Knowledge Relevance BERT: Integrating Noisy Knowledge into Language Representations

Karan Samel[1*] **Jun Ma**[2] **Zhengyang Wang**[2] **Tong Zhao**[3] **Irfan Essa**[1,4]
Georgia Tech[1] Amazon[2] Uber[3] Google[4]
ksamel@gatech.edu, junmaa@amazon.com, zhengywa@amazon.com, tongz@uber.com, irfan@gatech.edu

Abstract

Integrating structured knowledge into language model representations increases recall of domain-specific information useful for downstream tasks. Matching between knowledge graph entities and text entity mentions can be easily performed when entity names are unique or there exists entity linking data. When extending this setting to new domains, newly mined knowledge contains ambiguous and incorrect information, with no explicit linking information. In such settings, we design a framework to robustly link relevant knowledge to input texts as an intermediate modeling step while performing end-to-end domain fine-tuning tasks. This is done by first computing the similarity of the existing task labels with candidate knowledge triplets to generate relevance labels. We use these labels to train a relevance model, which predicts the relevance of the inserted triplets to the original text. This relevance model is integrated within a language model, leading to our Knowledge Relevance BERT (KR-BERT) framework. We test KR-BERT for linking and ranking tasks on a real-world e-commerce dataset as well as a public entity linking task, where we show performance improvements over strong baselines.

Introduction

Learning language representations is a fundamental task for many natural language applications, such as classification, question answering, summarization, and topic modeling. These language representations have recently been learned from deep self-attention based Transformers(Vaswani et al. 2017), BERT(Devlin et al. 2019) and their variants (Liu et al. 2019; Lan et al. 2019). Such models learn contextualized input text embeddings, but many tasks benefit from external knowledge beyond the input text through knowledge graphs (KGs). These KG-integrated Transformers (Liu et al. 2020; Sun et al. 2019; Wang et al. 2021; Peters et al. 2019) can leverage well-curated knowledge linked to their text representations. This linked knowledge enables explicitly recalling sparse facts or decoupled information related to the input text, which improves performance on domain-specific tasks over their standard Transformer counterparts. Given these successes, how can we leverage these KG-integrated Transformers in *new domains* where such well-curated knowledge and linking data may not exist? We investigate this noisy knowledge incorporation into Transformers within new domains.

New Domain Knowledge When curating knowledge for a domain that is specialized, such as e-commerce or molecular chemistry, corresponding KGs for these domains may not exist. While such structured KG information is unavailable, typically unstructured information is present through documents within that domain. For example, there are product details or research papers within e-commerce and chemistry domains respectively. Knowledge can be weakly mined from these unstructured data sources, where heuristics or knowledge extraction techniques transferred from other domains are used. Since KG construction within the new domain is not studied, the mined graph will inevitably contain ambiguous and incorrect knowledge. Therefore a model leveraging the mined KG in this domain must be robust to such noise.

Mined Knowledge Noise By mining documents, entities corresponding to common or *ambiguous* terms appear and can be associated with different intents. For example, in e-commerce, we want to find products associated with hiking. We have mined knowledge (water, occurs in, *nature*) and the product text “Merrell Moab 2 Boot: New Merrell water resistant boots...”. We can associate “Merrel Moab 2” to hiking since a model can learn that hiking products are used in *nature*. However, if there are multiple valid triplets such as (water, required for, painting), we need to choose the correct triplet to augment our product. When the incorrect intent is inserted into the text representations, they can detract from the original text’s meaning. Typically such ambiguous knowledge can be resolved by entity linking methods. However, in new domains such linking datasets are not available, thus we must disambiguate relevant knowledge only through the available data for that domain.

Additionally, Our newly mined KGs are also more likely to have *incorrect* triplets when the mining techniques used are not well researched or fine-tuned within that domain. Methods that leverage KGs for their task must be able to mitigate the effects of these triplets as well without explicit supervision.

[*]Work partially completed during an internship at Amazon.
Accepted to Workshop on Knowledge Augmented Methods for Natural Language Processing, in conjunction with AAAI 2023.

Contributions We focus on optimizing new domain-specific end tasks with noisy KGs by exploring tractable approaches to identify ambiguous and incorrect knowledge as intermediate modeling steps. We present these steps within our Knowledge Relevance BERT (KR-BERT) pipeline, where we contribute the following:

- To prevent ambiguous knowledge, we present a knowledge ranking scheme to source the top most relevant KG triplets.
- We further suppress ambiguous and incorrect triplets through a triplet relevance model, which only uses domain-specific task data for supervision. This relevance model is integrated directly into our Transformer for end-to-end training.
- We present our framework's performance on a real-world e-commerce dataset and a public entity linking task, where we show that KR-BERT outperforms existing baseline KG integrated Transformers.

Related Works

To understand how to integrate knowledge into language representations we first look at how entities are generally linked to texts, and then how linking done within existing language models.

