousands of categorical features combined in sparse high-dimensional vectors.Recent approaches (Medini et al., 2021; Liang et al., 2021) employ anchor-based hashing techniques to factorize sparse feature vectors into dense embeddings. Contrary to those setups, we do not expect availability of feature vectors for arbitrary KGs and rather learn vocabulary embeddings from scratch.

Entity descriptions and language models. A recent line of work such as KG-BERT (Yao et al., 2019), MLMLM (Clouâtre et al., 2021), BLP (Daza et al., 2021) utilize entity descriptions passed through a language model (LM) encoder as entity embeddings suitable for link prediction. We would like to emphasize that such approaches are rather orthogonal to NodePiece. Textual features are

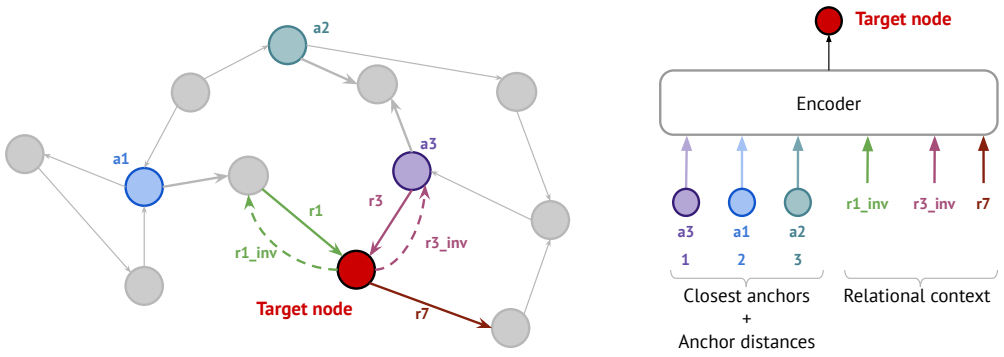


Figure 1: NodePiece tokenization strategy. Given three anchors a_1, a_2, a_3, a target node can be tokenized into a hash of top-k closest anchors, their distances to the target node, and the relational context of outgoing relations from the target node. This hash sequence is passed through an injective encoder to obtain a unique embedding. Inverse relations are added to ensure connectivity.

mostly available in Wikipedia-derived KGs like Wikidata but are often missing in domain-specific graphs like social networks and product graphs. We therefore assume textual features are not available and rather learn node representations based on their spatial characteristics. Still, textual features can be easily added by concatenating NodePiece-encoded features with LM-produced features.

Out-of-sample representation learning. This task focuses on predictions involving previously unseen, or *out-of-sample*, entities that attach to a known KG with a few edges. These new edges are then utilized as a context to compute its embedding. Previous work (Wang et al., 2019; Hamaguchi et al., 2017; Albooyeh et al., 2020) proposed different neighborhood aggregation functions for this process or resorted to meta-learning (Chen et al., 2019; Baek et al., 2020; Zhang et al., 2020a). However, all of them follow the shallow embedding paradigm. Instead, NodePiece uses the new edges as a basis for anchor-based tokenization of new nodes in terms of an existing vocabulary.

3 NodePiece Vocabulary Construction

Given a directed KG $G = (N, E, R)$ consisting of $|N|$ nodes, $|E|$ edges, and $|R|$ relation types, our task is to reduce the original vocabulary size of $|N|$ nodes to a smaller, fixed-size vocabulary of *node pieces* akin to *subword units*. In this work, we represent node pieces through *anchor* nodes $a \in A, A \subset N$, a pre-selected set of nodes in a graph following a deterministic or stochastic strategy. A full NodePiece vocabulary is then constructed from anchor nodes and relation types, i.e, $V = A + R$. Note that in order to maintain reachability of each node and balance in- and out-degrees we enrich G with inverse edges with inverse relation types, such that $|R|_{inverse} = |R|_{direct}$ and $|R| = |R|_{direct} + |R|_{inverse}$. Using elements of the constructed vocabulary each node n can be uniquely *tokenized* into $hash(n)$ as a sequence of k closest anchors, discrete anchor distances, and a relational context of m immediate relations. Then, any injective function $enc(n) : hash(n) \rightarrow \mathbb{R}^d$ can be applied to embed the hash into a d-dimensional vector. An intuition of the approach is presented in Fig. 1 with each step explained in more detail below.

3.1 Anchor Selection

Subword tokenization algorithms such as BPE (Sennrich et al., 2016) employ deterministic strategies to create tokens and construct a vocabulary, e.g., based on frequencies of co-occurring n-grams, such that more frequent words are tokenized with fewer subword units. On graphs, such strategies might employ centrality measures like degree centrality or Personalized PageRank (Page et al., 1999). However, in our preliminary experiments, we found random anchor selection to be as effective as centrality-based strategies. A choice for deterministic strategies might be justified when optimizing for certain task-specific topological characteristics, e.g., degree and PPR strategies indeed skew the distribution of shortest anchor distances towards smaller values thus increasing chances to find anchors in 2- or 3-hop neighborhood of any node (we provide more evidence for that in Appendix C).

3.2 NODE TOKENIZATION

Once the vocabulary $V = A + R$ is constructed, each node n can be hashed (or *tokenized*) into a unique *hash*(n) using 1) k nearest anchors and their discrete distances; 2) m immediate outgoing relations from the relational context of n. Since anchor nodes are concrete nodes in G, they get hashed in the same way as other non-anchor nodes.

Anchors per node. Given $|A|$ anchor nodes, it is impractical to use all of them for encoding each node. Instead, we select k anchors per node and describe two possible strategies for that, i.e., *random* and *deterministic*. The basic random strategy uniformly samples an unordered set of k anchors from A yielding $\binom{|A|}{k}$ possible combinations. To avoid collisions when hashing the nodes, $|A|$ and k are to be chosen according to the lower bound on possible combinations that is defined by the total number of nodes, e.g., $\binom{|A|}{k} \geq |N|$. Note that running depth-first search (DFS) to random anchors at inference time is inefficient and, therefore, *hash*(n) of the random strategy has to be pre-computed.

On the other hand, the deterministic strategy selects an ordered sequence of k nearest anchors. Hence, the anchors can be obtained via breadth-first search (BFS) in the l-hop neighborhood of n at inference time (or pre-computed for speed reasons). However, the combinatorial bound is not applicable in this strategy and we need more discriminative signals to avoid hash collisions since nearby nodes will have similar anchors. Such signals have to better ground anchors to the underlying graph structure, and we accomplish that using *anchor distances*[2] and *relational context* described below.

A node residing in a disconnected component is assigned with an auxiliary `[DISCONNECTED]` token or can be turned into an anchor. However, the majority of existing KGs are graphs with one large connected component with very few disconnected nodes, such that this effect is negligible.

Anchor Distances. Given a target node n and an anchor a_i, we define anchor distance $z_{a_i} \in [0; \textit{diameter}(G)]$ as an integer denoting the shortest path distance between a_i and n in the original graph G. Note that when tokenizing an anchor a_j with the deterministic strategy, the nearest anchor among top-k is always a_j itself with distance 0. We then map each integer to a learnable d-dimensional vector $f_z : z_{a_i} \rightarrow \mathbb{R}^d$ akin to *relative distance encoding* scheme.

Relational Context. We also leverage the multi-relational nature of an underlying KG. Commonly[3], the amount of unique edge types in G is orders of magnitude smaller than the total number of nodes, i.e., $|R| \ll |N|$. This fact allows to include the entire $|R|$ in the NodePiece vocabulary V_{NP} and further featurize each node with a unique relational context. We construct a relational context of a node n by randomly sampling a set of m immediate unique outgoing relations starting from n, i.e., $\textit{rcon}_n = \{r_j\}^m \subseteq \mathcal{N}_r(n)$ where $\mathcal{N}_r(n)$ denotes all outgoing relation types. Due to a non-uniform degree distribution, if $|\mathcal{N}_r(n)| < m$, we add auxiliary `[PAD]` tokens to complete $\textit{rcon}_n$ to size m.

3.3 ENCODING

At this step, a node n is tokenized into a sequence of k anchors, their k respective distances, and relational context of size m:

$$\textit{hash}(n) = \Big[\{a_i\}^k, \{z_{a_i}\}^k, \{r_j\}^m\Big] \tag{1}$$

Taking anchors vectors $\mathbf{a_n}$ and relation vectors $\mathbf{r_n}$ from the learnable NodePiece vocabulary $\mathbf{V} \in \mathbb{R}^{|V| \times d}$, and anchor distances $\mathbf{z_{a_v}}$ from $\mathbf{Z} \in \mathbb{R}^{(diameter(G)+1) \times d}$, we obtain a vectorized hash:

$$\textit{hash}(n) = \Big[\mathbf{a_n} + \mathbf{z_{a_n}}, \mathbf{r_n}\Big] = \Big[\mathbf{\hat{a}_n}, \mathbf{r_n}\Big] \in \mathbb{R}^{(k+m) \times d} \tag{2}$$

Although other operations are certainly possible, in this work, we use anchor distances as positional encodings of corresponding anchors and sum up their representations that helps to maintain the overall hash dimension of $(k + m) \times d$.

