AutoAlign: Fully Automatic and Effective Knowledge Graph Alignment enabled by Large Language Models

Rui Zhang, Yixin Su, Bayu Distiawan Trisedya, Xiaoyan Zhao, Min Yang, Hong Cheng, and Jianzhong Qi

Abstract— The task of entity alignment between knowledge graphs (KGs) aims to identify every pair of entities from two different KGs that represent the same entity. Many machine learning-based methods have been proposed for this task. However, to our best knowledge, existing methods all require *manually crafted* seed alignments, which are expensive to obtain. In this paper, we propose the first fully automatic alignment method named AutoAlign, which does not require any manually crafted seed alignments. Specifically, for predicate embeddings, AutoAlign constructs a predicate-proximity-graph with the help of large language models to automatically capture the similarity between predicates across two KGs. For entity embeddings, AutoAlign first computes the entity embeddings of each KG independently using TransE, and then shifts the two KGs' entity embeddings into the same vector space by computing the similarity between entities based on their attributes. Thus, both predicate alignment and entity alignment can be done without manually crafted seed alignments. AutoAlign is not only fully automatic, but also highly effective. Experiments using real-world KGs show that AutoAlign improves the performance of entity alignment significantly compared to state-of-the-art methods.

Index Terms—knowledge base, entity alignment, attribute embeddings, knowledge graph, knowledge graph alignment, representation learning, deep learning, predicate proximity graph, large language model

1 Introduction

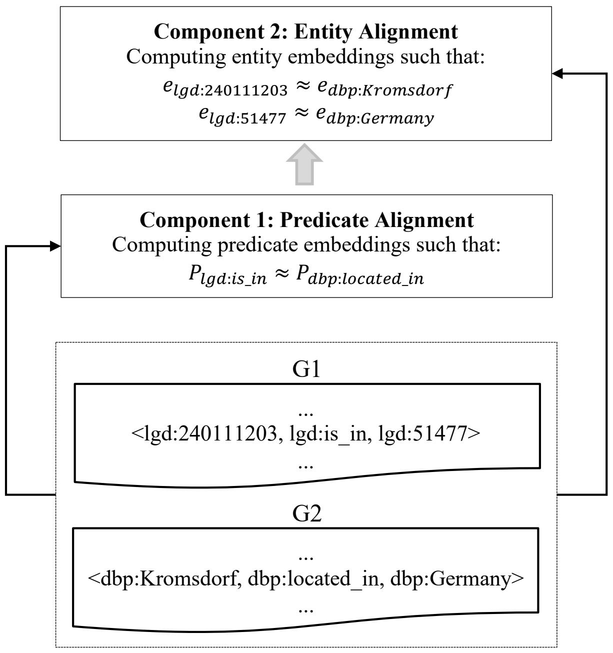
knowledge bases in the form of *knowledge graphs* (KGs) have been used in many applications, including question answering systems [1], dialogue systems [2], and recommender systems [3]. Many KGs have been created separately for particular purposes. The same real-world entity may exist in different forms in different KGs. For example, a village named `Kromsdorf` in Germany is a real-world entity that exists in two different KGs, LinkedGeoData [4] and DBpedia [5]. This entity is denoted in the form of `lgd:240111203` in LinkedGeoData but in the form of `dbp:Kromsdorf` in DBpedia. Usually, these KGs complement each other in terms of the number of entities each KG contains, and the types of information related to each entity. Therefore, we may merge two KGs into one with more entities and richer information related to each entity. To merge two KGs, a core task is *entity alignment*, which is to identify every pair of entities from the two KGs that correspond to the same real-world entity. Existing methods require significant manual work (e.g., manually crafted seed alignments), and the performance of the alignment is low. In this paper, we propose a novel method to this problem, which is fully automatic and effective (i.e., the alignment result is of high accuracy).

- R. Zhang is with Tsinghua University. E-mail: rayteam@yeah.net.
- Y. Su is with The University of Melbourne. E-mail: yixin.su@outlook.com.
- B.D. Trisedya is with Universitas Indonesia. E-mail: b.distiawan@cs.ui.ac.id.
- X. Zhao, and H. Cheng are with The Chinese University of Hong Kong, Hong Kong, China. E-mail: {xzhao, hcheng}@se.cuhk.edu.hk.
- M. Yang is with the Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China. E-mail: min.yang@siat.ac.cn.
- J. Qi is with The University of Melbourne. E-mail: jianzhong.qi@unimelb.edu.au.

TABLE 1: Knowledge graph alignment example.

$\mathcal{G}_1$
⟨`lgd:240111203,lgd:population,1595`⟩
⟨`lgd:240111203,rdfs:label,"Kromsdorf"`⟩
⟨`lgd:240111203,geo:lat,50.9988888889`⟩
⟨`lgd:240111203,lgd:alderman,"B. Grobe"`⟩
⟨`lgd:240111203,lgd:is_in,lgd:51477`⟩
$\mathcal{G}_2$
⟨`dbp:Kromsdorf,rdfs:label,"Kromsdorf"`⟩
⟨`dbp:Kromsdorf,geo:lat,50.9989`⟩
⟨`dbp:Kromsdorf,dbp:populationTotal,1595`⟩
⟨`dbp:Kromsdorf,dbp:located_in,dbp:Germany`⟩
⟨`dbp:Kromsdorf,dbp:district,dbp:Weimarer`⟩
Merged $\mathcal{G}_M$
⟨`lgd:240111203,:population,1595`⟩
⟨`lgd:240111203,:label,"Kromsdorf"`⟩
⟨`lgd:240111203,:lat,50.9988888889`⟩
⟨`lgd:240111203,:alderman,"B. Grobe"`⟩
⟨`lgd:240111203,:is_in,lgd:51477`⟩
⟨`lgd:240111203,:district,dbp:Weimarer`⟩

We use an example as shown in Table 1 to illustrate the entity alignment problem in detail. Typically, knowledge or real-world facts in KGs are stored in the form of triples, and a triple consists of three elements in the form of ⟨*head*, *predicate*, *tail*⟩, where *head* denotes an entity and *tail* denotes either another entity or a literal (attribute value) of the head entity. Here, if *tail* is an entity, the triple is called a *relation triple* and the predicate is called *relation predicate*; if *tail* is a literal, the triple is called an *attribute triple* and



Component 2: Entity Alignment
Computing entity embeddings such that:
$e_{lgd:240111203} \approx e_{dbp:Kromsdorf}$
$e_{lgd:51477} \approx e_{dbp:Germany}$
Component 1: Predicate Alignment
Computing predicate embeddings such that:
$P_{lgd:is\_in} \approx P_{dbp:located\_in}$
G1
...
<lgd:240111203, lgd:is_in, lgd:51477>
...
G2
...
<dbp:Kromsdorf, dbp:located_in, dbp:Germany>
...

Fig. 1: Components of knowledge graph alignment: *predicate alignment* and *entity alignment*.

the predicate is called *attribute predicate*. Table 1 gives an example of two subsets of triples from two KGs, denoted by $\mathcal{G}_1$ and $\mathcal{G}_2$ (we use prefixes `lgd:` and `dbp:` to simplify the original spell out). The head entities in these two subsets refer to the same entity `Kromsdorf`, even though they are in different forms, `lgd:240111203` and `dbp:Kromsdorf`. We aim to identify such entities and give them a unified ID such that both KGs can be merged together through them. In Table 1, $\mathcal{G}_M$ denotes the merged KG with entities aligned, where `lgd:240111203` is used as the unified ID for the entity `Kromsdorf` which has a set of properties that is the union of the sets of properties from both KGs.

As illustrated by the above example, to align entities, we also need to have the corresponding predicates aligned (e.g., `lgd:is_in` and `dbp:located_in`). The task of *knowledge graph alignment* is to have both entity alignment and predicate alignment between two KGs. Recent KG alignment methods are mainly based on representation learning [6]. Fig 1 shows the two critical components, *predicate alignment* and *entity alignment*: (i) the embeddings of predicates that represent the same relationship in the two KGs should have similar embeddings in the aligned vector space, e.g., `lgd:is_in` and `dbp:located_in` should have close embeddings, and (ii) if entity e_{g_1} from $\mathcal{G}_1$ corresponds to the same real-world entity as entity e_{g_2} from $\mathcal{G}_2$, then e_{g_1} should have similar embeddings to that of e_{g_2} in the aligned vector space, e.g., `lgd:240111203` and `dbp:Kromsdorf` in Fig. 1 should have similar embeddings.

There are mainly two paradigms of KG embedding, *translation-based methods* and *Graph Neural Network (GNN)-based methods*. See [6] for a comprehensive survey. Translation-based methods [7]–[9] first learn an embedding space for each KG separately, then learn a *transition matrix* to map the embedding space from one KG to the other. The mapping relies on large numbers of seed alignments (i.e., a set of manually crafted aligned triples from the two KGs) to compute the transition matrix. The other paradigm, GNN-based methods [10]–[13], aggregates information from the neighborhood of entities together with the graph structure to compute entity embeddings. Then they align the space of the two KGs via manually crafted seed alignments, which is similar to translation-based methods. Moreover, all the existing studies have focused only on entity alignment, whereas for predicate alignment, they also rely on manually crafted seeds. In summary, to our best knowledge, existing methods of both paradigms rely on manually crafted seed alignments.

Relying on manually created seed alignments have significant drawbacks: 1) Manually created seeds require careful human curation and usually domain expertise, which is expensive. For large datasets, a substantial number of manual alignments are required, which is prohibitive. 2) The portability of manually created seeds is poor. For each new alignment task, we have to manually create seeds again. 3) Different annotators have different biases and manually created seeds are error-prone, which results in manual seeds of highly varying quality and hence uncertain quality of alignment results.

To address the above problems, we propose a novel method for KG alignment that is not only *fully automatic* (i.e., not involving any manual seed alignments) but also much more accurate in aligning entities and predicates (i.e., more effective). We name our method AutoAlign as it is an automatic KG alignment method without needing the human annotation of seed alignments. For predicate alignment, AutoAlign constructs a predicate-proximity-graph to automatically capture the similarity between predicates across two KGs by learning the attentions of entity types. The predicate-proximity-graph construction is made automatic by leveraging recent large language models (such as ChatGPT and Claude) for aligning entity types of two KGs. For Entity alignment, AutoAlign computes the entity embeddings of each KG independently using TransE, and then shifts the two KGs' entity embeddings into the same vector space by computing the similarity between entities based on their attributes. The learning process of the above predicate alignment and entity alignment are jointly performed, which yields the final aligned KG.

