OKGIT: Open Knowledge Graph Link Prediction with Implicit Types

Chandrahas
Indian Institute of Science, Bangalore
chandrahas@iisc.ac.in

Partha Pratim Talukdar
Indian Institute of Science, Bangalore
ppt@iisc.ac.in

Abstract

Open Knowledge Graphs (OpenKG) refer to a set of (head noun phrase, relation phrase, tail noun phrase) triples such as (tesla, return to, new york) extracted from a corpus using OpenIE tools. While OpenKGs are easy to bootstrap for a domain, they are very sparse and far from being directly usable in an end task. Therefore, the task of predicting new facts, i.e., link prediction, becomes an important step while using these graphs in downstream tasks such as text comprehension, question answering, and web search query recommendation. Learning embeddings for OpenKGs is one approach for link prediction that has received some attention lately. However, on careful examination, we found that current OpenKG link prediction algorithms often predict noun phrases (NPs) with incompatible types for given noun and relation phrases. We address this problem in this work and propose OKGIT that improves OpenKG link prediction using novel type compatibility score and type regularization. With extensive experiments on multiple datasets, we show that the proposed method achieves state-of-the-art performance while producing type compatible NPs in the link prediction task.

1 Introduction

An Open Knowledge Graph (OpenKG) is a set of factual triples extracted from a text corpus using Open Information Extraction (OpenIE) tools such as TEXTRUNNER (Banko et al., 2007) and ReVerb (Fader et al., 2011). These triples are of the form (noun phrase, relation phrase, noun phrase), e.g., (*tesla*, *return to*, *new york*). An OpenKG can be viewed as a multi-relational graph where the noun phrases (NPs) are the nodes, and the relation phrases (RPs) are the labeled edges between pairs of nodes. It is easy to bootstrap OpenKGs from a domain-specific corpus, making them suitable for newer domains. However, they are extremely sparse and may not be directly usable for an end task. Therefore, tasks such as NP canonicalization (merging mentions of the same entity) and link prediction (predicting new facts) become an important step in downstream applications. Some example applications are text comprehension (Mausam, 2016), relation schema induction (Nimishakavi et al., 2016), canonicalization (Vashishth et al., 2018), question answering (Yao and Van Durme, 2014), and web search query recommendation (Huang et al., 2016). In this work, we focus on improving OpenKG link prediction.

Triple	(*tesla, return to, ?*)				
CaRE	*polytechnic institute*	*2009*	*1986*	*jp morgan*	*patent*
BERT	*chicago*	*earth*	*england*	*america*	*detroit*
OKGIT	*new york*	*america*	*paris*	*california*	*london*

Table 1: Some sample tail NP predictions by CaRE, BERT, and OKGIT. The true tail NP is underlined. As we can see, both CaRE and BERT fail to predict the correct tail NP. However, BERT predictions are type compatible with the query. OKGIT predicts the correct NP while improving the type compatibility with the query.

Although OpenKGs are structurally similar to Ontological KGs, they come with a different set of challenges. They are extremely sparse, NPs and RPs are not canonicalized, and no type information is present for NPs. There has been much work on learning embeddings for Ontological KGs in the past years. However, this task has not received much attention in the context of OpenKGs. CaRE (Gupta et al., 2019) is a recent method which addresses this problem. It learns embeddings for NPs and RPs in an OpenKG while incorporating NP canonicalization information. However, even after incorporating canonicalization, we find that CaRE struggles to predict NPs whose types are compati-

Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, pages 2546–2559

ble with given head NP and RP.

As observed by Petroni et al. (2019), modern pre-trained language representation models like BERT can store factual knowledge and can be used to perform link prediction in KGs. However, in our explorations with OpenKGs, we found that even though BERT may not predict the correct NP on the top, it predicts type compatible NPs (Table 1). A similar observation was also made in the context of entity linking (Chen et al., 2020). As OpenKGs do not have any underlying ontology and obtaining type information can be expensive, BERT predictions can help improve OpenKG link prediction.

Motivated by this, we employ BERT for improving OpenKG link prediction, using novel type compatibility score (Section 4.2) and type regularizer term (Section 4.4). We propose OKGIT, a method for OpenKG link prediction with improved type compatibility. We test our model on multiple datasets and show that it achieves state-of-the-art performance on all of these datasets.

We make the following contributions:

- We address the problem of OpenKG link prediction, focusing on improving type compatibility of predictions. To the best of our knowledge, this is the first work that addresses this problem.
- We propose OKGIT, a method for OpenKG link prediction with novel type compatibility score and type regularization. OKGIT can utilize NP canonicalization information while improving the type compatibility of predictions.
- We evaluate OKGIT on the link prediction across multiple datasets and observe that it outperforms the baseline methods. We also demonstrate that the learned model generates more type compatible predictions.

Source code for the proposed model and the experiments from this paper is available at https://github.com/Chandrahasd/OKGIT.

2 Related Work

OpenKG Embeddings: Learning embeddings for OpenKGs has been a relatively under-explored area of research. Previous work using OpenKG embeddings has primarily focused on canonicalization. CESI (Vashishth et al., 2018) uses KG embedding models for the canonicalization of noun phrases in OpenKGs. The problem of incorporating canonicalization information into OpenKG embeddings was addressed by Gupta et al. (2019). Their method for OpenKG embeddings (i.e., CaRE) performs better than Ontological KG embedding baselines in terms of link prediction performance. The challenges in the link prediction for OpenKGs were discussed in Broscheit et al. (2020), and methods similar to CaRE were proposed. In spirit, CaRE (Gupta et al., 2019) comes closest to our model; however, they do not address the problem of type compatibility in the link prediction task.

Entity Type: Entity typing is a popular problem where given a sentence and an entity mention, the goal is to predict *explicit* types of the entity. It has been an active area of research, and many models and datasets, such as (Mai et al., 2018), (Hovy et al., 2006), and (Choi et al., 2018), have been proposed. However, unlike this task, we aim to incorporate unsupervised **implicit** type information present in the pre-trained BERT model into OpenKG embeddings, rather than predicting *explicit* entity types present in ontologies or corpora.

For unsupervised cases, the problem of type compatibility in link prediction was addressed in (Jain et al., 2018). They employ a type compatibility score by learning a type vector for each NP and two type vectors (head and tail) for each relation. This score is multiplied with the triple score function, and the type vectors are trained jointly with embedding vectors. Although their method addresses the type compatibility issue, it is based on Ontological KG embedding models and shares the same limitations. In another work (Xie et al., 2016), hierarchical type information available in the dataset is incorporated while learning embeddings. However, their model is suitable only for Ontological KGs where the type information is readily available.

BERT in KG Embedding: BERT architecture has been used for scoring KG triples (Yao et al., 2019; Wang et al., 2019). However, their methods work on Ontological KGs without any explicit attention to NP types. In other work (Petroni et al., 2019), pre-trained BERT models are used for predicting links in KG. However, their focus was to evaluate knowledge present in the pre-trained BERT models instead of improving the existing link prediction model. BERT embeddings were also used for extracting entity type information (Chen et al., 2020). However, it was used for Entity Linking compared to OpenKG link prediction in our case.