Finally, an injective encoder function $\textit{enc} : \mathbb{R}^{(k+m) \times d} \rightarrow \mathbb{R}^d$ is applied to the vectorized hash to bootstrap an embedding of n. In our experiments, we probe two basic encoders: 1) MLP

[2] A full relational path can be mined as well but it has proven to be not scalable as each path needs to be encoded separately through a sequence encoder, e.g., GRU.

[3] As of 2021, one of the largest open KGs Wikidata contains about 100M nodes and 6K edge types

Table 2: Dataset statistics. LP - link prediction, RP - relation prediction, NC - node classification, OOS - out-of-sample. In OOS-LP, *Nodes* also shows the amount of unseen nodes in validation/test.

Dataset	Task	Nodes	Relations	Edges	Train	Validation	Test
FB15k-237 (Toutanova & Chen, 2015)	LP, RP	14,505	237	310,079	272,115	17,526	20,438
WN18RR (Dettmers et al., 2018)	LP, RP	40,559	11	92,583	86,835	2824	2924
CoDEx-Large (Safavi & Koutra, 2020)	LP	77,951	69	612,437	551,193	30,622	30,622
YAGO 3-10 (Mahdisoltani et al., 2015)	LP, RP	123,143	37	1,089,000	1,079,040	4978	4982
OGB WikiKG 2 (Hu et al., 2020)	LP	2,500,604	535	17,137,181	16,109,182	429,456	598,543
WD50K	NC	46,164	526	222,563	4600 (N)	4600 (N)	4600 (N)
oFB15k-237 (Albooyeh et al., 2020)	OOS-LP	11k/1395/1395	234	292,173	193,490	44,601	54,082
oYAGO 3-10	OOS-LP	117k/2960/2959	37	1,086,416	988,124	47,112	51,180

that takes as input a concatenated hash vector $\mathbb{R}^{1\times(k+m)d}$ projecting it down to $\mathbb{R}^d$; 2) Transformer encoder (Vaswani et al., 2017) with average pooling that takes as input an original sequence $\mathbb{R}^{(k+m)\times d}$. While MLP is faster and better scales to graphs with more edges, Transformer is slower but requires less trainable parameters. As the two encoders were chosen to illustrate the general applicability of the whole approach, we leave a study of even more efficient and effective encoders for future work.

While the nearest-neighbor hashing function has a greater number of collisions, its non-arbitrary mapping means that it is effectively permutation invariant. We show this in Proposition 1 through the framework of Janossy pooling and permutation sampling based SGD, π-SGD (Murphy et al., 2019). A proof is provided in Appendix H.

Proposition 1. *The nearest-anchor encoder with $\binom{|A|}{k}$ anchors and $|m|$ subsampled relations, can be considered a π-SGD approximation of $(k+|m|)$-ary Janossy pooling with a canonical ordering induced by the anchor distances.*

Janossy pooling with π-SGD can be used to learn a permutation-invariant function from a broad class of permutation-sensitve functions such as MLPs (Murphy et al., 2019). The permutation-invariant nature of the nearest-neighbor encoding scheme combined with the lack of transductive features such as node-specific embeddings mean that NodePiece can be used for inductive learning tasks as well.

With a fixed-size vocabulary V_{NP}, the overall complexity and parameter budget of downstream models are largely defined by the complexity of the encoder and its inductive biases. By design, the NodePiece *smaller vocabulary - larger encoder* framework is similar to various Transformer-based language models (Qiu et al., 2020) whose vocabulary size remains rather stable with the encoder being the most important part responsible for the final performance.

4 Experiments

We design the experimental program not seeking to outperform the best existing approaches but to show the versatility of NodePiece on a variety of KG-related tasks: transductive and out-of-sample link prediction, node classification, and relation prediction. With this desiderata, we formulate the following research questions: **RQ 1)** Is it necessary to map each node to a unique vector for an acceptable performance on KG tasks?; **RQ 2)** What is the effect of hashing features?; **RQ 3)** Is there an optimal number of anchors per node, after which diminishing returns hit the performance?

4.1 Transductive Link Prediction

Setup. We run experiments on five KGs of different sizes (Table 2) varying the total number of nodes from ~15K to ~2.5M. As a baseline, we compare to RotatE (Sun et al., 2019) that remains one of state-of-the-art shallow embedding models for transductive link prediction tasks. To balance with NodePiece, RotatE operates on a graph with added inverse edges as well. We report MRR with Hits@10 in the *filtered* (Bordes et al., 2013) setting as evaluation metrics, and count parameters for all models. On larger KGs, we also compare to a smaller RotatE with a similar parameter budget.

In this task, NodePiece is equipped with a 2-layer MLP encoder. For a fair comparison, we also adopt the RotatE scoring function as a link prediction decoder. As to the NodePiece configuration, we generally keep the number of anchors below 10% of total nodes in respective graphs. We select 1k/20

Table 3: Transductive link prediction on smaller KGs. † results taken from (Sun et al., 2019). $|V|$ denotes vocabulary size (anchors + relations), #P is a total parameter count (millions). % denotes the Hits@10 ratio based on the strongest model.

	FB15k-237					WN18RR				
	$\lvert V\rvert$	#P (M)	MRR	H@10	%	$\lvert V\rvert$	#P (M)	MRR	H@10	%
RotatE	15k + 0.5k	29	0.338†	0.533†	100	40k + 22	41	0.476†	0.571†	100
NodePiece + RotatE	1k + 0.5k	3.2	0.256	0.420	79	500 + 22	4.4	0.403	0.515	90
- no rel. context	1k + 0.5k	2	0.258	0.425	80	500 + 22	4.2	0.266	0.465	81
- no distances	1k + 0.5k	3.2	0.254	0.421	79	500 + 22	4.4	0.391	0.510	89
- no anchors, rels only	0 + 0.5k	1.4	0.204	0.355	67	0 + 22	0.3	0.011	0.019	0.3

Table 4: Transductive link prediction on bigger KGs. The same denotation as in Table 3. Second RotatE has a similar parameter budget as a NodePiece-based model.

	CoDEx-L					YAGO 3-10				
	$\lvert V\rvert$	#P (M)	MRR	H@10	%	$\lvert V\rvert$	#P (M)	MRR	H@10	%
RotatE (500d)	77k + 138	77	0.258	0.387	100	123k + 74	123	0.495†	0.670†	100
RotatE (20d)	77k + 138	3.8	0.196	0.322	83	123k + 74	4.8	0.121	0.262	39
NodePiece + RotatE	7k + 138	3.6	0.190	0.313	81	10k + 74	4.1	0.247	0.488	73
- no rel. context	7k + 138	3.1	0.201	0.332	86	10k + 74	3.7	0.249	0.482	72
- no distances	7k + 138	3.6	0.179	0.302	78	10k + 74	4.1	0.250	0.491	73
- no anchors, rels only	0 + 138	0.6	0.063	0.121	31	0 + 74	0.5	0.025	0.041	6

for FB15k-237 (i.e., total 1000 anchors and 20 anchors per tokenized node) with 15 unique outgoing relations in the relational context; 500/50 with 4 relations for WN18RR; 7k/20 with 6 relations for CoDEx-L; 10k/20 with 5 relations for YAGO 3-10. Other hyperparameters are listed in Appendix A.

Discussion. Generally, the results suggest that a fixed-size NodePiece vocabulary of <10% of nodes sustains 80-90% of Hits@10 compared to 10x larger best shallow models. Some performance loss is expected due to the compositional and compressive nature of entity tokenization. On smaller graphs (Table 3), parameter saving might not be well pronounced due to the overall small number of nodes to embed. Still, taking even as few as 500 nodes as anchors on WN18RR retains 90% of the best model performance. On bigger graphs (Table 4), parameter efficiency is more pronounced, i.e., on YAGO 3-10, a RotatE model of comparable size is 20 Hits@10 points worse than a NodePiece-based one. This observation can be attributed to the fact the shrinking shallow models results in shrinking the embedding dimension of each node (20d for RotatE) which is inefficient on small parameter budgets. In contrast, a small fixed-size vocabulary allows for larger anchor embedding dimensions (100d for NodePiece with RotatE) since most of the parameter budget is defined by the encoder.

We further study the effect of different anchor selection combinations (Fig. 2). On WN18RR, fewer anchors with fewer anchors per node ($|A|/k$) yield relatively low accuracy but starting from $50/20$ (~0.1% of 40k nodes in the graph) the Hits@10 performance starts to saturate. On FB15k-237, as few as 25 anchors already exhibit the signs of saturation where a further increase to 500 or 1000 anchors only marginally improves the performance. We hypothesize such a difference can be explained by graph density, e.g., WN18RR is a sparse graph with a diameter of 23 and average anchor distance of 6 hops; while FB15k-237 is a denser graph with an average anchor distance of 2-3. Hence, on a sparse graph with longer distances, it takes more anchors to properly encode a node.