Entity Linking Entity linking associates which specific entity in the KG is associated with a found entity mentioned in the text. They traditionally have been carried out using features-based methods to operate on short texts, such as Twitter data (Guo, Chang, and Kiciman 2013; Yang and Chang 2016). Similarly linking is performed over larger KGs in domains with more complex document data, such as in news articles (Kolitsas, Ganea, and Hofmann 2018) or the biomedical setting (Angell et al. 2021).

In our setting with new domains, there will not be high quality KGs and supervision. Here neural-based models have been proposed to learn representations of the entity in its text context and match them to entities in KGs with limited to no supervision (Nayak and Bach 2020; Logeswaran et al. 2019; Wu et al. 2019). However there are many candidate triplets to be checked within the same input text, thus a large computational overhead is required to perform such disambiguation. This makes it unclear how to efficiently integrate such large-scale entity linkers within Transformers to optimize end-to-end. We address this by computing the embedding similarity of the triplets to the sample *labels* as a lightweight method of selecting the most relevant top-k triplets.

KG integrated Transformers Once entities are linked to the text some works fuse external triplet information into text representations. One method is to insert knowledge triplets at the input text level, as done in K-BERT (Liu et al. 2020). ERNIE (Sun et al. 2019) learns a cross-modal KG entity and token embeddings within the intermediate Transformer layers. KEPLER (Wang et al. 2021) keeps the original Transformer architecture and adds a KG entity embedding objective(Bordes et al. 2013) based on the triplets found in the text to fine-tune their token representations. These methods don't explicitly disambiguate between similar entities to use since they often use well curated KGs, such as Wikidata[1], DBpedia[2], or YAGO(Pellissier Tanon, Weikum, and Suchanek 2020), which contain minimal ambiguous knowledge. KnowBERT(Peters et al. 2019) integrates an entity linking sub-module to infer the correct entity, but requires entity linking data or relies on heuristics to perform embedding alignment when such linking data doesn't exist. In KR-BERT we relax the entity linking data requirement through our relevance model, which we integrate into our KG-based Transformer to optimize knowledge relevance and our domain-specific task end-to-end.

Method

Preliminaries

To define our problem, we are provided a dataset where each sample consists of a text string and domain labels $(T, \mathbf{y})$, where the task is to infer the labels from the text $T \rightarrow \mathbf{y}$. We further refer to the token level strings in the sentence as $[x_1, x_2, \ldots, x_N] = T$ for a text with N tokens. For our e-commerce example the input pairs consist of product text T and concept labels $\mathbf{y}$ that we want to predict. For tasks such as entity linking the problem is similarly posed for the reference text and the entity labels associated to the text.

To better associate the text to the labels, we leverage a knowledge graph (KG) consisting of subject, relation, and object (s, r, o) triplets. We integrate this knowledge by augmenting the *input level text* T. This is done by finding all subject text s mentions in T, and then inserting the corresponding relation r and object o text right after each subject text. For example, given the triplet (water, occurs in, *nature*) and the text T "...Merrell water resistant boots...", our augmented text T' would be "...Merrell water occurs in *nature* resistant boots...". Such KG information is useful to infer the concept labels, such as $\mathbf{y} = \{\texttt{hiking}\}$, from T'.

If there are multiple candidate triplets to insert, we would have to choose the most relevant triplets through entity linking methods. In our previous example we might also insert the triplet (water, required for, painting), but does not fit into the context of the product. To address this we design KR-BERT, which first sources the relevant knowledge triplets, predicts if the selected triplets are relevant for the corresponding text, and finally controls the knowledge augmentation through a gating mechanism within the Transformer's self-attention mechanism.

Knowledge Triplet Selection

Triplet Candidate Sourcing In the triplet candidate sourcing step, we form a match set C that contains all triplets with subject tokens s that text match tokens in the original text T. We then apply a simple unsupervised filtering step leveraging each sample label $y \in \mathbf{y}$ to refine the knowledge triplet match set.

Specifically, we compute $\mathbf{e}^{\mathbf{y}_i}$, the averaged token embedding of each label $y \in \mathbf{y}$ as well as $\mathbf{e}^{\mathbf{o}_j}$, the averaged token

[1]https://www.wikidata.org/

[2]https://www.dbpedia.org/

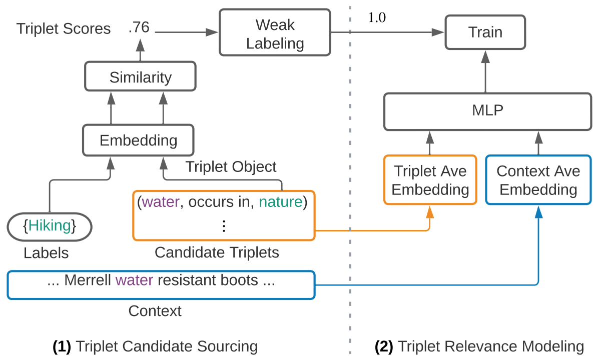

Figure 1: In **(1)** we show triplet sourcing where we determine the score of triplets based on the objects embedding similarity with the sample label. It generates weak labels for triplet relevance modeling in **(2)**, which predicts the similarity using the triplet and the text.

