Separate-and-Aggregate: A Transformer-based Patch Refinement Model for Knowledge Graph Completion

Chen Chen[1], Yufei Wang[2], Yang Zhang[2], Quan Z. Sheng[2], and Kwok-Yan Lam[1(✉)]

[1] Nanyang Technological University, Singapore
{s190009,kwokyan.lam}@ntu.edu.sg
[2] Macquarie University, Sydney, Australia
yufei.wang@students.mq.edu.au
yang.zhang21@hdr.mq.edu.au
michael.sheng@mq.edu.au

Abstract. Knowledge graph completion (KGC) is the task of inferencing missing facts from any given knowledge graphs (KG). Previous KGC methods typically represent knowledge graph entities and relations as trainable continuous embeddings and fuse the embeddings of the entity h (or t) and relation r into hidden representations of query $(h, r, ?)$ (or $(?, r, t)$) to approximate the missing entities. To achieve this, they either use shallow linear transformations or deep convolutional modules. However, the linear transformations suffer from the expressiveness issue while the deep convolutional modules introduce unnecessary inductive bias, which could potentially degrade the model performance. Thus, we propose a novel Transformer-based Patch Refinement Model (**PatReFormer**) for KGC. **PatReFormer** first segments the embedding into a sequence of patches and then employs cross-attention modules to allow bi-directional embedding feature interaction between the entities and relations, leading to a better understanding of the underlying KG. We conduct experiments on four popular KGC benchmarks, WN18RR, FB15k-237, YAGO37 and DB100K. The experimental results show significant performance improvement from existing KGC methods on standard KGC evaluation metrics, e.g., MRR and H@n. Our analysis first verifies the effectiveness of our model design choices in **PatReFormer**. We then find that **PatReFormer** can better capture KG information from a large relation embedding dimension. Finally, we demonstrate that the strength of **PatReFormer** is at complex relation types, compared to other KGC models [3].

Keywords: Knowledge Graph Completion · Transformer · Cross-Attention.

1 Introduction

Knowledge graphs (KGs) have emerged as a powerful tool for representing structured knowledge in a wide range of applications, including information retrieval, question answering and recommendation systems. A typical KG is represented as a large collection of triples $(head\ entity, relation, tail\ entity)$, denoted as (h, r, t). Despite having large amount of KG triples, many real-world KGs still suffer from incompleteness issue. To

---

[3] Source code is at https://github.com/chenchens190009/PatReFormer

alleviate this issue, the task of knowledge graph completion (KGC) is proposed [1,7,8,13], which is to predict the missing entity given the query $(h, r, ?)$ or $(?, r, t)$.

Existing methods for KGC generally learn continuous embeddings for entities and relations, with the goal of capturing the inherent structure and semantics of the knowledge graph. They define various scoring functions to aggregate the embeddings of the entity and relation, forming a hidden representation of query $(h, r, ?)$ (or $(?, r, t)$) and determine the plausibility between the query representation and missing entity embedding. Essentially, these scoring functions are a set of computation operations on interactive features of the head entity, relation and tail entity. Early KGC models like TransE [1], DistMult [2] and ComplEx [3] use simple linear operations, such as addition, subtraction and multiplication. Despite the computational efficiency, these simple and shallow architectures are incapable of capturing complicated features, e.g., poor expressiveness. To improve the model expressiveness, some recent KGC models integrate the deep neural operations into the scoring function. ConvE [8], as the start of this trend, applies standard convolutional filters over reshaped embeddings of input entities and relations, and subsequent models [6,9] follow this trend to further improve the expressiveness of the feature interaction between entities and relations. Although these convolution-based KGC models have achieved significant empirical success, they impose unnecessary image-specific inductive bias (i.e., locality and translation equivariance) to the KGC embedding models, potentially degrading the model performance.

To combat these limitations, in this paper, we propose a novel Transformer-based Patch Refinement Model (**PatReFormer**) for the KGC task. The Transformer model is first proposed to handle Natural Language Processing (NLP) tasks [24] and demonstrates superior capability in other visual tasks [37]. More recently, with the recent progress of Vision Transformer (ViT) [28], attention-based modules achieve comparable or even better performances than their CNN counterparts on many vision tasks. Through attention mechanism, ViT-based models could dynamically focus on different embedding regions to obtain high-level informative features. What is more, ViT-based models do not impose any image-specific inductive bias, allowing them to handle a wider range of input data. Motivated by this, **PatReFormer** follows a "Separate-and-Aggregate" framework. In the separation stage, **PatReFormer** segments the input entity and relation embeddings into several patches. We explore three different separation schemes: *1)* directly folding the embedding vector into several small patches; *2)* employing several trainable mapping matrices to obtain patches; and *3)* using randomly initialized, but orthogonal mapping matrix to obtain patches. In the aggregation stage, unlike [28,32] which use standard Transformer architecture, **PatReFormer** uses a cross-attentive architecture that deploys two separate attention modules to model the bi-directional interaction between the head entities and relations.

To evaluate our proposed approach, we conduct experiments on several benchmark datasets, including WN18RR, FB15k-237, YAGO37, and DB100K, for the KGC tasks. Our experiments show that **PatReFormer** successfully outperforms both non-Transformer-based and Transformer-based KGC methods, demonstrating the effectiveness of our approach. Our analysis shows the effectiveness of our cross-attention module design, patch-based position design, and embedding segmentation design. We find that **PatReFormer** is capable to learn useful KG knowledge using a large embedding

dimension, while previous KGC models cannot. Finally, we demonstrate the advantages of `PatReFormer` in complex relation types, compared to previous KGC methods.