However, more precise predictions (e.g., Hits@1) reflected in the MRR metric (see Appendix E) still remain a challenging task for small vocabulary NodePiece setups, and bigger $|A|/k$ combinations alleviate this issue. We also observe that diminishing returns, which make further vocabulary increase less rewarding, start to appear from anchor set sizes of ~1% of total nodes.

Ablations. In the ablation study, we measure the impact of relational context and anchor distances on link prediction (Table 3). Removing relational context and anchor distances does not tangibly affect the denser FB15k-237 data but does impair the accuracy on the sparser WN18RR. Pushing vocabulary sizes to the limit, we also investigate NodePiece behavior in the absence of anchors at all, i.e., when hashes are defined only by the relational context of size m. Interestingly, this still yields fair performance on FB15k-237 with just 7 points Hits@10 drop, but drops to zero the WN18RR

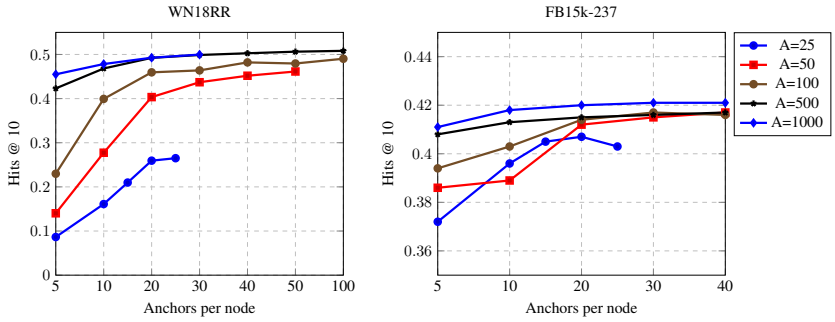


Figure 2: Combinations of total anchors A and anchors per node. Denser FB15k-237 saturates faster on smaller A while sparse WN18RR saturates at around 500 anchors.

performance. The fact that node embeddings might not be at all necessary but relations are more important supports the recent findings of Teru et al. (2020) that relies only on relations seen in a small subgraph around a target node. However, at this point, it seems to be a virtue of graphs with a diverse set of unique relations. That is, FB15k-237 has 20x more unique relations than WN18RR and resulting hashes have more diverse combinations of relations which lead to more discriminative node representations. Additionally, we visualize anchor embedding projections in Appendix D.

4.1.1 OGB WIKIKG 2

To measure the benefits of NodePiece on large-scale KGs, we run a link prediction experiment on OGB WikiKG 2 (Hu et al., 2020), a subset of Wikidata that consists of 2.5M nodes and 16M edges. NodePiece is configured to sample a vocabulary 20K anchor nodes ($< 1\%$ of total nodes) where each node is represented with $k = 20$ nearest anchors and a relational context of size $m = 12$, and we use a 2-layer MLP as a hash encoder (other hyperparameters as in Appendix A). Generally, such a NodePiece configuration can be paired with any link prediction decoder and we chose a non-parametric AutoSF (Zhang et al., 2020b) as one of the strongest decoders on this graph. Overall, the NodePiece + AutoSF model has only 6.9M parameters, about $70\times$ smaller than top shallow models. Compared to the best reported shallow approaches, the NodePiece-enabled model exhibits (cf. Table 5, averaged over 10 seeds) even better performance achieved with a orders of magnitude smaller parameter budget. We believe this result shows the effectiveness of NodePiece on large KGs with a dramatic parameter size reduction without significant performance losses. Ablations report the duality of a relational context, i.e., removing it from hashes leads to even higher MRR scores. On the other hand, the relational context alone with 0 learnable anchors still yields considerably better results than $1000\times$ larger shallow models RotatE and TransE.

Table 5: Test MRR and parameter budget on OGB WikiKG 2.

Model	#Params	MRR
NP + AutoSF	6.9M	0.570 ± 0.003
- rel. context	5.9M	0.592 ± 0.003
- anc. dists	6.9M	0.570 ± 0.004
- no anchors	1.3M	0.476 ± 0.001
AutoSF	500M	0.546 ± 0.005
PairRE	500M	0.521 ± 0.003
RotatE	1250M	0.433 ± 0.002
TransE	1250M	0.426 ± 0.003

4.2 RELATION PREDICTION

Setup. We conduct the relation prediction experiment on the same FB15k-237, WN18RR, and YAGO 3-10 datasets. While link prediction deals with entities, the relation prediction model has to rank a correct relation given a *(head, ?, tail)* query. We report MRR and Hits@10 in the filtered setting as evaluation metrics. Similar to the link prediction configuration, we use NodePiece + 2-layer MLP and compare against RotatE of the same total parameter count.

Discussion. The reported results (Table 6) demonstrate a competitive performance of NodePiece-based models with reduced vocabulary sizes bringing more than 97% Hits@10 across graphs of different sizes. In the case of WN18RR and YAGO 3-10, NodePiece models with fewer anchors even slightly improve the accuracy upon the shallow embedding baseline. The ablation study suggests that on dense graphs with a reasonable amount of unique relations having explicit learnable node embeddings might not be needed at all for this task. That is, we see that on FB15k-237 and YAGO

Table 6: Relation prediction results. $|V|$ denotes vocabulary size (anchors + relations).

	FB15k-237			WN18RR			YAGO 3-10		
	$\|V\|$	MRR	H@10	$\|V\|$	MRR	H@10	$\|V\|$	MRR	H@10
RotatE	15k + 0.5k	0.905	0.979	40k + 22	0.774	0.897	123k + 74	0.909	0.992
NodePiece + RotatE	1k + 0.5k	0.874	0.971	500 + 22	0.761	0.985	10k + 74	0.951	0.997
- no rel. context	1k + 0.5k	0.876	0.968	500 + 22	0.541	0.958	10k + 74	0.898	0.993
- no distances	1k + 0.5k	0.877	0.970	500 + 22	0.746	0.975	10k + 74	0.943	0.997
- no anchors, rels only	0 + 0.5k	0.873	0.971	0 + 22	0.545	0.947	0 + 74	0.951	0.998

Table 7: Node classification results. $|V|$ denotes vocabulary size (anchors + relations), #P is a total parameter count (millions).

			WD50K (5% labeled)			WD50K (10% labeled)		
	$\|V\|$	#P (M)	ROC-AUC	PRC-AUC	Hard Acc	ROC-AUC	PRC-AUC	Hard Acc
MLP	46k + 1k	4.1	0.503	0.016	0.001	0.510	0.017	0.002
CompGCN	46k + 1k	4.4	0.836	0.280	0.176	0.834	0.265	0.161
NodePiece + GNN	50 + 1k	0.75	0.981	0.443	0.513	0.981	0.450	0.516
- no rel. context	50 + 1k	0.64	0.982	0.446	0.534	0.982	0.449	0.530
- no distances	50 + 1k	0.74	0.981	0.448	0.516	0.981	0.448	0.513
- no anchors, rels only	0 + 1k	0.54	0.984	0.453	0.532	0.984	0.456	0.533

3-10 the NodePiece hashes comprised only of the relational context deliver the same performance without any performance drop confirming the findings from the previous experiment.

4.3 Node Classification

Setup. Due to the lack of established node classification datasets on multi-relational KGs, we design a multi-class multi-label task based on a triple version of a recent WD50K (Galkin et al., 2020) extracted from Wikidata. The pre-processing steps are described in Appendix F, and the final graph consists of 46K nodes and 222K edges. The task belongs to the family of transductive (the whole graph is seen during training) semi-supervised (only a fraction of nodes are labeled) problems, where labels are 465 classes as seen in Wikidata. In a semi-supervised mode, we test the models on a graph with 5% and 10% of labeled nodes. Node features have to be learned as node embeddings.

As baselines, we compare to a 2-layer MLP and CompGCN (Vashishth et al., 2020) in a full-batch mode which is one of the strongest GNN encoders for multi-relational KGs. Both baselines learn a full entity vocabulary. We report ROC-AUC, PRC-AUC, and Hard Accuracy metrics as commonly done in standard graph benchmarks like OGB (Hu et al., 2020). For PRC-AUC and Hard Accuracy, we binarize predicted logits using a threshold of 0.5. Hard Accuracy corresponds to the exact match of a predicted sparse 465-dimensional vector to a sparse 465-dimensional labels vector.

NodePiece is configured to have only 50 anchors and use 10 nearest anchors per node with 5 unique relations in the relational context. The dimensionality of anchors and relations is the same as in the baseline CompGCN. Each epoch, we first materialize all entity embeddings through the NodePiece encoder and then send the materialized matrix to CompGCN with the class predictor.

Discussion. Surprisingly, ~1000x vocabulary reduction ratio (50 anchors against 46k for shallow models) greatly outperforms the baselines (Table 7). MLP, as expected, is not able to cope with the task producing random predictions. CompGCN, in turn, outperforms MLP demonstrating non-random outputs as seen by the ROC-AUC score of 0.836 and higher PRC-AUC and Hard Accuracy metrics. Still, a NodePiece-equipped CompGCN with 50 anchors reaches even higher ROC-AUC of 0.98 with considerable improvements along other metrics, i.e., +16-19 PRC-AUC points and 3x boost along the hardest accuracy metric. We attribute such a noticeable performance difference to better generalization capabilities of the NodePiece model. That is, a generalization gap between training and validation metrics of the NodePiece + CompGCN is much smaller compared to the baselines who overfit rather heavily (cf. the training curves in the Appendix G). The effect remains after increasing the number of labeled nodes to 10%. Even with 50 anchors, the overall performance is saturated as the further increase of the vocabulary size did not bring any improvements.

