Inference over Unseen Entities, Relations and Literals on Knowledge Graphs

Caglar Demir[a,*,1], **N'Dah Jean Kouagou**[a,1], **Arnab Sharma**[a,1] **and Axel-Cyrille Ngonga Ngomo**[a]

[a]Data Science Research Group, Paderborn University, Germany

Abstract. In recent years, knowledge graph embedding models have been successfully applied in the transductive setting to tackle various challenging tasks, including link prediction, and query answering. Yet, the transductive setting does not allow for reasoning over unseen entities, relations, let alone numerical or non-numerical literals. Although increasing efforts are put into exploring inductive scenarios, inference over unseen entities, relations, and literals has yet to come. This limitation prohibits the existing methods from handling real-world dynamic knowledge graphs involving heterogeneous information about the world. Here, we propose a remedy to this limitation. We propose the attentive byte-pair encoding layer (BYTE) to construct a triple embedding from a sequence of byte-pair encoded subword units of entities and relations. Given a triple, BYTE acts as an encoder and constructs an embedding vector by combining embeddings of subword units of head entities, relations, and tail entities. Thereafter, BYTE applies a knowledge graph embedding model as a decoder to compute the likelihood of an input triple being true. Compared to the conventional setting, BYTE leads to massive feature reuse via weight tying, since it forces a knowledge graph embedding model to learn embeddings for subword units instead of entities and relations directly. Consequently, the sizes of embedding matrices are no longer bound to the unique number of entities and relations of a knowledge graph. Experimental results show that BYTE improves the link prediction performance of 4 knowledge graph embedding models on datasets where the syntactic representations of triples are semantically meaningful. However, benefits of training a knowledge graph embedding model with BYTE dissipate on knowledge graphs where entities and relations are represented with plain numbers or URIs. We provide an open source implementation of BYTE to foster reproducible research.

1 Introduction

The field of natural language processing (NLP) has reached an unprecedented level with the advent of large language models (LLMs) [24, 9]. Such models have demonstrated significant capabilities in understanding and generating natural language text. Behind the success of LLMs, tokenizers play a fundamental role [27]. Tokenizers allow the transformation of plain text into smaller pieces (tokens) which serve as the building blocks for text understanding and generation [18, 4]. Through tokenization, LLMs can not only handle language variability and ambiguity (e.g., syntactic errors) but also process long sequences more efficiently. Current knowledge graph embedding (KGE) models are developed to work in a transductive setting[2], with a few partially supporting the inductive setting [3]. These KGE models initialize a fixed-size vocabulary from an input Knowledge Graph (KG) and map each entity and relation to an element (e.g., TransE [3]) or a sequence of elements (e.g., NodePiece [11]) of the vocabulary before projecting them into a d-dimensional vector space. Specifically, NodePiece represents each entity as a hash of its immediate outgoing relations and its closest anchor nodes together with their respective discrete distances. While this approach can handle unseen entities with a known neighborhood, it cannot handle unseen relations nor entities for which no neighborhood information is given.

Knowledge graph embedding research has mainly focused on designing embedding models tailored towards the transductive link prediction [22, 38, 8, 2, 41, 26]. This task is often formulated as the problem of learning a parameterized scoring function $\phi_\Theta : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \to \mathbb{R}$ such that $\phi_\Theta(\mathtt{h}, \mathtt{r}, \mathtt{t})$ ideally signals the likelihood of $(\mathtt{h}, \mathtt{r}, \mathtt{t})$ being true [8]. In contrast, inductive link prediction on a knowledge graph refers to the task of predicting missing links between new entities that are not observed during training [14, 31]. To handle unseen entities, a few inductive methods focus on learning entity-independent relational patterns using logical rule mining [20], while others exploit graph neural networks (GNNs) [11, 14]. Yet, most existing methods assume that only entities can be new, and all relations should be observed during training. Thus, they perform inductive inference for entities but transductive inference for relations. An exception is ULTRA [12] which introduces a graph of relations to learn fundamental interactions between relations. The learned patterns can effectively be transferred to new, unseen graphs. However, to answer queries of the form $(h, r, ?)$, ULTRA requires a subgraph containing the query relation r. This limits its applicability to many downstrean tasks, including standard link prediction where no subgraph information is available.

In this work, we propose an attentive byte-pair encoding layer (BYTE) to make existing KGE methods support the three inference regimes: transductive, inductive, and out-of-vocabulary (i.e., unseen) entities and relations. BYTE represents each entity and relation as a sequence of byte-pair encoded subword units (tokens).

During training, given a triple $(\mathtt{h}, \mathtt{r}, \mathtt{t})$, a KGE model predicts its likelihood by combining embeddings of subword units representing $(\mathtt{h}, \mathtt{r}, \mathtt{t})$. Hence, a KGE model does not plainly retrieve embedding vectors of entities and relations but construct them on the fly. To the best of our knowledge, BYTE is the first attempt to make most KGE models support inference over unseen entities, relations, and literals.

Our extensive experiments with 4 state-of-the-art KGE models

* Corresponding Author. Email: caglar.demir@upb.de

[1] Equal contribution.

[2] Entities and relation involved in this setting are also seen during training

[3] In the inductive setting, KGE approaches are tasked to perform inference over unseen entities or relations by leveraging learned patterns

over 8 benchmark knowledge graphs suggest that BYTE improves the link prediction performance of KGE models on knowledge graphs where semantic information on syntactic representations of triples are visible. Yet, benefits of training a KGE model with BYTE dissipates (even reverses) on knowledge graphs where syntactic representations of entities are not semantically meaningful to a domain expert, e.g., a triple (`06cv1`, `person/profession`, `02jknp`) from FB15k-237. We provide an open-source implementation of BYTE within the library `dice-embeddings` [4], where it can be applied to any transductive KGE model by adding `--byte_pair_encoding` to the command for training.

2 Background and Related Works

2.1 Knowledge Graphs

A knowledge graph (KG) is often formally defined as a set of triples $\mathcal{G} \subset \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, where each triple $(\mathtt{h}, \mathtt{r}, \mathtt{t}) \in \mathcal{G}$ contains two entities/nodes $\mathtt{h}, \mathtt{t} \in \mathcal{E}$ and a relation/edge $\mathtt{r} \in \mathcal{R}$. Therein, $\mathcal{E}$ and $\mathcal{R}$ denote a finite set of entities/nodes and relations/edges. These collections of facts have been used in a wide range of applications, including web search, question answering, and recommender systems [23, 15]. Despite their wide application domains, most KGs on the web are incomplete [23].