2   Related Work

Non-Neural-based methods. A variety of non-neural based models are proposed for KGC leveraging simple vector space operations, such as dot product and matrix multiplication, to compute scoring function. TransE [1] and its subsequent extensions [34,33] learn embeddings by representing relations as additive translations from head to tail entities. DistMult [2] uses multi-linear dot product to characterize three-way interactions among entities and relations. ComplEx [3] represents entities and relations as complex-valued vectors, achieving an optimal balance between accuracy and efficiency. HolE [15] utilizes cross-correlation, the inverse of circular convolution, for matching entity embeddings. More recently, SEEK [17] proposes a framework for modeling segmented knowledge graph embeddings and demonstrates that several existing models, including DistMult, ComplEx, and HolE, can be considered special cases within this framework.
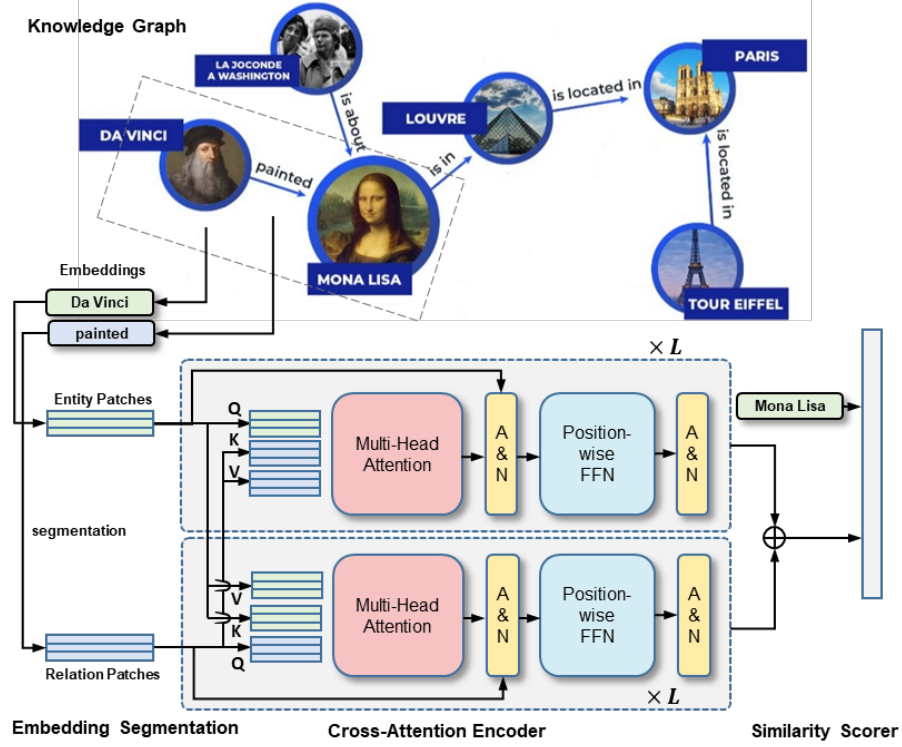
Neural-based methods. Neural network (NN) based methods have also been explored. Approaches such as [35,36] employ a Multi-Layer Perceptron (MLP) to model the scoring function. Moreover, Convolutional Neural Networks (CNN) have been utilized for KGC tasks. ConvE [8] aplies convolutional filters over reshaped head and relation embeddings to compute an output vector, which is then compared with all other entities in the knowledge graph. Subsequent works, including ConvR [6] and InteractE [9] enhance ConvE by fostering interactions between head and relation embeddings.

Transformer-based methods. The Transformer model known for employing self-attention to process token sequences has achieved remarkable success in NLP tasks. This success is attributed not only to its capacity for handling long-range dependencies but also to its tokenization concept. Recently, this concept has been extended to other domains, such as computer vision through Vision Transformers [28] and multi-modality with Two-stream Transformers [31]. These approaches have a common thread: they decompose the data (text or images) into smaller patches and process them using attention mechanisms. In the field of KGC, recent works have incorporated textual information and viewed entity and relation as the corresponding discrete descriptions. These methods often utilize pre-trained Transformers for encoding. However, high-quality textual KG data is not always accessible. As a result, our proposed method eschews additional textual information, instead integrating the tokenization concept into KGC to enhance performance.

3   Method

3.1   Knowledge Graph Completion

A *Knowledge Graph* can be represented as $(\mathcal{E}, \mathcal{R}, \mathcal{T})$ where $\mathcal{E}$ and $\mathcal{R}$ denote the sets of entities and relations respectively. $\mathcal{T}$ is a collection of tuples $[(h, r, t)_i]$ where head and tail entity $h, t \in \mathcal{E}$ and relation $r \in \mathcal{R}$. The task of *Knowledge Graph Completion*

Fig. 1: An overview of **PatReFormer**

includes the head-to-tail prediction (e.g., predicting the head entity h in the query $(?, r, t)$) and the tail-to-head prediction (e.g., predicting the tail entity t in the query $(h, r, ?)$).