Table 8: Out-of-sample link prediction. † results are taken from (Albooyeh et al., 2020). $|V|$ denotes vocabulary size (anchors + relations), #P is a total parameter count (millions).

	oFB15k-237					oYAGO 3-10 (117k)				
	$\|V\|$	#P (M)	MRR	H@10	%	$\|V\|$	#P (M)	MRR	H@10	%
oDistMult-ERAvg	11k + 0.5k	2.4	0.256†	0.420†	100	117k + 74	23.4	OOM	OOM	-
NodePiece + DistMult	1k + 0.5k	1	0.206	0.372	88	10k + 74	2.7	0.133	0.261	100
- no rel. context	1k + 0.5k	1	0.173	0.329	78	10k + 74	2.7	0.125	0.245	94
- no distances	1k + 0.5k	1	0.208	0.372	88	10k + 74	2.7	0.133	0.260	99
- no anchors, rels only	0 + 0.5k	0.8	0.069	0.127	30	0 + 74	0.7	0.015	0.017	6

Ablations. We probe setups where NodePiece hashes use only anchors or only relational context, and find they both deliver a similar performance. Following the previous experiments on dense graphs with lots of unique relations, it appears that node classification can be performed rather accurately based only on the node relational context which is captured by NodePiece hashes.

4.4 Out-of-sample Link Prediction

Setup. In the out-of-sample setup, validation and test splits contain unseen entities that arrive with a few edges connected to the seen nodes. For this experiment, we adopt the out-of-sample FB15k-237 split (oFB15k-237) as designed in Albooyeh et al. (2020). We do not employ their version of WN18RR as the split contains too many disconnected entities and components in the train graph. Instead, using the authors script, we sample a much bigger out-of-sample version of YAGO 3-10.

Conventional link prediction models like the baseline RotatE are not able to work in inductive setups, e.g., out-of-sample cases, as new entities arriving at inference time do not have trained embeddings nor node features. As a baseline, we compare to oDistMult (Albooyeh et al., 2020) which aggregates embeddings of all seen neighboring nodes around the unseen one (akin to 1-layer message passing with mean aggregator). We adopt the same evaluation protocol - given an unseen node with its connecting edges, we mask one of the edges and predict its tail or head using the rest of the edges, repeating this procedure for each edge. We report filtered MRR and Hits@10 as main metrics.

NodePiece enables traditional transductive-only models to perform inductive inference as both seen and unseen nodes are tokenized using the same vocabulary. For a smaller oFB15k-237 the NodePiece vocabulary has 1k/20 configuration with 15 relations, while in a bigger oYAGO 3-10 we use 10k/20 with 5 relations. For this task, we apply a transformer encoder instead of MLP. For a fair comparison, we use DistMult as a scoring function as well.

Discussion. The results in Table 8 show that a simple NodePiece-based model retains ~90% of the baseline performance on oFB15k-237, but achieved faster and computationally inexpensive compared to oDistMult. Moreover, while oDistMult is tailored specifically for the out-of-sample task, we did not do any task-specific modifications to the NodePiece-enabled model as it is inductive by design. Furthermore, oDistMult is not able to scale to a bigger oYAGO 3-10 on a 256 GB RAM machine due to the out of memory crash. Conversely, a NodePiece-equipped model has the same computational requirements as in other tasks and converges rather quickly (40 epochs). The experiment shows that the NodePiece hashing strategy can be effectively used in inductive scenarios on various KGs with a fixed-size node vocabulary. Performed ablations underline the importance of having both anchors and relational context for tokenizing unseen entities. We elaborate more on possible inference strategies for transductive and inductive tasks in Appendix A.5.

5 Conclusion

In this paper, we have introduced NodePiece, a compositional approach for representing nodes in multi-relational graphs with a fixed-size vocabulary. Similar to subword units, NodePiece allows to tokenize every node as a combination of anchors and relations where the number of anchors can be 10–100× smaller than the total number of nodes. We show that in some tasks, node embeddings are not even necessary for getting an acceptable accuracy thanks to a rich set of relation types. Moreover, NodePiece is inductive by design and is able to tokenize unseen entities and perform downstream prediction tasks in the same fashion as on seen ones.

Reproducibility Statement. The source code is available in the supplementary materials for reviewers. All hyperparameters and implementation details are presented in Appendix A. Information on the used datasets is presented in Table 2 and we provide more details on dataset construction for node classification and out-of-sample link prediction tasks in Appendix F. The proof for Proposition 1 is given in the Appendix H.

Ethics Statement. As NodePiece is a general graph representation learning method, we do not foresee immediate ethical consequences pertaining to the method itself.

REFERENCES

Marjan Albooyeh, Rishab Goel, and Seyed Mehran Kazemi. Out-of-sample representation learning for knowledge graphs. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 2657–2666, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.241. URL https://www.aclweb.org/anthology/2020.findings-emnlp.241.

Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Mikhail Galkin, Sahand Sharifzadeh, Asja Fischer, Volker Tresp, and Jens Lehmann. Bringing light into the dark: A large-scale evaluation of knowledge graph embedding models under a unified framework. *CoRR*, abs/2006.13365, 2020.

Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Sahand Sharifzadeh, Volker Tresp, and Jens Lehmann. PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings. *Journal of Machine Learning Research*, 22(82):1–6, 2021. URL http://jmlr.org/papers/v22/20-825.html.

Jinheon Baek, Dong Bok Lee, and Sung Ju Hwang. Learning to extrapolate knowledge: Transductive few-shot out-of-graph link prediction. *Advances in Neural Information Processing Systems*, 33, 2020.

Ivana Balazevic, Carl Allen, and Timothy M. Hospedales. Multi-relational poincaré graph embeddings. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 4465–4475, 2019.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. doi: 10.1162/tacl_a_00051. URL https://www.aclweb.org/anthology/Q17-1010.

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pp. 2787–2795, 2013.

Ines Chami, Adva Wolf, Da-Cheng Juan, Frederic Sala, Sujith Ravi, and Christopher Ré. Low-dimensional hyperbolic knowledge graph embeddings. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pp. 6901–6914. Association for Computational Linguistics, 2020.

Mingyang Chen, Wen Zhang, Wei Zhang, Qiang Chen, and Huajun Chen. Meta relational learning for few-shot link prediction in knowledge graphs. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4208–4217, 2019.

Louis Clouâtre, Philippe Trempe, Amal Zouaq, and Sarath Chandar. MLMLM: link prediction with mean likelihood masked language model. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto

Navigli (eds.), *Findings of the Association for Computational Linguistics: ACL/IJCNLP 2021, Online Event, August 1-6, 2021*, volume ACL/IJCNLP 2021 of *Findings of ACL*, pp. 4321–4331. Association for Computational Linguistics, 2021.

Daniel Daza, Michael Cochez, and Paul Groth. Inductive entity representations from text via link prediction. In Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia (eds.), *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pp. 798–808. ACM / IW3C2, 2021.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In Sheila A. McIlraith and Kilian Q. Weinberger (eds.), *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 1811–1818. AAAI Press, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 4171–4186. Association for Computational Linguistics, 2019.

Bahare Fatemi, Perouz Taslakian, David Vázquez, and David Poole. Knowledge hypergraphs: Prediction beyond binary relations. In Christian Bessiere (ed.), *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pp. 2191–2197. ijcai.org, 2020.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Mikhail Galkin, Priyansh Trivedi, Gaurav Maheshwari, Ricardo Usbeck, and Jens Lehmann. Message passing for hyper-relational knowledge graphs. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7346–7359. Association for Computational Linguistics, 2020.

Takuo Hamaguchi, Hidekazu Oiwa, Masashi Shimbo, and Yuji Matsumoto. Knowledge transfer for out-of-knowledge-base entities : A graph neural network approach. In Carles Sierra (ed.), *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pp. 1802–1808. ijcai.org, 2017.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. A survey on knowledge graphs: Representation, acquisition and applications. *CoRR*, abs/2002.00388, 2020.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2869–2878. PMLR, 2018.

Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. PyTorch-BigGraph: A Large-scale Graph Embedding System. In *Proceedings of the 2nd SysML Conference*, Palo Alto, CA, USA, 2019.

Paul Pu Liang, Manzil Zaheer, Yuan Wang, and Amr Ahmed. Anchor & transform: Learning sparse embeddings for large vocabularies. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Vd7lCMvtLqg.

Farzaneh Mahdisoltani, Joanna Biega, and Fabian M. Suchanek. YAGO3: A knowledge base from multilingual wikipedias. In *Seventh Biennial Conference on Innovative Data Systems Research, CIDR 2015, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*. www.cidrdb.org, 2015.