embedding for each triplet object o in the candidate set C. We use a pre-trained Word2vec (Mikolov et al. 2013) model to generate the embeddings. Next we compute the cosine similarity between each pairwise label $\mathbf{e^{y_i}}$ and triplet object embedding $\mathbf{e^{o_j}}$. For each triplet, we take its max score across all label embeddings $\mathbf{e^y}$ to determine the triplet's overall relevance score. This triplet score score$_j$ is computed for all candidate triplets as shown in Equation 1, where S is the set of scores for all triplets.

$$\text{score}_j = \max_{\mathbf{e^{y_i}} \in \mathbf{e^y}} \cos(\mathbf{e^{y_i}}, \mathbf{e^{o_j}}) \quad \forall\ \text{score}_j \in S \tag{1}$$

We provide a visual example of the triplet scoring on the left of Figure 1. In the next step, we refine our candidate set C by the top-k scoring triplets $C = \{(s, r, o)_j \mid \text{score}_j \in \text{top-}k(S)\}$, which are then inserted into our sentence T to produce our model's text input T'.

Even though we rank the triplets to insert, we don't guarantee that the triplets are relevant within context. The best triplets in our KG can still have a low score, so we still need a model to determine if the inserted triplets are truly relevant to the text as a whole. Therefore we also use the triplet scores as weak labels to train a triplet relevance model.

Triplet Relevance Prediction Given the sourced top-k triplets, we use a binary classifier to predict the relevance of these triplets to the *text*. This is done through a triplet relevance model, a 2-layer MLP with 100 hidden units containing two inputs, as shown on the right of Figure 1.

The first input is an average embedding $\mathbf{e^c}$ of the context text T^c. This context text is a representative sub-text of the original text $T^c \subseteq T$, such as a product title. The second is an average embedding $\mathbf{e^t}$ of all triplet tokens T^* obtained from our top-k candidate triplets C.

The triplet relevance model outputs the relevance probability score of the top-k triplets for the context text: $p = \sigma(\text{MLP}(\mathbf{e^t}; \mathbf{e^c})) \in [0, 1]$. We use weak labels generated in Equation 2 to train the relevance prediction model.

$$y_{weak} = \begin{cases} 1, & \text{if } \min(\text{top-}k(S)) > 0.5 \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Given top-$k(S)$ we consider the top-k triplets as relevant if all scores are above a threshold, otherwise they are irrelevant.

We then optimize a standard cross entropy loss between the MLP output and the weak labels $\mathcal{L}_{rel} = \text{CE}(p, y_{weak})$. To summarize, we first leverage triplet sourcing to score relevant triplets based on the similarity of the triplet object to the sample labels. Then we threshold these scores to generate labels to train a small MLP model to predict the relevance of the triplet embedding to the text. This relevance model is integrated into a Transformer to efficiently compute relevance during end-to-end training, described in the next section.

Knowledge Relevance BERT

Building on top of K-BERT (Liu et al. 2020), we propose Knowledge Relevance BERT (KR-BERT), which integrates the triplet relevance model into a transformer to regulate the impact of the inserted triplets during concept linking.

Knowledge Masking In K-BERT, the text is also augmented by directly inserting triplets into the text. Concretely the original text T and the triplet tokens T^* make up the full augmented text $T \cup T^* = T'$ used as the model input. However, attention across the input text and inserted triplets may corrupt the original semantic meaning of the text. In our example "...Merrell water occurs in *nature* resistant boots..." we don't want the inserted triplet tokens "occurs in nature" to attend to "Merrell" or "resistant" since it was not in the original context.

To address this problem, K-BERT utilizes a visible matrix M (Equation 3) to mask out the self-attention weights between text and knowledge triplet text tokens.

$$M_{ij} = \begin{cases} 0 & \text{if } x_i, x_j \in T \text{ or } x_i, x_j \in T^* \\ -\infty & \text{otherwise} \end{cases} \tag{3}$$

Since it uses triplet subjects s as anchor points for triplet text insertion, s will be available in both T and T^* ($T \cap T^* = s$). With the visible matrix M as a mask, all text tokens in T will attend to subject tokens s, but not to r and o in T^*. Conversely, the triplet tokens r and o can attend to the subject tokens s, but not the rest of T. Therefore only the subject tokens s will be contextualized by both the input text and knowledge triplet within the self-attention layer.