In this paper, following previous works [1,8,6], we represent head and tail entities h and t as $\boldsymbol{e}_h$ and $\boldsymbol{e}_t \in \mathbb{R}^{d_e}$ and relation r as $\boldsymbol{e}_r \in \mathbb{R}^{d_r}$. Our objective is to learn a function $\mathrm{F} : \mathbb{R}^{d_e} \times \mathbb{R}^{d_r} \to \mathbb{R}^{d_e}$ such that given tuple (h, r, t), the output of $\mathrm{F}(\boldsymbol{e}_h, \boldsymbol{e}_r)$ closely approximates $\boldsymbol{e}_t$. For tail-to-head prediction, we additionally generate the reversed tuple (t, r^{-1}, h) and train the output of $\mathrm{F}(\boldsymbol{e}_t, \boldsymbol{e}_{r^{-1}})$ to be closed to $\boldsymbol{e}_h$.

3.2 **PatReFormer**

In this section, we will introduce the details of **PatReFormer**. Fig.1 shows the overview of our **PatReFormer** model, which comprises three components: *Embedding Segmentation*, *Cross-Attention Encoder*, and *Similarity Scorer*.

Embedding Segmentation. At this stage, **PatReFormer** converts entity and relation embeddings into sequences of patches. Formally, a segmentation function $pat(\cdot)$ is defined as follows:

$$\boldsymbol{p}_0, \boldsymbol{p}_1, \cdots, \boldsymbol{p}_k = pat(\boldsymbol{e}) \qquad (1)$$
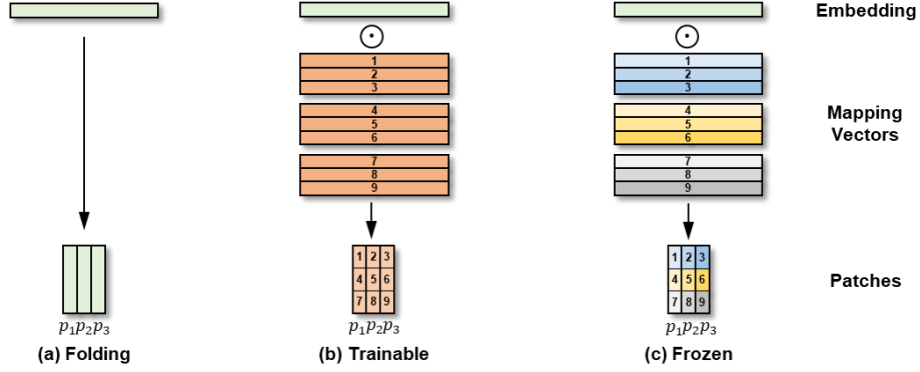
Fig. 2: Variants for Embedding Segmentation. $\odot$ denotes dot product operation. The mapping vectors with similar color (blue, yellow, grey) of frozen segmentation are mutually orthogonal.

where $\boldsymbol{e} \in \mathbb{R}^{k \cdot d}$ is the input entity or relation embeddings. $\boldsymbol{p}_i \in \mathbb{R}^d$ are segmented patches. k is the sequence length of the generated patches and d is the dimension of each patch. Our method considers three segmentation variants, as shown in Fig. 2:

Folding involves reshaping the original embeddings $\boldsymbol{e}$ into a sequence of equally-sized, smaller patches. Formally,

$$pat(\cdot) : \boldsymbol{p}_{i,j} = \boldsymbol{e}_{i*d+j} \tag{2}$$

Trainable Segmentation employs a set of mapping vectors v with adaptable parameters, enabling the model to learn and optimize the mapping function during training. This function can be written as:

$$pat(\cdot) : \boldsymbol{p}_{i,j} = u_{i,j} \odot \boldsymbol{e} \tag{3}$$

where $u_{i,j}$ are trainable vectors.

Frozen Segmentation utilizes the function with fixed parameters, precluding updates during the training process. Notably, the frozen Segmentation function comprises a set of matrices populated with mutually orthogonal vectors. This design choice aims to facilitate the generation of embedding patches that capture distinct aspects of an entity or relation, thereby enhancing the model's ability to represent diverse features. The patches are generated by:

$$pat(\cdot) : \boldsymbol{p}_{i,j} = u_{i,j} \odot \boldsymbol{e} \text{ , where } u_{i,j} \odot u_{i,k} = 0 \text{ for all } j, k \tag{4}$$

The value of $u_{i,j}$ is obtained from the orthogonal matrix U_i, which is generated through singular value decomposition (SVD) of a randomly initialized matrix M_i. i.e.,

$$M_i = U_i \Sigma V_i^{\top} \tag{5}$$

Cross-Attention Patch Encoder. After segmenting entity and relation embedding into patches, we then aggregate these patches together via *Cross-Attention Patch Encoder* which is based on a Siamese-Transformer architecture. We will discuss its details below.

Positional Embedding. The original Transformer model encodes them with either fixed or trainable positional encoding to preserve ordering information. However, unlike visual patches from images or words from the text, in the `PatReFormer` model, the patches from embeddings do not hold any much spatial information (i.e., the values in the first and last dimension alone do not carry particular semantic meaning). We thus remove the positional embedding in `PatReFormer`. We verify the effectiveness of this design in Section 5.