Tharun Medini, Beidi Chen, and Anshumali Shrivastava. {SOLAR}: Sparse orthogonal learned and random embeddings. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=fw-BHZ1KjxJ.

Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pp. 3111–3119, 2013.

Ryan L. Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs, 2019.

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proc. IEEE*, 104(1):11–33, 2016.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 8024–8035, 2019.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans (eds.), *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pp. 1532–1543. ACL, 2014.

Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, pp. 1–26, 2020.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Paolo Rosso, Dingqi Yang, and Philippe Cudré-Mauroux. Beyond triplets: Hyper-relational knowledge graph embedding for link prediction. In *Proceedings of The Web Conference 2020*, pp. 1885–1896, 2020.

Mrinmaya Sachan. Knowledge graph embedding compression. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2681–2691, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.238. URL https://www.aclweb.org/anthology/2020.acl-main.238.

Tara Safavi and Danai Koutra. CoDEx: A Comprehensive Knowledge Graph Completion Benchmark. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8328–8350, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.669. URL https://www.aclweb.org/anthology/2020.emnlp-main.669.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019. URL http://arxiv.org/abs/1910.01108.

Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, volume 10843 of *Lecture Notes in Computer Science*, pp. 593–607. Springer, 2018.

Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2012, Kyoto, Japan, March 25-30, 2012*, pp. 5149–5152. IEEE, 2012. doi: 10.1109/ICASSP.2012.6289079. URL https://doi.org/10.1109/ICASSP.2012.6289079.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://www.aclweb.org/anthology/P16-1162.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Komal K. Teru, Etienne Denis, and Will Hamilton. Inductive relation prediction by subgraph reasoning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9448–9457. PMLR, 2020.

Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pp. 57–66, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.18653/v1/W15-4007. URL https://www.aclweb.org/anthology/W15-4007.

Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BylA_C4tPr.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017.

Denny Vrandecic and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, 2014.

Kai Wang, Yu Liu, Qian Ma, and Quan Z Sheng. Mulde: Multi-teacher knowledge distillation for low-dimensional knowledge graph embeddings. *arXiv preprint arXiv:2010.07152*, 2020.

Peifeng Wang, Jialong Han, Chenliang Li, and Rong Pan. Logic attention based neighborhood aggregation for inductive knowledge graph embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 7152–7159, 2019.

Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. KEPLER: A unified model for knowledge embedding and pre-trained language representation. *Trans. Assoc. Comput. Linguistics*, 9:176–194, 2021.

Liang Yao, Chengsheng Mao, and Yuan Luo. KG-BERT: BERT for knowledge graph completion. *CoRR*, abs/1909.03193, 2019. URL http://arxiv.org/abs/1909.03193.

Chuxu Zhang, Huaxiu Yao, Chao Huang, Meng Jiang, Zhenhui Li, and Nitesh V Chawla. Few-shot knowledge graph completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3041–3048, 2020a.

Yongqi Zhang, Quanming Yao, Wenyuan Dai, and Lei Chen. Autosf: Searching scoring functions for knowledge graph embedding. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pp. 433–444. IEEE, 2020b.

Yongqi Zhang, Quanming Yao, Wenyuan Dai, and Lei Chen. Autosf: Searching scoring functions for knowledge graph embedding. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pp. 433–444. IEEE, 2020c.

Yushan Zhu, Wen Zhang, Hui Chen, Xu Cheng, Wei Zhang, and Huajun Chen. Distile: Distiling knowledge graph embeddings for faster and cheaper reasoning. *arXiv preprint arXiv:2009.05912*, 2020.

A IMPLEMENTATION & HYPERPARAMETERS

NodePiece is implemented[4] in Python using *igraph* library (licensed under GNU GPL 2) for computing centrality measures and perform basic tokenization. Downstream tasks employ NodePiece in conjunction with PyTorch (Paszke et al., 2019) (BSD-style license), PyKEEN (Ali et al., 2021) (MIT License), and PyTorch-Geometric (Fey & Lenssen, 2019) (MIT License). We ran experiments on a machine with one RTX 8000 GPU and 64 GB RAM. The OGB WikiKG 2 experiments were executed on a single Tesla V100 16 GB VRAM and 64 GB RAM. All used datasets are available under open licenses.

For all downstream tasks and datasets we employ the deterministic anchor selection strategy where 40% of the total number of anchors $|A|$ are nodes with top PPR scores, 40% are top degree nodes, and remaining 20% are selected randomly. All anchor sets are non-overlapping and disjoint, i.e., if some top degree nodes have already been selected with the PPR policy, they will be skipped in favor of next nodes in the sorted list. The choice for this mixed anchor selection strategy is motivated in Appendix C.

A.1 TRANSDUCTIVE LINK PREDICTION

The optimizer is Adam for all experiments. As RotatE is a scoring function in the complex space, the reported embedding dimensions are a sum of real and imaginary dimensions, e.g., 1000d means that both real and imaginary vectors are 500d.

Table 9: NodePiece hyperparameters for transductive link prediction experiments

Parameter	FB15k-237	WN18RR	CoDEx-L	YAGO 3-10	OGB WikiKG 2
# Anchors, $\|A\|$	1000	500	7000	10000	20000
# Anchors per node, k	20	50	20	20	20
Relational context, m	15	4	6	5	12
Vocabulary dim, d	200	200	200	200	200
Batch size	512	512	256	512	512
Learning rate	0.0005	0.0005	0.0005	0.00025	0.0001
Epochs	400	600	120	600	300k (steps)
Encoder type	MLP	MLP	MLP	MLP	MLP
Encoder dim	400	400	400	400	400
Encoder layers	2	2	2	2	2
Encoder dropout	0.1	0.1	0.1	0.1	0.1
Loss function	BCE	NSSAL	BCE	NSSAL	NSSAL
Margin	-	15	-	50	50
# Negative samples	-	20	-	10	128
Label smoothing	0.4	-	0.3	-	-
Training time, hours	7	5.5	26	23	11

Table 10: RotatE hyperparameters for transductive link prediction experiments. CoDEx-L and YAGO 3-10 also list the hyperparameters (after the symbol /) for smaller models (reported in Table 4) of the same parameter budget as NodePiece

Parameter	FB15k-237	WN18RR	CoDEx-L	YAGO 3-10
Embedding dim, d	2000	1000	1000 / 50	1000 / 40
Batch size	1024	512	512 / 512	1024 / 512
Loss function	NSSAL	NSSAL	NSSAL	NSSAL
Margin	9	6	25 / 9	24 / 15
# Negative samples	256	1024	100 / 100	400 / 100

[4]Source code and data are available for reviewers

A.2 RELATION PREDICTION

Configurations (Table 11) for the compared models are almost identical to those of the transductive link prediction experiment. We mostly reduce the number of epochs and negative samples as models converge faster on this task.

Table 11: Hyperparameters for relation prediction experiments. The content is largely identical to Table 9, only changed parameters are listed

	NodePiece + RotatE			RotatE		
Parameter	FB15k-237	WN18RR	YAGO 3-10	FB15k-237	WN18RR	YAGO 3-10
Batch size	512	512	512	512	512	512
Epochs	20	150	7	150	150	150
Loss function	NSSAL	NSSAL	NSSAL	NSSAL	NSSAL	NSSAL
Margin	15	12	25	9	3	5
# Negative samples	20	20	20	20	20	20
Training time, min	25	30	25	28	10	57

A.3 NODE CLASSIFICATION

In this experiment, NodePiece is used at the initial step to bootstrap a node embeddings matrix which is then sent to the CompGCN graph encoder. In contrast, CompGCN and MLP baselines use directly a trained node embedding matrix as their initial input.

Table 12: Hyperparameters for node classification experiments

Parameter	NodePiece + CompGCN	CompGCN	MLP
# Anchors, $\|A\|$	50	-	-
# Anchors per node, k	10	-	-
Relational context, m	5	-	-
Vocabulary dim, d	100	100	100
Batch size	512	512	512
Learning rate	0.001	0.001	0.001
Epochs	4000	4000	4000
NodePiece encoder	MLP	-	-
NodePiece encoder dim	200	-	-
NodePiece encoder layers	2	-	-
NodePiece encoder dropout	0.1	-	-
GNN (MLP) layers	3	3	3
GNN (MLP) dropout	0.5	0.5	0.5
Loss function	BCE	BCE	BCE
Label smoothing	0.1	0.1	0.1
Training time, hours	14	22	6

A.4 OUT-OF-SAMPLE LINK PREDICTION

The set of NodePiece hyperparameters is similar to the set of the transductive experiments except the scoring function (DistMult), encoder function (Transformer), and number of epochs as the model converges faster. We do not provide a setup for the baseline oDistMult on oYAGO 3-10 as the model was not able to pre-process the dataset on a machine with 256 GB RAM. Reported training times for NodePiece models exclude evaluation. Training times of the baseline oDistMult were not reported by its authors.