Knowledge Relevance To integrate our triplet relevance model we first define a triplet position matrix P in Equation 4 which is a self-attention mask between triplet tokens.

$$P_{ij} = \begin{cases} 1 & \text{if } x_i, x_j \in T^* \\ 0 & \text{otherwise } (x_i \text{ or } x_j \in T - s) \end{cases} \tag{4}$$

Given the relevance score p computed from our relevance model, we update the self-attention scores between text and triplet tokens using $D = \mathbf{1} - (1 - p) \cdot P$. Given the visible

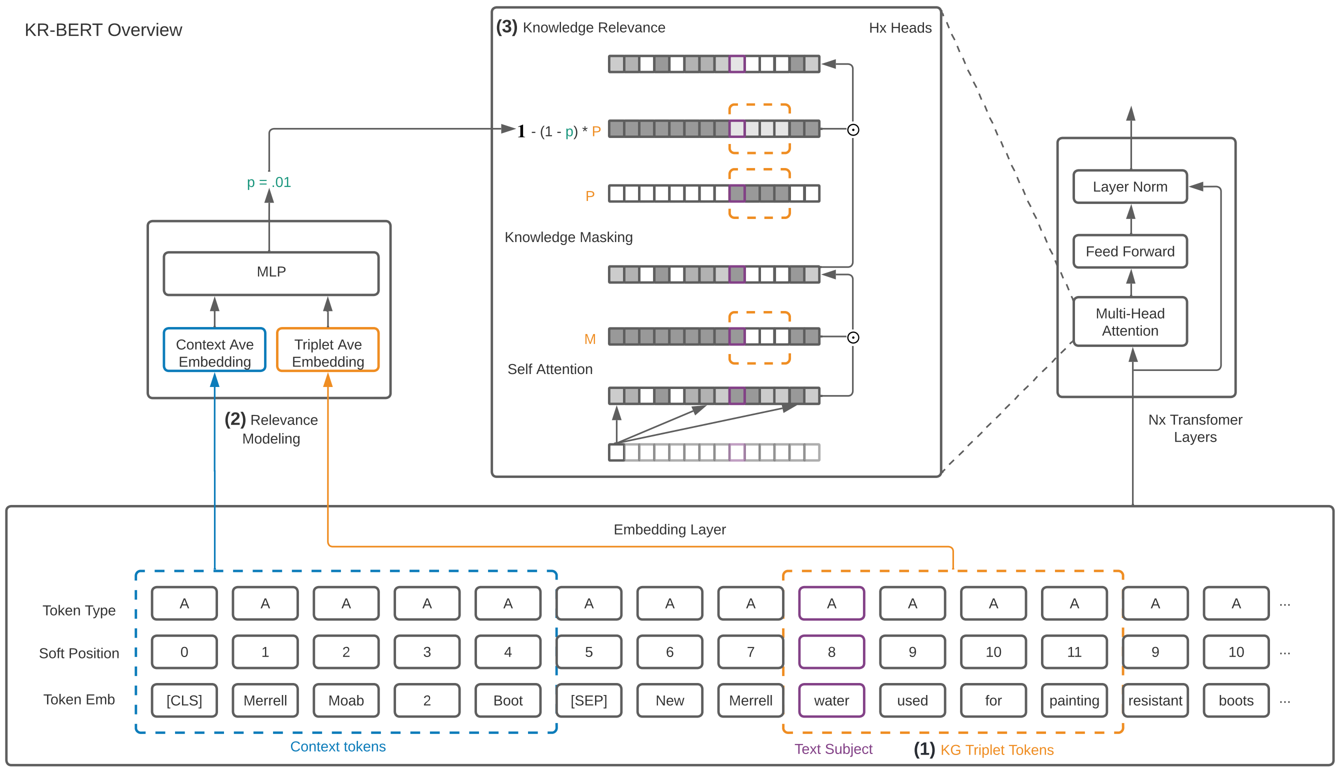

Figure 2: Our KR-BERT framework with key steps: **(1)** We insert triplet tokens into the text input. **(2)** The triplet relevance model determines if the triplet inserted into the input is relevant to the text context. **(3)** This relevance score is used to gate the impact of the triplet's attention scores on other tokens if deemed irrelevant.

matrix M and knowledge relevance scaling D, we define the knowledge relevant self-attention as in Equation 5.

$$\text{self-attn} = (D \odot \text{softmax}(\frac{QK^{\top} + M}{\sqrt{d}}))V \tag{5}$$

Here $\odot$ indicates element-wise multiplication. We illustrate this full process in the attention layer in the center of Figure 2. Note that in case p is 0 (low relevance), D will mask out the attention weights between the original text and inserted triplet tokens, reverting to the original text T self-attention. In case p is 1 (high relevance), D will not change any attention weights, and it uses the K-BERT self-attention over the augmented text T'. This full KR-BERT model is illustrated in Figure 2.

Model Training We first pre-train the model with a masked language model (MLM) objective to predict masked out tokens from T', which are randomly masked out 15% of the time. During pre-training, we remove the relevance model since it may try to suppress a triplet token that the MLM objective is trying to simultaneously predict.

After pre-training we train our triplet relevance MLP on our relevance weak labels, using the pre-trained frozen input embeddings from the MLM stage. This ensures the input embeddings used for the relevance model and the pre-trained KR-BERT layers are aligned for future fine-tuning.