Cross-Attention Layer. Our proposed cross-attention layer process the entity and relation patches interactively with two separated attention modules:

$$
\boldsymbol{h}_h^i = \begin{cases} \mathrm{MHA}_{\mathrm{ER}}^{\mathrm{i}}(\boldsymbol{h}_h^{i-1}, \boldsymbol{h}_r^{i-1}, \boldsymbol{h}_r^{i-1}) & i > 0 \\ pat(e_h) & i = 0 \end{cases} \tag{6}
$$

$$
\boldsymbol{h}_r^i = \begin{cases} \mathrm{MHA}_{\mathrm{RE}}^{\mathrm{i}}(\boldsymbol{h}_r^{i-1}, \boldsymbol{h}_h^{i-1}, \boldsymbol{h}_h^{i-1}) & i > 0 \\ pat(e_r) & i = 0 \end{cases} \tag{7}
$$

where $\boldsymbol{h}_h^i, \boldsymbol{h}_r^i$ denote hidden representation of the i-th layer for head entity and relation respectively. $\mathrm{MHA}_{\mathrm{ER}}$ and $\mathrm{MHA}_{\mathrm{RE}}$ denotes Entity-to-Relation and Relation-to-Entity Attention module respectively. Both modules are based on the multi-head attention (MHA) mechanism, though they have different sets of parameters and inputs. The MHA module operates as follows:

$$
\mathrm{MHA}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, \mathrm{head}_2, \cdots, \mathrm{head}_{\mathrm{H}})W^o, \tag{8}
$$

$$
\text{where } \mathrm{head}_{\mathrm{i}} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{9}
$$

$W_i^Q \in \mathcal{R}^{d \times d_s}, W_i^K \in \mathcal{R}^{d \times d_s}, W_i^V \in \mathcal{R}^{d \times d_s}$ are projection matrix. $d_s = d/H$ where H is the predefined number of attention heads. $\mathrm{Attention}(\cdot)$ is the scaled dot-product attention module:

$$
\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^\intercal}{\sqrt{d}})V \tag{10}
$$

where $Q \in \mathcal{R}^{N \times d}$, $K \in \mathcal{R}^{M \times d}$, $V \in \mathcal{R}^{M \times d}$, and N and M denote the lengths of queries and keys (or values).

Position-wise Feed-Forward Network Layer. The position-wise feed-forward network (FFN) refers to fully connected layers, which perform the same operation on each position of the input independently.

$$
\mathrm{FFN(X)} = \mathrm{ReLU}(XW_1 + b_1)W_2 + b_2 \tag{11}
$$

where X is the output of the *Cross-Attention Layer* i.e., $\boldsymbol{h}_h^i$ or $\boldsymbol{h}_r^i$. $W_1 \in \mathcal{R}^{d \times d_f}$, $b_1 \in \mathcal{R}^{d_f}$, $W_2 \in \mathcal{R}^{d_f \times d}$, $b_2 \in \mathcal{R}^d$ are trainable weights and bias. To facilitate the optimization on deep networks, `PatReFormer` employs a residual connection [29] and Layer Normalization [30] on *Corss-Attention Layer* and *FFN*.

Similarity Scorer. We employ a scoring function to evaluate the relevance between the output from the Cross-Attention Encoder and the target entity embedding. Specifically, we concatenate the hidden representations obtained from the two Transformers sub-modules and project them back to the entity dimension using a linear layer.

$$\boldsymbol{e}' = \text{Concat}(\overline{X_e}, \overline{X_r})W_o + b_o \tag{12}$$

In this context, $W_o \in \mathcal{R}^{(d_e+d_r) \times d_e}$, $b_o \in \mathcal{R}^{d_e}$ are weights and bias of the linear layer, respectively. $\bar{\cdot}$ is the operation to reshape Transformer output into a vector. Subsequently, we compute the dot product of the projected vector $\boldsymbol{e}'$ and the target entity embedding $\boldsymbol{e}_t$. A sigmoid function is then applied to the result to ensure the final output falls within the $[0, 1]$ range. This scorer can be expressed as:

$$s = \text{Sigmoid}(\boldsymbol{e}' \odot \boldsymbol{e}_t) \tag{13}$$

Algorithm 1 provides a full procedure of our proposed **PatReFormer** method.

Algorithm 1 PatReFormer for Computing the Score of a Triple in a KG

Input: Embedding for entities and relations, E and R; head entity h, relation r and tail entity t; tokenization function $tok(\cdot)$
Output: the score of triple (h, r, t)
1: $\boldsymbol{e}_h, \boldsymbol{e}_r, \boldsymbol{e}_t \leftarrow E.get(h), R.get(r), E.get(t)$ # get embeddings for h, r and t
2: $\boldsymbol{e}_h \leftarrow tok(\boldsymbol{e}_h)$
3: $\boldsymbol{e}_r \leftarrow tok(\boldsymbol{e}_r)$
4: **for** $i = 1 \ to \ L$ **do**
5:     $\boldsymbol{e}_h \leftarrow \text{LayerNorm}(\text{MHA}(\boldsymbol{e}_h, \boldsymbol{e}_r, \boldsymbol{e}_r) + \boldsymbol{e}_h)$
6:     $\boldsymbol{e}_h = \text{LayerNorm}(\text{FFN}(\boldsymbol{e}_h) + \boldsymbol{e}_h)$
7:     $\boldsymbol{e}_r \leftarrow \text{LayerNorm}(\text{MHA}(\boldsymbol{e}_r, \boldsymbol{e}_h, \boldsymbol{e}_h) + \boldsymbol{e}_r)$
8:     $\boldsymbol{e}_r = \text{LayerNorm}(\text{FFN}(\boldsymbol{e}_r) + \boldsymbol{e}_r)$
9: **end for**
10: $e' \leftarrow \text{Concat}(\overline{e_h}, \overline{e_r})W_o + b_o$
11: $s \leftarrow \text{Sigmoid}(e' \odot \boldsymbol{e}_t)$
12: **return** s

3.3 Training and Inference

For training, we leverage the standard binary cross entropy loss with label smoothing:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(s_i) + (1 - y_i) \log(1 - s_i)] \tag{14}$$

where p_i and y_i are the score and label of the i-th training instance respectively. $y_i \in [\epsilon, 1-\epsilon]$, where ϵ is the label smoothing value. For inference, **PatReFormer** computes the scores of the query $(h, r, ?)$ for every possible entities and rank them based on the corresponding scores. More details are presented in Section 4.1.