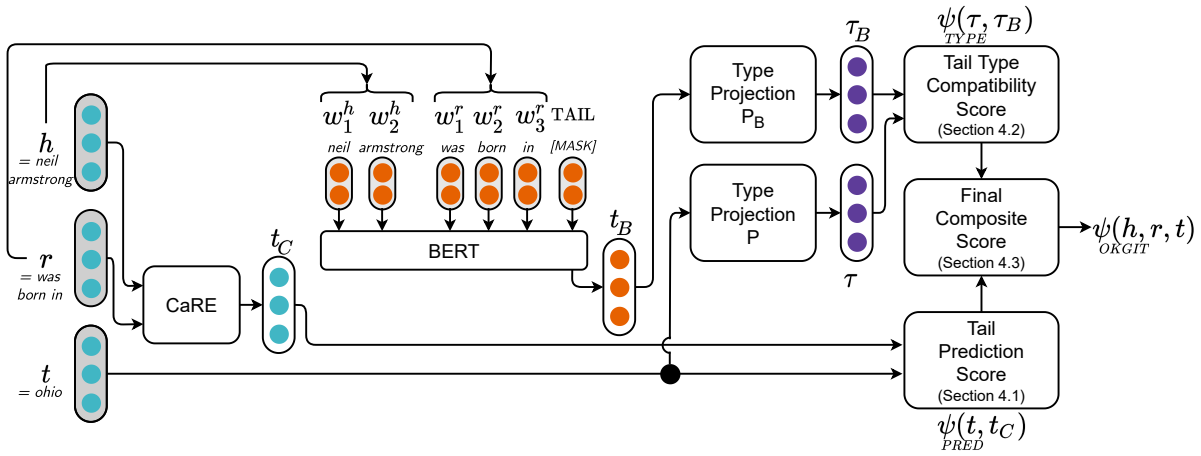

Figure 1: OKGIT Architecture. OKGIT learns embeddings for Noun Phrases (NP) and Relation Phrases (RP) present in an OpenKG by augmenting a standard tail prediction loss with type compatibility loss. Guidance for the tail type is obtained through type projection out of BERT's tail embedding prediction. In the figure, h, r, and t are the head NP, relation (RP), and tail NP. $h = w_1^h \dots w_{k_h}^h$ and $r = w_1^r \dots w_{k_r}^r$ are tokens in the head NP and relation, respectively. t_C and t_B are the tail NP vectors predicted by CaRE and BERT models (Please see Section 3 for background on these two models). Vectors τ_B and τ are the type vectors obtained using type projections P_B and P, respectively. ψ_{PRED} represents tail prediction score (Section 4.1) while ψ_{TYPE} represents type compatibility score (Section 4.2). ψ_{OKGIT} is the combined score generated by OKGIT for the input triple (h, r, t) (Section 4.3). Please refer to Section 4 for more details.

3 Background

We first introduce the notation used in this paper, followed by brief descriptions of BERT and CaRE.

Notation: An Open Knowledge Graph OpenKG = $(\mathcal{N}, \mathcal{R}, \mathcal{T})$ contains a set of noun phrases (NPs) $\mathcal{N}$, a set of relation phrases (RPs) $\mathcal{R}$ and a set of triples $(h, r, t) \in \mathcal{T}$ where $h, t \in \mathcal{N}$ and $r \in \mathcal{R}$. Here, h and t are called the head and tail NPs, and r is the RP between them. Each of them contains tokens from a vocabulary $\mathcal{V}$, specifically, $h = (w_1^h, w_2^h, \dots, w_{k_h}^h)$, $t = (w_1^t, w_2^t, \dots, w_{k_t}^t)$ and $r = (w_1^r, w_2^r, \dots, w_{k_r}^r)$. Here, k_h, k_r, and k_t are the numbers of tokens in the head NP, the relation, and the tail NP. OpenKG embedding methods learn vector representations for NPs and RPs. Specifically, vectors for an NP $e \in \mathcal{N}$ and an RP $r \in \mathcal{R}$ are represented by boldface letters $\mathbf{e} \in \mathbb{R}^{d_e}$ and $\mathbf{r} \in \mathbb{R}^{d_r}$. Here, d_e and d_r are dimensions of NP and RP vectors. Usually, $d_e = d_r$. A score function $\psi(h, r, t)$ represents the plausibility of a triple. Similarly, BERT represents tokens by d_B-dimensional vectors. A type projection matrix P takes the vectors to a common d_τ-dimensional type space $\mathbb{R}^{d_\tau}$. The vectors in the type space are denoted by τ.

BERT (Devlin et al., 2019): BERT is a bi-directional language representation model based on the transformer architecture (Vaswani et al., 2017), which has shown performance improvements across multiple NLP tasks. It is pre-trained on two tasks, (1) Masked Language Modeling (MLM), where the model is trained to predict randomly masked tokens from the input sentences, and (2) Next Sentence Prediction (NSP), where the model is trained to predict whether an input pair of sentences occurs in a sequence or not. In our case, we use a pre-trained BERT model (without fine-tuning) for predicting a masked tail NP in a triple.

CaRE (Gupta et al., 2019): CaRE is an OpenKG embedding method that can incorporate NP canonicalization information while learning the embeddings. NP canonicalization is the problem of grouping all surface forms of a given entity in one cluster, e.g., inferring that *Barack Obama*, *Barack H. Obama*, and *President Obama* all refer to the same underlying entity. CaRE consists of three components: (1) a canonicalization cluster encoder (CN), which generates NP embeddings by aggregating embeddings of canonical NPs from the corresponding cluster, (2) a bi-directional GRU based phrase encoder (PN), which encodes the tokens in RPs to generate RP embeddings, and (3) a base model, which is an Ontological KG embedding method like ConvE (Dettmers et al., 2018). It uses NP and

RP embeddings for scoring triples. These triple scores are then fed to a loss function (e.g., pairwise ranking loss with negative sampling (Bordes et al., 2013) or binary cross-entropy loss (BCE) (Dettmers et al., 2018)). In this paper, we use CaRE with ConvE as the base model. This model generates a candidate tail NP vector for a given NP h and RP r, denoted by CaRE($\mathbf{h}$, $\mathbf{r}$).

4 OKGIT: Our Proposed Method

Motivation: As illustrated in Table 1, top NPs predicted by CaRE may not always be type compatible with the input query. On the other hand, BERT's top predictions are usually type compatible (Chen et al., 2020), although they may not be factually correct. Thus, we hypothesize that a combination of these two models can produce correct as well as type compatible predictions. Motivated by this, we develop OKGIT, which combines the best of both of these models. The complete architecture of the proposed model can be found in Figure 1. In the following section, we present various components of the proposed model.

4.1 ψ_{PRED}: Tail Prediction Score

The correctness of tail prediction in a triple is measured by the triple score function ψ_{PRED}. Given a triple (h, r, t), it uses the corresponding vectors ($\mathbf{h}$, $\mathbf{r}$, $\mathbf{t}$) and assigns high scores to correct triples and low scores to incorrect triples. We follow CaRE (Gupta et al., 2019) for scoring triples, which internally uses ConvE (Dettmers et al., 2018) as the base model. For a given triple (h, r, t), the CaRE model first predicts a tail NP vector $\mathbf{t_C}$ as

$$\mathbf{t_C} = \text{CaRE}(\mathbf{h}, \mathbf{r}) \tag{1}$$

The predicted tail NP vector $\mathbf{t_C}$ is then matched against the given tail NP vector $\mathbf{t}$ using dot product to generate the triple score ψ_{PRED}.

$$\psi_{\text{PRED}}(\mathbf{t}, \mathbf{t_C}) = \mathbf{t_C}^\top \mathbf{t}. \tag{2}$$

The score ψ_{PRED} represents tail prediction correctness, and CaRE model uses only this score.