Table 13: Hyperparameters for out-of-sample link prediction experiments. The content is largely identical to Table 9, only changed parameters are listed

	NodePiece + DistMult		oDistMult
Parameter	oFB15k-237	oYAGO 3-10	oFB15k-237
# Anchors, $\|A\|$	1000	10000	-
# Anchors per node, k	20	20	-
Relational context, m	15	5	-
Vocabulary dim, d	200	200	200
Batch size	256	256	1000
Learning rate	0.0005	0.0005	0.01
Epochs	40	40	1000
NodePiece encoder	Transformer	Transformer	-
NodePiece encoder dim	512	512	-
NodePiece encoder layers	2	2	-
NodePiece encoder dropout	0.1	0.1	-
Loss function	Softplus	Softplus	Softplus
# Negative samples	5	5	1
Training time, hours	2	8	-

A.5 Deployment in Real-world Dynamic Knowledge Graphs

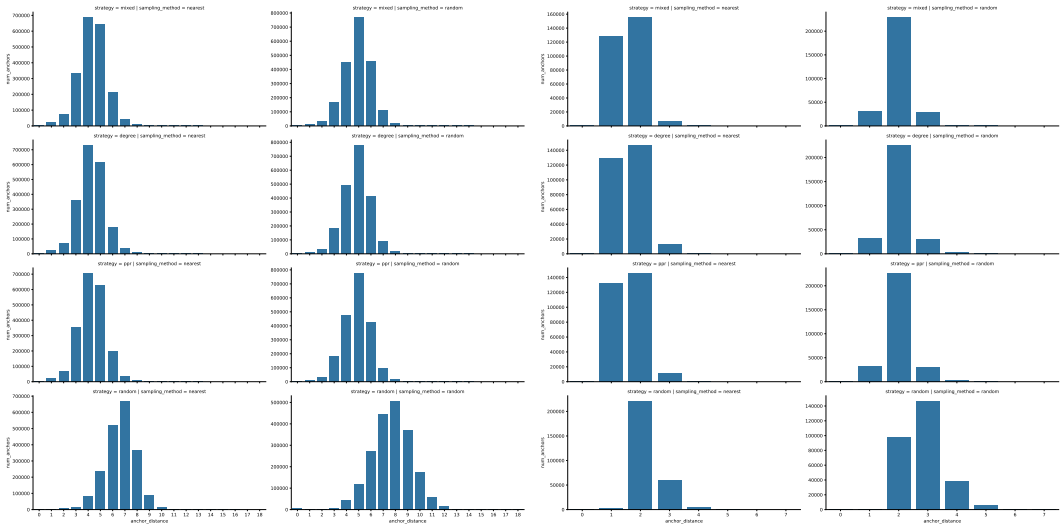
NodePiece, on account of its compositional representation, can be applied to dynamic real-world knowledge graphs where nodes are added and removed over time. That is, training on a graph snapshot we can obtain the embeddings of new nodes without re-computing and updating representations of every other node in the graph. This is valuable in settings where there are latency requirements such as many online services. For example, if a user creates an account on a social media service and begins liking content (represented as a "like" edge in the social network graph between the user and the content), it would be desirable to make future content recommendations rapidly reflect this new data without waiting for the next batched retraining to update the users embedding. In order to reduce latency the entire embedding matrix can be materialized and cached ahead of time, updating embeddings as new nodes and edges are added. The parameter efficiency and compositionality of NodePeice means that for large real-world graphs NodePiece subsumes what would before have been a complex system of a large-scale embedding framework like Pytorch-BigGraph (Lerer et al., 2019), an OOV embedding method (e.g,. ERAvg) and a shallow embedding method (e.g., RotatE).

B Limitations and Future Work

Our experimental results demonstrate the promise of using NodePeice to significantly reduce the parameter complexity of node embeddings. While it is difficult to prove, we also hypothesize that the parameters required by NodePeice to maintain the same level of performance (as the graph scales) increase sublinearly according to the size of the graph. The intuition for this is twofold. First, the number of unique anchor combinations of size k that can be encoded increases according to $\binom{|A|}{k}$ (i.e. $\mathcal{O}(|A|^k)$) if randomly sampled — if the sampling is done via nearest neighbor anchor selection then the number of unique permutations is expected to increase polynomially. Second, increasing the size of the graph will only require sublinear increase in the number of anchors in order to maintain the same average node-anchor distance. Although proving causality is difficult, we believe that maintaining hashing uniqueness and node-anchor distances stable will be sufficient to maintain equivalent performance.

C Anchor Selection Strategies

Here, we provide more details as to anchor configurations (k nearest from total A anchors) and anchor distances. Recall that there exist several ways to select the total set of anchors A as stated in Section 3.1, i.e., random or centrality-measure based. Then, k anchors per node can be chosen



(a) WN18RR (b) FB15k-237

Figure 3: Distribution of anchor distances under various anchor selection strategies. Top-bottom: mixed, degree-based, PPR-based, random. For each dataset, left: selecting k nearest anchors, right: k random anchors. (a) Selecting a fixed 500/50 configuration on WN18RR; (b) Selecting a fixed 1000/20 configuration on FB15k237. Generally, all strategies except random ones skew the distributions towards nearest anchors.

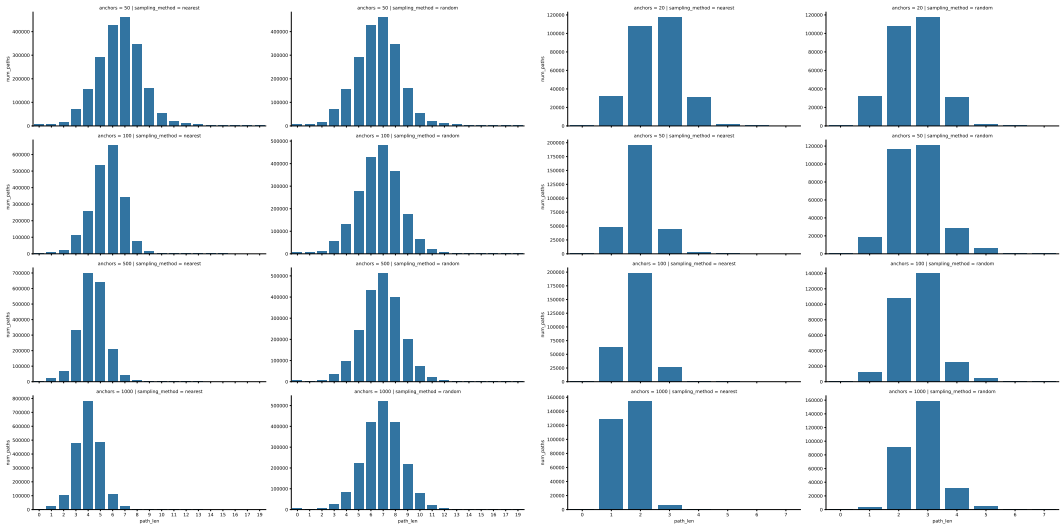
either as k nearest (default NodePiece mode) or k random anchors. Figure 3 depicts the effect of those strategies on the distribution of anchor distances (number of hops between a target node and its anchors). We use the configurations used in the main experiments, i.e., 1000 anchors and 20 anchors per node for FB15k-237, and 500 anchors with 50 anchors per node on WN18RR.

First, we observe that PPR, degree, and mixed (40% PPR, 40% degree, 20% random) strategies generally skew the distribution towards smaller anchor distances compared to random strategies. This fact supports the hypothesis that deterministic strategies improve the chances to find an anchor in a closer l-hop neighborhood of a target node. Second, varying the way of selecting k anchors per node between nearest (left column) and random (right column), we also observe the skew of a distribution of anchor distances.

Next, we fix the anchor selection strategy to the *mix*, fix the number of anchors per node (50 for WN18RR and 20 for FB15k-237), and vary a total number of anchors A (50 to 1000 for WN18RR and 20 to 1000 for FB15k-237) along with the method of sampling k anchors per node, i.e., nearest and random. Figure 4 shows that increasing the total number of anchors together with k nearest anchors again skews the distribution of anchor distances towards smaller values and, hence, to higher probabilities of finding anchors in a closer neighborhood of a target node.

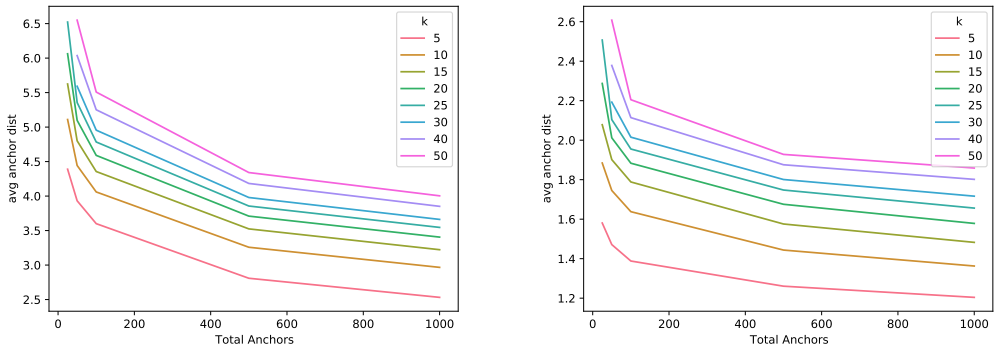
We would recommend using centrality-based strategies to select A with k *nearest* anchors per node if anchor distances and probability of finding anchors in a closer neighborhood are of higher importance.