4 Experimental Results

In this section, we evaluate **PatReFormer** against various baselines in the KGC task on multiple benchmark KGs.

4.1   Experimental Setup

Dataset.   Our proposed method is evaluated on four publicly available benchmark datasets: FB15K-237 [19], WN18RR [8], YAGO37 [20] and DB100K [21]. A summary of these datasets is provided in Table 1. FB15K-237 and WN18RR are widely-used benchmarks derived from FB15K and WN18 [1], respectively. They are free from the inverting triples issue. FB15K-237 and WN18RR were created by removing the inverse relations from FB15K and WN18 to address this issue. DB100K and YAGO37 are two large-scale datasets. DB100K was generated from the mapping-based objects of core DBpedia [22], while YAGO37 was extracted from the core facts of YAGO3 [23].

Table 1: Statistics of datasets.

Dataset	#Ent	#Rel	#Train	#Valid	#Test
WN18RR	$40,943$	11	$86,835$	$3,034$	$3,134$
FB15K-237	$14,541$	237	$272,115$	$17,535$	$20,466$
DB100K	$99,604$	470	$597,572$	$50,000$	$50,000$
YAGO37	$123,189$	37	$989,132$	$50,000$	$50,000$

Evaluation protocol.   Our experiment follows the filtered setting proposed in [1]. Specifically, for each test triple (h, r, t), two types of triple corruption are considered, i.e., tail corruption $(h, r, ?)$ and $(t, r^{-1}, ?)$. Every possible candidate in the knowledge graph is used to replace the entity, forming a set of valid and invalid triples. The goal is to rank the test triple among all the corrupted triples. In the filtered setting, any true triples observed in the train/validation/test set except the test triple (h, r, t) are excluded during evaluation. The evaluation metrics include the mean reciprocal rank (MRR) and the proportion of correct entities ranked in the top n (H@n) for $n = 1, 3, 10$. The evaluation is performed over all test triples on both types of triple corruption.

Table 2: Optimal hyperparameters for various KGC benckmarks

	η	L	d_e	d_r	p_1	p_2	p_3
WN18RR	1e-3	2	100	5000	0.1	0.1	0.4
FB15K-237	1e-3	12	100	3000	0.3	0.1	0.4
DB100K	5e-4	4	200	5000	0.1	0.1	0.4
YAGO37	1e-4	4	200	1000	0.1	0.1	0.1

Implementation details.   We implement **PatReFormer** in PyTorch[4]. In this experiment, we fix mini-batch size $\mathcal{B}$ to 256, Transformer dimensions d to 50, and label smoothing value ϵ to 0.1. The other hyper-parameters are tuned via grid search. Specifically, we select learning rate η from {1e-4, 5e-4, 1e-3}, number of layers L from {2, 4, 12}, entity

---

[4] https://pytorch.org/

Table 3: Experimental results of baseline models on FB15K-237, WN18RR.

	FB15K-237				WN18RR			
	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10
Non-Transformer-Based Methods								
TransE [1]	.279	.198	.376	.441	.243	.043	.441	.532
DistMult [2]	.241	.155	.263	.419	.430	.390	.440	.490
ComplEx [3]	.247	.158	.275	.428	.440	.410	.460	.510
R-GCN [4]	.249	.151	.264	.417	-	-	-	-
SACN [5]	.350	.260	.390	.540	.470	.430	.480	.540
ConvR [6]	.350	.261	.385	.528	.475	.443	.489	.537
RotatE [7]	.338	.241	.375	.533	.476	.428	.492	.571
ConvE [8]	.325	.237	.356	.501	.430	.400	.440	.520
InteractE [9]	.354	.263	-	.535	.463	.430	-	.528
AcrE [10]	.358	.266	.393	.545	.459	.422	.473	.532
Transformer-based methods								
KG-BERT [11]	-	-	-	.420	.216	.041	.302	.524
MTL-KGC [12]	.267	.172	.298	.458	.331	.203	.383	.597
StAR [13]	.296	.205	.322	.482	.401	.243	.491	**.709**
GenKGC [14]	-	.192	.355	.439	-	.287	.403	.535
PatReFormer (ours)	**.364**	**.271**	**.400**	**.551**	**.480**	.439	**.499**	.558

embedding size d_e $\{100, 200\}$, relation embedding size d_r $\{1000, 3000, 5000\}$. All dropout ratios, i.e., p_1 on embedding patches, p_2 on Cross-Attention Encoder and p_3 on the linear layer in Similarity Scorer, are tuned in $\{0.1, 0.2, 0.3, 0.4\}$. We use Adam [18] to optimize our model. On each dataset, we select the optimal configuration according to the best MRR on the validation set within 2500 epochs. The optimal configurations of **PatReFormer** on the four datasets are listed in Table 2.