4.2 ψ_{TYPE}: Tail Type Compatibility Score

The type compatibility between a given (head NP, RP) pair and a tail NP is measured by the type compatibility score function ψ_{TYPE}. It assigns a high score when an NP t has suitable types as candidate tail NP for given head NP h and RP r. We employ a Masked Language Model (MLM) for measuring type compatibility, specifically BERT (Devlin et al., 2019). Following (Petroni et al., 2019), we can generate a candidate tail NP vector using BERT. Specifically, given a triple (h, r, t), we replace the head NP h and RP r with their tokens and tail NP t with a special MASK token. The resulting sentence $(w_1^h, \ldots, w_{k_h}^h, w_1^r, \ldots, w_{k_r}^r, \text{MASK})$ is sent as input to the BERT model. We denote the output vector from BERT corresponding to the MASK tail token as $\mathbf{t_B}$.

$$\mathbf{t_B} = \text{BERT}(h, r, \text{MASK}) \tag{3}$$

We can predict tail NPs for a given (h, r) by finding the nearest neighbors of $\mathbf{t_B}$ from the BERT vocabulary (Appendix D). These predicted NPs may not be the correct tail NP present in KG; however, they tend to be type compatible with the given (h, r) pair.

Motivated by this, we extract the implicit NP type information from this vector using a type projector $P_B \in \mathbb{R}^{d_\tau \times d_B}$. The output vector from BERT t_B is high-dimensional and can be used as a proxy for NP's type (Chen et al., 2020). Therefore, P_B projects the $\mathbf{t_B}$ vector to a lower dimensional space such that only relevant information is retained. We do a similar operation on tail NP embedding $\mathbf{t}$ and use a type projector $P \in \mathbb{R}^{d_\tau \times d_e}$ to extract type information. Both P_B and P are trained jointly with the model. Thus, the type vectors are given by

$$\boldsymbol{\tau_B} = P_B \mathbf{t_B} \quad \text{and} \quad \boldsymbol{\tau} = P\mathbf{t} \tag{4}$$

for BERT and CaRE, respectively. Here, both $\boldsymbol{\tau_B}, \boldsymbol{\tau} \in \mathbb{R}^{d_\tau}$. Then, the type compatibility score between these can be measured by negative of Euclidean distance, i.e.,

$$\psi_{\text{TYPE}}(\boldsymbol{\tau}, \boldsymbol{\tau_B}) = -||\boldsymbol{\tau_B} - \boldsymbol{\tau}||_2^2.$$

We also experimented with a dot product version of the type score, $\psi_{\text{TYPE}}^{\text{Dot}}(\boldsymbol{\tau}, \boldsymbol{\tau_B}) = \boldsymbol{\tau_B}^\top \boldsymbol{\tau}$, and found its performance to be comparable to the Euclidean distance version. Therefore, we use the Euclidean distance version for all our experiments.

4.3 ψ_{OKGIT}: Final Composite Score

The score functions ψ_{PRED} and ψ_{TYPE} may contain complementary information. Therefore, we use a combination of triple and type compatibility scores as final score for a given triple.

$$\psi_{\text{OKGIT}}(h, r, t) = \psi_{\text{PRED}}(\mathbf{t}, \mathbf{t_C}) + \gamma \times \psi_{\text{TYPE}}(\boldsymbol{\tau}, \boldsymbol{\tau_B}). \tag{5}$$

Dataset	#NPs	#RPs	#Gold Clusters	#Average NPs Per Cluster
ReVerb20K	11,064	11,057	10,897	1.02
ReVerb45K	27,007	21,622	18,626	1.45
ReVerb20KF	3,524	6,076	3,406	1.03
ReVerb45KF	9,400	11,249	6,749	1.39

	#Train	#Validation	#Test
ReVerb20K	15,498	1,549	2,324
ReVerb45K	35,969	3,597	5,394
ReVerb20KF	6,685	1,015	1,517
ReVerb45KF	14,775	1,781	2,650

Table 2: Dataset Statistics. Please refer to Section 5 for more details.

Please recall that $\mathbf{t_C}$ and $\boldsymbol{\tau_B}$ are in turn dependent on h and r ((1) and (3)), while $\boldsymbol{\tau}$ is dependent on t (4). Here, γ controls the relative weights given to individual scores. This final score takes care of both, i.e., triple correctness as well as type compatibility. For training, we feed the sigmoid of this score function to the Binary Cross Entropy (BCE) loss function following (Dettmers et al., 2018).

4.4 Learning with Type Regularization

Let $\mathcal{X} = \{(h_i, r_i) | (h_i, r_i, t_i) \in \mathcal{T} \text{for some } t_i \in \mathcal{N}\}$ be the set of all head NPs and RPs which appear in the OpenKG. Let y_i be the label for the triple (h_i, r_i, t_i) which is 1 if $(h_i, r_i, t_i) \in \mathcal{T}$ and 0 otherwise. We apply the logistic sigmoid function σ on score ψ_{OKGIT} to get the predicted label

$$\hat{y}_i = \sigma(\psi_{\text{OKGIT}}(h_i, r_i, t_i))$$

Finally, we use the following binary cross-entropy (BCE) loss for triple correctness.

$$\text{TripleLoss}(h_i, r_i, t_i) = y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)$$

To further reinforce the type compatibility in the model, we include an additional loss term which forces the type vectors of correct triples to be closer in the type space. Similar to TripleLoss, we use the binary cross-entropy loss for type regularization as well. The type regularization term is shown below.

$$\text{TypeLoss}(h_i, r_i, t_i) = y_i \cdot \log(\hat{p}_i) + (1 - y_i) \cdot \log(1 - \hat{p}_i)$$

where $\hat{p} = \sigma(\psi_{\text{TYPE}}(\boldsymbol{\tau}, \boldsymbol{\tau_B}))$. The cumulative loss function is then given as below.

$$\sum_{i=1}^{n} \text{TripleLoss}(h_i, r_i, t_i) + \lambda \times \text{TypeLoss}(h_i, r_i, t_i) \tag{6}$$

where n is the number of training instances. We consider $\mathcal{X} \times \mathcal{N}$ as our training data where triples present in $\mathcal{T}$ have label 1 and rest have label 0.

Dataset	$d_e = d_r$	d_τ	λ	γ	BERT model
ReVerb20K	300	300	0.01	5.0	large
ReVerb45K	300	100	0.0	2.0	large
ReVerb20KF	300	300	0.001	5.0	base
ReVerb45KF	300	300	0.001	0.25	base

Table 3: Optimal Hyperparameter values. Please refer to Section 5 for more details.

5 Experiments

Datasets: Following (Gupta et al., 2019), we use two subsets of English OpenKGs created using ReVerb (Fader et al., 2011), namely ReVerb20K and ReVerb45K. We follow the same train-validation-test split for these datasets. As noted in (Petroni et al., 2019), predicting multi-token NPs using BERT could be challenging and it might require special pre-training (Joshi et al., 2020). To understand this difference, we create filtered subsets of these datasets such that they contain only single token NPs [1]. Specifically, we create ReVerb20KF (ReVerb20K-Filtered) and ReVerb45KF (ReVerb45K-Filtered) which contain only single token NPs. More details about these datasets can be found in Table 2.

Setup and hyperparameters: We use d_e = d_r = 300 for NP and RP vectors. For other hyperparameters, we use grid-search and select the model based on MRR on validation split. For type vectors, we select d_τ from {100, 300, 500}. The weight for type regularization term λ is selected from the range $\{10^{-3}, 10^{-2} \ldots, 10^{1}\} \cup \{0\}$. Type composition weight γ is selected from {0.25, 0.5, 1.0, 2.0, 5.0}. For the language model, we try both BERT-base as well as BERT-large. The optimal values for hyperparameters are shown in Table 3. The experiments run for 1.5 hours (for filtered subsets) and 9 hours (for full datasets) on GeForce GTX 1080 Ti GPU.