2.2 Link Prediction and Knowledge Graph Embeddings

The link prediction task on KGs refers to predicting whether a triple is likely to be true. This task is often formulated as the problem of learning a parameterized scoring function $\phi_\Theta : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \to \mathbb{R}$ such that $\phi_\Theta(\mathtt{h}, \mathtt{r}, \mathtt{t})$ ideally signals the likelihood of $(\mathtt{h}, \mathtt{r}, \mathtt{t})$ is true [8]. For instance, given the triples (`western_europe`, `locatedin`, `europe`) and (`germany`, `locatedin`, `western_europe`) $\in \mathcal{G}$, a good scoring function is expected to return high scores for (`germany`, `locatedin`, `europe`), while returning a considerably lower score for (`europe`, `locatedin`, `germany`).

Most KGE models are designed to learn continuous vector representations of entities and relations tailored towards predicting missing triples. In our notation, the embedding vector of the entity $e \in \mathcal{E}$ is denoted by $\mathbf{e} \in \mathbb{R}^{d_e}$ and the embedding vector for the relation $r \in \mathcal{R}$ is denoted by $\mathbf{r} \in \mathbb{R}^{d_r}$. $\mathbf{E} \in \mathbb{R}^{|\mathcal{R}| \times d_e}$ and $\mathbf{R} \in \mathbb{R}^{|\mathcal{E}| \times d_r}$ are often called as an entity and a relation embedding matrices, respectively. Three training strategies are commonly used for KGE models. Bordes et al. [3] designed a negative sampling technique via perturbing an entity in a randomly sampled triple. A triple $(\mathtt{h}, \mathtt{r}, \mathtt{t}) \in \mathcal{G}$ is considered as a positive example, whilst $\{(\mathtt{h}, \mathtt{r}, \mathtt{x}) \mid \forall x \in \mathcal{E}\} \cup \{(\mathtt{x}, \mathtt{r}, \mathtt{t}) \mid \forall x \in \mathcal{E}\}$ is considered as a set of possible candidate negative examples. For each positive triple $(\mathtt{h}, \mathtt{r}, \mathtt{t}) \in \mathcal{G}$, a negative triple is sampled from the set of corresponding candidate negative triples. Given $(\mathtt{h}, \mathtt{r}, \mathtt{t})$, a triple score (e.g. or a distance) is computed by retrieving row vectors $\mathbf{h}, \mathbf{t} \in \mathbf{E}$ and $\mathbf{r} \in \mathbf{R}$ and applying the scoring function (e.g. element-wise multiplication followed by an inner product $\mathbf{h} \circ \mathbf{r} \cdot \mathbf{t}$).

Lacroix et al. [19] proposed 1vsAll/1vsN the training strategy that omits the idea of randomly sampling negative triples. For each positive triple $(\mathtt{h}, \mathtt{r}, \mathtt{t}) \in \mathcal{G}$, all possible tail perturbed set of triples are considered as negative triples regardless of whether a perturbed triple exists in the input knowledge graph KG ($\{(\mathtt{h}, \mathtt{r}, \mathtt{x}) | \forall x \in \mathcal{E} : x \neq t\}$).. Given that this setting does not involve negative triples via head perturbed entities, a data augmentation technique is applied to add inverse triples (also known as reciprocal triples [2]) $(\mathtt{t}, \mathtt{r}^{-1}, \mathtt{h})$ for each $(\mathtt{h}, \mathtt{r}, \mathtt{t})$. In the 1vsAll training strategy, a training data point consists of $(\mathtt{h}, \mathtt{r})$ and a binary vector containing a single "1" for the $\mathtt{t}$ and "0"s for other entities. Therefore, a KGE model is trained in a fashion akin to multi-class classification problem. Dettmers et al. [8] extended 1vsAll into KvsAll[5] via constructing multi-label binary vectors for each $(\mathtt{h}, \mathtt{r})$. A training data point consists of a pair $(\mathtt{h}, \mathtt{r})$ and a binary vector containing "1" for $\{x | x \in \mathcal{E} \wedge (\mathtt{h}, \mathtt{r}, \mathtt{x}) \in \mathcal{G}\}$ and "0"s for other entities. During training, for a given pair $(\mathtt{h}, \mathtt{r})$, predicted scores (logits) for all entities are computed, i.e., $\forall x \in \mathcal{E} : \phi((\mathtt{h}, \mathtt{r}, \mathtt{x}))) =: \mathbf{z} \in \mathbb{R}^{|\mathcal{E}|}$. Through the logistic sigmoid function $\sigma(\mathbf{z}) = \frac{1}{1+\exp(-\mathbf{z})}$, scores are normalized to obtain predicted probabilities of entities denoted by $\hat{\mathbf{y}}$. A loss incurred on a training data point is then computed as

$$l(\hat{\mathbf{y}}, \mathbf{y}) = -\frac{1}{|\mathcal{E}|} \sum_{i=1}^{|\mathcal{E}|} \mathbf{y}^{(i)} \log(\hat{\mathbf{y}}^{(i)}) + \big(1 - \mathbf{y}^{(i)}\big) \log\big(1 - \hat{\mathbf{y}}^{(i)}\big), \quad (1)$$

where $\mathbf{y} \in [0, 1]^{|\mathcal{E}|}$ is the binary sparse label vector. If $(\mathtt{h}, \mathtt{r}, \mathtt{e}_i) \in \mathcal{G}$, then $\mathbf{y}^{(i)} = 1$, otherwise $\mathbf{y}^{(i)} = 0$. Recent works show that learning Θ by means of minimizing Equation 1 often leads to state-of-the-art link prediction performance [2, 6]. Expectedly, 1vsAll and KvsAll are computationally more expensive than the negative sampling. As $|\mathcal{E}|$ increases, 1vsAll and KvsAll training strategies become less applicable. Yet, recent KGE models are commonly trained with 1vsAll or KvsAll [25].

2.3 Transductive Learning Approaches

A plethora of KGE models have been developed over the last decade [35, 5, 36]. Most KGE models map entities $e \in \mathcal{E}$ and relations $r \in \mathcal{R}$ found in a KG $\mathcal{G} \subset \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ to $\mathbb{V}$, where $\mathbb{V}$ is a d-dimensional vector space and $d \in \mathbb{N} \backslash \{0\}$ [15]. This family of models is currently one of the most popular means to make KGs amenable to *vectorial* machine learning [35] and has been used in applications including drug discovery, community detection, recommendation systems, and question answering [33, 13, 1]. While early models (e.g., RESCAL [21], TransE [3], DistMult [39]) compute embeddings in $\mathbb{R}^d$ and perform particularly well when trained appropriately [25], recent results suggest that embedding using the more complex division algebras $\mathbb{C}$ and $\mathbb{H}$ can achieve a superior link prediction performance (measured in terms of hits at n) [41, 40]. The superior performance of the latter is partially due to the characteristics of (hyper)complex algebras (e.g., $\mathbb{C}$, $\mathbb{H}$) being used to account for logical properties such as symmetry, asymmetry, and compositionality [33] of relations. Although recent works continue improving the predictive performance of these models by adding more complex neural architectures, the resulting models inherit the fundamental limitations of base models: The size of the embedding layer increases linearly w.r.t. the number of entities in the input knowledge graph, and unseen entities and relations cannot be handled at inference time. An overview of the transductive KGE models are given in Table 1. Here, we focused on multiplicative KGE models as recent results show that the aforementioned models reach state-of-the-art performance in the link prediction task while retaining computational efficiency over complex models [26, 6].