Finally, the pre-trained KR-BERT weights and relevance weights are loaded to start the fine-tuning procedure. Here we train our classifier heads to predict our labels $\hat{\mathbf{y}} = \text{classifier}(\text{BERT}(T'))$, where our classifier heads are a dropout layer followed by a single dense layer. This is optimized through cross entropy as $\mathcal{L}_{cls} = \text{CE}(\hat{\mathbf{y}}, \mathbf{y})$. We also keep our relevance optimization objective during fine-tuning to maintain consistent relevance predictions for knowledge regulation. This gives our final end-to-end fine tuning loss as $\mathcal{L} = \mathcal{L}_{cls} + \mathcal{L}_{rel}$.

Model Inference During model training we will have labels $\mathbf{y}$ for the triplet candidate sourcing, while at inference we do not. Therefore during inference we first predict the task labels $\hat{\mathbf{y}}_{pre}$ without the sourcing step. Then we can provide $\hat{\mathbf{y}}_{pre}$ to bootstrap the triplet sourcing step to provide triplets for the rest of the pipeline and predict the final labels. This can be thought as a two step process to first compute an initial estimation of the distribution of labels $\hat{\mathbf{y}}_{pre}$, similar to models such as K-BERT or ERNIE that are triplet agnostic. Then in the second pass, we source triplets using $\hat{\mathbf{y}}_{pre}$ to insert relevant triplets into the text and re-rank the final label predictions $\hat{\mathbf{y}}$.

E-Commerce Experiments

We first test KR-BERT on e-commerce product linking and search tasks, where product descriptions and KGs provide rich information for e-commerce applications. Having product level data provides *precision* on which products are most relevant to a user query. Conversely, a knowledge graph linking products would enable better *recall* of relevant or useful products beyond a typical query.

The challenge to enable this recall is that there are no established KGs that focus on linking products to common concepts, or high-level topics of interest that customers search for. For example, if the query pertains to the concept of hiking, an existing product taxonomy might show products related to hiking boots and poles. However, we would also want to extend the coverage of these products to include categories such as water bottles, bug spray, small first aid kits, etc. which would also be relevant to the user and would ideally come from a KG.

Before mining a KG, we first define a set of concepts that customers are interested in, where we use the Wikipedia list of hobbies page[3]. Examples of such hobby concepts are hiking, baking, sketching, e-sports, and so forth spanning 572 unique concepts. These concepts are common entities with which we can associate relevant products in a KG.

Concept KG Construction

Starting from these hobby concepts, we next construct a knowledge graph of products via rule-based extraction from the corresponding wiki pages. For each hobby concept wiki page, we look for subsections containing the following keywords: tool, material, equipment, and accessories. If such subsection exists, we check two structures for products: 1) linked pages which may lead to another product page, such as a hiking boots page link within the equipment section in the hiking page. 2) items in comma delimited lists, which often describe products required for that hobby concept. We then insert triplets in the form of (product, required for, hobby concept) into the KG. As a result, our final product to concept KG contains 2.6k triplets covering 87 out of 572 total hobby concepts. With these heuristics, we mine a KG that covers a spectrum of e-commerce products but contains significant noise as well.

Task Data Set Construction

To build our dataset we use customer search logs which contain search queries associated with purchased products. We then assign our hobby concepts as labels to these query-product pairs if the concept text occurs in the search query. To overcome the low-recall issues due to direct text matching, we expanded by also linking the matched concepts with *all* queries and purchased products within the same search *session*. This led to 4M query-product pairs associated with multi-label concepts. These include over 2M unique products spanning 273 hobby concepts, which are used to develop our e-commerce tasks. With this e-commerce dataset and KG we generated, we test two tasks.

Product Concept Linking The first task is product concept linking, where given a product we predict the concept label. Such classification benefits customers who explore and purchase products grouped by common concepts. For the product text T we concatenate its title, description, and any bullet points. For evaluation we sample 10 product-concept pairs for each concept and manually cleaned the concept labels. After correction, we had a gold test set of 2.7k samples spanning 253 unique hobby concepts.

[3]https://en.wikipedia.org/wiki/List_of_hobbies

Search Ranking The second task we test is a search ranking task. Given our query-product pairs, the query is concatenated with augmented product text T' to serve as the input. Then the model predicts if the product was purchased given the query as a binary prediction task. For each query, we sampled 100 products that didn't have the same concept label to serve as negative samples during training. For testing, we use the query-products from our manually corrected test set as positive pairs. Given a fixed query and its corresponding concept, we use two strategies to obtain products for negative sampling:

- Any products where the corresponding concept is in the original test labels, but removed during test set correction. The concept corresponding to these products was incorrect, but likely a close match (hard cases).
- Random product sampling to fill in the rest of the negative labels (easier cases).

For each positive test query-product pair, we generate 99 other negative product samples using these two strategies and test the model's product recall over 100 products.