Finally, we fix the anchor selection strategy as *mix*, obtain *nearest* anchors per node, and under this setup study average anchor distances varying k - the number of anchors per node in various combinations of total anchors A. The results presented on Figure 5 suggest that sparser graphs (like WN18RR) benefit more from increasing the number of anchors A, i.e., the delta between distances is much larger than that of dense FB15k-237. The difference in distances on Figure 5 might also explain the performance on Figure 8, i.e., generally, smaller A/k configurations like 25/5 are inferior on sparser graphs but perform competitively on denser ones.



(a) WN18RR (b) FB15k-237

Figure 4: Distribution of anchor distances under the fixed *mix* anchor selection strategy when varying the total number of anchors A (50–1000 for WN18RR, 20–1000 for Fb15k-237). For each dataset, left column - k nearest anchors, right - k random anchors. On both graphs, increase in A with the nearest anchors always leads to shorter anchor distances.



(a) WN18RR (b) FB15k-237

Figure 5: Average node-anchor distances when varying the total number of anchors A from 25 to 1000 and k nearest anchors per node from 5 to 50. Note that on a sparser WN18RR the gap between min and max values is much wider than of denser FB15k-237. Signs of saturation suggest that further increasing A is not beneficial.

D EMBEDDING VISUALIZATIONS

To further study learned representations of anchors and capabilities of the encoder, we build tSNE and UMAP projections from subsamples of FB15k-237 and WN18RR based on trained models from the transductive link prediction experiments (hyperparameters listed in Table A).

For FB15k-237, we randomly sample 1000 entities (out of total 15K) and find their top-100 most common anchors. The anchor embeddings are extracted from the learned tensor while 1000 entity embeddings are obtained through the NodePiece encoder. Similarly for WN18RR, we sample 4000 entities (out of total 40K) keeping their top-100 most common anchors. As we use the RotatE decoder

that assumes entities and anchors are modeled in a complex space, we visualize their real parts (e.g., first 100 dimensions out of 200).

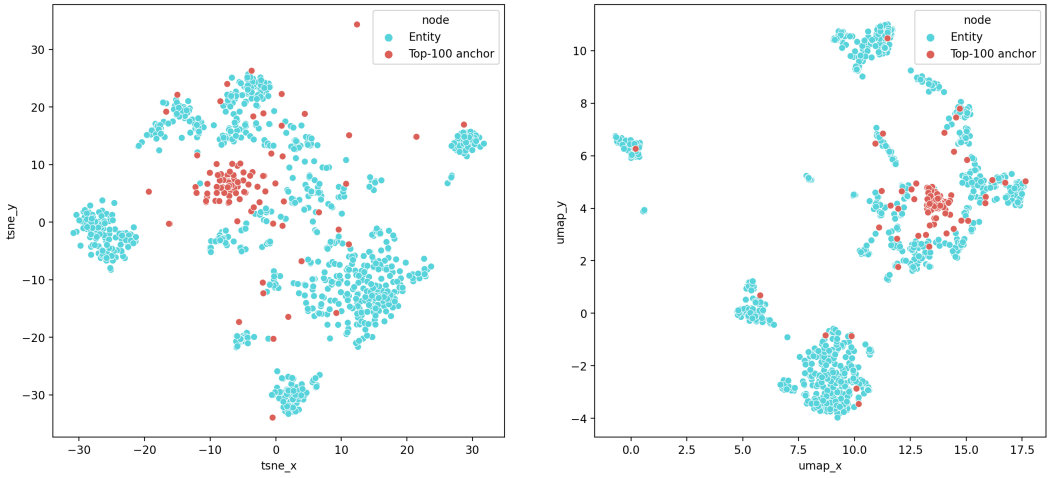


Figure 6: tSNE (left) and UMAP (right) projections of 1000 encoded entities sampled randomly from FB15k-237 and their top 100 most common anchors.

Recalling that link prediction performance of NodePiece + RotatE retains 80-90% of the state of the art models performance, the results on Fig. 6 and Fig. 7 demonstrate that (1) NodePiece encoder is able to reconstruct clusters of similar entities; (2) anchors are well-scattered among communities. Albeit entity embeddings are built as a composition of k anchors, it can be seen that all communities have "specialized" nearby anchors. On a higher level, common anchors tend to be well-scattered in the space. Less common anchors, as seen on FB15k-237 and Fig. 6, tend to group together. However, thanks to the non-linear nature of the NodePiece encoder, resulting entity embeddings still form different clusters and communities not concentrated around one point. We believe this is the effect of a compositional encoder and plan to investigate this phenomenon further.

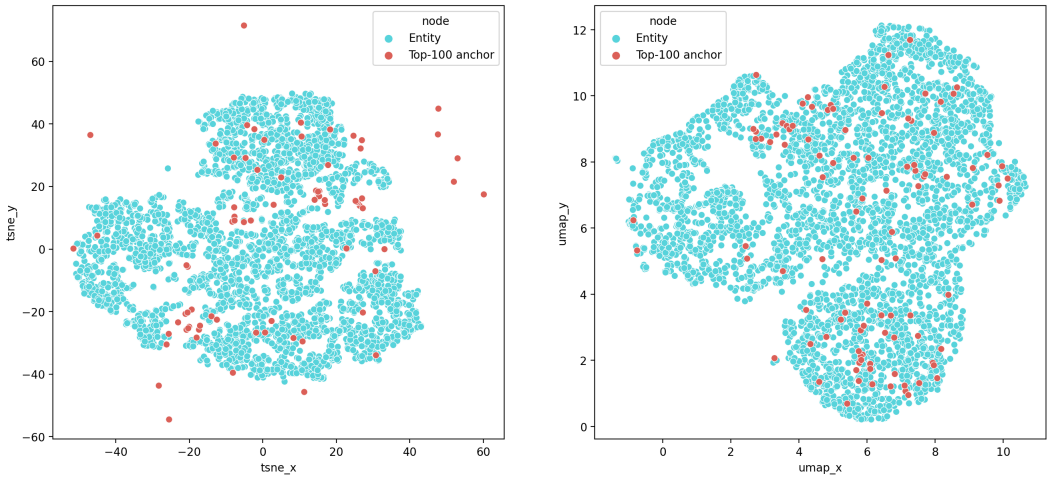


Figure 7: tSNE (left) and UMAP (right) projections of 4000 encoded entities sampled randomly from WN18RR and their top 100 most common anchors.

E TRANSDUCTIVE LINK PREDICTION RESULTS: MRR

In addition to Figure 2 that presents Hits@10, we report variations of mean reciprocal rank (MRR) depending on combinations of A and k on Figure 8 from the same set of experiments. On sparser

WN18RR, smaller A/k combinations like 25/5 or 50/10 struggle with more precise predictions like Hits@1 which is captured by low values of MRR. Starting from 500/10, the WN18RR performance starts to saturate. On the other hand, on denser FB15k-237, the difference between minimum and maximum MRR is less than 4 points, and performance exhibits signs of saturation already at 50/10.

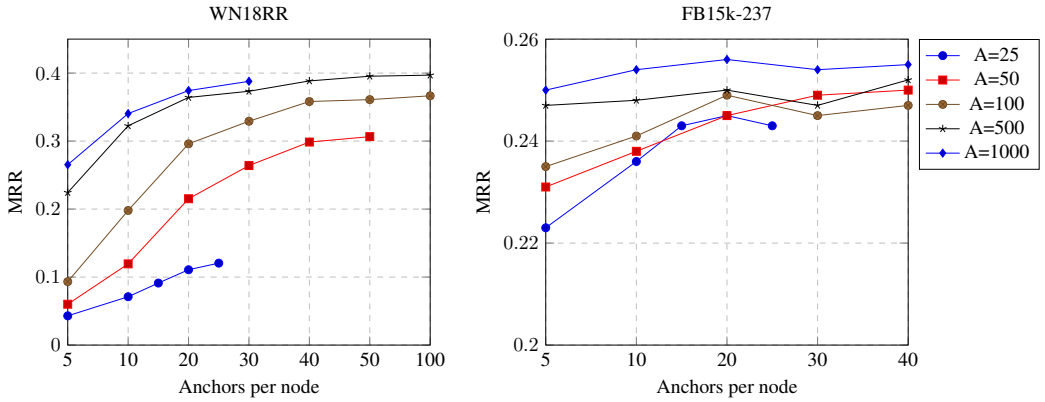


Figure 8: Combinations of total anchors A and anchors per node. Denser FB15k-237 saturates faster on smaller A while sparse WN18RR saturates at around 500 anchors. MRR metric captures all ranks. Note that performance gap on FB15k-237 is very small indicating that saturation has occurred already with small anchor configurations.