2.4 Inductive Learning Approaches

This family of approaches handle unseen entities at inference time by using rule mining techniques (e.g., DRUM [29], AnyBURL [20])

[4] BYTE Code: https://github.com/dice-group/dice-embeddings

[5] We use the terminology introduced by Ruffinelli et al. [25].

Table 1: Overview of KGE models. $\mathbf{e}$ denotes an embedding vector, d is the embedding vector size, $\overline{\mathbf{e}} \in \mathbb{C}$ corresponds to the complex conjugate of $\mathbf{e}$. $\times_n$ denotes the tensor product along the n-th mode. $\otimes, \circ, \cdot$ stand for the Hamilton, Hadamard, and inner product, respectively.

Model	Scoring Function	Vector Space	Additional
RESCAL [21]	$\mathbf{e}_h \cdot \mathcal{W}_r \cdot \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_t \in \mathbb{R}^d$	$\mathcal{W}_r \in \mathbb{R}^{d^2}$
DistMult [39]	$\mathbf{e}_h \circ \mathbf{e}_r \cdot \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{R}^d$	-
ComplEx [34]	$\mathrm{Re}(\langle \mathbf{e}_h, \mathbf{e}_r, \overline{\mathbf{e}_t} \rangle)$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{C}^d$	-
TuckER [2]	$\mathcal{W} \times_1 \mathbf{e}_h \times_2 \mathbf{e}_r \times_3 \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{R}^d$	$\mathcal{W} \in \mathbb{R}^{d^3}$
QMult [7]	$\mathbf{e}_h \otimes \mathbf{e}_r \cdot \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{H}^d$	-
Keci-[6]	$\mathbf{e}_h \circ \mathbf{e}_r \cdot \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in Cl_{p,q}(\mathbb{R}^d)$	-

or graph neural networks (e.g., NodePiece [11], GraphSAGE [14], GraIL [28]). Specifically, DRUM is a differential rule mining approach that learns rule structures and confidence values simultaneously. DRUM employs shared bidirectional RNNs to model relation interactions in rules, e.g., the relation `wife_of` cannot follow the relation `father_of` due to type constraints. AnyBURL is an anytime bottom-up rule mining approach specifically designed for large KGs. It mines rules in a sequence of time spans through random walks in the input graph, and stores rules which satisfy a given quality criterion. Both approaches (DRUM and AnyBURL) handle unseen entities by reasoning over the learned rules. NodePiece computes an embedding for an unseen entity by leveraging its relational context, i.e., by representing that entity as a hash of its known incident relation types and its closest anchor nodes. GraphSAGE uses node features (e.g., textual descriptions) of a local neighborhood to bootstrap an embedding for unseen entities. GraIL is a relation prediction approach that leverages sub-graph structures around target entities and message-passing to compute the likelihood of a triple. While these approaches achieve a remarkable performance on benchmark inductive link prediction tasks, they can only handle unseen entities for which the relational context (i.e., links) is known; in particular, they cannot be applied to tasks involving unseen relations. Our approach overcomes these limitations by operating at the subword/token level, and ensuring that every entity and relation can be encoded regardless of whether it was encountered during training or not. Most importantly, our approach is generic and can be applied to any KGE model.

2.5 Byte-pair Encoding Tokenization

Subword unit tokenization techniques are an effective way to address the open vocabulary problem in various domains [30, 17]. Most techniques convert raw sentences into unique subword sequences. Although subword segmentation is potentially ambiguous and multiple segmentations are possible even with the same vocabulary, subword unit tokenization techniques played an important role in the recent success of LLMs [24]. Byte Pair Encoding (BPE) [10] is a data compression technique that iteratively replaces the most frequent pairs of bytes in a sequence with a single, unused byte. Sennrich et al. [30] extend the initial BPE algorithm for word segmentation by merging characters or character sequences instead of merging frequent pairs of bytes. This modification turned out to be effective for neural machine translation with up to 1.3 absolute improvement in BLEU over baselines. The BPE technique in GPT-2 [24] is one of the most used techniques to convert natural language text into subword units which are then mapped to positive integers (`ids`) for embedding lookup. Although BPE is predominantly used in NLP, we employ it in this work to make KGEs support unseen entities and relations at inference time.

3 Methodology

In this section, we describe our overall approach to developing an embedding model that can be used in the three inference settings mentioned earlier, i.e., effectively handling unseen entities and relations. Our approach (BYTE) essentially consists of three main phases, **(i)** `Tokenization` 1 which decomposes each of the components of $(\mathtt{h}, \mathtt{r}, \mathtt{t})$ into sequences of subword units, **(ii)** `Embedding Lookup` 2 which fetches an embedding for each of the subword units from the embedding matrix, and **(iii)** `Linear Mapping` 3 which maps from the packed tokens' space dimension to the initial embedding dimension d (more details below). We define the steps **(i)–(iii)** by using the functions $\mathbf{Tok}(\cdot)$, $\mathbf{Emb}(\cdot)$, and $\mathbf{Lin}(\cdot)$, respectively. Since BYTE sequentially applies these three functions to a given input triple $(\mathtt{h}, \mathtt{r}, \mathtt{t}) \in \mathcal{G}$, it can be defined as the composition of the latter:

$$\text{BYTE} = \mathbf{Lin} \circ \mathbf{Emb} \circ \mathbf{Tok}. \tag{2}$$

The overall workflow is depicted in Figure 1 and described in the following subsections.

3.1 Tokenization

Step 1 (i.e., the function $\mathbf{Tok}(\cdot)$) works as the initial step which basically takes an input triple $(\mathtt{h}, \mathtt{r}, \mathtt{t})$ and generates a single token or multiple ones for each of the components $(\mathtt{h}, \mathtt{r}, \mathtt{t})$ depending on their string representation. Moreover, the type of the tokenizer (e.g., pre-trained on a specific corpus) chosen, and the size of the vocabulary also affect the output of the tokenization step. Since in our case the tokenizer is fixed (e.g., `GPT2`'s pre-trained tokenizer with a fixed vocabulary), the output of the tokenization step solely depends on the string representation of entities and relations. Formally, the tokenization step can be described by the following

$$\mathbf{Tok} : \mathcal{G} \rightarrow \mathbb{N}^m \times \mathbb{N}^m \times \mathbb{N}^m. \tag{3}$$

Herein m denotes the maximum number of subword units that can be found in an entity or a relation. Once m is fixed, longer entities/relations are truncated, and shorter ones are padded to the maximum length m. We include the padding/truncation operation within the tokenization step to generate equal-sized integer-valued arrays representing the indices of subword units. To explain it further, we consider an example triple with which we describe the tokenization step for $m = 2$.

