# Scalable Knowledge Graph Construction over Text using Deep Learning based Predicate Mapping

Aman Mehta[*]
International Institute of Information Technology Hyderabad
India
aman.mehta@research.iiit.ac.in

Aashay Singhal[*]
International Institute of Information Technology Hyderabad
India
aashay.singhal@research.iiit.ac.in

Kamalakar Karlapalem
International Institute of Information Technology Hyderabad
India
kamal@iiit.ac.in

## ABSTRACT

Automatic extraction of information from text and its transformation into a structured format is an important goal in both Semantic Web Research and computational linguistics. Knowledge Graphs (KG) serve as an intuitive way to provide structure to unstructured text. A fact in a KG is expressed in the form of a triple which captures entities and their interrelationships (predicates). Multiple triples extracted from text can be semantically identical but they may have a vocabulary gap which could lead to an explosion in the number of redundant triples. Hence, to get rid of this vocabulary gap, there is a need to map triples to a homogeneous namespace. In this work, we present an end-to-end KG construction system, which identifies and extracts entities and relationships from text and maps them to the homogenous DBpedia namespace. For Predicate Mapping, we propose a Deep Learning architecture to model semantic similarity. This mapping step is computation heavy, owing to the large number of triples in DBpedia. We identify and prune unnecessary comparisons to make this step scalable. Our experiments show that the proposed approach is able to construct a richer KG at a significantly lower computation cost with respect to previous work.

## CCS CONCEPTS

• **Computing methodologies** → **Knowledge representation and reasoning**; **Neural networks**; • **Information systems** → *Entity resolution*; *Ontologies*.

## KEYWORDS

Knowledge Graph, Predicate Mapping, Deep Learning, Scalability, Sentence Simplification

**ACM Reference Format:**
Aman Mehta, Aashay Singhal, and Kamalakar Karlapalem. 2019. Scalable Knowledge Graph Construction over Text using Deep Learning based Predicate Mapping. In *Companion Proceedings of the 2019 World Wide Web Conference (WWW '19 Companion), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3308560.3317708

| | |
|---|---|
| *(a)* | *Barack Obama was born in Honolulu.* |
| *(b)* | *Obama was elected in 2009 as the president of the United states. He belongs to Honolulu.* |
| *(c)* | *Barack served as the 44th president of the United States and grew up in Honolulu.* |

**Table 1: Example text (referred in multiple sections)**

## 1 INTRODUCTION

**Motivation**. In any industry, text documents that contain important data are a common occurrence. Hence, improving the ability of machines to understand the intent and context of information to the level of humans is one of biggest challenges faced today. Achieving this would involve transformation of unstructured text to a structured format. Knowledge graphs provide an intuitive way of giving shape and structure to the initially unstructured information. A fact in a KG is represented by a triple of the form *<S;P;O>*. *S*, *O* denote the *subject* and *object*, respectively and *P* is the *predicate* which describes the relationship between *S* and *O*.

A KG built over any unstructured text document helps in making the information in the document queryable. This makes the previously inoperable information usable for tasks such as search result ranking, recommendation, question answering, etc [2, 13, 30, 31, 37]. Another interesting application of constructing KG over text is to add new or missing information to an existing KG like DBpedia [2], Freebase [4].

Traditional methods ([9]) focus on building KGs from infobox templates and categorization information in the Wikipedia articles. However, the unstructured text of these articles is left unprocessed. In order to make this natural language text structured and usable, generating a KG over text becomes an important task. Previous works such as [3, 6, 7, 11, 20, 36] use information extraction system to extract facts from NL text. But these facts do not necessarily follow the paradigm of a KG (such as DBpedia), making the KG construction task challenging. For instance, consider the output of sample sentence *(a)* from (table 1) using OLLIE information extractor - (Barack Obama, was born in, Honolulu). This fact is identical to (Barack Obama, birthPlace, Honolulu) in DBpedia. Although these two facts are identical, there is a vocabulary gap between them. In the fact extracted from text, *"was born in"* is a natural language phrase, whereas *"birthPlace"* is a formatted predicate in DBpedia. These two relational phrases have different surface forms and hence they cannot be mapped just on the basis of string similarity.

Furthermore, same DBpedia entities and predicate could be used in multiple NL excerpts. For instance, *"He"*, *"Barack Obama"* and

[*] represents equal contribution.

*WWW '19 Companion, May 13–17, 2019, San Francisco, CA, USA*

ACM ISBN 978-1-4503-6675-5/19/05.
https://doi.org/10.1145/3308560.3317708

*"Barack"* in the facts extracted for *(a), (b)* and *(c)* (from table 1) correspond to *"Barack_Obama"* in DBpedia. Similarly, the predicates such as *"was born in"*, *"grew up in"*, *"belongs to"* are related to the same predicate *"birthPlace"* in DBpedia. As shown in fig 1, this leads to a high amount of redundancy in the constructed KG, giving rise to an unnecessarily large sized KG. This redundancy leads to wastage of space for its storage and wastage of time in execution of graph retrieval algorithms [22, 23, 38]. Therefore, it is important to efficiently resolve entities and their relationships in triples to facilitate a queryable Knowledge Graph. The most common approach to resolve them is by mapping them to a single homogeneous namespace, such as DBpedia namespace.