Baselines and Experimental Setup

During our experiments we used a 12 layer ALBERT (Lan et al. 2019) as our backbone, which is used for *all* our models and baselines to establish a fair comparison. The baselines we test against are:

- Keyword Search: A keyword baseline that searches for the concept token in the text.
- KG Lookup: Since our concepts and knowledge graphs were mined from Wikipedia pages, we also test a KG lookup. In this setting, if a candidate triplet subject (product) matches a text substring, then the triplet's object (hobby concept) is predicted.
- ALBERT: The standard ALBERT model with MLM pre-training and fine-tuning on the task.
- K-BERT: This is the K-BERT implementation, but with the ALBERT backbone. Unlike the original paper, we run MLM pre-training given the knowledge insertions, which improves performance on our tasks.
- ERNIE: We use the ERNIE implementation with the ALBERT backbone. We perform pre-training using MLM and entity prediction as described in the ERNIE paper (Sun et al. 2019). For its entity embeddings, we trained a TransE (Bordes et al. 2013) model on top of our KG.

For all methods use the AdamW optimizer (Loshchilov and Hutter 2018) with the default settings provided in the original paper, except we updated our learning rate to $1e^{-5}$ for all experiments. For MLM pre-training we used our product text corpus, where we randomly masked out the token 15% of the time. We pre-trained and fine-tuned all models till we see training loss convergence, which usually happened within 5 epochs.

Method	Micro			Macro		
	F1	P	R	F1	P	R
Keyword Search	.407	.343	**.760**	.565	.564	.743
KG Lookup	.361	.422	.580	.446	.497	.606
ALBERT	.574	**.602**	.655	.692	.695	.788
ERNIE	.576	.555	.707	.701	.671	.825
K-BERT	.586	.574	.684	.711	.686	.817
KR-BERT	**.598**	.582	.725	**.717**	**.697**	**.826**

Table 1: Results for predicting the hobby concept given a product. The best results are in bold while the runner ups are underlined.

Method	Micro			Macro		
	R@1	R@5	R@10	R@1	R@5	R@10
TF-IDF	.642	.900	.945	.694	.941	.973
ERNIE	**.869**	.956	.977	.874	.973	.984
K-BERT	.865	.953	.976	.885	**.977**	**.987**
KR-BERT	.864	**.957**	**.978**	**.891**	**.977**	**.987**

Table 2: Results for ranking the correct product given the query.

Method	Relevance Model	Triplet Sourcing	Micro			Macro		
			F1	P	R	F1	P	R
ERNIE	no	yes	.554	.545	.671	.690	.664	.807
K-BERT	no	yes	.567	.542	.708	.699	.677	.803
KR-BERT	yes (frozen)	no	.589	**.615**	.655	.708	.687	.821
KR-BERT	yes ($\mathcal{L}_{cls}$ only)	no	.583	.611	.633	.707	.686	.822
KR-BERT	yes ($\mathcal{L}_{cls} + \mathcal{L}_{rel}$)	no	.568	.572	.680	.702	.686	.808
KR-BERT	yes (frozen)	yes	.591	.581	.675	.703	.673	.814
KR-BERT	yes ($\mathcal{L}_{cls}$ only)	yes	.583	.603	.646	.700	.696	.793
KR-BERT (full)	yes ($\mathcal{L}_{cls} + \mathcal{L}_{rel}$)	yes	**.598**	.582	.725	**.717**	**.697**	**.826**

Table 3: Ablating the triplet sourcing and relevance model in KR-BERT as well as our baselines for the concept linking task.

Task Evaluation Results

Product Concept Linking We present the concept linking results in Table 1. Since keyword search matches all concepts, it has the best recall but poor precision. The KG lookup results prove that the mined KG is too noisy to use directly to predict concept labels. Our neural baselines can further adapt to this noise, where ERNIE and K-BERT also directly leverage knowledge. However, our full KR-BERT best adapts to the noisy knowledge and provides the best performance.

Search Ranking For our retrieval task in Table 2 KR-BERT also shows strong performance. Here the baseline Transformer methods also show comparable performance in this task. Learning from query-product pairs leads to large performance improvements for all Transformer methods over traditional lexical retrievers, such as TF-IDF.

Model Ablation Studies

We further investigate how the different components of our model lead to performance differences. First, we investigate the interaction of our triplet sourcing and relevance modeling stages. Then we further dive into selecting the top-k triplets used to augment our text. Finally, we view the efficacy of our triplet relevance model.

Triplet Sourcing and Relevance Within KR-BERT we ablate multiple setups for the relevance model during the fine-tuning stage as shown in Table 3. Without triplet sourcing, simply freezing the pre-trained relevance model weights outperforms end-to-end task fine-tuning. This is because we cannot efficiently sample positive triplets during fine-tuning $\mathcal{L}_{rel}$, thus mostly predict negative labels. Then during inference, any input triplets are suppressed. Adding the triplet sourcing, we produce better relevance labels and jointly tune the classification loss, providing better results.