Specifically, to achieve predicate embedding alignment without manually crafted seed alignments, we propose a *predicate-proximity-graph* for approximately computing the predicate embeddings, including both relation predicates and attribute predicates, where each predicate is a vertex that represents a relationship between entity types or literal types (instead of entities or literals). We create such a graph by replacing the head entity and tail entity of KG triples by their corresponding types, which are provided as `rdfs:type` relationship in the knowledge graph. For example, we replace the triples ⟨`dbp:Kromsdorf`, `dbp:located_in`, `dbp:Germany`⟩ and ⟨`lgd:240111203`, `lgd:is_in`, `lgd:51477`⟩ with the triples ⟨`village`, `dbp:located_in`, `country`⟩ and ⟨`village`, `lgd:is_in`, `country`⟩, respectively. Using the predicate-proximity-graph, AutoAlign can learn the similarity between predicates from two KGs that represent the same relationships, e.g., the predicates `dbp:located_in` and `lgd:is_in`.

Capturing predicate similarity in different KGs via a predicate-proximity-graph has a few **challenges**. First, each entity often has multiple types, which makes it difficult to directly align the predicates through the entity types. For example, the entity `Germany` may have multiple entity types {`thing`, `place`, `location`, `country`} in a KG. Second, different KGs may correspond to different sets of entity types, e.g., in another KG, the entity `Germany` may have entity types {`place`, `country`}. Hence, in the predicate-proximity-graph, the head entity and the tail entity may be replaced by multiple entity types. To address the above challenges, we propose two algorithms for aggregating multiple types of an entity and highlighting the most distinctive entity type (e.g., focusing more on `country` than on `thing`) via *pseudo-type embedding*, which is a representation obtained from aggregating multiple entity types' information according to their importance. Such an approximate predicate algorithm provides an automatic way of aligning predicates between two KGs, which not only complements the latent type information but also can be optimized by further joint learning for better predicate embeddings.

To achieve entity alignment, we exploit attribute triples and propose *attribute character embeddings* for capturing the similarity between attributes; entities that have similar attributes should also be similar. Before our work, there was one study that has proposed an embedding for attributes [14]. However, it only uses the attribute types for computing embedding, which loses all the content information of the attributes and is ineffective in capturing attribute (dis)similarity. *We are the first to propose attribute embedding that is based on the textual contents of the attributes [15].* Capturing the similarity of attributes of two KGs attribute similarity between entities in two KGs helps the attribute embedding to yield a unified embedding space for two KGs. This enables us to use attribute embeddings to shift the entity embeddings of two KGs into the same vector space and hence allows the entity embeddings to capture the similarity between entities from two KGs.

With the above two components, we achieve the first fully automatic method for KG alignment. The contributions of this paper are as follows.

C1: We propose AutoAlign, a fully automatic KG alignment method that aligns two KGs with no seed alignments required (neither predicate nor entity seed alignments). Specifically, we propose automatic predicate alignment algorithms, automatic entity alignment algorithms, and a scheme to perform joint learning of entity, attribute, and predicate embeddings.

C2: We are the first to propose attribute embedding based on the textual contents of the attributes, which enables automatic entity alignment.

C3: We propose an automatic predicate alignment algorithm, enabled by two techniques: (i) we use a predicate-proximity-graph powered by large language models to capture predicates as relationships of entity types, and (ii) we use pseudo-type embeddings that aggregate multiple entity types in the proximity graph as the vector representation for predicates.

C4: We conduct an extensive experimental study, which shows that our method is highly effective while being fully automatic. Compared to existing methods, which all rely on manually crafted seeds, AutoAlign outperforms the best baseline by up to 10.65% in hits@10.

This paper is an extended version of our earlier conference paper [15]. There, we presented the basic idea of the attribute character embeddings (**C2**). However, the method in the previous paper [15] requires manually crafted predicate alignments. Specifically, it uses edit distance to compute the similarity score between predicates, and manual inspection is required to remove false positives. In this journal extension, we have made substantial new contributions. First, we propose novel algorithms to align predicates without seed alignments including exploiting the most recent large language models (**C3**). Second, we propose a scheme to put all these components together and perform joint learning of entity, attribute and predicate embeddings, achieving a fully automatic KG alignment method (**C1**). Third, we have conducted a much more comprehensive experimental study, comparing with more baselines such as GNN-based ones, and using more recent benchmarks with more realistic and much larger datasets (**C4**).

2 Related Work

This section discusses the related work, including knowledge graph embedding methods and knowledge graph alignment methods.

2.1 Knowledge Graph Embedding Methods

Knowledge graph embedding methods are to address KB completion tasks [16] that aim to predict missing entities (i.e., head entity or tail entity) or relations (i.e., predicates) based on triples in a knowledge graph. TransE [17], is a simple yet effective knowledge graph embedding method. TransE represents a relationship between a pair of entities as a *translation* between the embeddings of the entities: a triple that consists of $\langle$`head`, `predicate`, `tail`$\rangle$ denoted as $\langle h, p, t\rangle$ should hold the property of $\mathbf{h} + \mathbf{p} \approx \mathbf{t}$. This representation indicates that the embedding vector of the tail entity approximately equals the embedding vector of the head entity after a vector sum operation of the embedding vector of predicate $\mathbf{p}$. A scoring function $f(\mathbf{h}, \mathbf{t}) = \|\mathbf{h} + \mathbf{p} - \mathbf{t}\|_2$ is used to measure the plausibility of a triple.There are subsequent variants of TransE such as TransH [18] and TransR [19] by capturing more complex relationships.

Another popular paradigm of KG embedding is via graph neural networks, such as Graph Convolutional Networks (GCN) [20] and Graph Attention Networks (GAT) [21]. Recently, graph embedding based on Transfomers [22] have been proposed such as HGT [23] and RHGT [24]. These methods learn entity embeddings via information propagation between nodes in a graph.

2.2 Knowledge Graph Alignment Methods

The embedding methods above aim to preserve the structural information of the entities, i.e., entities that share similar neighbor structures in the KB should have similar representations in the embedding space. The advancement of such embedding methods motivates researchers to study embedding-based entity alignment. Chen et al. [7] propose MTransE, an embedding-based method for multilingual entity alignment based on TransE. In the follow-up work, Chen et al. [8] propose a generalized affine-map-based method to improve the alignment method of MTransE for handling various forms of invertible transformations. Zhu et al. [9] propose an iterative method for entity alignment via joint embeddings. Sun et al. [14] propose a joint attribute embedding method (called JAPE) for cross-lingual entity alignment. JAPE improves the alignment by capturing the correlations of attributes via the attribute type similarity.

As the other popular paradigm for KG embedding, many GNN-based entity alignment methods have been proposed [10]–[13]. To improve the performance of GNN for entity alignment, existing methods [10] combine entity embeddings and attribute type embeddings in the computation of GNN. AttrGNN [25] performs entity alignment by focusing on attributes, values, and structures. UPLR [26] effectively learns alignment information from pseudo-labeled datasets containing noisy data. MHGCN [27] employs a multiview highway graph convolutional network to consider entity alignment from different views. These works share a common goal of leveraging different types of information present in knowledge graphs to improve entity alignment performance. However, our proposed method stands out by being fully automated and leveraging both predicate and attribute triples without requiring seed alignments or multiple views.

In summary, all existing embedding-based entity alignment methods require a set of manually crafted seed alignments. There has been no existing work on automatic predicate alignment. Due to the significant drawbacks of using seed alignments, our work focuses on automatic KG alignment methods. A recent survey [6] shows that translation-based methods outperform GNN-based ones in accuracy and efficiency, and our method AutoAlign is a translation-based one.

There are studies on cross-lingual entity alignment such as [14]. Recent studies [6], [28] have shown that aligning KGs originating from different sources but the same language is more common in real-life industrial applications [28], [29]; a recent benchmark DWY-NB has been proposed [6], which better reflects real-world scenarios for KG alignment and has addressed several limitations of existing cross-lingual KG alignment datasets such as DBP15K [14]. Therefore, in this paper, we focus on monolingual KG alignment rather than cross-lingual KG alignment. Moreover, [6] has shown that translating two languages into one and then performing monolingual alignment achieves similar accuracy to what direct cross-lingual entity alignment gets.

3 Preliminary

We start with the problem definition. A knowledge graph $\mathcal{G}$ consists of relation triples and attribute triples. A relationship triple is in the form of $\langle h, p, t \rangle$, where p is a relationship (*predicate*) between two entities h (*head*) and t (*tail*). An attribute triple is in the form of $\langle h, p, v \rangle$ where v is an attribute value of entity h with respect to the predicate p.

Given two knowledge graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, the task of entity alignment aims to find every pair $\langle h_1, h_2 \rangle$ where $h_1 \in \mathcal{G}_1$, $h_2 \in \mathcal{G}_2$, and h_1 and h_2 represent the same real-world entity. We use an embedding-based method that assigns a continuous representation for each element of a triple in the forms of $\langle \mathbf{h}, \mathbf{p}, \mathbf{t} \rangle$ and $\langle \mathbf{h}, \mathbf{p}, \mathbf{v} \rangle$, where the bold-face letters denote the vector representations of the corresponding element.

Our proposed method is built on top of a translation-based embedding method. We first discuss translation-based embedding methods and their limitations when being used for entity alignment before presenting our method.

3.1 Translation-based Embedding Method

Given a relationship triple $\langle h, p, t \rangle$, a translation-based embedding method, such as TransE [17], suggests that the embedding of the tail entity t should be close to the embedding of the head entity h plus the embedding of the relationship p, i.e., $\mathbf{h} + \mathbf{p} \approx \mathbf{t}$. Such an embedding method aims to preserve the structural information of the entities, i.e., entities that share similar neighbor structures in a knowledge graph should have similar representations in the embedding space. We refer to the modeling of the structural information as *structure embedding* and the modeling should preserve the translation property of $\mathbf{h} + \mathbf{p} \approx \mathbf{t}$. To learn the structure embedding, TransE minimizes a margin-based objective function $\mathcal{J}_{SE}$:

$$\mathcal{J}_{SE} = \sum_{t_r \in \mathcal{T}_r} \sum_{t'_r \in \mathcal{T}'_r} \max\left(0, [\gamma + f(t_r) - f(t'_r)]\right) \tag{1}$$

$$f(t_r) = \|\mathbf{h} + \mathbf{p} - \mathbf{t}\|_2 \tag{2}$$

$$\mathcal{T}_r = \{\langle h, p, t \rangle | \langle h, p, t \rangle \in \mathcal{G}\} \tag{3}$$

$$\mathcal{T}_r{}' = \{\langle h', p, t \rangle \mid h' \in \mathcal{E}\} \cup \{\langle h, p, t' \rangle \mid t' \in \mathcal{E}\} \tag{4}$$