4.2 Experimental Results

Table 3 presents a comprehensive comparison of our proposed **PatReFormer** model, against the baseline models on two popular FB15K-237 and WN18RR benchmarks. Our experimental results indicate that **PatReFormer** is highly competitive against the state-of-the-art models. Specifically, **PatReFormer** achieves improvements of 0.009 in MRR and 0.6% in H@10 compared to the previous models used on WN18RR. On WN18RR, **PatReFormer** obtains better results in terms of MRR (0.480 vs. 0.476) and H@3 (0.499 vs. 0.492) and is competitive in the H@10 and H@1 metrics. We attribute this discrepancy to the fact that the WN18RR dataset is a lexicon knowledge graph that relies heavily on textual information. As a result, the KGC models that incorporate pre-trained language models, such as StAR and MTL-KGC, achieve better performance than **PatReFormer** in those metrics.

To further verify the effectiveness of **PatReFormer** on larger KG, we evaluate our method on DB100K and YAGO37. Table 4 presents the performance comparison of **PatReFormer** with other baseline KGC models. On both benchmarks, **PatReFormer** outperforms existing methods on all evaluation criteria. In particular, **PatReFormer** demonstrates superiority on YAGO37 with a significant relative improvement of 15.2% (0.523 vs 0.454) and 5.5% (0.656 vs 0.622) in MRR and H@10

Table 4: Experimental results of several models evaluated on DB100K, YAGO37.

	DB100K				YAGO37			
	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10
TransE [1]	.111	.016	.164	.270	.303	.218	.336	.475
DistMult [2]	.233	.115	.301	.448	.365	.262	.411	.575
HolE [15]	.260	.182	.309	.411	.380	.288	.420	.551
ComplEx [3]	.242	.126	.312	.440	.417	.320	.471	.603
Analogy [16]	.252	.142	.323	.427	.387	.302	.426	.556
SEEK [17]	.338	.268	.370	.467	.454	.370	.498	.622
AcrE [10]	.413	.314	.472	.588	-	-	-	-
PatReFormer (ours)	**.436**	**.353**	**.479**	**.589**	**.523**	**.449**	**.567**	**.656**

respectively. These findings indicate the feasibility and applicability of **PatReFormer** on real-world large-scale knowledge graphs.

5 Analysis

In this section, we investigate **PatReFormer** from various perspectives. In the first place, we show the effectiveness of the design choices in **PatReFormer**. We then show that **PatReFormer** is capable of capturing more knowledge via a large embedding dimension. Finally, we demonstrate the advantages of **PatReFormer** in complex knowledge relations. All experiments are conducted on FB15K-237.

5.1 Impact of Cross Attention

In this section, we aim to examine the effectiveness of cross-attention in our proposed model by comparing it with two variants: 1) full self-attention, in which entity and relation patches are combined together before being fed into the model, and full self-attention is applied on the combined input; and 2) separate self-attention, in which each Transformer conducts self-attention on entity and relation patches independently before concatenating their results in the Similarity Scorer. The experimental results demonstrate that our proposed cross-attention method outperforms both the full self-attention and separate self-attention variants. We hypothesize that the cross-attention mechanism only learns to connect patches from different embeddings (i.e., patches from the same embedding never interact with each other), avoiding unnecessary interference from a single embedding. This could be the primary reason why cross-attention outperforms the full self-attention variant. Furthermore, the separate self-attention variant lacks interaction between entities and relations, which could explain the significant performance drop.

5.2 Impact of Positional Encoding

The original Transformer model [24] involves positional encoding to convey positional information of sequential tokens. To examine the impact of positional encoding on **PatReFormer**, we conduct an experiment with two variants: 1) trainable positional

Table 5: Analysis for model structure on FB15K-237. Att. denotes attention.

	MRR	H@1	H@3	H@10
PatReFormer	.3640	.2708	.3997	.5506
Full Self-Att.	$.3599_{\downarrow.0041}$	$.2656_{\downarrow.0052}$	$.3966_{\downarrow.0031}$	$.5476_{\downarrow.0030}$
Sep. Self-Att.	$.3387_{\downarrow.0253}$	$.2503_{\downarrow.0205}$	$.3698_{\downarrow.0299}$	$.5161_{\downarrow.0345}$

encoding (TPE) and 2) fixed positional encoding (FPE). Our experimental results demonstrate that the model without positional encoding (**PatReFormer**) outperforms the other two variants. We believe that this is due to the nature of embeddings patches, which inherently capture the features of entities or relations in a non-sequential manner. As a result, integrating positional encoding into the model introduces extraneous positional information, causing a decline in performance.

Table 6: Analysis for positional encoding (PE) on FB15K-237. Our proposed **PatReFormer** does not apply PE.

Models	MRR	H@1	H@3	H@10
PatReFormer	.3640	.2708	.3997	.5506
w/ TPE	$.3354_{\downarrow.0286}$	$.2474_{\downarrow.0234}$	$.3660_{\downarrow.0337}$	$.5107_{\downarrow.0399}$
w/ FPE	$.2580_{\downarrow.1060}$	$.1897_{\downarrow.0811}$	$.2789_{\downarrow.1208}$	$.3907_{\downarrow.1599}$

5.3   Impact of Segmentation

In this section, we explore the impact of segmentation on our proposed model, specifically examining the performance without using segmentation, and employing folding, trainable, and frozen segmentation. Our experimental results in Table 7 present that the utilization of segmentation yields a substantial performance improvement. With respect to the segmentation methods, frozen segmentation outperforms the other two variants. We believe this is due to the orthogonal vectors employed in frozen segmentation, which enhance the model's capacity to discern features of embeddings from distinct perspectives. Conversely, trainable segmentation, which allows parameters freely update during training, may face difficulties in achieving this. These findings emphasize the importance of selecting segmentation variants in the context of knowledge graph completion tasks. The superior performance of frozen segmentation suggests that these orthogonal vectors can be advantageous in extracting diverse features from entity and relation embeddings.