Let us assume that we have the following triple ("Obama", "bornIn", "NewYork"). Using the `GPT2`'s pre-trained tokenizer with a vocabulary of size 50257, we get the following token `ids` for this triple.

Subject:	"Obama"	$\rightarrow$	[15948]
Predicate:	"bornIn"	$\rightarrow$	[6286, 818]
Object:	"NewYork"	$\rightarrow$	[3791, 49278]

Figure 2: Tokenization of a triple

In this case, due to their string representation, the tokenizer identifies different numbers of tokens. For instance, the subject "Obama" is identified with a single token while the predicate "bornIn" and the object "NewYork" comprise two subwords each and are identified with two tokens. This aligns well with the way humans would read and interpret each of these terms. Oppositely, traditional KGE methods fail to detect subword units and consider each entity and relation as a single word which is then mapped to a single `id` in the embedding layer.

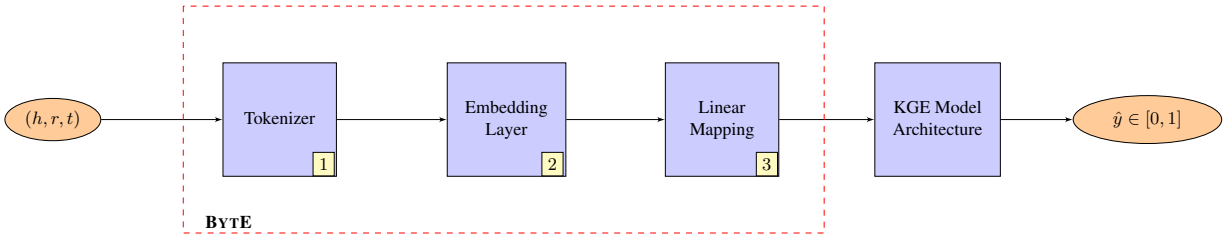

Figure 1: Workflow of a KGE model using our byte-pair encoding approach.

To make batch processing possible, we need to ensure that the head entity "Obama" is also represented with two indices (`ids`). This can be done by simply using the token "`\t`" as padding since it is often assigned an embedding as opposed to the traditional padding token "`|<pad>|`". With this, "Obama" is now represented by $[15948, 197]$.

3.2 Embedding Lookup

Step 2 (i.e., the function $\mathbf{Emb}(\cdot)$) gets the token `ids` generated in the previous step and assigns an embedding vector of size d (that is to be fixed beforehand) to each of the corresponding tokens. Thus, this step can be formally defined by the following function

$$\mathbf{Emb} : \mathbb{N}^m \times \mathbb{N}^m \times \mathbb{N}^m \rightarrow \mathbb{R}^{m \times d} \times \mathbb{R}^{m \times d} \times \mathbb{R}^{m \times d}. \quad (4)$$

Here, each of the tokens generated for each component of the triple is mapped to a real-valued vector $\mathbb{R}^d$. Hence, for a specific triple $(\mathtt{h}, \mathtt{r}, \mathtt{t})$, we get 3 matrices of size $m \times d$ each. During training, the embeddings that are generated for each of the tokens for a specific $(\mathtt{h}, \mathtt{r}, \mathtt{t})$ are trainable parameters; they are tuned to optimize the training loss. Assuming $d = 4$, we illustrate this further in Figure 3 by using our running example.

$[15948, 197] \rightarrow$ [[2.30, -1.87, 7.82, -5.91],
[8.10, -5.39, -1.08, 4.46]]
$[6286, 818] \rightarrow$ [[-1.81, -3.95, 4.84, -8.91],
[0.81, 0.95, -2.84, 3.48]]
$[3791, 49278] \rightarrow$ [[3.05, 0.08, -9.66, 4.01],
[-2.95, 9.34, 1.66, 13.01]]

Figure 3: Generating embeddings of size 4 for each token

As can be seen in Figure 3, each token is mapped to an embedding vector of size 4. With this, entities and relations are represented by matrices in $\mathbb{R}^{2 \times 4}$. In the next subsection, we describe how these matrices are mapped back to the embedding space $\mathbb{R}^d$.

3.3 Linear Mapping

Step 3 (i.e., the function $(\mathbf{Lin}(\cdot))$ maps each of the embedding matrices (of size $m \times d$) generated for the components of the input triple to a real-valued vector of size d. This step works as a sort of bridge that connects BYTE to the traditional KGE framework. Formally, this step is defined as follows

$$\mathbf{Lin} : \mathbb{R}^{m \times d} \times \mathbb{R}^{m \times d} \times \mathbb{R}^{m \times d} \rightarrow \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^d. \quad (5)$$

Thus, at the end of this step for a specific triple $(\mathtt{h}, \mathtt{r}, \mathtt{t})$, we get 3 vectors of size d. To achieve this, a *flattening* operation is first applied to the embedding matrix of each input triple's component, resulting in a vector v with md entries, i.e., an element of $\mathbb{R}^{md \times 1}$. Next, a trainable weight matrix $W \in \mathbb{R}^{d \times md}$ and optionally a trainable bias vector $b \in \mathbb{R}^{d \times 1}$ are applied to project v onto $\mathbb{R}^d$ as $Wv + b$. The matrix W and the bias vector b are shared across all components and across all triples in a given knowledge graph. Note that, before the flattening operation is applied, an *attention* layer can also be applied to capture the relationship between different subword units within the components of a triple. In any case, the embedding model expects inputs to be vectors in $\mathbb{R}^d$, and this is what the linear mapping takes care of. We exemplify this step further with our running example as follows:

$[[2.30, -1.87, 7.82, -5.91],$
$[8.10, -5.39, -1.08, 4.46]] \rightarrow$ [7.06, -3.81, 6.19, 9.73]
$[[-1.81, -3.95, 4.84, -8.91],$
$[0.81, 0.95, -2.84, 3.48]] \rightarrow$ [3.63, -5.37, -9.14, -2.55]
$[[3.05, 0.08, -9.66, 4.01],$
$[-2.95, 9.34, 1.66, 13.01]] \rightarrow$ [1.86, 2.88, 6.51, -3.56].