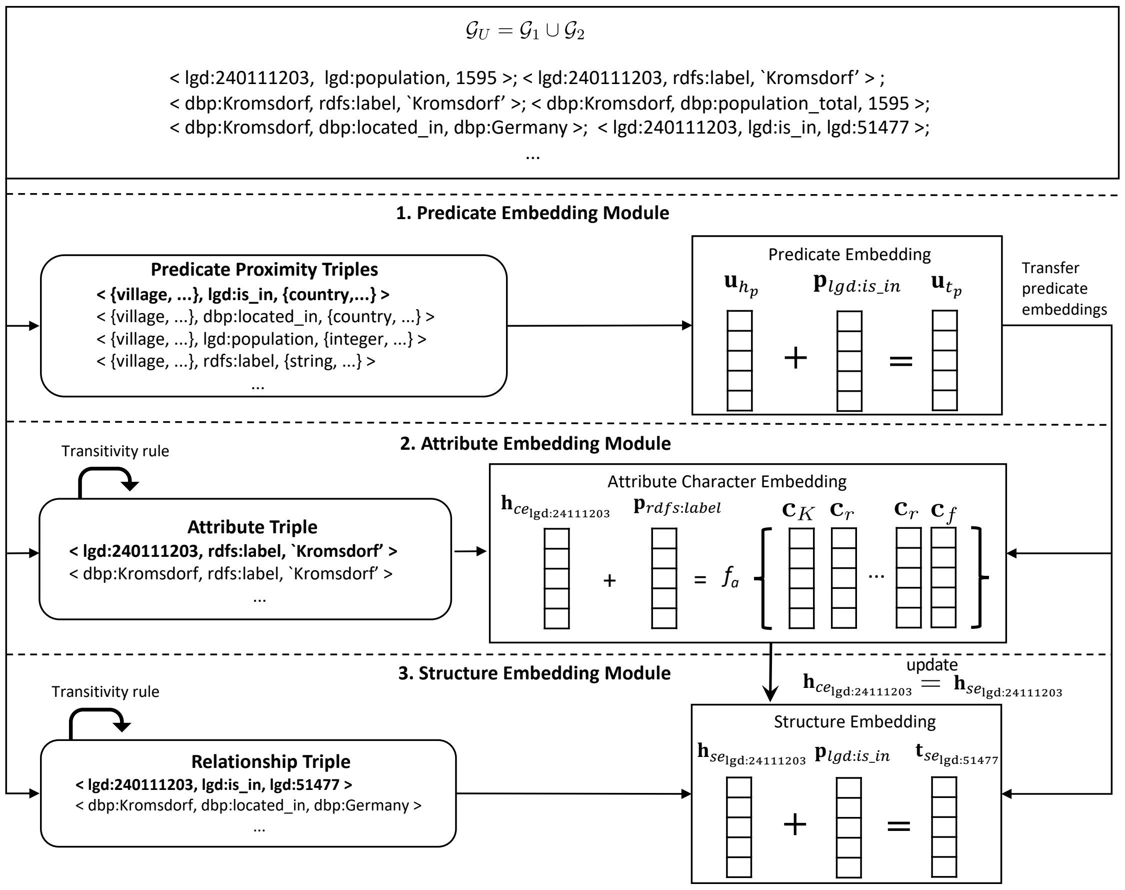
Here, $\|\mathbf{x}\|_2$ is the L2-Norm of vector $\mathbf{x}$, γ is a margin hyperparameter, $\mathcal{T}_r$ is the set of valid relation triples, and $\mathcal{T}'_r$ is the set of corrupted relation triples ($\mathcal{E}$ is the set of entities in $\mathcal{G}$). The corrupted triples are used as negative samples created by replacing the head or tail entity of a valid triple in $\mathcal{T}_r$ with a random entity.

The advantages of structure embeddings drive further studies of embedding-based entity alignment. However, a straightforward implementation of structure embedding for entity alignment has limitations: the entity embeddings computed on different KGs may fall in different spaces, where similarity cannot be computed directly. Existing methods [7], [9], [14] address this limitation by computing a transition matrix to map the embedding spaces of different KGs into the same space, as discussed earlier. However, such methods require manually collecting a seed set of aligned entities from the different KGs to compute the transition matrix, which does not scale and is vulnerable to the quality of the manually crafted seed aligned entities.

Next, we detail our method to address these limitations.

4 Proposed Method

We give an overview of AutoAlign in Section 4.1. We then explain the various components of AutoAlign, which include the *predicate embedding* module in Section 4.2, the *structure embedding* module in Section 4.3, the *attribute embedding* module in Section 4.4, the joint learning scheme in Section 4.5, entity alignment in Section 4.6, and triple enrichment in Section 4.7. We discuss the scalability of AutoAlign in Section 4.8.



$\mathcal{G}_U = \mathcal{G}_1 \cup \mathcal{G}_2$
< lgd:240111203, lgd:population, 1595 >; < lgd:240111203, rdfs:label, `Kromsdorf' > ;
< dbp:Kromsdorf, rdfs:label, `Kromsdorf' >; < dbp:Kromsdorf, dbp:population_total, 1595 >;
< dbp:Kromsdorf, dbp:located_in, dbp:Germany >; < lgd:240111203, lgd:is_in, lgd:51477 >;
1. Predicate Embedding Module
Predicate Proximity Triples
< {village, ...}, lgd:is_in, {country,...} >
< {village, ...}, dbp:located_in, {country, ...} >
< {village, ...}, lgd:population, {integer, ...} >
< {village, ...}, rdfs:label, {string, ...} >
Predicate Embedding
Transfer predicate embeddings
2. Attribute Embedding Module
Transitivity rule
Attribute Triple
< lgd:240111203, rdfs:label, `Kromsdorf' >
< dbp:Kromsdorf, rdfs:label, `Kromsdorf' >
Attribute Character Embedding
3. Structure Embedding Module
update
Transitivity rule
Relationship Triple
< lgd:240111203, lgd:is_in, lgd:51477 >
< dbp:Kromsdorf, dbp:located_in, dbp:Germany >
Structure Embedding

Fig. 2: Overview of our proposed AutoAlign method for entity alignment.

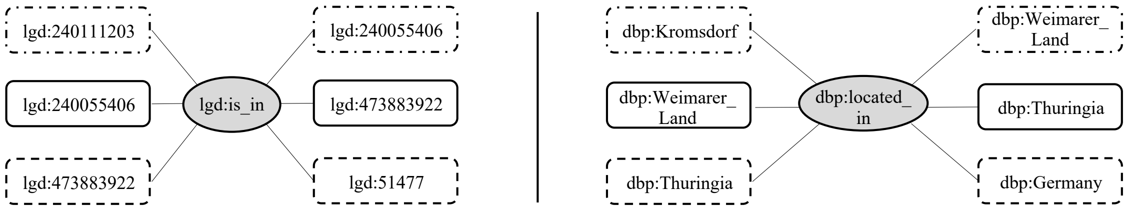


Fig. 3: Predicate graph of the similar relationship `lgd:is_in` (left) and `dbp:located_in` (right) in two KGs. Each type of dotted line represents the similar entity types in two predicate graphs.

4.1 Overview of AutoAlign

AutoAlign has three core embedding modules: the predicate embedding, the attribute embedding, and the structure embedding. Fig. 2 gives an overview and shows the interaction between the three modules.

In order to execute the embedding-based KG alignment, it's crucial to ensure that both predicate and entity embeddings of two KGs coexist in the identical vector space. To meet this criterion, we start by simply making a union of the two knowledge graphs, denoted as $\mathcal{G}_U = \mathcal{G}_1 \cup \mathcal{G}_2$. This put all the triples from both KGs together in their original form. Note that $\mathcal{G}_U$ is different from $\mathcal{G}_M$ as the entities in $\mathcal{G}_U$ are not aligned yet. We will obtain three types of triples from $\mathcal{G}_U$: predicate-proximity triples $\mathcal{T}_p$, relation triples $\mathcal{T}_r$, and attribute triples $\mathcal{T}_a$. In the set of predicate-proximity triples, each predicate corresponds to a connection between entity types. For instance, the set of predicate-proximity triples may contain triples such as ⟨`village`, `dbp:located_in`, `country`⟩ and ⟨`village`, `lgd:is_in`, `country`⟩. These triples help train a predicate embedding module (elaborated in Section 4.2) to capture the similarity between predicates from two KGs, such as `dbp:located_in` and `lgd:is_in`. This process not only generates a unified embedding space for predicates but also captures their similarity between the two KGs. Predicate embeddings are then utilized for computing attribute and structure embeddings.

The structure and entity embeddings are obtained from the set of relation triples $\mathcal{T}_r$ as discussed in Section 4.3, and the attribute embedding, as discussed in Section 4.4, leveraging the set of attribute triples $\mathcal{T}_a$. Initially, due to the unique naming schemes in each KG, entity embeddings from $\mathcal{G}_1$ and $\mathcal{G}_2$ exist in two different vector spaces. However, we unify the attribute embeddings derived from attribute triples $\mathcal{T}_a$ into a common vector space. This unification is via the character embeddings learned from attribute strings, which can manifest similarity despite their origin from different KGs. The obtained attribute embeddings are then utilized to align the entity embeddings into a common vector space, thus enabling the entity embeddings to reflect the similarity between entities across both KGs.

Upon acquiring the embeddings for all entities in $\mathcal{G}_1$ and $\mathcal{G}_2$, the entity alignment module (explained in Section 4.6) identifies every pair $\langle h_1, h_2 \rangle$, with $h_1 \in \mathcal{G}_1$ and $h_2 \in \mathcal{G}_2$, that has a similarity score exceeding a threshold β.

To bolster the effectiveness of AutoAlign, we implement the transitivity rule to expand an entity's properties, thereby fostering a more resilient attribute embedding for gauging entity similarities. This procedure is discussed in Section 4.7.

4.2 Predicate Embedding Module

The same predicates from two KGs typically connect the same entity type, albeit in different surface forms. Consider the example in Fig. 3, where the predicate `lgd:is_in` in LinkedGeoData and the predicate `dbp:located_in` in DBpedia connect three entity pairs. In LinkedGeoData, the predicate `lgd:is_in` connects ⟨`lgd:240111203`,`lgd:240055406`⟩, ⟨`lgd:240055406`, `lgd:473883922`⟩, and ⟨`lgd:473883922`,`lgd:51477`⟩. Meanwhile, in DBpedia, the predicate `dbp:located_in` connects ⟨`dbp:Kromsdorf`,`dbp:Weimarer_Land`⟩, ⟨`dbp:Weimarer_Land`,`dbp:Thuringia`⟩, and ⟨`dbp:-Thuringia`,`dbp:Germany`⟩. Here, the entity pairs from both knowledge graphs correspond to the same real-world entity pairs. For instance, the head and tail entities of the entity pair ⟨`lgd:240111203`,`lgd:240055406`⟩ and ⟨`dbp:Kromsdorf`,`dbp:Weimarer_Land`⟩ represent a village named Kromsdorf and a district named Weimarer Land, respectively. However, due to the distinct naming schemes of the two knowledge graphs, entity embedding methods may not capture this similarity. Applying an entity embedding method to the raw knowledge graph could result in the predicate embeddings of `lgd:is_in` and `dbp:located_in` being placed in different vector spaces.