6 Results

We evaluate the proposed model on the link prediction task. We follow the same evaluation process as in (Gupta et al., 2019). From our experiments, we try to answer the following questions:

1. Is OKGIT effective in the link prediction task? (Section 6.1)

[1] Please note that the single-token limitation is only valid for BERT, not for OKGIT (Appendix D).

Model	ReVerb20K					ReVerb45K				
	MRR(%)↑	MR↓	Hits(%)↑			MRR(%)↑	MR↓	Hits(%)↑		
			@1	@3	@10			@1	@3	@10
ConvE (Dettmers et al., 2018)	26.2	2177.0	20.2	29.1	36.3	18.4	6625.0	13.3	20.6	28.3
CaRE (Gupta et al., 2019)	30.6	851.1	24.4	33.1	41.7	32.0	1276.8	25.3	35.0	44.6
CaRE [BERT initialization]	31.6	837.0	24.8	35.0	44.2	31.2	925.5	24.2	34.4	44.3
OKGIT [Our model]	**35.9**	**527.1**	**28.2**	**39.4**	**49.9**	**33.2**	**773.9**	**26.1**	**36.3**	**46.4**
Model	**ReVerb20KF**					**ReVerb45KF**				
	MRR(%)↑	MR↓	Hits(%)↑			MRR(%)↑	MR↓	Hits(%)↑		
			@1	@3	@10			@1	@3	@10
BERT (Devlin et al., 2019)	4.9	1116.5	2.2	5.0	9.7	18.9	536.5	12.3	20.8	32.5
ConvE (Dettmers et al., 2018)	22.3	836.6	16.1	25.5	33.4	16.5	2398.1	10.9	18.9	27.6
CaRE (Gupta et al., 2019)	29.3	308.3	22.1	31.6	43.2	26.6	692.7	20.1	28.8	39.1
CaRE [BERT initialization]	31.8	207.6	24.2	34.8	46.2	24.9	557.3	17.8	27.6	38.3
OKGIT [Our model]	**34.6**	**214.7**	**26.5**	**38.0**	**50.2**	**29.7**	**500.2**	**22.5**	**32.4**	**43.3**

Table 4: Results of link prediction task. Here ↑ indicates higher values are better while ↓ indicates lower values are better. We can see that the OKGIT model outperforms the baseline models on all the datasets (Section 6.1).

2. Does OKGIT generate more type compatible NPs in link prediction? (Section 6.2)

3. Is the Type Projector effective in extracting type vectors from embeddings? (Section 6.3)

6.1 Effectiveness of OKGIT Embeddings in Link Prediction

We evaluate our model on the link prediction task. Given a held-out triple (h_i, r_i, t_i), all the NPs $e \in \mathcal{N}$ in the KG are ranked as candidate tail NP based on their score $\psi_{\text{OKGIT}}(h_i, r_i, e)$. Let the rank of the correct tail NP t be denoted by $rank_i^t$. Similarly, ranks are also calculated for predicting head NPs instead of tail NPs using inverse relations (Dettmers et al., 2018; Gupta et al., 2019); let it be denoted by $rank_i^h$. These ranks are then used to find Mean Reciprocal Rank (MRR), Mean Rank (MR) and Hit@k (k=1,3,10) as follows.

$$\text{MRR} = \frac{1}{2 \times n_{test}} \sum_{i=1}^{n_{test}} \left(\frac{1}{rank_i^h} + \frac{1}{rank_i^t} \right),$$

$$\text{MR} = \frac{1}{2 \times n_{test}} \sum_{i=1}^{n_{test}} \left(rank_i^h + rank_i^t \right), \text{and}$$

$$\text{Hits@k} = \sum_{i=1}^{n_{test}} \frac{\mathbb{1}(rank_i^h \leq k) + \mathbb{1}(rank_i^t \leq k)}{2 \times n_{test}}.$$

Here, n_{test} is the number of test triples and $\mathbb{1}$ is the indicator function. As noted in (Gupta et al., 2019), ranking individual NPs is not suitable for OpenKGs due to the lack of canonicalization. Hence, following their approach, we rank gold canonicalization clusters instead of individual NPs. The gold canonicalization partitions the NPs into clusters such that NPs mentioning the same entity belong to the same cluster. For ranking these clusters, we first find ranks of all NPs $e \in \mathcal{N}$. Then for each cluster, we keep the NP with minimum rank as representative and discard others. The representative NPs are then ranked again and the new ranks are assigned to the corresponding clusters. The rank of the cluster containing the true NP is then used for evaluating the performance. For better readability, the MRR and Hits@k metrics have been multiplied by 100.

We compare OKGIT with BERT (MLM), ConvE (Ontological KGE) and CaRE (OpenKGE). We also compare against a version of CaRE where phrase embeddings have been initialized with BERT (CaRE [BERT initialization]). As we can see from the results in Table 4, the proposed model OKGIT outperforms baseline methods in link prediction task across all datasets. This suggests that the implicit type scores from BERT help in improving ranks of correct NPs. Moreover, OKGIT outperforms CaRE with BERT initialization, suggesting the importance of type projectors [2].

The performance gain is higher for ReVerb20K and ReVerb20KF (+5.3 MRR) than ReVerb45K and ReVerb45KF (+1.2 and +3.1 MRR) datasets. As we can see from Table 2, the number of NPs are very close to the number of gold clusters in the 20K

[2]Please refer to Appendix A for a detailed comparison.

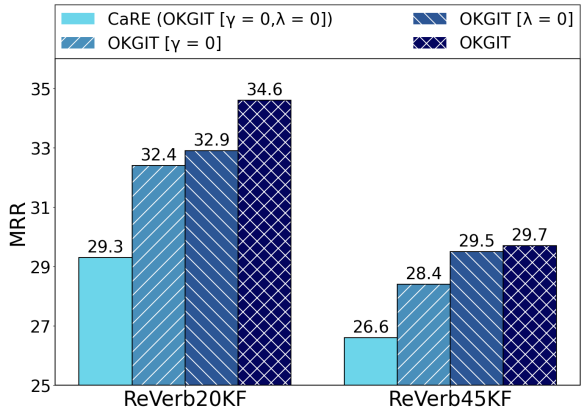

Figure 2: Effect of type compatibility score and type regularization on link prediction performance. While the type compatibility score with $\lambda = 0$ gives better gains in MRR (11%-12%) than type regularization term with $\gamma = 0$ (7%-11%), the combined model performs the best, achieving 12%-18% gains in MRR (Section 6.1).

datasets. Thus, the canonicalization information is slightly weaker in the 20K datasets than the 45K datasets. Due to this, CaRE achieved better gains in the ReVerb45K dataset as noted in (Gupta et al., 2019). This leaves more scope of improvements in the 20K datasets. By including the type information from BERT, OKGIT is able to fill this gap. It achieves better gains in the 20K datasets and is able to alleviate the lack of canonicalization information. Moreover, OKGIT is able to improve ranks of correct NPs ranked lower by CaRE. This can be seen by significant improvements in the MR.

Other Language Models: Using RoBERTa instead of BERT results in similar performance improvements (Appendix B). However, our primary focus is to understand the impact of *implicit type information* present in pre-trained MLMs, such as BERT, and *not* to compare multiple MLMs themselves.