Figure 4: Example input and output of the linear mapping

Finally, the output of the linear mapping is forwarded to the KGE model which generates $\hat{y} \in [0, 1]$, representing the likelihood of the given triple $(\mathtt{h}, \mathtt{r}, \mathtt{t})$ being true. Herein, the KGE model can be of any type (for e.g., DistMult, ComplEx, and others), and more importantly, our approach BYTE does not depend on it. That gives us the flexibility to use the KGE model of our choice to obtain the best possible results. To the best of our knowledge, our work is the first in this line to propose such an approach using tokenizers from LLMs to make KGE models handle unseen entities, relations, and literals.

4 Experimental Setup

4.1 Datasets

We used the benchmark datasets UMLS, KINSHIP, NELL-995 h25, NELL-995 h75, NELL-995 h100, WN18RR, FB15K-237, and YAGO3-10 for the link prediction problem. UMLS describes medical entities, e.g., `cell`, `immunologic_factor`, and the relationships

Table 2: An overview of datasets in terms of number of entities, relations, and train split along with the number of triples in each split of the dataset.

Dataset	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{G}^{\text{Train}}\rvert$	$\lvert\mathcal{G}^{\text{Validation}}\rvert$	$\lvert\mathcal{G}^{\text{Test}}\rvert$
Countries-S1	271	2	1111	24	24
Countries-S2	271	2	1063	24	24
Countries-S3	271	2	985	24	24
UMLS	135	46	5,216	652	661
KINSHIP	104	25	8,544	1,068	1,074
NELL-995 h100	22,411	43	50,314	3,763	3,746
NELL-995 h75	28,085	57	59,135	4,441	4,389
NELL-995 h25	70,145	344	245,236	18,388	18,374
FB15K-237	14,541	237	272,115	17,535	20,466
YAGO3-10	123,182	37	1,079,040	5,000	5,000

Table 3: Link prediction results on Countries benchmark datasets.

	S1				S2				S3			
	MRR	@1	@3	@10	MRR	@1	@3	@10	MRR	@1	@3	@10
DistMult-train	0.515	0.396	0.571	0.749	0.458	0.340	0.516	0.693	0.481	0.370	0.525	0.704
DistMult-test	0.273	0.167	0.292	0.479	0.166	0.083	0.167	0.333	0.115	0.062	0.104	0.208
DistMult-BYTE-train	0.763	0.599	0.918	0.960	0.757	0.608	0.896	0.956	0.679	0.505	0.837	0.942
DistMult-BYTE-test	**0.612**	**0.500**	**0.729**	**0.812**	**0.611**	**0.542**	**0.646**	**0.750**	**0.330**	**0.208**	**0.354**	**0.562**
ComplEx-train	0.260	0.158	0.280	0.469	0.274	0.168	0.298	0.480	0.250	0.145	0.272	0.462
ComplEx-test	0.183	0.062	0.229	0.438	0.162	0.083	0.167	0.333	0.065	0.021	0.042	0.146
ComplEx-BYTE-train	0.910	0.855	0.961	0.982	0.943	0.913	0.967	0.988	0.832	0.756	0.891	0.951
ComplEx-BYTE-test	**0.441**	**0.271**	**0.542**	**0.708**	**0.422**	**0.250**	**0.521**	**0.688**	**0.178**	**0.083**	**0.229**	**0.312**
QMult-train	0.133	0.060	0.126	0.270	0.187	0.098	0.196	0.366	0.164	0.087	0.170	0.313
QMult-test	0.112	0.021	0.125	0.333	0.131	0.062	0.146	0.292	**0.124**	0.062	**0.125**	**0.229**
QMult-BYTE-train	0.925	0.883	0.964	0.986	0.889	0.830	0.943	0.981	0.621	0.528	0.660	0.814
QMult-BYTE-test	**0.293**	**0.167**	**0.333**	**0.521**	**0.483**	**0.375**	**0.542**	**0.667**	0.110	**0.062**	0.083	0.188
Keci-train	0.947	0.914	0.977	0.993	0.943	0.908	0.976	0.995	0.949	0.918	0.975	0.988
Keci-test	0.208	0.104	0.229	0.479	0.278	0.104	0.396	0.604	0.072	0.301	0.083	0.146
Keci-BYTE-train	0.997	0.994	1.000	1.000	0.958	0.917	1.000	1.000	0.986	0.973	1.000	1.000
Keci-BYTE-test	**0.566**	**0.354**	**0.688**	**0.917**	**0.557**	**0.417**	**0.625**	**0.833**	**0.218**	**0.146**	**0.208**	**0.375**

between them, e.g., `disrupts`. KINSHIP describes the 25 different kinship relations of the Alyawarra tribe.

FB15K-237 is a subset of Freebase which is a collaborative knowledge graph of general knowledge. Terms found in this knowledge graph include `Stephen_Hawking`, `Copley_Medal`, and more. YAGO3-10 is a subset of YAGO [8], which mostly contains information about people, with relation names such as `actedIn` and `hasWonPrize`. The Never-Ending Language Learning datasets NELL-995 h25, NELL-995 h50, and NELL-995 h100 are designed to evaluate multi-hop reasoning capabilities of various approaches for learning on KGs [37]. An overview of the datasets is provided in Table 2.

4.2 Training and Optimization

Throughout our experiments, we followed a standard training setup: we used the cross-entropy loss function, KvsAll scoring technique, Adam optimizer and we performed a grid search over learning rate $\{0.1, 0.01, 0.011\}$, embedding dimension $d \in \{32, 64\}$, number of epochs 500 on each dataset [6]. Unless stated otherwise, we did not use any regularization technique (e.g., dropout technique or L2 regularization). In our parameter analysis experiments, we explored a large range of embedding dimensions $\{2, 4, 8, 16, 32, 64, 128, 256\}$. We report the training and test results to show a fine-grained performance overview across datasets and models. Each entity and relation is represented with a d-dimensional real-valued vector across datasets and models. Hence, DistMult, ComplEx, QMult, and OMult learn embeddings in $\mathbb{R}^d, \mathbb{C}^{d/2}$, and $\mathbb{H}^{d/4}$, respectively. We evaluated the link prediction performance of models with benchmark metrics such as filtered MRR, and Hits@k. In evaluation results, Hits@k is abbreviated as k. At test time, learned embeddings of subword units are used to compute triple scores.