We also test our triplet sourcing method within ERNIE and K-BERT and observe that even with triplet sourcing, the performance does not improve. Therefore an additional mechanism needs to filter relevant triplets, performed through the relevance model in KR-BERT.

Method	top-k	Micro F1	Macro F1
ERNIE	1	**.576**	**.701**
	3	.571	.674
K-BERT	1	**.586**	**711**
	3	.577	.700
KR-BERT	1	**.598**	**.717**
	3	.589	.705

Table 4: Product concept linking performance with varying number of triplet insertions.

Top-k Analysis In Table 4, we show the impact of our top-k hyper-parameter for number of inserted triplets. We find that inserting more than 1 triplet leads to degraded performance regardless of the model choice. This is likely due to the noise introduced by more insertions under the noisy knowledge setting, as opposed to inserting all triplets from well curated KGs (Sun et al. 2019; Liu et al. 2020). Using

(Subject, Object)	Category	y_{weak}	Pred p
(bikini, swimming)	swimwear	1	.99
(foundation, makeup)	concealer	1	.94
(racket, table tennis)	sport table	1	.99
(action figure, dolls)	toy figure	0	.97
(marker, journaling)	writing	0	.99
(trail shoe, hiking)	shoes	0	.99

Table 5: Relevance success cases and generalization.

(Subject, Object)	Category	y_{weak}	Pred p
(height, table tennis)	shoes	0	.99
(strike, cricket)	sandal	0	.97
(protection, rock climbing)	sunglasses	0	.97
(sew, sewing)	sewing button	1	.06
(compressed air, paintball)	air gun	1	.01

Table 6: Relevance prediction failure cases.

$k = 1$ makes it easier for the model to understand the relevance for each triplet disjointly, rather than jointly estimating the relevance of multiple $k > 1$ insertions.

Relevance Model Predictions We observe the predictions made by our relevance model to see if it is learning useful relevance scores. This is an important step to verify correctly operating model logic, instead of having performance gains just from the additional free parameters from additional the relevance model.

In Table 5 we observe where the model was able to correctly predict a positive weak label y_{weak} from our relevance prediction training. For brevity, we only show the subject and the object for the triplet and the category of the corresponding product for which we are predicting the relevance score of. We also see generalization successes, where even though we had negative weak labels, our model still predicts the relevance between the triplet and context correctly. We also observe false positive and false negative cases in Table 6. Here we see relevance prediction failure, as well as in some cases failure to suppress incorrectly mined triplets, as seen in row 1. Notably, this example shows noisy knowledge and how models are susceptible to it.

Triplet Label Overlap Due to the nature of our e-commerce dataset construction, the hobby concepts in our triplets are also the concept labels that we are trying to predict. For example, during triplet sourcing we will be matching sample labels $\mathbf{y} = \{\texttt{hiking}\}$ directly to triplets containing *hiking* as well as other closely related triplet objects such as *nature*. However, we don't have these ground truth concept labels during inference and are sourcing triplets given inferred labels. Therefore our model still learns the appropriate triplet insertions to do well in our tasks. Additionally, this concept-label overlap is not always present in general and our model still shows benefits, as shown in further entity linking experiments.

Model	top-k	F1	P	R
ALBERT	-	.564	.600	.641
K-BERT	no-limit	.554	.544	.635
ERNIE	no-limit	.593	.601	**.670**
KnowBERT	no-limit	.489	.475	.568
+ $\mathcal{L}_{MLM}$	no-limit	.556	.599	.631
+ $\mathcal{L}_{MLM}$ + $\mathcal{L}_{rel}$	1	.570	.589	.641
+ $\mathcal{L}_{MLM}$ + $\mathcal{L}_{rel}$	3	.578	.591	.655
+ $\mathcal{L}_{MLM}$ + $\mathcal{L}_{rel}$	no-limit	.582	.591	$\underline{.661}$
KR-BERT	1	$\underline{.599}$	**.644**	.610
KR-BERT	3	**.601**	$\underline{.622}$	.658
KR-BERT	no-limit	.582	.615	.631

Table 7: Performance results on FIGER data.

FIGER Entity Linking

In addition to e-commerce data, we also tested KR-BERT on Wikipedia sentences tagged by FIGER (Ling and Weld 2012), a public fine-grained entity linking framework. Each sentence has on average 4.4 entities, of which one of the entities is labeled with a subset of 112 unique entity type labels. For example, if the relevant entity in the sentence was Tom Cruise, the corresponding entity labels would be [person, person/actor]. The data set contains 2M training samples, 10k validation, and 563 test samples obtained from the ERNIE codebase[4].

Following the training setup from ERNIE, the Wikidata KG was used to augment the representations for this task. Unlike the product linking data where there were overlaps between the labels and concepts, this is not enforced with the Wikidata KG. For example, the labels of Tom Cruise are person/actor while the closest Wikidata entry is human. This provides an opportunity to test KR-BERT's entity linking performance.