In our previous work [15], a semi-automatic predicate alignment process was employed to tackle this challenge. This involved renaming akin predicates from the two KGs via a uniform naming scheme, thereby generating a shared vector space for relationship embeddings. To pinpoint similar predicates across the two KGs, we leveraged string edit distance and subsequently examed any false positives manually. However, such a method presents limitations in real-world applications, given its reliance on manual intervention for aligning predicates, or creating 'seed alignments', across the two KGs.

To address the above problem, AutoAlign introduces a fully automatic predicate alignment procedure by learning predicate embeddings from a *predicate-proximity-graph* of two KGs. The predicate-proximity-graph replaces entities with their corresponding entity types. Through learning the graph, we can effectively identify predicate similarities between two KGs without the need to manually compare predicates' surface forms. The automatic predicate alignment is detailed in Sections 4.2.1 and 4.2.2.

4.2.1 Predicate-proximity-graph Construction

A predicate-proximity-graph is essentially a graph depicting the relationships between *entity types* rather than *entities*. Entity types indicate the broad categories of entities, which automatically link different entities. Even if some predicates have different surface forms (e.g., `lgd:is_in` v.s. `dbp:located_in`), we can effectively identify them as being similar by learning the predicate-proximity-graph. This is because the head/-tail entities of these predicates usually have similar entity types (e.g., ⟨`place`, `lgd:is_in`,`country`⟩,⟨`place`, `dbp:located_in`,`country`⟩). To create the predicate-proximity-graph, we start with $\mathcal{G}_U$ the union of the two KGs' triples in their original forms, and then replace each triple's head and tail entities with their respective entity types. Next, the entity types are obtained through two steps: *entity type extraction* and *type alignment enabled by large language models*, which are described as follows.

1) Entity Type Extraction. We extract entity types by taking the values of the `rdfs:type` predicate for every entity from each KG. It's common for each entity to have multiple types. For instance, the entity `Germany` might have several entity types like `place`, `location`, `country` within a KG. Furthermore, varying KGs may adopt different schemes of entity types, e.g., in another KG, the entity `Germany` might have entity types like `place`, `country`. To accommodate these variations, within the predicate-proximity-graph, we substitute both the head and tail entities of each triple in a knowledge graph with a collection of entity types. For example, we replace the triples ⟨`dbp:Kromsdorf`, `dbp:located_in`, `dbp:Germany`⟩ with the triples ⟨$\mathcal{U}_{kromsdorf}$, `dbp:located_in`, $\mathcal{U}_{germany}$⟩. Here, $\mathcal{U}x$ is a set of types for entity x, e.g., $\mathcal{U}germany$= {`thing`, `place`, `location`, `country`}[1].

2) Type Alignment enabled by Large Language Models. After obtaining the entity types of the two KGs, we need to perform an alignment between the entity types of two KGs. This is because the types of two KGs may refer to the same meaning but using different surface forms, e.g., person v.s. people. Therefore, we need to align such types into one for the best effect of predicate-proximity-graph training. In previous work, we manually align the types since the number of the types is not large. With the recent breakthrough of large language models such as ChatGPT and Claude [30]–[32], we can eliminate such manual effort and make it fully automatic.

Specifically, we use Claude[2], which is a free and powerful large language model. We construct a prompt as input to Claude as follows:

1. We provide the details of automatically obtain and align the entity types from two KGs in Appendix A
2. https://www.anthropic.com/index/introducing-claude

"Now you are an expert in linguistics and knowledge graphs. I will give you two sets of words, indicating the entity types from two knwoledge graphs. You need to identify all the word pairs from the two sets that are synonyms. For example, if the first set has the word 'people' and the second set has the word 'person', you need to identify the two words being synonyms and return me the pair (people, person). Now the following are the two sets: Set 1: {people, music,...} Set 2: {person, thing,...} Please return all the pairs that are synonyms from the two sets reagarding entity types. Do not output the pairs if they are exactly the same. Remember you only need to return the pairs, each pair in one line. Each pair contain two types, one from Set 1 and another from Set 2, in the format (type1, type2)."

The type set in *Set 1* and *Set 2* can be filled with the types extracted from corresponding KGs, and the other contents are fixed for any two KGs. We feed the prompt to Claude, and it will return the pairs of types that are similar to each other in the format of (type1, type2). After identifying type1 of $\mathcal{G}_1$ is similar to type2 in $\mathcal{G}_2$, we replace all the type2 by type1 so that similar types are represented by the same surface form, e.g., replace "people" with "person" so that two KGs both use the type "person". This way we obtain type alignment without human intervention.

After extracting the entity types for each KG and aligning the extracted types enabled by LLMs, we obtain the predicate-proximity-graph that has the same number of triples as $\mathcal{G}_U$, with each triple's entity being replaced by their types.

4.2.2 Module Learning

To capture the predicate similarity, the module should focus on the most distinctive entity types, e.g., emphasizing `country` more than `thing`. We propose two ways for aggregating multiple entity types: 1) weighted sum function, and 2) attention-based function. In the experimental study, we show that the attention-based function works better.

1) **Weighted Sum Function:** Given the entity type embedding $\mathbf{U}_x = (\vec{\mathbf{z}}_0, \vec{\mathbf{z}}_1, \cdots, \vec{\mathbf{z}}_M)$ of entity type $z \in \mathcal{U}_x$ with M types, we calculate the *pseudo-type embedding* $\mathbf{u}$ as follows.

$$\mathbf{u} = \sum_{i=0}^{M} w_i \vec{\mathbf{z}}_i, \ w_i = \text{softmax}\left(\frac{l_i}{a_i r_i} k_i\right) \tag{5}$$

Here, the weight w_i controls the distribution of $\vec{\mathbf{z}}_i$, which is the vector representation of an entity type z in $\mathcal{U}_x$. To give a larger weight to the most distinctive type, we use l_i, which is the level of specificity of an entity type in WordNet [33]. An entity type with a deeper level of specificity has a larger value of l_i, e.g., `country` has a deeper level of specificity than `thing`. This way, the predicate embedding module can emphasize the most distinct type. We normalize the weight using three variables. The first is a_i, which is the number of attributes in $\mathcal{U}_x$. The second is r_i, which is the number of occurrences of type z in a KG. Intuitively, an entity type that appears in almost all entities, such as `thing`, is less distinctive, and the predicate embedding module should filter this entity type to obtain a better predicate representation. The third variable is k_i, which is the number of KGs that contain the entity type z to indicate the agreements between two KGs on the entity type z. Lastly, we use softmax to transform the weights into probability distributions for each type in $\mathcal{U}_x$.

2) **Attention-based Function:** In contrast to the most distinctive entity types, there may be some "noise" entity types that contribute little when learning the meaning of a predicate. For example, in the set of entity types for entity `Germany`, $\mathcal{U}_{germany}$= `thing`, `place`, `location`, `country`, the entity type `thing` $\in \mathcal{U}_{germany}$ may be less relevant to the triple ⟨`dbp:Kromsdorf`, `dbp:located_in`, `dbp:Germany`⟩. In this case, `thing` can be seen as a "noise" entity type that is not essential for learning the representation of predicate `dbp:located_in`.

To better capture the importance of different entity types for a predicate, we propose an attention-based algorithm, which allows the module to adaptively evaluate the entity type weight and ignore potential type noise. We calculate the attention weight z_i of the i-th entity type as:

$$z_i = softmax(U_x^T W_z \vec{\mathbf{z}}_i) \tag{6}$$

where W_z denotes the trainable weight matrix of entity type embedding $\vec{\mathbf{z}}_i$. The final pseudo-type embedding $\mathbf{u}$ is obtained through a weighted sum of all corresponding entity type vectors $\vec{\mathbf{z}}_i$ representing different semantic meanings:

$$\mathbf{u} = \sum_{i=0}^{M} z_i \vec{\mathbf{z}}_i \tag{7}$$

where $\vec{\mathbf{z}}_i$ is the embedding of the i-th entity type.

The pseudo-type embeddings computed by Equations 5 or 7 are used as the proximity entity embeddings, which are used next to train the predicate embeddings as follows.

We use the pseudo-type embeddings $\mathbf{u_{h_p}}$ and $\mathbf{u_{t_p}}$ to represent the corresponding head entity and tail entity in the predicate proximity triples $\mathcal{T}_p$, respectively. We then compute the predicate embeddings by minimizing the following objective function:

$$\mathcal{J}_{PE} = \sum_{t_p \in \mathcal{T}_p} \sum_{t'_p \in \mathcal{T}'_p} \max\left(0, \left[\gamma + f(t_p) - f(t'_p)\right]\right) \tag{8}$$

$$f(t_p) = \left\| \mathbf{u_{h_p}} + \mathbf{p} - \mathbf{u_{t_p}} \right\|_2 \tag{9}$$

where t_p is a triple in the predicate-proximity-graph and t'_p is a corrupted triple (i.e., for negative samples) generated based on the predicate-proximity-graph. Here, $\mathbf{u_{h_p}}$ and $\mathbf{u_{t_p}}$ can be obtained using the above two functions, Eq. 5 and Eq. 7. We use AutoAlign-W and AutoAlign-A to denote AutoAlign utilizing Eq. 5 and Eq. 7, respectively.

Note that the procedure discussed can be extended to compute embeddings for attribute predicates by modifying Eq. 9 to replace the entity types of the tail entity in relation triples with the literal types (e.g., string, integer, and long data type) of attribute values in attribute triples. Through optimizing the objective function, AutoAlign cultivates a unified predicate embedding space from two knowledge graphs. This method empowers us to transition these embeddings into the learning of structure and attribute embeddings.

4.3 Structure Embedding Module

Our method of computing the structure embeddings is built based on TransE. Although TransE typically assigns equivalent weights to each neighbor when computing an entity's embeddings, we adjust TransE to give different weights to an entity's different neighbors. The underlying reasoning is to assign higher weights to neighbors that are linked by predicates already aligned, as they serve as a significant indicator for entity alignment.

As illustrated in Table 1, we can categorize the predicates of KGs into three groups. The first group comprises the already aligned predicates, such as `geo:lat`, `geo:long`, and `rdfs:label`, which adhere to the predicate naming scheme convention[3] in knowledge graphs. The second group contains implicitly aligned predicates, such as `lgd:is_in` and `dbp:located_in`. These predicates are beneficial for entity alignment if we can identify the alignment between them. We address this issue with our predicate embedding module (see Section 4.2). The final group consists of non-aligned predicates, such as `lgd:alderman` and `dbp:district`. These predicates do not aid entity alignment and are treated as noise.