5 Results

Tables 3, 5 and 6 report the link prediction results on the Countries, UMLS, KINSHIP, and NELL-955 benchmark datasets. Overall, results suggest that BYTE often improves the link prediction results on knowledge graphs where syntactic representations of triples are semantically meaningful. For instance, on the countries datasets (i.e., Countries-S1, Countries-S2, and Countries-S3) BYTE improves the link prediction performances of DistMult, ComplEx, QMult, and Keci on the training and the test splits. Therein, semantic information on syntactic representations of triples is visible, e.g., (`western_europe`, `locatedin`, `europe`), (`germany`, `locatedin`, `western_europe`). With BYTE, KGE models in this case learn embeddings for subword units, e.g., `western`, _, and `europe`. At test time, learned embeddings of subword units are combined (see linear mapping in Section 3.3) to compute triple scores, while transductive KGE models learn to represent each entity and relation with respective embedding vectors independently. Hence, BYTE does not only improve the generalization performance of models but also leads to a better fit to the training datasets. These three datasets contain triples whose syntactic representations are semantically meaningful, e.g., (`western_europe`, `locatedin`, `europe`), (`germany`, `locatedin`, `western_europe`), and (`germany`, `locatedin`, `europe`). Incorporating such knowledge into the learning process improves the link prediction results across models and across the three datasets. Table 5 reports the link prediction results on the UMLS and KINSHIP datasets. Overall, the results continue to indicate that BYTE often improves the link prediction results on the training and the test splits if the given knowledge graph contains triples whose representations are semantically meaningful, e.g., (`lipid`, `affects`, `physiologic_function`). Yet, on the KINSHIP dataset, BYTE does not seem to improve the link prediction results. This could be explained by using the fact that KINSHIP does not contain triples whose syntactic representations are as semantically meaningful as those triples on the countries and UMLS benchmark datasets, e.g., (`person20`, `term11`, `person46`) and (`person83`, `term8`, `person25`).

Table 6 reports the link prediction results on the NELL-995-h100,-h75, and -h25 datasets. Herein, the results on h100 and h75 suggest that using BYTE improves the link prediction performance on the test dataset in 28 out of 32 cases. However, results on h25 indicate that training a KGE model with BYTE leads to poor link prediction results with especially DistMult, ComplEx, and Keci (e.g. ≤ 0.07 MRR). These results were quite surprising, and to investigate further we delved into the details and observed that QMult applies either standard unit or batch normalization over embeddings of entities and relations, whereas DistMult, ComplEx, and Keci do not.

Table 4: Predicted unnormalized log-likelihood of triples on Countries dataset. "Seen Terms" denotes a triple containing an unseen entity/relation.

Triples	Seen Terms	**Keci**	**BYTE**	**BYTE** + L2 reg.	**BYTE** + L2 + Dropout
(germany, locatedin, europe)	✓	2.4	1151.9	453.1	487.1
(germany, locatedin, western_europe)	✓	1.6	1596.7	625.5	623.6
(western_europe, locatedin, europe)	✓	2.9	1335.7	345.5	217.6
(germany, located, europe)	✗	-	1237.1	661.7	74.9
(western_europ, located, europe)	✗	-	1427.2	611.5	2.11
(germany, located_in, europe)	✗	-	612.3	276.8	222.2

We observed that the sizes of the byte-pair encoded triples are larger on h25 as compared to the other 7 benchmark datasets. Thus, we presume that these two factors might have contributed to the poor performance of some of the KGE models.

We have furthermore conducted experiments considering the WN18RR, FB15k-237, and YAGO3-10 datasets, where BYTE has mostly led to poor link prediction results—MRR $\leq$ 0.1. This again could be attributed to the fact that the syntactic representations of triples on these datasets are neither semantically meaningful nor ambiguous. For instance, consider (/m/06cv1, person/profession, /m/02jknp) from FB15k-237, (01455754, _hypernym, 01974062) from WN18RR, and (Taribo_West, playsFor, Inter_Milan) from YAGO3-10. Entities such as 01455754 are represented with numbers and do not convey much information after tokenization. While the entity Taribo_West is the name of a football player, the token _West is not necessarily understood as a name. Thus, to deal with datasets where such triples are more prevalent, sophisticated neural network architectures such as LLMs are needed to improve the performance.

Inference over Unseen Entities, Relations, and Literals. Table 4 reports the predicted likelihoods of existing triples and hand-crafted out-of-vocabulary triples. The results show that the KGE model Keci with or without BYTE correctly assigns positive scores for existing triples, while Keci alone cannot do inference over triples containing unseen entities or relations. BYTE effectively allows Keci to assign positive scores for out-of-vocabulary triples that are only perturbed by removing or adding few characters in their string representation (e.g. adding _ or removing in). These results also suggest that BYTE leads to over-confidence in predictions. After observing overconfident scores, we conducted two additional experiments by applying L2 regularization and/or dropout on embeddings to quantify possible reduction in the magnitude of predicted normalized likelihood. Results indicate that L2 regularization and dropout can be used to reduce the overconfidence in the predictions over unseen entities and relations, thereby improving the effectiveness of BYTE.

Impact of the Number of Byte-pair Encoded Subword Units. In Table 8, we report the maximum performance of the BYTE-augmented KGE models on different datasets and the corresponding byte-pair encoded sequence lengths. The results suggest that as the number of byte-pair encoded subword units grows, the performance decreases. This may be due to the fact that computing triple score via element-wise operations (e.g. element-wise multiplication followed by an inner product in DistMult) over unnormalized embedding vectors leads to unstable training, even with a small learning rate (e.g. Adam with 0.005 learning rate).

Impact of Subword Unit Dimensions on Link Prediction. Table 7 reports our parameter analysis for link prediction performance with different subword unit dimensions. The performance (on the test set) of both the base model Keci and the augmented model Keci-BYTE first increases (for $d < 32$) then decreases (for $d > 64$).

Both models achieve similar average test results although Keci-BYTE is slightly better. However, as the number of embedding dimensions increases ($d > 64$), Keci starts to overfit on the training set while Keci-BYTE fails to train properly. This suggests that KGE models employing BYTE should use a moderate number of embedding dimensions for optimal performance.

6 Discussion

Overall, we see that BYTE could be used to perform inference over unseen entities and relations thereby alleviating the typical transductive settings of KGE models. However, based on the evaluation results, we can conclude that the effectiveness of BYTE can further be improved by the following two techniques. (1) Applying multi-head self-attention mechanism on byte-pair encoded subword unit embeddings of entities and relations may improve the link prediction performance. Currently, interactions between subword units composing an entity or relation is not captured. Therefore, modelling such interactions by computing attention weights between subword units can improve the link prediction performance. (2) A pre-trained publicly available large language model (e.g., Mistral [16], Llama 2 2 [32]) can be used to initialize the embedding vectors of subword units in BYTE. With this, not only the training runtimes of BYTE can be reduced but also, at testing time, its performance can become more stable. A KGE model using BYTE only updates embeddings of subword units if such subword units are encountered during training. Therefore, during testing, although a KGE model can do inference over unseen entities and relations, its predictions may involve randomly initialized embeddings. During our experiments, we also observe that the performance of BYTE degrades as the size of the byte-pair-encoded subword units increases, see Table 8. Similarly, on some benchmark datasets, we observe that increasing the number of embedding dimensions decreases the training as well as the testing performance. This can be attributed to the fact that KGE models often apply linear operations (e.g. dot product) without any scaling or normalization.