Ablations: We perform ablation experiments to compare the relative importance of type compatibility score ψ_{TYPE} and type regularization term. We evaluate OKGIT with disabled type compatibility score (i.e., $\gamma = 0$ in Equation (5)) and disabled type regularization term (i.e., $\lambda = 0$ in Equation (6)) separately. Please note that CaRE model is equivalent to OKGIT with $\gamma = 0$ and $\lambda = 0$. The results of this experiment are shown in Figure 2. We find that while type compatibility score gives more performance gain (11%-12% gain in MRR) than type regularization (7%-11% gain in MRR), the combined model achieves the best performance (12%-18% gain in MRR). It suggests that both the components are important. Please refer to the Appendices A, B, C for more ablation experiments.

Dataset	CaRE	OKGIT
ReVerb20KF	0.23	**0.30**
ReVerb20K	0.35	**0.36**
ReVerb45KF	0.22	**0.31**
ReVerb45K	0.34	**0.35**

Table 5: Results of type evaluation in CaRE and OKGIT predictions. We find that OKGIT performs better than CaRE in all datasets in terms of F1-score. Also, the results are statistically significant for all the datasets (Section 6.2).

6.2 Type Compatibility in Predicted NPs

As noted in (Chen et al., 2020), BERT vectors contain NP type information[3]. OKGIT utilizes this type information for improving OpenKG link prediction. In this section, we evaluate whether OKGIT improves upon CaRE in predicting type compatible NPs. For such an evaluation, we require type annotations for the NPs in the OpenKGs. However, OpenKGs do not have an underlying ontology or explicit gold NP type annotations, making a direct evaluation impossible. Therefore, we employ a pre-trained entity typing model UFET (Choi et al., 2018). Given a sentence and an entity mention, the entity typing model predicts the mentioned entity's types. Using this model, we obtain types for true as well as predicted NPs by CaRE and OKGIT and use it for the evaluation. Please note that this evaluation is limited to the coverage and quality of the UFET model.

Evaluation Protocol: The type vocabulary in UFET model contains $10,331$ types including 9 general, 121 fine-grained, and $10,201$ ultra-fine types. The model takes a sentence $(w_1^h, \ldots, w_{k_h}^h, w_1^r, \ldots, w_{k_r}^r, w_1^t, \ldots, w_{k_t}^t)$ formed from a triple (h, r, t) along with an entity mention (either t or h) as inputs and outputs a distribution over types. We use the top five predicted types for our experiments[4]. For a triple (h, r, t), we consider the types predicted for the true tail NP t as true types $\Gamma(t)$. Let $\hat{t}_{CaRE}$ and $\hat{t}_{OKGIT}$ be the top predicted tail NP by CaRE and OKGIT for the (h, r) pair. Then the types $\Gamma(\hat{t}_{CaRE})$ predicted for $\hat{t}_{CaRE}$

[3]We also verify this using Freebase, an ontological KG. Please refer to the Appendix G for more details.

[4]We observe similar behaviour with top one and three types.

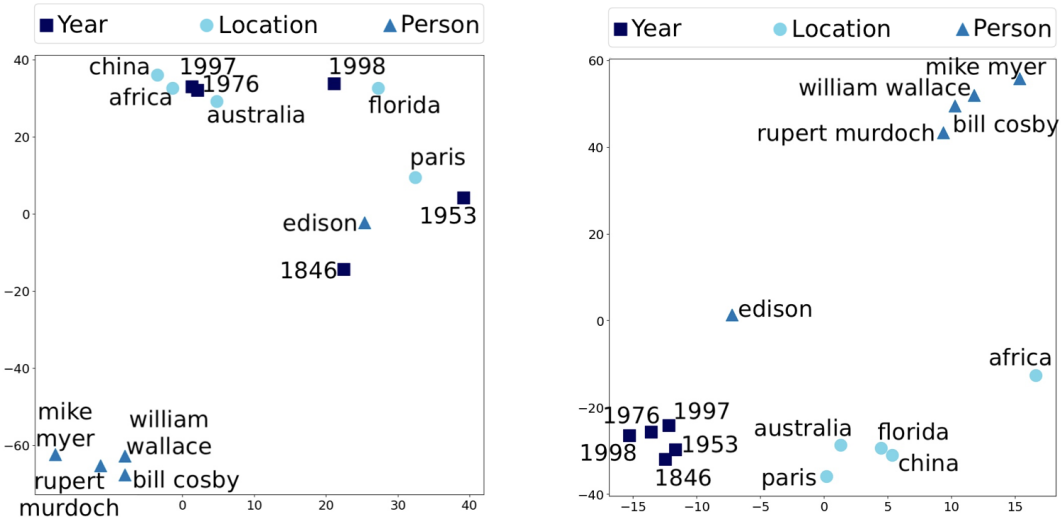

Figure 3: t-SNE projections of tail NP embeddings (left) and type vectors (right) extracted by the Type Projector from tail NP embeddings (Section 4.2) in the ReVerb20K dataset. We find that the Type Projector is able to extract informative type vectors from the tail embeddings. This is evident from the fact that the tail embeddings corresponding to person, location, and dates were inter-mixed in the left plot, while they have been separated into type specific clusters in the right plot. Please see Section 6.3 for details.

in the triple $(h, r, \hat{t}_{CaRE})$ is used as predicted types for CaRE. Similarly the types $\Gamma(\hat{t}_{OKGIT})$ predicted for $\hat{t}_{OKGIT}$ in the triple $(h, r, \hat{t}_{OKGIT})$ are used as predicted types for OKGIT. For evaluation, we calculate the mean F1-score as follows [5]

$$\text{F1} = \frac{2}{n_{test}} \sum_{i=1}^{n_{test}} \frac{|\Gamma(t_i) \cap \Gamma(\hat{t}_i)|}{|\Gamma(\hat{t}_i)| + |\Gamma(t_i)|}.$$

Here, $|\Gamma(t)|$ denotes the number of types present in $\Gamma(t)$ and $\hat{t}$ represents $\hat{t}_{CaRE}$ or $\hat{t}_{OKGIT}$. We can obtain the F1-scores for head NP similarly. We evaluate the mean F1-scores across head and tail NP prediction tasks on the test data and compare CaRE with OKGIT.

As we can see from the results in Table 5, OKGIT performs better than CaRE, suggesting that OKGIT generates more type compatible NPs than CaRE in the link prediction task. OKGIT achieves higher gains in the single-token datasets (i.e., ReVerb20KF and ReVerb45KF) than multi-token dataset (i.e., ReVerb20K and ReVerb45K). Upon investigation, we found that the types obtained using the entity typing model (true as well as predicted) for the multi-tokens datasets often contain common noisy types, leading to the small difference between CaRE and OKGIT. Following Dror et al. (2018), we also check the results for statistical significance using Permutation, Wilcoxon, and t-test with $\alpha = 0.05$, and found it to be significant for all the datasets.

6.3 Effectiveness of Type Projector

To better understand the effect of type projection, we visualize the vectors in NP-space from CaRE and Type-space (i.e., after type projection) from OKGIT. For this experiment, we randomly select 5 NPs from 3 categories, namely Person, Location and Year. More details about this selection process can be found in the Appendix E. We project the NP vectors (i.e., $\mathbf{t}$) corresponding to these NPs to a 2-dimensional NP-space using t-SNE (Maaten and Hinton, 2008)[6]. Similarly, we also project the corresponding type vectors (i.e., τ) to 2-dimensional Type-space. We plot the resulting vectors, color and shape coded by their respective categories, in Figure 3.

We can see that the vectors from different categories in the NP-space are mixed. However, after the type projection, the vectors in the Type-space are clustered together based on their categories.