To mitigate the influence of noise, we adjust TransE by incorporating a weight factor α to govern the learning of embeddings across the triples. As a result, the entity embedding approach can sift out non-aligned triples grounded on non-aligned predicates. To deduce the structure embedding in AutoAlign, we aim to minimize the objective function $\mathcal{J}_{SE}$, which is adapted from Eq. (1), as follows:

$$\mathcal{J}_{SE} = \sum_{t_r \in \mathcal{T}_r} \sum_{t'_r \in \mathcal{T}'_r} \max\left(0, \gamma + \alpha\left(f(t_r) - f(t'_r)\right)\right) \tag{10}$$

$$\alpha = \frac{count(r)}{|\mathcal{T}|} \tag{11}$$

where $\mathcal{T}_r$ represents the set of valid relation triples, $\mathcal{T}r'$ denotes the set of corrupted relation triples, $count(r)$ is the occurrence count of relationship r, and $|\mathcal{T}|$ is the total number of triples in the merged KG $\mathcal{G}U$. Typically, the occurrence count of already aligned and implicitly aligned predicates is greater than that of non-aligned predicates (as aligned predicates are present in both KGs, while non-aligned predicates only appear in one of the KGs). Therefore, our weighting algorithm enables the embedding method to learn more effectively from the aligned triples. For instance, in the triples shown in Table 1, the weight α assists the embedding method in prioritizing relationships like `rdfs:label`, `geo:lat`, and `geo:long` ($\alpha = 2/12$ for each of these predicates) over relationships such as `lgd:alderman` or `dbp:district` ($\alpha = 1/12$ for each of these predicates).

4.4 Attribute Embedding Module

For attribute embedding, we construe the attribute predicate p as a transition from the head entity h to the attribute value v. An attribute might be represented in multiple forms across two KGs. For instance, `50.9989` versus `50.9988888889` as an entity's latitude, or `"Barack Obama"` against `"Barack Hussein Obama"` as a person's name. We adopt a compositional function to code the attribute value and establish the relationship of each component in an attribute triple as $\mathbf{h} + \mathbf{p} \approx f_a(v)$. Here, $f_a(v)$ signifies a compositional function, and v denotes a sequence of the characters of the attribute value $v = \{c_1, c_2, c_3, ..., c_t\}$.

3. https://www.w3.org/TR/rdf-schema/

This compositional function compiles the attribute value into a single vector, thereby associating similar attribute values to a like vector representation. We present three compositional functions as described below.

Sum Compositional Function (**SUM**): The first compositional function is defined as the sum of all character embeddings of the attribute value.

$$f_a(v) = \mathbf{c_1} + \mathbf{c_2} + \mathbf{c_3} + ... + \mathbf{c_t} \tag{12}$$

where $\mathbf{c_1}, \mathbf{c_2}, ..., \mathbf{c_t}$ represent the character embeddings of the attribute value. While this compositional function is straightforward, it suffers from a major limitation: two strings with the same character set but arranged in a different order will have the same vector representation (i.e., order invariant). For instance, two coordinates, `"50.15"` and `"15.05"`, will result in the same vector representation.

LSTM-based Compositional Function (**LSTM**). To address the above problem, we propose an LSTM-based compositional function. This function uses LSTM networks to encode a sequence of characters into a single vector. We use the final hidden state of the LSTM networks as a vector representation of the attribute value.

$$f_a(v) = f_{lstm}(\mathbf{c_1}, \mathbf{c_2}, \mathbf{c_3}, ..., \mathbf{c_t}) \tag{13}$$

where f_{lstm} is an LSTM network [34].

N-gram-based Compositional Function (**N-gram**). LSTM-based compositional function handles the order invariant problem. However, it only considers the unigram features of a string. To capture rich compositional information of a string, we further propose an N-gram-based compositional function as an alternative to the above two compositional functions. Here, we use the summation of the n-gram combination of the attribute value.

$$f_a(v) = \sum_{n=1}^{N} \left(\frac{\sum_{i=1}^{l} \sum_{j=i}^{n} \mathbf{c_j}}{t - i - 1} \right) \tag{14}$$

where N indicates the maximum value of n used in the n-gram combinations ($N = 10$ in our experiments), and l is the length of the attribute value.

To learn the attribute embedding, we minimize the following objective function $\mathcal{J}_{CE}$:

$$\begin{gathered}\mathcal{J}_{CE} = \sum_{t_a \in \mathcal{T}_a} \sum_{t'_a \in \mathcal{T}'_a} \max\left(0, \left[\gamma + \alpha\left(f(t_a) - f(t'_a)\right)\right]\right) \\ f(t_a) = \|\mathbf{h} + \mathbf{p} - f_a(v)\|_2\,,\ \mathcal{T}_a = \{\langle h, p, v\rangle \in \mathcal{G}_U\} \\ \mathcal{T}_a{}' = \{\langle h', p, v\rangle | \, h' \in \mathcal{E}_U\} \cup \{\langle h, p, v'\rangle \, | v' \in \mathcal{A}_U\}\end{gathered} \tag{15}$$

Here, $\mathcal{T}_a$ is the set of valid attribute triples from the training dataset, and $\mathcal{T}'_a$ is the set of corrupted attribute triples ($\mathcal{A}_U$ is the set of attributes in $\mathcal{G}_U$). The corrupted triples are used as negative samples by replacing the head entity with a random entity or the attribute with a random attribute value. $f(t_a)$ is the plausibility score computed based on the embedding of the head entity h, the embedding of the attribute predicate p, and the vector representation of the attribute value computed using function $f_a(v)$.

4.5 Joint Learning of the Embeddings

AutoAlign jointly learns the predicate embeddings, the structure embeddings, and the attribute embeddings. The proposed method first trains over the predicate-proximity-graph to yield the unified predicate embedding space. AutoAlign then uses these predicate embeddings to jointly learn the structure and attribute embeddings. However, the attribute embedding module yields a unified embedding space for two knowledge graphs but lacks structure information. On the other hand, the structure embedding module may yield different embedding space for two knowledge graphs. Thus, we use the attribute embedding $\mathbf{h_{ce}}$ to shift the structure embedding $\mathbf{h_{se}}$ into the same vector space by minimizing the following objective function $\mathcal{J}_{SIM}$:

$$\mathcal{J}_{SIM} = \sum_{s \in \mathcal{G}_1 \cup \mathcal{G}_2} [1 - \cos(\mathbf{h_{se}}, \mathbf{h_{ce}})] \tag{16}$$

Here, $\cos(\mathbf{h_{se}}, \mathbf{h_{ce}})$ is the cosine similarity of vector $\mathbf{h_{se}}$ and $\mathbf{h_{ce}}$. As a result, the structure embedding captures the similarity of entities between two KGs based on entity relationships, while the attribute embedding captures the similarity of entities based on attribute values. The overall objective function of the joint learning is:

$$\mathcal{J} = \mathcal{J}_{PE} + \mathcal{J}_{SE} + \mathcal{J}_{CE} + \mathcal{J}_{SIM} \tag{17}$$

4.6 Entity Alignment

The existing embedding-based entity alignment methods are supervised when obtain the resulting embeddings since they need seed alignments to learn entity alignments from two knowledge graphs. Unlike the existing methods, AutoAlign captures the similarity between entities from two knowledge graphs by learning a unified entity embedding space via predicate and attribute embeddings. AutoAlign does not need seed alignments. Our joint learning embedding scheme lets similar entities from $\mathcal{G}_1$ and $\mathcal{G}_2$ have close vector representations. Thus, the resultant embeddings can be used for entity alignment. We compute the following equation for entity alignment.

$$h_{map} = \underset{h_2 \in \mathcal{G}_2}{\operatorname{argmax}} \cos(\mathbf{h_1}, \mathbf{h_2}) \tag{18}$$

Given an entity $h_1 \in \mathcal{G}_1$, we compute the similarity between h_1 and all entities $h_2 \in \mathcal{G}_2$; $\langle h_1, h_{map} \rangle$ is the expected pair of aligned entities. We use a similarity threshold β to filter the pairs of entities that are too dissimilar to be aligned.

4.7 Triple Enrichment via Transitivity Rule

In translation-based embedding methods such as TransE, the embedding of an entity is learned by aggregating information from its immediate neighbors (i.e., one-hop neighbors). These methods may implicitly learn the multi-hop relationships between entities via information propagation after many training epochs. However, the information propagation of the multi-hop relationship is weak. On the other hand, the explicit inclusion of multi-hop relationships (e.g., transitive relationships) increases the number of attributes and related entities for each entity, which helps identify the similarity between entities. For example, given triples $\langle$`dbp:Emporium_Tower, :locatedIn, dbp:London`$\rangle$ and $\langle$`dbp:London, :country, dbp:England`$\rangle$, we can infer that `dbp:Emporium_Tower` has a relationship (i.e., "`:locatedInCountry`") with `dbp:England`. In fact, this information can be used to enrich the related entity `dbp:Emporium_Tower`. We treat the one-hop transitive relation as follows. Given transitive triples $\langle h_1, p_1, t_1 \rangle$ and $\langle t_1, p_2, t_2 \rangle$, we interpret $p_1.p_2$ as a relation from head entity h_1 to tail entity t_2. Therefore, the relationship between these transitive triples is defined as $\mathbf{h_1} + (\mathbf{p_1}.\mathbf{p_2}) \approx \mathbf{t_2}$. The objective functions of the transitivity-enhanced embedding methods are adapted from the Eq. (10) and Eq. (15) by replacing the relationship vector $\mathbf{p}$ with $\mathbf{p_1}.\mathbf{p_2}$.

4.8 Scalability Discussion

AutoAlign has three main modules, predicate embedding module, structure embedding module and attribute embedding module; their most time-consuming operations are to iterate through the corresponding training samples (i.e., the triples) on the proximity graph using Equation 9, the relation graph using Equation 10, and the attribute graph using Equation 15, respectively. The numbers of triples of the three graphs are all upper-bounded by the total number of edges (i.e., triples) in the two KGs, which we denote as $\mathcal{M}$. Therefore, the time complexities of the predicate embedding module, structure embedding module and attribute embedding module are all $\mathcal{O}(\mathcal{M})$. The triple enrichment via transitivity rule modifies part of the triples without bringing new ones, and does not increase the complexity. Therefore, the time complexity of AutoAlign is still $\mathcal{O}(\mathcal{M})$.

As analysed by [6], translation-based methods [7], [17], [35] typically have the same time complexity, $\mathcal{O}(\mathcal{M})$, since their most time-consuming operations are to iterate through all the training samples (i.e., triples) in the two KGs. GNN-based models also have $\mathcal{O}(\mathcal{M})$ time complexity and require loading the whole graph into memory due to the message passing mechanism. Therefore, AutoAlign has the same time complexity as state-of-the-art KG alignment methods.

We have conducted experiments to compare the running time of AutoAlign to two recent baselines AttrGNN [25] and UPLR [26]. We observe that the running time of AutoAlign is twice that of AttrGNN and half that of UPLR. This is acceptable and reasonable, which is consistent with our complexity analysis above.

5 Experiments

We evaluate AutoAlign from three different aspects. First, we show the performance of AutoAlign in entity alignment, which is the main task in this paper. Second, we show that our predicate embedding module effectively aligns predicates from different knowledge graphs. Third, we show that the resulting embeddings of AutoAlign preserve the structure information of knowledge graphs, enabling them to be used in broader applications such as KG completion.