Note that, in our evaluation, we do not include other link prediction methods in the inductive setting (such as NodePiece [11]). This is because, we solely focus on extending the existing transductive KGE models to work in the inductive setting. Therefore, we compare the link prediction performances to the inductive link performances of the current state-of-the-art transductive KGE models such as DistMult, ComplEX, and others. In other words, our evaluation solely focuses on showing the alleviation of the capabilities of the current transductive models in performing inductive tasks. Moreover, models like NodePiece require additional information, for instance, the local structure of each node at testing time. Hence, to use inductive KGE models at testing, large KGs must be stored in memory. On the contrary, since our approach uses the existing transductive KGE models, they require only the embeddings of entities and relations.

Table 5: Link prediction results on UMLS and KINSHIP.

	UMLS				KINSHIP			
	MRR	@1	@3	@10	MRR	@1	@3	@10
DistMult-train	0.497	0.364	0.563	0.756	0.493	0.327	0.570	0.863
DistMult-test	0.414	0.275	0.468	0.696	0.405	0.235	0.463	**0.819**
DistMult-BYTE-train	0.792	0.699	0.862	0.954	0.500	0.350	0.565	0.838
DistMult-BYTE-test	**0.715**	**0.609**	**0.779**	**0.911**	**0.435**	**0.284**	**0.491**	0.786
ComplEx-train	0.435	0.304	0.491	0.698	0.529	0.370	0.615	0.869
ComplEx-test	0.368	0.241	0.414	0.620	0.453	0.288	**0.530**	**0.822**
ComplEx-BYTE-train	0.895	0.835	0.947	0.987	0.519	0.371	0.584	0.854
ComplEx-BYTE-test	**0.826**	**0.744**	**0.889**	**0.974**	**0.458**	**0.299**	0.522	0.818
QMult-train	0.515	0.392	0.572	0.766	0.497	0.341	0.576	0.822
QMult-test	0.439	0.308	0.501	0.702	0.423	0.264	0.494	0.754
QMult-BYTE-train	0.899	0.825	0.968	0.988	0.657	0.518	0.745	0.931
QMult-BYTE-test	**0.832**	**0.727**	**0.928**	**0.976**	**0.604**	**0.453**	**0.694**	**0.911**
Keci-train	0.733	0.614	0.821	0.935	0.575	0.418	0.663	0.912
Keci-test	0.623	0.480	0.720	0.873	**0.463**	**0.290**	**0.541**	**0.860**
Keci-BYTE-train	0.845	0.754	0.920	0.976	0.516	0.361	0.588	0.863
Keci-BYTE-test	**0.757**	**0.644**	**0.841**	**0.941**	0.445	0.284	0.500	0.824

Table 6: Link prediction results on NELL-995 h100, h75, and h50.

	h100				h75				h25			
	MRR	@1	@3	@10	MRR	@1	@3	@10	MRR	@1	@3	@10
DistMult-train	0.586	0.494	0.636	0.761	0.750	0.675	0.796	0.888	0.927	0.891	0.957	0.984
DistMult-test	0.225	0.159	0.254	0.357	0.225	0.162	0.252	0.346	**0.223**	**0.167**	**0.248**	**0.328**
DistMult-BYTE-train	0.307	0.224	0.341	0.468	0.262	0.190	0.290	0.402	0.002	0.000	0.000	0.007
DistMult-BYTE-test	**0.266**	**0.193**	**0.292**	**0.413**	**0.247**	**0.180**	**0.270**	**0.381**	0.007	0.002	0.000	0.007
ComplEx-train	0.548	0.460	0.595	0.715	0.692	0.614	0.737	0.837	0.964	0.949	0.976	0.988
ComplEx-test	0.226	0.160	0.254	0.349	0.235	0.172	0.259	0.361	**0.206**	**0.152**	**0.228**	**0.313**
ComplEx-BYTE-train	0.333	0.245	0.371	0.504	0.313	0.232	0.348	0.470	0.000	0.000	0.000	0.000
ComplEx-BYTE-test	**0.277**	**0.201**	**0.307**	**0.431**	**0.273**	**0.201**	**0.302**	0.415	0.000	0.000	0.000	0.000
QMult-train	0.398	0.303	0.440	0.582	0.570	0.476	0.618	0.756	0.856	0.802	0.896	0.954
QMult-test	0.202	0.133	0.230	0.337	0.223	0.157	0.249	0.355	**0.249**	**0.188**	**0.274**	**0.369**
QMult-BYTE-train	0.337	0.245	0.379	0.516	0.323	0.239	0.358	0.486	0.269	0.202	0.293	0.396
QMult-BYTE-test	**0.281**	**0.202**	**0.316**	**0.433**	**0.272**	**0.198**	**0.301**	**0.415**	0.242	0.182	0.266	0.356
Keci-train	0.804	0.738	0.849	0.922	0.863	0.814	0.897	0.949	0.924	0.890	0.951	0.977
Keci-test	**0.231**	**0.164**	**0.255**	**0.366**	0.214	0.149	0.237	0.342	**0.216**	**0.159**	**0.240**	**0.326**
Keci-BYTE-train	0.053	0.026	0.060	0.104	0.266	0.192	0.295	0.408	0.003	0.002	0.003	0.004
Keci-BYTE-test	0.050	0.024	0.057	0.098	**0.252**	**0.180**	**0.280**	**0.391**	0.003	0.002	0.003	0.004

Table 7: Impact of subword unit embedding dimensions in the link prediction of BYTE on the UMLS dataset.

Dim.	Keci		Keci BYTE	
	Train MRR	Test MRR	Train MRR	Test MRR
2	0.377	0.382	0.337	0.350
4	0.645	0.631	0.518	0.509
8	0.850	0.812	0.755	0.705
16	0.948	0.830	0.879	0.784
32	0.990	0.709	0.934	0.794
64	1.000	0.579	0.947	0.734
128	1.000	0.544	0.878	0.714
256	1.000	0.536	0.820	0.613
512	1.000	0.565	0.643	0.497
Avg.	**0.868**	0.621	0.746	**0.633**

7 Conclusion

In this paper, we proposed a technique (BYTE) to endow knowledge graph embedding models with inductive capabilities. BYTE effectively leverages a byte-pair encoding technique to obtain subword units representing entities and relations of a knowledge graph. In this way, knowledge embedding models are able to perform well on unseen entities, relations as well as literals by learning embeddings over a sequence of subword units. Our extensive experiments on benchmark datasets indicate that BYTE often improves link prediction results provided that the syntactic representation of triples are semantically meaningful. In the future, we plan to use an attention mechanism to capture pair-wise interactions between subword units to improve the predictive performance of KGEs employing BYTE. We will also investigate the use of pretrained LLMs as few-shot learners on knowledge graphs.