From the results in Table 7 we see strong performance versus the baselines, even with the fact that Wikidata KG is significantly cleaner than the e-commerce KG. ERNIE provides competitive performance, where it has the best recall. One explanation is that since it leverages the entire Wikidata KG for its embedding, it can capture more variance for the entity representations. This pattern similarly continues with KnowBERT where we also test adding pre-training ($\mathcal{L}_{MLM}$) and our additional relevance weak label tuning ($\mathcal{L}_{rel}$). It demonstrates the benefits of controlling which entities are used when no gold entity-linking labels are available.

We see benefits going from 1 to 3 insertions in both KnowBERT and KR-BERT since Wikidata is a clean KG. However, in KR-BERT, adding *all* the triplets lowers the performance overall since only a subset of the triplets is relevant to the target entity in each sample. Interestingly, this is not the case for KnowBERT which learns more dense neural representations to capture the interaction between entities, similar to ERNIE.

[4]https://github.com/thunlp/ERNIE#fine-tune

Conclusion and Future Work

We develop our KR-BERT framework for noisy KG-based inference, which occurs when working on new domains. It first ranks the candidate triplets based on the similarity between triplet objects and sample labels. Then a relevance model looks at the top-k ranked triplets and determines if the triplets are relevant to the text context. Finally, based on the relevance, the triplet tokens are gated within the Transformer self-attention mechanism. We empirically evaluate our method and our design choices for concept linking and ranking tasks on an e-commerce dataset and on a public entity linking task, where we outperform existing KG integrated transformers.

For future work, such a framework can be improved by computing the relevance prediction at the individual triplet level instead of the aggregate level, similar to KnowBERT, for finer control and reducing the constraint on setting top-k. We saw the recall benefits of using KG embeddings with ERNIE and KnowBERT, so a relevance model using entity and relevance embeddings could better capture this variance. By controlling the quality of the knowledge inserted into different layers of the model, we can better understand how to best leverage these noisy representations for end-to-end tasks.

References

Angell, R.; Monath, N.; Mohan, S.; Yadav, N.; and McCallum, A. 2021. Clustering-based Inference for Biomedical Entity Linking. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2598–2608.

Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, 4171–4186.

Guo, S.; Chang, M.-W.; and Kiciman, E. 2013. To link or not to link? a study on end-to-end tweet entity linking. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1020–1030.

Kolitsas, N.; Ganea, O.-E.; and Hofmann, T. 2018. End-to-end neural entity linking. *arXiv preprint arXiv:1808.07699*.

Lan, Z.; Chen, M.; Goodman, S.; Gimpel, K.; Sharma, P.; and Soricut, R. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Ling, X.; and Weld, D. S. 2012. Fine-grained entity recognition. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Liu, W.; Zhou, P.; Zhao, Z.; Wang, Z.; Ju, Q.; Deng, H.; and Wang, P. 2020. K-bert: Enabling language representation with knowledge graph. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 2901–2908.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Logeswaran, L.; Chang, M.-W.; Lee, K.; Toutanova, K.; Devlin, J.; and Lee, H. 2019. Zero-shot entity linking by reading entity descriptions. *arXiv preprint arXiv:1906.07348*.

Loshchilov, I.; and Hutter, F. 2018. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Nayak, N. V.; and Bach, S. H. 2020. Zero-shot learning with common sense knowledge graphs. *arXiv preprint arXiv:2006.10713*.

Pellissier Tanon, T.; Weikum, G.; and Suchanek, F. 2020. Yago 4: A reason-able knowledge base. In *European Semantic Web Conference*, 583–596. Springer.

Peters, M. E.; Neumann, M.; Logan IV, R. L.; Schwartz, R.; Joshi, V.; Singh, S.; and Smith, N. A. 2019. Knowledge enhanced contextual word representations. *arXiv preprint arXiv:1909.04164*.

Sun, Y.; Wang, S.; Li, Y.; Feng, S.; Chen, X.; Zhang, H.; Tian, X.; Zhu, D.; Tian, H.; and Wu, H. 2019. Ernie: Enhanced representation through knowledge integration. *arXiv preprint arXiv:1904.09223*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

Wang, X.; Gao, T.; Zhu, Z.; Zhang, Z.; Liu, Z.; Li, J.; and Tang, J. 2021. KEPLER: A unified model for knowledge embedding and pre-trained language representation. *Transactions of the Association for Computational Linguistics*, 9: 176–194.

Wu, L.; Petroni, F.; Josifoski, M.; Riedel, S.; and Zettlemoyer, L. 2019. Scalable zero-shot entity linking with dense entity retrieval. *arXiv preprint arXiv:1911.03814*.

Yang, Y.; and Chang, M.-W. 2016. S-mart: Novel tree-based structured learning algorithms applied to tweet entity linking. *arXiv preprint arXiv:1609.08075*.