5.1 Datasets

We evaluate our method on the latest comprehensive benchmark for KG alignment, *DWY-NB* [6], which consists of two datasets *DW-NB* and *DY-NB*. The two KGs of DW-NB are subsets of DBpedia [5] and Wikidata [36], respectively.

TABLE 2: Statistics of the datasets for entity alignment.

Subset	Unique entities	Predicates	Relationship triples	Attribute triples	Entity types
DW-NB					
DBpedia	84,911	545	203,502	221,591	93
Wikidata	86,116	703	198,797	223,232	257
DY-NB					
DBpedia	58,858	211	87,676	173,520	50
Yago	60,228	91	66,546	186,328	61

The two KGs of DY-NB are subsets of DBpedia [5] and Yago [37], respectively. Specifically, DW-NB has more than 84,911 unique entities and contains 50,000 aligned entities, DY-NB has more than 58,858 unique entities and contains 15,000 aligned entities. 36% of the aligned entities have different entity names, which makes the datasets more realistic and the entity alignment task more challenging. To compare with baselines that require entity seeds. We randomly select 50% of the seed alignment as a test set, and the rest of them are used as entity seeds. The statistics of the datasets are summarized in Table 2.

5.2 Implementation Details

We use grid search to find the best hyperparameters for AutoAlign. We choose the embeddings dimensionality d among $\{50, 75, 100, 200\}$, the learning rate of the Adam optimizer among $\{0.001, 0.01, 0.1\}$, and the margin γ among $\{1, 5, 10\}$. We train AutoAlign with a batch size of 100 and a maximum of 400 epochs. We compare with representative state-of-the-art methods, and have used the hyperparameters suggested by their corresponding papers.

5.3 Compared Methods

AutoAlign has two ways for aggregating multiple entity types, weighted sum function and attention-based function as described in Section 4.2.2. We use **AutoAlign-W** and **AutoAlign-A** to represent AutoAlign with weighted sum function and attention-based function respectively. Other compared existing entity alignment methods are described below.

MTransE [7] is the state-of-the-art embedding-based alignment method built on top of TransE. MTransE learns a transition matrix from seed alignments to yield a unified embedding space from two KGs. **IPTransE** [35] is an improved version of TransE. IPTransE adopts two soft strategies to add newly-aligned entities to the seeds to mitigate error propagation. **BootEA** [17] models EA as a one-to-one classification problem where the counterpart of an entity is regarded as the label of the entity. It iteratively learns the classifier via bootstrapping from both labeled and unlabeled data. **TransEdge** [38] proposes an edge-centric translational embedding method addressing the deficiency of TransE in that its relation predicate embeddings are entity-independent. **JAPE** [14] is another state-of-the-art embedding-based entity alignment method built on top of TransE. It combines the relation triples with masked attribute triples. A masked attribute triple is an attribute triple in which its object is replaced by its data type. **MultiKE** [39] uses multi-view learning on various kinds of features. The embedding module of MultiKE divides the features of KGs into three subset views. Entity embeddings are learned for each view and then combined. **AttrE** [15] is the first method that makes use of attribute values and the only EA method that needs no seed alignments. **MuGNN** [10] is the state-of-the-art embedding-based entity alignment built on top of GCN, which uses two GCN as different channels to encode a KG. One channel is for completing missing links in a KG, and the other channel is for filtering unnecessary entities. **AliNet** [11] learns entity embeddings by a controlled aggregation of entity neighborhood information, and shares similar neighborhood structures by considering both direct and distant neighbors. **KECG** [40] aims to reconcile the issue of structural heterogeneity between KGs by jointly training both a GAT-based cross-graph module and a TransE-based knowledge embedding module. **GCN-Align** [41] is the first study on GNN-based EA, which learns entity embeddings from structural information of entities and exploits attribute triples by treating them as relation triples. **HGCN** [42] explicitly utilizes relation representation to improve the alignment process in EA. It incorporates the relation information by jointly learning entity and relation predicate embeddings. **GMNN** [43] formulates the EA task as graph matching between two topic entity graphs. It uses a graph matching module to model the similarity of two topic entity graphs, which indicates the probability of the two corresponding entities being aligned. **RDGCN** [44] utilizes relation information and extends GCNs with highway gates to capture the neighborhood structural information. It differs from HGCN in that it incorporates relation information by the attentive interaction. **CEA** [45] proposes a collective EA method which considers the dependency of alignment decisions among entities. It uses structural, semantic, and string signals to capture different aspects of the similarity between entities in the source and the target KGs, which are represented by three separate similarity matrices. **MRAEA** [46] considers meta relation semantics including relation predicates, relation direction, and inverse relation predicates, in addition to structural information learned from merely the structure of relation triples. **NMN** [47] aims to tackle the structural heterogeneity between KGs. The method learns both the KG structure information and the neighborhood difference so that the similarities between entities can be better captured. **AttrGNN** [25] performs entity alignment by combining attribute graph learning, value graph learning, and structure graph learning, and selects the best performance by comparing different combinations. **UPLR** [26] constructs pseudo-labeled datasets containing noisy data and leverage the graph attention nets to capture the similarities between two KGs.

5.4 Entity Alignment Results

This experiment evaluates the performance of EA while varying the amount of seed entity alignments used for training from 10% to 50% of the total available set of seed entity alignments (50,000 for DW-NB and 7,500 for DY-NB). We fix the test set in all the settings for all the models. That is, the baseline methods with different ratios of seed alignments will have the same test set. This setting ensures that the results in different settings are comparable

TABLE 3: The effect of the amount of seed entity alignments on EA performance in terms of Hits@k (%). The numbers with bold/underlined indicate the highest/sub-optimal values in each group compared to baseline methods.

	Method	Seed: 0%		Seed: 10%		Seed: 20%		Seed: 30%		Seed: 40%		Seed: 50%	
		Hits@1	Hits@10	Hits@1	Hits@10	Hits@1	Hits@10	Hits@1	Hits@10	Hits@1	Hits@10	Hits@1	Hits@10
	DW-NB												
Translation-based	MTransE	N/A	N/A	2.82	10.45	5.42	18.72	7.88	25.75	10.42	31.44	12.98	36.00
	IPTransE	N/A	N/A	5.98	13.45	7.54	18.78	12.90	24.61	16.32	32.86	23.54	35.98
	BootEA	N/A	N/A	8.12	16.15	12.54	20.13	17.92	28.38	21.46	35.16	25.44	37.57
	TransEdge	N/A	N/A	22.98	48.12	38.29	56.22	45.27	68.95	49.26	75.25	54.85	79.68
	JAPE	N/A	N/A	4.62	7.87	8.62	14.43	12.57	19.96	17.20	27.32	19.91	30.63
	MultiKE	N/A	N/A	80.25	87.58	82.56	88.92	84.06	90.05	84.87	91.24	85.21	95.06
	AttrE	87.98	95.80	87.98	95.80	87.98	95.80	87.98	95.80	87.98	95.80	87.98	95.80
	AutoAlign-W	87.81	95.86	87.81	95.86	87.81	95.86	87.81	95.86	87.81	95.86	87.81	95.86
	AutoAlign-A	**88.73**	**96.91**	**88.73**	**96.91**	**88.73**	**96.91**	**88.73**	**96.91**	**88.73**	**96.91**	**88.73**	**96.91**
GNN-based	MuGNN	N/A	N/A	13.49	37.79	20.96	49.28	26.92	56.77	31.09	61.43	34.41	64.96
	AliNet	N/A	N/A	14.58	31.46	18.55	35.84	24.34	50.46	28.39	55.46	35.31	58.22
	KECG	N/A	N/A	18.95	34.17	24.32	40.78	30.24	48.66	35.29	52.12	39.40	62.31
	GCN-Align	N/A	N/A	12.40	30.18	20.04	41.56	24.76	48.52	29.02	53.43	31.80	56.20
	HGCN	N/A	N/A	58.08	62.15	63.14	68.15	78.97	86.51	84.25	90.75	88.54	91.54
	GMNN	N/A	N/A	71.32	74.24	75.34	79.23	80.98	82.23	82.67	85.87	84.59	88.64
	RDGCN	N/A	N/A	59.22	62.98	64.22	68.98	79.02	87.12	85.34	90.45	88.21	93.23
	CEA	N/A	N/A	50.13	52.31	63.25	64.12	80.32	84.21	84.34	85.54	86.58	88.34
	MRAEA	N/A	N/A	53.75	54.74	64.58	66.12	81.54	85.97	83.54	86.02	84.06	87.55
	NMN	N/A	N/A	51.45	59.78	68.21	72.54	84.03	88.21	85.65	90.54	88.69	95.46
	AttrGNN	N/A	N/A	45.79	78.28	51.67	82.85	54.65	84.30	59.48	86.18	62.08	88.74
	UPLR	0.34	1.62	0.34	1.62	0.34	1.62	0.34	1.62	0.34	1.62	0.34	1.62
	DY-NB												
Translation-based	MTransE	N/A	N/A	0.01	0.15	0.01	0.39	0.08	0.68	0.08	1.39	0.13	1.89
	IPTransE	N/A	N/A	1.54	9.87	5.67	25.76	14.55	36.45	15.77	45.81	17.33	52.18
	BootEA	N/A	N/A	2.15	14.19	8.47	38.15	15.77	48.32	17.22	57.15	19.24	58.14
	TransEdge	N/A	N/A	22.98	47.50	37.85	64.85	48.98	72.15	58.95	76.54	62.49	78.54
	JAPE	N/A	N/A	0.70	1.83	1.57	3.37	1.40	3.27	1.37	1.77	2.37	4.97
	MultiKE	N/A	N/A	81.87	88.05	82.11	89.26	84.97	90.84	87.22	92.05	89.25	93.58
	AttrE	90.44	94.23	90.44	94.23	90.44	94.23	90.44	94.23	90.44	94.23	90.44	94.23
	AutoAlign-W	90.42	94.35	90.42	94.35	90.42	94.35	90.42	94.35	90.42	94.35	90.42	94.35
	AutoAlign-A	**91.27**	**95.62**	**91.27**	**95.62**	**91.27**	**95.62**	**91.27**	**95.62**	**91.27**	**95.62**	**91.27**	**95.62**
GNN-based	MuGNN	N/A	N/A	19.16	51.41	27.40	62.69	31.60	68.56	34.73	71.24	37.15	74.07
	AliNet	N/A	N/A	13.54	28.53	14.25	31.69	25.39	58.31	28.98	56.12	34.59	64.12
	KECG	N/A	N/A	11.19	23.65	14.89	27.25	20.95	34.48	22.81	35.44	24.71	37.15
	GCN-Align	N/A	N/A	8.56	25.09	17.88	43.88	24.36	53.43	31.29	62.44	33.56	67.88
	HGCN	N/A	N/A	52.54	64.51	65.87	77.40	71.14	85.64	71.45	85.64	74.54	87.48
	GMNN	N/A	N/A	62.34	70.34	64.32	67.34	75.57	77.47	78.65	82.65	82.34	85.62
	RDGCN	N/A	N/A	53.13	65.30	67.28	78.21	74.54	85.22	77.45	87.43	78.67	89.45
	CEA	N/A	N/A	55.24	58.97	64.35	65.42	74.56	78.42	77.78	80.95	78.91	83.24
	MRAEA	N/A	N/A	52.46	53.20	60.33	64.54	73.71	78.52	74.25	78.66	76.22	80.15
	NMN	N/A	N/A	55.74	64.78	62.54	70.54	75.87	80.54	84.55	88.69	90.78	94.77
	AttrGNN	N/A	N/A	77.21	88.03	79.44	89.76	80.16	90.19	81.31	90.84	83.98	91.95
	UPLR	89.84	93.11	89.84	93.11	89.84	93.11	89.84	93.11	89.84	93.11	89.84	93.11

* Methods that use attribute triples are underlined. The rest tables and figures follow this convention.
* AttrE, AutoAlign-W and AutoAlign-A do not use any seed alignments.

and hence valid for evaluating the model performance. We evaluate the performance of AutoAlign (note that it does not need any seed alignments) using $\mathbf{Hits@k}(k = 1, 10)$ (i.e., the proportion of correctly aligned entities ranked in the top k predictions). A higher value indicates better performance.