Table 8: Impact of Byte-pair encoded triple size on training MRR.

Datasets	Triple size with BYTE	Max. MRR
Countries	13×3	0.997
UMLS	16×3	0.899
KINSHIP	5×3	0.657
h100	29×3	0.337
h75	29×3	0.323
h25	56×3	0.268

Acknowledgements

This work has received funding from the European Union's Horizon Europe research and innovation programme within the project

ENEXA under the grant No 101070305, and the Ministry of Culture and Science of North Rhine Westphalia (MKW NRW) within the project SAIL under the grant No NW21-059D. This work has also been supported within the project "WHALE" (LFN 1-04) funded under the Lamarr Fellow Network programme by the Ministry of Culture and Science of North Rhine-Westphalia (MKW NRW).

References

[1] E. Arakelyan, D. Daza, P. Minervini, and M. Cochez. Complex query answering with neural link predictors. In *ICLR*. OpenReview.net, 2021.

[2] I. Balazevic, C. Allen, and T. M. Hospedales. Tucker: Tensor factorization for knowledge graph completion. In *EMNLP/IJCNLP (1)*, pages 5184–5193. Association for Computational Linguistics, 2019.

[3] A. Bordes, N. Usunier, A. García-Durán, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, pages 2787–2795, 2013.

[4] S. Choo and W. Kim. A study on the evaluation of tokenizer performance in natural language processing. *Applied Artificial Intelligence*, 37(1): 2175112, 2023. doi: 10.1080/08839514.2023.2175112. URL https://doi.org/10.1080/08839514.2023.2175112.

[5] Y. Dai, S. Wang, N. N. Xiong, and W. Guo. A survey on knowledge graph embedding: Approaches, applications and benchmarks. *Electronics*, 9 (5):750, 2020.

[6] C. Demir and A.-C. Ngonga Ngomo. Clifford embeddings–a generalized approach for embedding in normed algebras. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 567–582. Springer, 2023.

[7] C. Demir, D. Moussallem, S. Heindorf, and A. N. Ngomo. Convolutional hypercomplex embeddings for link prediction. In *ACML*, volume 157 of *Proceedings of Machine Learning Research*, pages 656–671. PMLR, 2021.

[8] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[9] L. Floridi and M. Chiriatti. GPT-3: its nature, scope, limits, and consequences. *Minds Mach.*, 30(4):681–694, 2020.

[10] P. Gage. A new algorithm for data compression. *C Users Journal*, 12(2): 23–38, 1994.

[11] M. Galkin, E. G. Denis, J. Wu, and W. L. Hamilton. Nodepiece: Compositional and parameter-efficient representations of large knowledge graphs. In *ICLR*. OpenReview.net, 2022.

[12] M. Galkin, X. Yuan, H. Mostafa, J. Tang, and Z. Zhu. Towards foundation models for knowledge graph reasoning. *arXiv preprint arXiv:2310.04562*, 2023.

[13] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[14] W. L. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1024–1034, 2017.

[15] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. d. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, et al. Knowledge graphs. *ACM Computing Surveys (CSUR)*, 54(4):1–37, 2021.

[16] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[17] T. Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. *arXiv preprint arXiv:1804.10959*, 2018.

[18] T. Kudo and J. Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *EMNLP (Demonstration)*, pages 66–71. Association for Computational Linguistics, 2018.

[19] T. Lacroix, N. Usunier, and G. Obozinski. Canonical tensor decomposition for knowledge base completion. In *International Conference on Machine Learning*, pages 2863–2872. PMLR, 2018.

[20] C. Meilicke, M. W. Chekol, D. Ruffinelli, and H. Stuckenschmidt. Anytime bottom-up rule learning for knowledge graph completion. In *IJCAI*, pages 3137–3143. ijcai.org, 2019.

[21] M. Nickel, V. Tresp, and H. Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, pages 809–816. Omnipress, 2011.

[22] M. Nickel, V. Tresp, and H.-P. Kriegel. A three-way model for collective learning on multi-relational data. In *Icml*, 2011.

[23] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015.

[24] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1 (8):9, 2019.

[25] D. Ruffinelli, S. Broscheit, and R. Gemulla. You CAN teach an old dog new tricks! on training knowledge graph embeddings. In *ICLR*. OpenReview.net, 2020.

[26] D. Ruffinelli, S. Broscheit, and R. Gemulla. You CAN teach an old dog new tricks! on training knowledge graph embeddings. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=BkxSmlBFvr.

[27] P. Rust, J. Pfeiffer, I. Vulić, S. Ruder, and I. Gurevych. How good is your tokenizer? on the monolingual performance of multilingual language models. *arXiv preprint arXiv:2012.15613*, 2020.

[28] A. Sadeghian, M. Armandpour, P. Ding, and D. Z. Wang. DRUM: end-to-end differentiable rule mining on knowledge graphs. In *NeurIPS*, pages 15321–15331, 2019.

[29] A. Sadeghian, M. Armandpour, P. Ding, and D. Z. Wang. DRUM: end-to-end differentiable rule mining on knowledge graphs. In *NeurIPS*, pages 15321–15331, 2019.

[30] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In K. Erk and N. A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, Aug. 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162.

[31] K. K. Teru, E. G. Denis, and W. L. Hamilton. Inductive relation prediction by subgraph reasoning. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 9448–9457. PMLR, 2020.

[32] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[33] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard. Complex embeddings for simple link prediction. In *International conference on machine learning*, pages 2071–2080. PMLR, 2016.

[34] T. Trouillon, C. R. Dance, É. Gaussier, J. Welbl, S. Riedel, and G. Bouchard. Knowledge graph completion via complex tensor factorization. *J. Mach. Learn. Res.*, 18:130:1–130:38, 2017.

[35] M. Wang, L. Qiu, and X. Wang. A survey on knowledge graph embeddings for link prediction. *Symmetry*, 13(3):485, 2021.

[36] Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 2017.

[37] W. Xiong, T. Hoang, and W. Y. Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. *arXiv preprint arXiv:1707.06690*, 2017.

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

[39] B. Yang, W. Yih, X. He, J. Gao, and L. Deng. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR (Poster)*, 2015.

[40] M. Yu, C. Bai, J. Yu, M. Zhao, T. Xu, H. Liu, X. Li, and R. Yu. Translation-based embeddings with octonion for knowledge graph completion. *Applied Sciences*, 12(8):3935, 2022.

[41] S. Zhang, Y. Tay, L. Yao, and Q. Liu. Quaternion knowledge graph embeddings. *Advances in neural information processing systems*, 32, 2019.