Table 7: Analysis for tokenization variants on FB15K-237.

	MRR	H@1	H@3	H@10
PatReFormer	.3640	.2708	.3997	.5506
w/o Seg.	$.3501_{\downarrow.0139}$	$.2592_{\downarrow.0116}$	$.3850_{\downarrow.0147}$	$.5316_{\downarrow.0190}$
Folding Seg.	$.3623_{\downarrow.0017}$	$.2695_{\downarrow.0013}$	$.3979_{\downarrow.0018}$	$.5488_{\downarrow.0018}$
Trainable Seg.	$.3572_{\downarrow.0068}$	$.2642_{\downarrow.0066}$	$.3936_{\downarrow.0061}$	$.5433_{\downarrow.0073}$

5.4   Effectiveness of `PatReFormer` via a Large Relation Embedding Dimension

In a typical KG, the number of relations is much less than the number of entities. Thus, we hypothesize that the KGC models that can effectively handle a large relation embedding dimension should achieve superior KGC performance. We verify this hypothesis in this section. Fig. 3 shows a clear performance increasing trend for `PatReFormer` as the length of relation embeddings increases. However, the other baseline KGC models, such as TransE, ConvE, and RotatE, do not deliver similar improvement; RotatE even suffers from performance delegations after the embedding dimension increases. This result shows that `PatReFormer` could capture more knowledge by using a large embedding dimension, while other methods cannot due to their insufficient modeling expressiveness. Such ability allows `PatReFormer` to capture more knowledge for relation embeddings and achieve better performance.
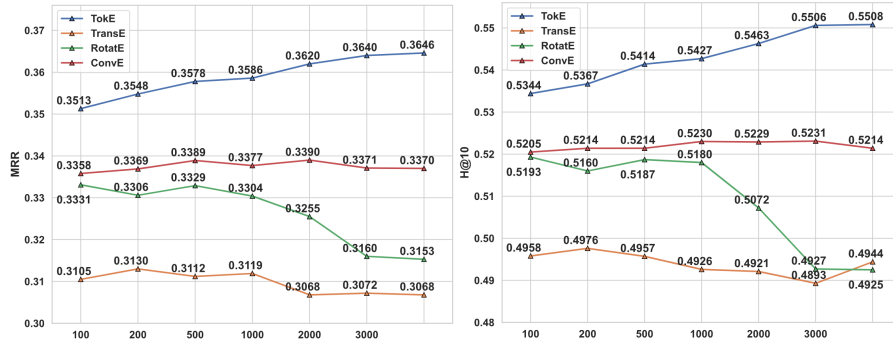

Fig. 3: Analysis of relation embedding size on FB15K-237

5.5   Analysis on Different Types of Relations

In this section, we analyze the performance of different types of relations for various models: TransE, ConvE, RotatE and `PatReFormer`. To categorize the relations, we considered the average number of tails per head and heads per tail, grouping them into four distinct types: 1-1 (one-to-one), 1-N (one-to-many), N-1 (many-to-one), and N-N (many-to-many). The results presented in Table 8 demonstrate that our `PatReFormer` model outperforms the other models in handling more complex relation types, such as 1-N, N-1, and N-N. This indicates that the increased interaction in our model allows it to capture intricate relationships more effectively. We note that TransE and ConvE perform better for simpler one-to-one relations. We believe there could be two reasons behind this phenomenon: 1) TransE and ConvE are intrinsically adept at representing simple relations (i.e., one-to-one), and 2) the limited number of evaluation instances for this category might result in biased results. Despite this, this experiment verifies the strength of our proposed `PatReFormer` model in modeling complex relation types

and highlights its potential applicability to a wide range of more complicated KGC tasks.

Table 8: Experimental results by relation categories for KGC methods on FB15K-237.

	#triples	TransE		ConvE		RotatE		PatReFormer	
		MRR	H@10	MRR	H@10	MRR	H@10	MRR	H@10
1-1	192	**.4708**	.5520	.4384	.5546	.3315	.5078	.3339	**.5625**
1-N	1,293	.2388	.3650	.2532	.3789	.2719	.4017	**.2828**	**.4203**
N-1	4,185	.3975	.4972	.4151	.5187	.4207	.5168	**.4647**	**.5698**
N-N	14,796	.2877	.5063	.3133	.5315	.3167	.5337	**.3432**	**.5564**

6 Conclusion

In this paper, motivated by the recent advances in Transformers, we propose a novel Transformer-based Patch Refinement model `PatReFormer` for knowledge graph completion. `PatReFormer` includes three main components: Embedding Segmentation, Cross-Attention Encoder, and Similarity Scorer. We first segment the knowledge graph embeddings into patches and then apply a Transformer-based cross-attention encoder to model interaction between entities and relations. Finally, the Similarity Scorer combines the encoded representations to compute the similarity between inputs and target entities. The experiments on four benchmark datasets (WN18RR, FB15k-237, DB100K and YAGO37) show that our proposed `PatReFormer` outperforms existing state-of-the-art knowledge graph completion (KGC) approaches. These results validate the effectiveness of our approach and highlight the potential advantages of incorporating patch-based embeddings and cross-attention mechanisms in such tasks.