Table 3 shows the results on the DWY-NB benchmark datasets [6]. Some of the results of the compared methods are obtained from [6]. Note that since our methods do not require entity seeds, the results of our model (and UPLR, which also does not require entity seeds) are the same for different seed ratios. We observe that the two variations of AutoAlign, AutoAlign-W and AutoAlign-A, are significantly better than all the other methods. The performance of AutoAlign-A is better than AutoAlign-W, which shows the importance to capture both the distinctive and noisy entity types, as done by AutoAlign-A. The underlined methods from both translation- and GNN-based methods exploit attribute triples. The methods that exploit attribute triples achieve much better performance than the methods that do not.

When no seed is provided (Seed: 0%), the baselines that require entity seeds simply cannot run, while our methods can still get the great performance. When seeds are available, other methods can run but perform considerably worse than our methods. For example, AutoAlign-A outperforms the best performing baseline, MultiKE, by 10.65% in hits@10 in the DW-NB dataset (96.91 v.s. 87.58). The performance of these methods get better with more seed alignments available, but they are still considerably worse than our methods, especially when fewer seed alignments are available.

5.5 Ablation Study

We conduct ablation tests from two perspectives to evaluate AutoAlign: the effect of attribute embedding module and the effect of predicate embedding module.

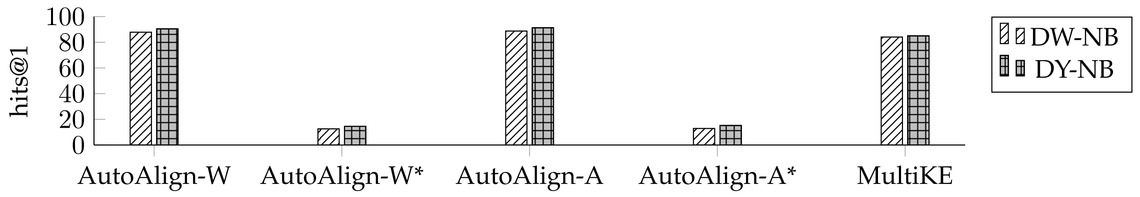


Fig. 4: The effect of attribute embedding module.

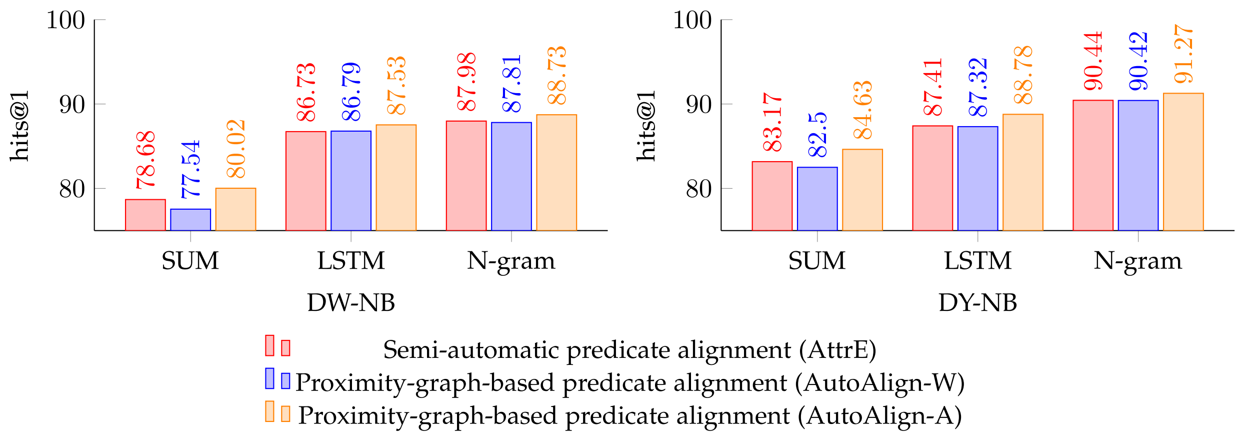


Fig. 5: The effect of predicate embedding module.

5.5.1 Effect of Attribute Embedding Module

To evaluate the effect of using attribute triples, we create a version of AutoAlign-W that does not use attribute triples to compute the entity embeddings, i.e., it only uses relation triples; we call this version AutoAlign-W*. Similarly, we create a version of AutoAlign-A that does not use attribute triples, which we call AutoAlign-A*. Figure 4 shows the performance of the four versions of AutoAlign on the benchmark measured by Hit@1. We can see that the performance of AutoAlign-W and AutoAlign-A are much higher than that of AutoAlign-W* and AutoAlign-A*, respectively. This shows that our idea of using attribute triples is highly effective. We also put the performance of MultiKE in the figure for comparison since MultiKE is the most accurate one among other existing methods; the proportion of seed entity alignments used for MultiKE is 30%. AutoAlign-W and AutoAlign-A both outperform MultiKE.

We also show the effect of different attribute embedding algorithms in Fig. 5. Here SUM, LSTM, and N-gram denote three algorithms with different attribute embedding functions, as described in Section 4.4. We see that the N-gram compositional function gives the best performance. This is because the N-gram compositional function better preserves string similarity when mapping attribute strings to their vector representations than the other two functions.

5.5.2 Effect of Predicate Embedding Module

To evaluate the effect of predicate embedding module proposed in Section 4.2, we compare with the semi-automatic predicate alignment module in our previously proposed method AttrE [15]. From Fig. 5, we see that the predicate embedding module helps our entity alignment method achieve comparable performance in terms of $hits@1$.

The same predicate may be stored in different surface forms in the KGs, e.g., one KG has the attribute predicate `birth_date` while the other KG has the attribute predicate `date_of_birth`. Previous methods exploit seed attribute predicate alignments and seed attribute alignments to address this difference. In comparison, our AutoAlign-W and AutoAlign-A do not need manual intervention, and they both yield competitive results. In particular, AutoAlign-A achieves the best performance since it enriches the related entity types information via the attention mechanism.

TABLE 4: The effect on the accuracy of downstream link prediction task in terms of Hits@10 (%). The numbers with bold/underline indicate the highest/sub-optimal values in each group compared to baseline methods.

Method	DW-NB (seed)			DY-NB (seed)		
	(10%)	(30%)	(50%)	(10%)	(30%)	(50%)
AutoAlign-A	**88.93**	88.93	88.93	**98.82**	98.82	**98.82**
MultiKE	88.76	**88.98**	**89.52**	98.62	**98.87**	98.07
AttrE	88.50	88.50	88.50	98.75	98.75	98.75
AutoAlign-W	88.41	88.41	88.41	98.66	98.66	98.66
TransE	87.45	87.45	87.45	98.42	98.42	98.42
TransEdge	85.27	85.71	86.40	93.24	93.54	93.76
JAPE	83.24	83.71	83.09	75.03	75.32	75.66
IPTransE	81.06	81.23	81.78	93.10	93.55	93.91
BootEA	80.41	80.90	81.66	94.11	94.54	94.85
MTransE	80.10	80.33	80.69	93.81	94.31	94.74

5.6 Effect of the Alignment Method on KG embeddings

We further evaluate the effect of AutoAlign on KG embeddings. This section experiments on how the quality of the KG embeddings obtained from EA methods is affected compared to the KG embeddings from pure KG embedding methods (TransE for translation-based and GCN for

GNN-based methods) via downstream applications of KGs. Following previous studies in EA methods [48], we conduct experiments using a common downstream task *link prediction* for this purpose, detailed as follows. The link prediction task aims to predict t given h and r of a relation triple. Specifically, first, a relation triple is corrupted by replacing its tail entity with all the entities in the dataset. Then, the corrupted triples are ranked in ascending order by the plausibility score computed as $\boldsymbol{h} + \boldsymbol{r} - \boldsymbol{t}$. Since true triples (i.e., the triples in a KG) are expected to have smaller plausibility scores and rank higher in the list than the corrupted ones, hits@10 (whether the true triples are in the top-10) is used as the measure for the link prediction task.

Table 4 shows the performance of link prediction on DW-NB and DY-NB with 10%, 30%, and 50% of seed entity alignments. The performance increases with the amount of seed alignments but not significantly. As mentioned earlier, the KG embeddings obtained from the KG alignment methods may not be optimized for downstream tasks. However, AutoAlign-A still achieves high performance, always in top-2 and top-1 in half of the cases, which shows that the learned predicate embeddings can also project entities into a unified embedding space.

6 Conclusion and Future Work

We have presented AutoAlign – the first fully automatic method for KG alignment enabled by large language models. To achieve that, we proposed attribute character embeddings and predicate-proximity-graph embeddings powered by large language models to compute a unified vector space for the entity and predicate embeddings from two KGs. Experimental results show that AutoAlign outperforms the competitors consistently. Further results on knowledge graph completion show that our joint learning of the entity, predicate, and attribute embeddings can capture the similarity between entities and predicates both within a KG and across KGs.

AutoAlign demonstrates the potential of leveraging large language models to improve the performance of KG alignment (e.g., requiring less manual work, incorporating the knowledge stored in large language models). In future work, we may investigate broader research domains based on graphs that can benefit from large language models enabled KG alignment. For example, leveraging large language models to align KGs with domain-specific graphs (e.g., feature graphs in recommender systems [49], [50], region graphs in point of interests learning [51]) to enrich their representation ability.

References

[1] Q. Wu, C. Shen, P. Wang, A. Dick, and A. v. d. Hengel, "Image captioning and visual question answering based on attributes and external knowledge," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 06, pp. 1367–1381, 2018.

[2] S. Yang, R. Zhang, S. M. Erfani, and J. H. Lau, "Unimf: A unified framework to incorporate multimodal knowledge bases intoend-to-end task-oriented dialogue systems." in *IJCAI*, 2021, pp. 3978–3984.