F Datasets Construction

F.1 Node Classification: WD50K NC

The original WD50K (Galkin et al., 2020) contains a triple-only KG version on which we base a new dataset for semi-supervised multi-class multi-label node classification. First, we remove all triples containing Wikidata properties `P31` (*instance of*) and `P279` (*subclass of*) as they already contain class information. We then remove nodes that became disconnected after removing those edges. Third, using SPARQL queries, for each remaining node in a graph, we extract a 3-hop class hierarchy of Wikidata classes and their superclasses. We only keep class labels that occur at least 50 times in the training set. Then, we sample 10% of nodes with labels for validation and 10% for test, and of remaining 80% we sample a set of nodes for the semi-supervised setup, i.e., we keep only 5% and 10% of those nodes. The resulting graph has 46k nodes, 526 distinct relation types, and 465 class labels.

F.2 Ouf-of-sample Link Prediction: oYAGO 3-10

For sampling the out-of-sample version of a bigger YAGO 3-10 we largely follow the same original procedure described in Section 4 of (Albooyeh et al., 2020). We first merge the train, validation and test triples from the original dataset for transductive link prediction. Then, from all entities appearing in at least two triples, we randomly sample 5% of nodes to be the out-of-sample entities for validation and 5% for test. All triples containing the out-of-sample entities on subject or object positions are put into validation or test, respectively, as edges that connect an unseen entity with the seen graph.

G Node Classification: Training Curves

Figure 9 depicts train and validation values of Hard Accuracy and PRC-AUC metrics for all the compared models on WD50K NC with 5% of labeled nodes. The NodePiece model has only 50 total anchors with 10 nearest anchors per node, and 5 unique relation types in the relational context. The performance on the dataset with 10% of nodes is almost the same, so we report the charts only on 5% dataset. By the generalization gap we understand the delta between training and validation values.

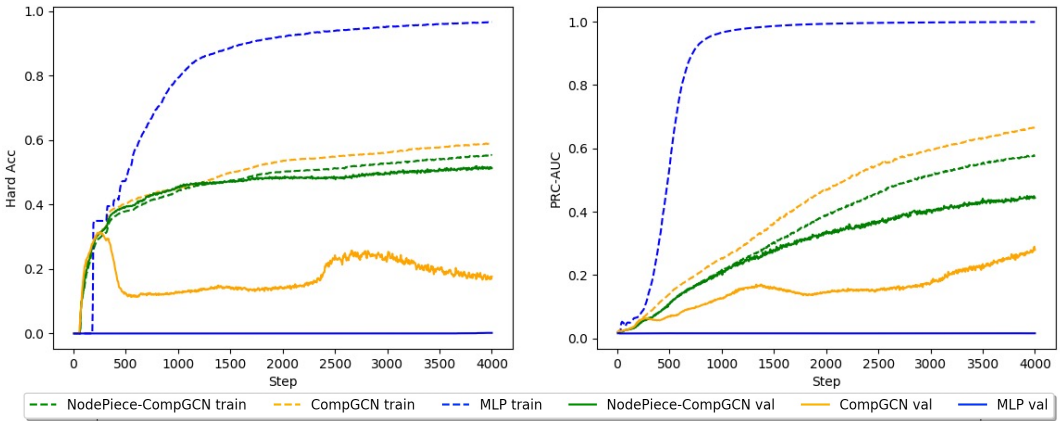


Figure 9: Generalization gap on WD50K (5% labeled nodes). NodePiece-bsed model has observably smaller generalization gaps compared to the baselines.

The MLP baseline quickly overfits but fails to generalize on the validation. The generalization gap of CompGCN is smaller compared to MLP but is still significant, i.e., validation performance is 2–3$\times$ smaller than train. Finally, the NodePiece-enabled model has the smallest generalization gaps, especially along the Hard Accuracy metric where the validation performance is very close to that of train. Similarly, the gap on PRC-AUC is smaller than 10 points.

As shown in the ablation study in Table 7, it appears that explicit node embeddings do not contribute to the classification performance. Hence, the baseline models tend to be overparameterized where learnable node embeddings add noise, while the NodePiece model has only a few anchors (or no anchors at all when using only the relational context), much fewer parameters, and therefore generalizes better. This hypothesis also explains the observation that the node classification performance does not improve when increasing A/k anchor configurations.

H PROOFS

Proposition 2. *The nearest-anchor encoder with $\binom{|A|}{k}$ anchors and $|m|$ subsampled relations, can be considered a π-SGD approximation of $(k + |m|)$-ary Janossy pooling with a canonical ordering induced by the anchor distances.*

Proof.

We begin by providing the definition of Janossy pooling as it was presented in the original paper Murphy et al. (2019).

Definition 1 (Janossy pooling)**.** *Let $\mathbb{H}^{\cup}$ be the union of all anchors and relations. Consider a function $\vec{f} : \mathbb{N} \times \mathbb{H}^{\cup} \times \mathbb{R}^d \to \mathbb{F}$ on variable-length but finite sequences $\mathbf{h}$, parameterized by $\boldsymbol{\theta}^{(f)} \in \mathbb{R}^d$, $d > 0$. A permutation-invariant function $\overline{\vec{f}} : \mathbb{N} \times \mathbb{H}^{\cup} \times \mathbb{R}^d \to \mathbb{F}$ is the Janossy function associated with $\vec{f}$ if*

$$\overline{\vec{f}}(|\mathbf{h}|, \mathbf{h}; \boldsymbol{\theta}^{(f)}) = \frac{1}{|\mathbf{h}|!} \sum_{\pi \in \Pi_{|\mathbf{h}|}} \vec{f}(|\mathbf{h}|, \mathbf{h}_{\pi}; \boldsymbol{\theta}^{(f)}), \tag{3}$$

where $\Pi_{|\mathbf{h}|}$ is the set of all permutations of the integers 1 to $|\mathbf{h}|$, and $\mathbf{h}_{\pi}$ represents a particular reordering of the elements of sequence $\mathbf{h}$ according to $\pi \in \Pi_{|\mathbf{h}|}$. We refer the operation used to construct $\overline{\vec{f}}$ from $\vec{f}$ as Janossy pooling.

While Janossy pooling provides a simple approach to construct permutation-invariant functions from arbitrary permutation sensitive functions, it is computationally intractable due to the need to sum over all computations. Three general strategies proposed under this framework to overcome this combinatorial challenge: canonical orderings, k-ary Janossy pooling, and π-SGD approximations.

A very effective way of reducing the complexity is to constrain the permutations to a canonical ordering that is independent of a specific adjacency matrix ordering over a given graph. More precisely, one defines as a function $\mathrm{CANONICAL} : \mathbb{H}^{\cup} \rightarrow \mathbb{H}^{\cup}$ such that $\mathrm{CANONICAL}(\mathbf{h}) = \mathrm{CANONICAL}(\mathbf{h}_{\pi}) \forall \pi \in \Pi_{|\mathbf{h}|}$ and only considers functions $\vec{f}$ based on the composition $\vec{f} = \mathrm{CANONICAL} \circ \vec{f}'$ (Murphy et al., 2019). In the case of NodePiece we are able to define this ordering for the anchors according to their distance to the target node. Assuming that the number of relations is fixed or grows at slow rate throughout the life-cycle of a graph we can define an arbitrary ordering for relations as a canonical ordering for the relational context. However, since anchors can be equidistant such a canonical ordering does fully satisfy permutation invariance. We propose a trivial relaxation of the original definition of canonical orderings simply requiring that an ordering greatly reduce the number of unique permutations since in practice an exact canonical ordering is rarely feasible. Specifically, $|\{\mathrm{CANONICAL}(\mathbf{h}_{\pi}) \forall \pi \in \Pi_{|\mathbf{h}|}\}| \ll |\{(\mathbf{h}_{\pi}) \forall \pi \in \Pi_{|\mathbf{h}|}\}|$.

To further reduce the number of permutations we can truncate our ordered sequence $\mathbf{h}$. This is known as k-ary Janossy pooling pooling (Definition 2) and is implicitly performed by the NodePeice algorithm by varying the anchor per node parameter, k, and the size of the relational context, $|m|$.

Definition 2 (k-ary Janossy pooling)**.** *Fix $k \in \mathbb{N}$. For any sequence $\mathbf{h}$, define $\downarrow_k (\mathbf{h})$ as its projection to a length k sequence; in particular, if $|\mathbf{h}| \geq k$, we keep the first k elements. Then, a k-ary permutation-invariant Janossy function $\overline{\vec{f}}$ is given by*

$$\overline{\vec{f}}(|\mathbf{h}|, \mathbf{h}; \boldsymbol{\theta}^{(f)}) = \frac{1}{|\mathbf{h}|!} \sum_{\pi \in \Pi_{|\mathbf{h}|}} \vec{f}(|\mathbf{h}|, \downarrow_k(\mathbf{h}_{\pi}); \boldsymbol{\theta}^{(f)}). \tag{4}$$

Since an imperfect truncated canonical ordering may still result in a potentially intractable number of permutations, we use permutation sampling also known as π-SGD to learn arbitrary functions that approximate $(k + |m|)$-ary Janossy pooling. This is done by randomly ordering anchors that are equidistant resulting in a uniform sampling of possible permutations during training and evaluation. For more details on the formal definition of π-SGD we point the reader to the original paper (Murphy et al., 2019).