6.4 Qualitative Evaluations

In this section, we present some examples of predictions made by CaRE and OKGIT methods. The result is shown in Table 6. As we see in Triple-1, both CaRE and OKGIT predict the correct NP (i.e.,

[5] Since we use a fixed number of types for ground truth and predictions, precision, recall, and F1-score have the same values. Therefore, we only report the F1-score.

[6] We run t-SNE for 2000 iterations with 15 perplexity.

	Triples	CaRE	OKGIT
1.	(*bach*, *moved to*, *?*)	*leipzig* *mobile* *vladimir h.* *horowitz* *yo yo*	*leipzig* *vienna* *stockholm* *sweden* *turin*
2.	(*clinton*, *lead by*, *?*)	*purchase* *sale* *video* *movie* *discount*	*1500* *260* *80* *99* *hire*

Table 6: Few example predictions made by CaRE and OKGIT models. We observe that the OKGIT predictions are more type compatible with the query. Please refer to Section 6.4 for more details.

leipzig) on top. However, more predictions from OKGIT are type compatible (i.e., all are locations) to the input query. On the other hand, CaRE predictions have mixed types (i.e., location, person, etc.). Also, CaRE makes an incorrect prediction, *vladimir horowitz*, possibly due to the presence of a training triple (*vladimir horowitz*, *had a great affinity for*, *bach*).

We see similar patterns in Triple-2, where the correct tail NP should be of type *number* indicating the count of votes. OKGIT is able to predict numbers in top predictions for Triple-2, while CaRE has mixed types in top predictions.

7 Conclusion

The task of link prediction for Open Knowledge Graphs (OpenKG) has been a relatively underexplored research area. Previous work on OpenKG embeddings has primarily focussed on improving or incorporating NP canonicalization information. While there are few methods for OpenKG link prediction, they often predict noun phrases with types incompatible with the query noun and relation phrases. Therefore, we use implicit type information from BERT to improve OpenKG link prediction and propose OKGIT. With the help of novel type compatibility score and type regularization term, OKGIT achieves significant performance improvement on the link prediction task across multiple datasets. We also find that OKGIT produces more type compatible predictions than CaRE, evaluated using an external entity typing model.

Acknowledgments

We thank the anonymous reviewers for their constructive comments. This work is supported by the Ministry of Human Resource Development (Government of India).

Broader Impact

OKGIT is the first attempt towards incorporating implicit type information in OpenKG link prediction without human intervention. It will greatly benefit densification and applications of OpenKGs where no underlying ontologies are available.

However, OKGIT predictions depend on various datasets, i.e., the corpus used for training the masked language model (e.g., BERT) and the corpus from which the OpenKG triples were extracted. A potential, possibly undesirable, bias may be introduced in the predictions by manipulating these corpora or adding a large number of malicious triples in the OpenKG.

We have tested OKGIT in English datasets. While the overall model architecture is independent of the language, the model's effectiveness might vary depending upon the quality of the masked language model, and it needs to be tested.

References

Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. 2007. Open information extraction from the web. In *IJCAI*, IJCAI'07, page 2670–2676, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *ACM SIGMOD*, pages 1247–1250. AcM.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Neural Information Processing Systems (NeurIPS)*, pages 1–9.

Samuel Broscheit, Kiril Gashteovski, Yanjie Wang, and Rainer Gemulla. 2020. Can we predict new facts with open knowledge graph embeddings? a benchmark for open link prediction. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2296–2308, Online. Association for Computational Linguistics.

Shuang Chen, Jinpeng Wang, Feng Jiang, and Chin-Yew Lin. 2020. Improving entity linking by modeling latent entity type information. In *Proceedings of*

the AAAI Conference on Artificial Intelligence, volume 34, pages 7529–7537.

Eunsol Choi, Omer Levy, Yejin Choi, and Luke Zettlemoyer. 2018. Ultra-fine entity typing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 87–96, Melbourne, Australia. Association for Computational Linguistics.

Tim Dettmers, Minervini Pasquale, Stenetorp Pontus, and Sebastian Riedel. 2018. Convolutional 2d knowledge graph embeddings. In *Proceedings of the 32th AAAI Conference on Artificial Intelligence*, pages 1811–1818.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. ACL.

Rotem Dror, Gili Baumer, Segev Shlomov, and Roi Reichart. 2018. The hitchhiker's guide to testing statistical significance in natural language processing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1383–1392. Association for Computational Linguistics.

Anthony Fader, Stephen Soderland, and Oren Etzioni. 2011. Identifying relations for open information extraction. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Swapnil Gupta, Sreyash Kenkre, and Partha Talukdar. 2019. CaRe: Open knowledge graph embeddings. In *EMNLP-IJCNLP*, pages 378–388, Hong Kong, China. ACL.

Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. OntoNotes: The 90% solution. In *NAACL-HLT, Companion Volume: Short Papers*, pages 57–60, New York City, USA. ACL.

Zhipeng Huang, Bogdan Cautis, Reynold Cheng, and Yudian Zheng. 2016. Kb-enabled query recommendation for long-tail queries. In *CIKM*, CIKM '16, pages 2107–2112, New York, NY, USA. ACM.

Prachi Jain, Pankaj Kumar, Mausam, and Soumen Chakrabarti. 2018. Type-sensitive knowledge base inference without explicit type supervision. In *ACL (Volume 2: Short Papers)*, pages 75–80, Melbourne, Australia. ACL.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *JMLR*, 9(Nov):2579–2605.

Khai Mai, Thai-Hoang Pham, Minh Trung Nguyen, Tuan Duc Nguyen, Danushka Bollegala, Ryohei Sasano, and Satoshi Sekine. 2018. An empirical study on fine-grained named entity recognition. In *COLING*, pages 711–722, Santa Fe, New Mexico, USA. ACL.

Mausam Mausam. 2016. Open information extraction systems and downstream applications. In *Proceedings of the twenty-fifth international joint conference on artificial intelligence*, pages 4074–4077.

Madhav Nimishakavi, Uday Singh Saini, and Partha Talukdar. 2016. Relation schema induction using tensor factorization with side information. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 414–423, Austin, Texas. Association for Computational Linguistics.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? In *EMNLP-IJCNLP*, pages 2463–2473, Hong Kong, China. ACL.

Shikhar Vashishth, Prince Jain, and Partha P. Talukdar. 2018. CESI: canonicalizing open knowledge bases using embeddings and side information. In *WWW 2018, Lyon, France, April 23-27, 2018*, pages 1317–1327.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *NeurIPS*, pages 5998–6008. Curran Associates, Inc.

Quan Wang, Pingping Huang, Haifeng Wang, Songtai Dai, Wenbin Jiang, Jing Liu, Yajuan Lyu, Yong Zhu, and Hua Wu. 2019. COKE: Contextualized Knowledge Graph Embedding. *arXiv preprint arXiv:1911.02168*.

Ruobing Xie, Zhiyuan Liu, and Maosong Sun. 2016. Representation learning of knowledge graphs with hierarchical types. In *IJCAI*, IJCAI'16, page 2965–2971. AAAI Press.

Liang Yao, Chengsheng Mao, and Yuan Luo. 2019. KG-BERT: BERT for Knowledge Graph Completion. *ArXiv*, abs/1909.03193.

Xuchen Yao and Benjamin Van Durme. 2014. Information extraction over structured data: Question answering with Freebase. In *ACL (Volume 1: Long Papers)*, pages 956–966, Baltimore, Maryland. ACL.