7 Acknowledgement

References

1. Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In NIPS, pages 1–9.
2. Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding entities and relations for learning and inference in knowledge bases. In ICLR, San Diego, CA, USA, May 7-9, 2015

3. Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In ICML, pages 2071–2080. PMLR.
4. Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In ESWC, pages 593–607. Springer.
5. Chao Shang, Yun Tang, Jing Huang, Jinbo Bi, Xiaodong He, and Bowen Zhou. 2019. End-to-end structure-aware convolutional networks for knowledge base completion. In Proceedings of the AAAI, volume 33, pages 3060– 3067.
6. Xiaotian Jiang, Quan Wang, and Bin Wang. 2019. Adaptive convolution for multi-relational learning. In Proceedings of the 2019 NAACL-HIT, pages 978–987.
7. Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. Rotate: Knowledge graph embedding by relational rotation in complex space. ICLR, New Orleans, LA, USA, May, 2019.
8. Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2d knowledge graph embeddings. In Proceedings of the AAAI, volume 32.
9. Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, Nilesh Agrawal, and Partha Talukdar. 2020. Interacte: Improving convolution-based knowledge graph embeddings by increasing feature interactions. In AAAI, volume 34, pages 3009–3016.
10. Feiliang Ren, Juchen Li, Huihui Zhang, Shilei Liu, Bochao Li, Ruicheng Ming, and Yujia Bai. 2020. Knowledge graph embedding with atrous convolution and residual learning. In Proceedings of COLING, Barcelona, Spain (Online), December 8-13, 2020, pages 1532–1543.
11. Liang Yao, Chengsheng Mao, and Yuan Luo. 2019. KG-BERT: BERT for knowledge graph completion. CoRR, abs/1909.03193.
12. Bosung Kim, Taesuk Hong, Youngjoong Ko, and Jungyun Seo. 2020. Multi-task learning for knowledge graph completion with pre-trained language models. In Proceedings of the 28th COLING 2020, Barcelona, Spain (Online), December 8-13, 2020, pages 1737–1743.
13. Bo Wang, Tao Shen, Guodong Long, Tianyi Zhou, Ying Wang, and Yi Chang. 2021. Structure-augmented text representation learning for efficient knowledge graph completion. In WWW '21, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021, pages 1737–1748.
14. Xin Xie, Ningyu Zhang, Zhoubo Li, Shumin Deng, Hui Chen, Feiyu Xiong, Mosha Chen, and Huajun Chen. 2022. From discrimination to generation: Knowledge graph completion with generative transformer. In WWW'22, Lyon, France, April, 2022, pages 162–165. ACM.
15. Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. 2016. Holographic embeddings of knowledge graphs. In Proceedings of the AAAI, volume 30.
16. Hanxiao Liu, Yuexin Wu, and Yiming Yang. 2017. Analogical inference for multi-relational embeddings. In ICML, pages 2168–2178. PMLR.
17. Wentao Xu, Shun Zheng, Liang He, Bin Shao, Jian Yin, and Tie-Yan Liu. 2020b. SEEK: segmented embedding of knowledge graphs. In Proceedings of the 58th ACL 2020, Online, July 5-10, 2020, pages 3888–3897.
18. Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In 3rd ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.
19. Kristina Toutanova and Danqi Chen. 2015. Observed versus latent features for knowledge base and text inference. In Proceedings of the 3rd workshop on continuous vector space models and their compositionality, pages 57–66.
20. Boyang Ding, Quan Wang, Bin Wang, and Li Guo. 2018. Improving knowledge graph embedding using simple constraints. In Proceedings of the 56th ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers, pages 110–121.
21. Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. 2018. Knowledge graph embedding with iterative guidance from soft rules. In Proceedings of the AAAI, volume 32.
22. Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. 2009. Dbpedia-a crystallization point for the web of data. Journal of web semantics, 7(3):154–165.

23. Farzaneh Mahdisoltani, Joanna Biega, and Fabian Suchanek. 2014. Yago3: A knowledge base from multilingual wikipedias. In 7th CIDR Conference.

24. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 5998–6008.

25. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 NAACL: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), pages 4171–4186.

26. Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. CoRR, abs/1907.11692.

27. Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res., 21:140:1–140:67.

28. Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In 9th ICLR 2021, Virtual Event, Austria, May 3-7, 2021.

29. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In 2016 IEEE Conference on CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, pages 770–778. IEEE Computer Society.

30. Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E.Hinton. 2016. Layer normalization. CoRR, abs/1607.06450.

31. Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. 2019. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In NIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pages 13–23.

32. Radford A, Kim JW, Hallacy C, Ramesh A, Goh G, Agarwal S, Sastry G, Askell A, Mishkin P, Clark J, Krueger G. Learning transferable visual models from natural language supervision. In ICML 2021 Jul 1 (pp. 8748-8763). PMLR.

33. Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. In Proceedings of AAAI, January 25-30, 2015, Austin, Texas, USA, pages 2181–2187. AAAI Press.

34. Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In Proceedings of AAAI, July 27 -31, 2014, Québec City, Québec, Canada, pages 1112–1119. AAAI Press.

35. Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In Proceedings of the 20th ACM SIGKDD, pages 601–610.

36. Srinivas Ravishankar, Chandrahas Dewangan, and Partha P. Talukdar. 2017. Revisiting simple neural networks for learning representations of knowledge graphs. In 6th AKBC@NIPS 2017, Long Beach, California, USA, December 8, 2017.

37. Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In ICML, pages 4055–4064. PMLR, 2018.