[3] F. Zhang, N. J. Yuan, D. Lian, X. Xie, and W.-Y. Ma, "Collaborative knowledge base embedding for recommender systems," in *KDD*, 2016, pp. 353–362.

[4] C. Stadler, J. Lehmann, K. Hoffner, and S. Auer, "Linkedgeodata: A core for a web of spatial open data," *Semantic Web*, vol. 3, no. 4, pp. 333–354, 2012.

[5] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives, "Dbpedia: A nucleus for a web of open data," in *ISWC*, 2007, pp. 722–735.

[6] R. Zhang, B. D. Trisedya, M. Li, Y. Jiang, and J. Qi, "A benchmark and comprehensive survey on knowledge graph entity alignment via representation learning," *The VLDB Journal*, pp. 1–26, 2022.

[7] M. Chen, Y. Tian, M. Yang, and C. Zaniolo, "Multilingual knowledge graph embeddings for cross-lingual knowledge alignment," in *IJCAI*, 2017, pp. 1511–1517.

[8] M. Chen, T. Zhou, P. Zhou, and C. Zaniolo, "Multi-graph affinity embeddings for multilingual knowledge graphs," in *NIPS Workshop on Automated Knowledge Base Construction*, 2017.

[9] H. Zhu, R. Xie, Z. Liu, and M. Sun, "Iterative entity alignment via joint knowledge embeddings," in *IJCAI*, 2017, pp. 4258–4264.

[10] Y. Cao, Z. Liu, C. Li, Z. Liu, J. Li, and T.-S. Chua, "Multi-channel graph neural network for entity alignment," in *ACL*, 2019, pp. 1452–1461.

[11] Z. Sun, C. Wang, W. Hu, M. Chen, J. Dai, W. Zhang, and Y. Qu, "Knowledge graph alignment network with gated multi-hop neighborhood aggregation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 01, 2020, pp. 222–229.

[12] Y. Wu, X. Liu, Y. Feng, Z. Wang, R. Yan, and D. Zhao, "Relation-aware entity alignment for heterogeneous knowledge graphs," in *IJCAI*, 2019, pp. 5278–5284.

[13] R. Ye, X. Li, Y. Fang, H. Zang, and M. Wang, "A vectorized relational graph convolutional network for multi-relational network alignment." in *IJCAI*, 2019, pp. 4135–4141.

[14] Z. Sun, W. Hu, and C. Li, "Cross-lingual entity alignment via joint attribute-preserving embedding," in *ISWC*, 2017, pp. 628–644.

[15] B. D. Trisedya, J. Qi, and R. Zhang, "Entity alignment between knowledge graphs using attribute embeddings," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 297–304.

[16] R. Socher, D. Chen, C. D. Manning, and A. Y. Ng, "Reasoning with neural tensor networks for knowledge base completion," in *NIPS*, 2013, pp. 926–934.

[17] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *NIPS*, 2013, pp. 2787–2795.

[18] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *AAAI*, 2014, pp. 1112–1119.

[19] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," in *AAAI*, 2015, pp. 2181–2187.

[20] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[21] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," in *ICLR*, 2018.

[22] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," *Advances in neural information processing systems*, vol. 32, 2019.

[23] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," in *Proceedings of The Web Conference 2020*, 2020, pp. 2704–2710.

[24] X. Mei, X. Cai, L. Yang, and N. Wang, "Relation-aware heterogeneous graph transformer based drug repurposing," *Expert Systems with Applications*, vol. 190, p. 116165, 2022.

[25] Z. Liu, Y. Cao, L. Pan, J. Li, and T.-S. Chua, "Exploring and evaluating attributes, values, and structures for entity alignment," in *EMNLP*, 2020.

[26] J. Li and D. Song, "Uncertainty-aware pseudo label refinery for entity alignment," in *The Web Conference*, 2022, pp. 829–837.

[27] J. Gao, X. Liu, Y. Chen, and F. Xiong, "Mhgcn: Multiview highway graph convolutional network for cross-lingual entity alignment," *Tsinghua Science and Technology*, vol. 27, no. 4, pp. 719–728, 2021.

[28] Z. Zhang, J. Chen, X. Chen, H. Liu, Y. Xiang, B. Liu, and Y. Zheng, "An industry evaluation of embedding-based entity alignment," in *COLING*, 2020.

[29] W. Liu, J. Pan, X. Zhang, X. Gong, Y. Ye, X. Zhao, X. Wang, K. Wu, H. Xiang, H. Yan *et al.*, "Cross-platform product matching based on entity alignment of knowledge graph with raea model," *WWW*, pp. 1–21, 2023.

[30] L. Floridi and M. Chiriatti, “Gpt-3: Its nature, scope, limits, and consequences,” *Minds and Machines*, vol. 30, pp. 681–694, 2020.
[31] T. L. Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon, M. Gallé *et al.*, “Bloom: A 176b-parameter open-access multilingual language model,” *arXiv preprint arXiv:2211.05100*, 2022.
[32] R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du *et al.*, “Lamda: Language models for dialog applications,” *arXiv preprint arXiv:2201.08239*, 2022.
[33] C. Fellbaum, *WordNet: An Electronic Lexical Database*. MIT Press, 1998.
[34] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, pp. 1735–1780, 1997.
[35] H. Zhu, R. Xie, Z. Liu, and M. Sun, “Iterative entity alignment via joint knowledge embeddings.” in *IJCAI 2017*, 2017.
[36] D. Vrandecic and M. Krötzsch, “Wikidata: a free collaborative knowledgebase,” *CACM*, vol. 57, no. 10, pp. 78–85, 2014.
[37] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum, “Yago2: A spatially and temporally enhanced knowledge base from wikipedia,” *Artificial Intelligence*, vol. 194, pp. 28–61, 2013.
[38] Z. Sun, J. Huang, W. Hu, M. Chen, L. Guo, and Y. Qu, “Transedge: Translating relation-contextualized embeddings for knowledge graphs,” in *International Semantic Web Conference*. Springer, 2019, pp. 612–629.
[39] Q. Zhang, Z. Sun, W. Hu, M. Chen, L. Guo, and Y. Qu, “Multi-view knowledge graph embedding for entity alignment,” in *IJCAI*, 2019.
[40] C. Li, Y. Cao, L. Hou, J. Shi, J. Li, and T. Chua, “Semi-supervised entity alignment via joint knowledge embedding model and cross-graph model,” in *EMNLP 2019*, 2019.
[41] Z. Wang, Q. Lv, X. Lan, and Y. Zhang, “Cross-lingual knowledge graph alignment via graph convolutional networks,” in *EMNLP 2018*, 2018.
[42] Y. Wu, X. Liu, Y. Feng, Z. Wang, and D. Zhao, “Jointly learning entity and relation representations for entity alignment,” in *EMNLP 2019*, 2019.
[43] K. Xu, L. Wang, M. Yu, Y. Feng, Y. Song, Z. Wang, and D. Yu, “Cross-lingual knowledge graph alignment via graph matching neural network,” in *ACL 2019*, 2019.
[44] Y. Wu, X. Liu, Y. Feng, Z. Wang, R. Yan, and D. Zhao, “Relation-aware entity alignment for heterogeneous knowledge graphs,” in *IJCAI 2019*, 2019.
[45] W. Zeng, X. Zhao, J. Tang, and X. Lin, “Collective entity alignment via adaptive features,” in *ICDE 2020*, 2020.
[46] X. Mao, W. Wang, H. Xu, M. Lan, and Y. Wu, “MRAEA: an efficient and robust entity alignment approach for cross-lingual knowledge graph,” in *WSDM 2020*, 2020.
[47] Y. Wu, X. Liu, Y. Feng, Z. Wang, and D. Zhao, “Neighborhood matching network for entity alignment,” in *ACL 2020*, 2020.
[48] Z. Sun, Q. Zhang, W. Hu, C. Wang, M. Chen, F. Akrami, and C. Li, “A benchmarking study of embedding-based entity alignment for knowledge graphs,” in *VLDB 2020*, 2020.
[49] Y. Su, R. Zhang, S. Erfani, and Z. Xu, “Detecting beneficial feature interactions for recommender systems,” in *AAAI*, 2021, pp. 4357–4365.
[50] Y. Su, Y. Zhao, S. Erfani, J. Gan, and R. Zhang, “Detecting arbitrary order beneficial feature interactions for recommender systems,” in *KDD*, 2022, pp. 1676–1686.
[51] Y. Zhao, J. Qi, B. D. Trisedya, Y. Su, R. Zhang, and H. Ren, “Learning region similarities via graph-based deep metric learning,” *TKDE*, 2023.

Appendix A
Automatically Obain Entity Types

To construct the predicate-proximity-graph, we automatically obtain the types of each entity by extracting them from the SPARQL Query Editor (https://dbpedia.org/sparql). Specifically, we obtain the types of entities through the following steps:

- For each entity, e.g., Barack Obama, we convert it into a DBpedia graph dataset format, e.g., http://dbpedia.org/resource/Barack_Obama.
- Then, we search for the types of the entity through the query:

```
PREFIX rdf: <\protect\vrule width0pt\
    ↪ protect\href{http://www.w3.org
    ↪ /1999/02/22-rdf-syntax-ns#}{http
    ↪ ://www.w3.org/1999/02/22-rdf-
    ↪ syntax-ns#}>
PREFIX rdfs: <\protect\vrule width0pt\
    ↪ protect\href{http://www.w3.org
    ↪ /2000/01/rdf-schema#}{http://www.
    ↪ w3.org/2000/01/rdf-schema#}>
PREFIX dbr: <\protect\vrule width0pt\
    ↪ protect\href{http://dbpedia.org/
    ↪ resource}{http://dbpedia.org/
    ↪ resource}>
PREFIX dbo: <\protect\vrule width0pt\
    ↪ protect\href{http://dbpedia.org/
    ↪ ontology}{http://dbpedia.org/
    ↪ ontology}>
SELECT DISTINCT ?obj WHERE {
<\protect\vrule width0pt\protect\href{
    ↪ http://dbpedia.org/resource/
    ↪ Barack_Obama}{http://dbpedia.org/
    ↪ resource/Barack_Obama}> rdf:type ?
    ↪ obj
FILTER strstarts(str(?obj), str(dbo:))
}
```

The first four lines define the prefixes for the namespaces used in the query. “rdf:” and “rdfs:” are standard namespaces for RDF and RDF Schema, respectively. ”dbr:” and ”dbo:” are prefixes for the DBpedia resource and ontology namespaces, respectively; the SELECT clause specifies that we want to retrieve the distinct values of the variable ”?obj”, which represents the types of which ”Barack Obama” is an entity.

- Finally, we get a set of types that belongs to the entity, e.g., Person, Politician, OfficeHolder, which will replace the entity ”Barack Obama” in the predicate-proximity-graph.
- If we cannot obtain any type from the entity, we will keep the entity as it is.