Appendices

A BERT Initialization vs Type Projectors

Here, we demonstrate the importance of type projectors by comparing OKGIT with multiple BERT-augmented versions of CaRE. Specifically, we initialize the phrase and word embeddings in CaRE with a pre-trained BERT model. The phrase (word) is passed as input to BERT and the output corresponding to the $[CLS]$ token is then used for initializing phrase (word) embedding for CaRE model. This modified CaRE model is trained similar to the base CaRE model. Based on different initialization methods, we experiment with following baselines.

CaRE [BERT NP]: NP embeddings are initialized using BERT and the rest of the model is same as CaRE. This model uses 768 (for BERT-base) or 1024 (for BERT-large) dimensional vectors.

CaRE [BERT NP+PROJ]: Since CaRE [BERT NP] uses higher dimensional vectors (768 or 1024) compared to other methods (300), the comparison may not be fair. To address this issue, we project BERT embeddings to 300 dimension. The projection is trained with the rest of the model.

CaRE [BERT NP+RP]: We initialize NP embeddings as well as the word embeddings in RP encoder using BERT embeddings. This method also uses 768 or 1024 dimensional vectors[7].

In all the methods, including OKGIT, we never fine-tune BERT, as our goal is to evaluate the type information already present in pre-trained BERT model. We experiment with both, BERT-base and BERT-large, and report the best performing model.

As we can see from the results in Table 7, OKGIT outperforms these baselines. Although BERT initialization improves the performance of CaRE model, the usage of explicit type-score and type regularization leads to significant performance improvements, suggesting their importance.

B Replacing BERT with other operations

In this section, we evaluate whether BERT module in OKGIT can be replaced by simple operations such as vector addition and concatenation. Specifically, we modify $\mathbf{t_B}$ in Equation (3) by replacing BERT with these operations leading to the following variants of OKGIT.

OKGIT-C: BERT is replaced by concatenation of head NP vector $\mathbf{h}$ and relation phrase vector $\mathbf{r}$

$$\mathbf{t_B} = [\mathbf{h}; \mathbf{r}].$$

OKGIT-A: BERT is replaced by vector addition

$$\mathbf{t_B} = \mathbf{h} + \mathbf{r}.$$

OKGIT-R: We also experiment with another masked language model RoBERTa in place of BERT.

$$\mathbf{t_B} = \text{RoBERTa}(h, r, \text{MASK}).$$

For this experiment, we use the ReVerb20KF and ReVerb45KF datasets as representatives. We perform grid-search with similar hyper-parameters as in Section 5 of the main paper and select the best model based on the MRR on the validation split. The results are reported in Table 8.

As we can see from the results, the OKGIT-C and OKGIT-A perform very similar to CaRE on both datasets. This suggests that the performance gains for OKGIT come from the BERT module. This observation is further reinforced because OKGIT-R results in similar improvements compared to CaRE as OKGIT. However, in all cases, we find that OKGIT with BERT outperforms other model variants.

C CaRE with Entity Typing

Entity typing is the task of predicting explicit types of an entity given a sentence and its mention. As we are interested in improving type compatibility of predictions in the link prediction task, we can also incorporate the output from an entity typing model. In this section, we explore this setting by replacing the BERT module in OKGIT with an entity typing model UFET from (Choi et al., 2018). Specifically, we replace the vector t_B in Equation (3) with the output of UFET representing the predicted probability distribution over types.

OKGIT(UFET) Model: The UFET model takes a sentence and an entity mention as input and produces a distribution over explicit set of types. In our case, the sentence is formed by concatenating the subject NP, relation phrase, and object NP, while the object NP is used as mention. The output distribution from UFET is used as t_B in our model. We call this version of the model as OKGIT(UFET) and compare it CaRE and OKGIT.

We run a grid-search for finding the best hyper-parameter similar to Section 5 and report the results

[7] we also tried using pre-trained BERT as RP encoder in CaRE, however, it performed poorly due to fixed RP encoder.

Model	ReVerb20K					ReVerb45K				
	MRR(%)↑	MR↓	Hits(%)↑			MRR(%)↑	MR↓	Hits(%)↑		
			@1	@3	@10			@1	@3	@10
CaRE	30.6	851.1	24.4	33.1	41.7	32.0	1276.8	25.3	35.0	44.6
CaRE [BERT-L NP]	31.6	837.0	24.8	35.0	44.2	31.2	925.5	24.2	34.4	44.3
CaRE [BERT NP+PROJ]§	27.4	950.2	21.9	29.2	38.0	30.7	952.8	23.0	34.4	45.5
CaRE [BERT-L NP+RP]	30.9	862.4	24.6	33.5	42.6	32.8	1015.6	25.9	35.9	45.6
OKGIT [Our model]	**35.9**	**527.1**	**28.2**	**39.4**	**49.9**	**33.2**	**773.9**	**26.1**	**36.3**	**46.4**

Model	ReVerb20KF					ReVerb45KF				
	MRR(%)↑	MR↓	Hits(%)↑			MRR(%)↑	MR↓	Hits(%)↑		
			@1	@3	@10			@1	@3	@10
CaRE	29.3	308.3	22.1	31.6	43.2	26.6	692.7	20.1	28.8	39.1
CaRE [BERT-B NP]	31.8	207.6	24.2	34.8	46.2	24.9	557.3	17.8	27.6	38.3
CaRE [BERT-L NP+PROJ]	27.6	258.6	21.0	29.1	40.7	24.7	600.5	17.4	27.4	39.2
CaRE [BERT-L NP+RP]	30.1	289.3	22.7	32.8	43.3	26.8	562.5	19.8	29.7	39.8
OKGIT [Our model]	**34.6**	**214.7**	**26.5**	**38.0**	**50.2**	**29.7**	**500.2**	**22.5**	**32.4**	**43.3**

Table 7: Results of the link prediction task. Here ↑ indicates higher values are better while ↓ indicates lower values are better. We can see that the OKGIT model outperforms the baseline models on all the datasets (Appendix A). Here, BERT-B and BERT-L denote BERT-base and BERT-large respectively. §For NP+PROJ models, BERT-large performs best for ReVerb20K, while BERT-base performs best for ReVerb45K.

Model	ReVerb20KF					ReVerb45KF				
	MRR(%)↑	MR↓	Hits(%)↑			MRR(%)↑	MR↓	Hits(%)↑		
			@1	@3	@10			@1	@3	@10
CaRE (Gupta et al., 2019)	29.3	308.3	22.1	31.6	43.2	26.6	692.7	20.1	28.8	39.1
OKGIT-C [$\mathbf{t_B} = [\mathbf{h}; \mathbf{r}]$]	30.0	309.3	22.9	32.4	43.8	27.1	666.5	20.2	29.8	39.9
OKGIT-A [$\mathbf{t_B} = \mathbf{h} + \mathbf{r}$]	30.4	331.7	23.5	32.9	43.6	27.1	660.5	19.9	30.6	40.2
OKGIT-R [RoBERTa]	32.7	221.0	25.3	35.1	46.5	29.0	596.7	21.8	32.0	43.0
OKGIT [Our model]	**34.6**	**214.7**	**26.5**	**38.0**	**50.2**	**29.7**	**500.2**	**22.5**	**32.4**	**43.3**

Table 8: Results of the ablation experiments. We replace the BERT module from OKGIT with simple operations such as vector addition (OKGIT-A) and vector concatenation (OKGIT-C). We also use RoBERTa in place of BERT(OKGIT-R). As we can see, replacing BERT with simple operations result in performance similar to CaRE. However, we do see better gains with RoBERTa, which performs better than CaRE and similar to OKGIT for ReVerb45KF. For all datasets, the OKGIT model outperforms other variants (Appendix B).

Model	ReVerb20KF					ReVerb45KF				
	MRR(%)↑	MR↓	Hits(%)↑			MRR(%)↑	MR↓	Hits(%)↑		
			@1	@3	@10			@1	@3	@10
CaRE	29.3	308.3	22.1	31.6	43.2	26.6	692.7	20.1	28.8	39.1
OKGIT (UFET)	8.8	1208.2	6.9	9.6	11.0	4.9	1156.8	1.5	4.0	11.5
OKGIT [Our model]	**34.6**	**214.7**	**26.5**	**38.0**	**50.2**	**29.7**	**500.2**	**22.5**	**32.4**	**43.3**

Table 9: Comparison of OKGIT with OKGIT(UFET). We can see that including UFET model in the system hurts the performance of the model (Appendix C).

on ReVerb20KF and ReVerb45KF datasets. The results are presented in the Table 9. As we can see from the results, OKGIT(UFET) performs poorly, even when compared to CaRE. It suggests that explicit type vectors from UFET model does not help in the link prediction task.

D BERT as Link Prediction Model

As mentioned in Section 4.2, t_B from Equation (3) can be used for predicting tail NPs by finding nearest neighbors in BERT vocabulary. However, this approach has a limitation. This model can only predict NPs that are single token and present in BERT vocabulary, restricting its applicability.

This limitation, however, is not valid for OKGIT. In OKGIT, the vector t_B is used for computing tail type compatibility score, instead of predicting tail NPs. Therefore, it is not restricted to BERT vocabulary or single-token NPs. As shown in Table 4, OKGIT is equally effective for single-token

Model	ReVerb20K					ReVerb45K				
	MRR(%)↑	MR↓	Hits(%)↑			MRR(%)↑	MR↓	Hits(%)↑		
			@1	@3	@10			@1	@3	@10
CaRE (Gupta et al., 2019)	30.7	879.2	24.4	33.5	41.7	32.9	1325.1	26.1	36.2	45.4
OKGIT [Our model]	**34.3**	**609.4**	**27.0**	**37.1**	**47.4**	**34.1**	**820.2**	**26.7**	**37.5**	**47.5**
Model	**ReVerb20KF**					**ReVerb45KF**				
	MRR(%)↑	MR↓	Hits(%)↑			MRR(%)↑	MR↓	Hits(%)↑		
			@1	@3	@10			@1	@3	@10
CaRE (Gupta et al., 2019)	28.0	326.0	21.2	30.5	41.3	28.0	683.8	21.0	31.4	41.3
OKGIT [Our model]	**31.7**	**258.1**	**24.2**	**34.0**	**46.3**	**31.2**	**483.3**	**23.8**	**34.4**	**45.4**

Table 10: Results of link prediction task on the validation split. We can see that the OKGIT model outperforms the baseline models on all the datasets (Appendix F).

datasets (e.g., ReVerb20KF and ReVerb45KF) and multi-token datasets (e.g., ReVerb20K and ReVerb45K).

E Selection of NPs for t-SNE

The OpenKGs do not have type annotations for the NPs. Therefore, we manually annotated a set of NPs and visualized a random subset. For this process, we first list all the NPs and shuffle them. Then we scan this list and note the first fifteen person names, locations, and years. Later, we select five NPs from each of these categories randomly and use them for the evaluation.

F Link Prediction Performance on Validation Split

The performance of CaRE and OKGIT on validation data on the link prediction task can be found in Table 10. These performance corresponds to the respective models which were used to report results in Table 4 of the main paper.

G Type Information in BERT Predictions

Our proposed OKGIT model is based on the hypothesis that BERT vectors (i.e., $\mathbf{t_B}$ in Equation (3) in Section 4.2) contain implicit type information. In this section, we evaluate this hypothesis that BERT vectors contain type information. It should be noted that evaluating OKGIT model for predicting NP types is not the goal here. We are interested in understanding whether pre-trained BERT vectors have sufficient type information, measured with respect to some existing anchors.

Evaluation Method: For this experiment, we use Freebase (Bollacker et al., 2008) which contains explicit gold type information for entities. Specifically, we use FB15K dataset (Bordes et al., 2013). We use the data from (Yao et al., 2019) for converting symbolic names in FB15k to textual descriptions. We only consider the subset of triples in FB15k which has single token in the tail node as BERT can only predict single token NPs.[8] This results in $n_T = 95,782$ triples. For type information, we use the data from (Xie et al., 2016). It contains 61 primary types (e.g., */award*). Please note that each node in FB15k can have multiple types. For a triple (h, r, t), we consider the types associated with the true tail NP t as true types $\Gamma(t)$. We then pass tokenized head NP and RP to BERT and find the top prediction $\hat{t} = \text{BERT}(h, r, \text{MASK})$ for tail position. The set of types associated with the predicted NP $\hat{t}$, denoted by $\Gamma(\hat{t})$, is then used as the predicted types. For evaluation, we calculate the following metrics

$$\text{Precision} = \frac{1}{n_T}\sum_{i=1}^{n_T}\frac{|\Gamma(t_i)\cap\Gamma(\hat{t_i})|}{|\Gamma(\hat{t_i})|},$$

$$\text{Recall} = \frac{1}{n_T}\sum_{i=1}^{n_T}\frac{|\Gamma(t_i)\cap\Gamma(\hat{t_i})|}{|\Gamma(t_i)|}, and$$

$$\text{F1} = \frac{2}{n_T}\sum_{i=1}^{n_T}\frac{|\Gamma(t_i)\cap\Gamma(\hat{t_i})|}{|\Gamma(\hat{t_i})|+|\Gamma(t_i)|}.$$

Here, $|\Gamma(t)|$ and $|\Gamma(\hat{t})|$ denotes the number of types present in $\Gamma(t)$ and $\Gamma(\hat{t})$ respectively. [9] For comparison, we use the following baseline methods to assign types to a given (h, r, t).

[8] Please note that this limitation is only valid for BERT, not for OKGIT.

[9] Please note that, since we have gold type annotations available for Freebase, the number of true and predicted types need not be the same. Therefore, we evaluate precision and recall along with F1-scores.

Model	Precision	Recall	F1
Random	0.13	0.10	0.10
MFT	0.45	0.30	0.31
BERT	0.44	**0.40**	**0.36**
Human	**0.87**	0.18	0.27

Table 11: Results of the experiment to test whether BERT embeddings are rich with type information. As we can see, BERT outperforms other methods in terms of F1 score, suggesting that it contains relevant type information. Please refer to Appendix G for more details.

Random: assign $|\Gamma(\hat{t})|$ randomly selected types.
Most Frequent Types (MFT): assign $|\Gamma(\hat{t})|$ most frequent types.
Human: We also evaluate the type annotations provided by human annotators on randomly selected 100 triples. Each triple is exposed to three annotators and they are asked to provide types to the tail NP. Since most of the annotations contain one type for each triple, we take the union of the types provided by different annotators to compensate for Recall. For 69% of the triples, the annotators agreed on the same type.

To be fair with the automated baselines, we use the same number of predicted types as BERT (i.e., $|\Gamma(\hat{t})|$). A comparison with a pre-trained explicit entity typing methods, such as (Choi et al., 2018), is not applicable here as their type vocabulary is different. As we can see from the results in Table 11, BERT achieves best F1 score, suggesting that it contains type information. The Recall for Human is low since most of the annotations contained only one type, resulting in lower F1 score.