A number of studies have proposed ideas for Entity Mapping [25]. However, only a few studies have worked on the task of Predicate Mapping. A few rule-based [10] and similarity-based approaches [19] have been proposed in recent years, but both of them have their own limitations. Simple rule-based approaches cannot generate rules efficiently, especially when text sources are sparse. This is due to the fact that these rules are manually generated and hence it cannot get rid of all redundancies. On the other hand, similarity-based solutions for Predicate mapping are challenging in two aspects:
a) To map a predicate to another namespace, they need to capture the accurate semantics for calculating similarity scores
b) comparing a predicate to each of the predicate in DBpedia is highly time consuming owing to the number of *candidates* (i.e. the number predicates in DBPedia namespace to which a particular text predicate can map to). T2KG [19] lacks in both of these aspects. Firstly, it uses a word2vec based model which is not able to capture all semantic features since it uses shallow neural networks[1]. Secondly, it scans all DBpedia triples in order to map a single predicate of KG to DBpedia namespace[2]. So, when the text sources are large, it becomes a time consuming task to map all text predicates to DBpedia predicates. Even tougher when the size of the DBpedia itself is as large as 1.3 billion[3] triples, such as in the case of English version of DBpedia. We address both of the above challenges in the proposed approach.

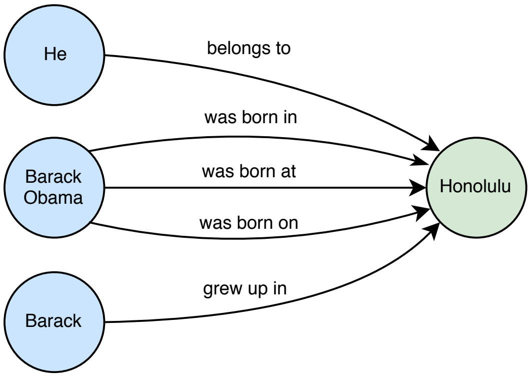


**Figure 1: Knowledge Graph obtained using information extraction (OLLIE [36])**

**Contributions**. In this work, we introduce an end-to-end KG construction system from unstructured text. In order to make this KG more useful, we map our KG to DBpedia namespace far more efficiently than the previous approaches. Under the Predicate Mapping step, we implement a pruning strategy to reduce a large number of unnecessary comparisons in order to make this step scalable. The end-to-end system is able to build a larger KG, with less redundancy while significantly reducing the computation cost in finding the Predicate Mapping step. Our main contributions are as follows:

- We introduce a system to construct KG from text which is consistent with the DBpedia namespace.
- We propose a Deep Learning model to calculate similarity between two predicates for the Predicate Mapping step.
- We develop optimization strategy to reduce the *Candidate Set* (described in section 3.2) for each text predicate.
- We introduce a sentence simplification component to improve triple extraction.
- We perform multiple experiments to evaluate our KG construction system with other systems. The experiments show the effectiveness of our Predicate Mapping component and the benefits of the Sentence Simplification component.
- A separate study to quantify the extent of redundancy reduction in different domains of Wikipedia articles in the KGs constructed using our system.

The rest of this paper is structured as follows: section 2 describes different components of our pipeline, section 3 describes in detail our Predicate Mapping model and section 4 describes the training details of our model. We perform experiments and show our results in section 5 and conclude our work in section 6.

## 2 END-TO-END PIPELINE: TEXT TO KG

In this section, we describe in detail, the steps to construct a Knowledge Graph from unstructured text. Our end-to-end pipeline has six components: The *Entity Mapping* component maps the entities in the text triples to the DBpedia entities. The *Sentence Simplifier* which simplifies sentences before triple extraction step to overcome limitations of triple extraction on complex sentences. The *Co-reference Resolution* component which finds and replaces all the expressions that refer to the same entity in the text. The *Triple Extractor* that uses information extraction techniques to extract relation triples from text (called text triples). The *Metadata Processing* component prepares and stores the set of candidate DBpedia predicates for a given text predicate, that could possibly be the mapping. The *Predicate Mapping* component maps a text triple's predicate to its matching predicate in DBpedia namespace.

**Entity Mapping:** An important step in KG construction is linking named entities to unique identifiers. In this component, we use a named entity recogniser [25] to mark all the named entities in the text. Now, for an entity which has a possible mapping in DBpedia, we use the URI of such a DBpedia entity as the representation of the text entity. Otherwise, we define a custom namespace to create their URIs. The output from Entity Mapping step are the same set of sentences, but entities replaced by the URIs as defined by above conditions. For eg: *"Barack Obama"*, *"Obama"* and *"Barack"* will be replaced by *DBpedia*:*Barack_Obama*, which is the URI of Barack

[1] https://en.wikipedia.org/wiki/Word2vec
[2] [19] Page 5: eq. (2) and eq. (3)
[3] https://wiki.dbpedia.org/develop/datasets/dbpedia-version-2016-10

Obama entity in DBpedia. Hence, output of this layer for above three instances will be,
*(a) DBpedia:Barack_Obama was born in DBpedia:Honolulu.*
*(b) DBpedia:Barack_Obama was elected in 2009 as the president of the DBpedia:United_States. He belongs to DBpedia:Honolulu.*
*(c) DBpedia:Barack_Obama served as the 44th president of the DBpedia:United_States and grew up in DBpedia:Honolulu.*

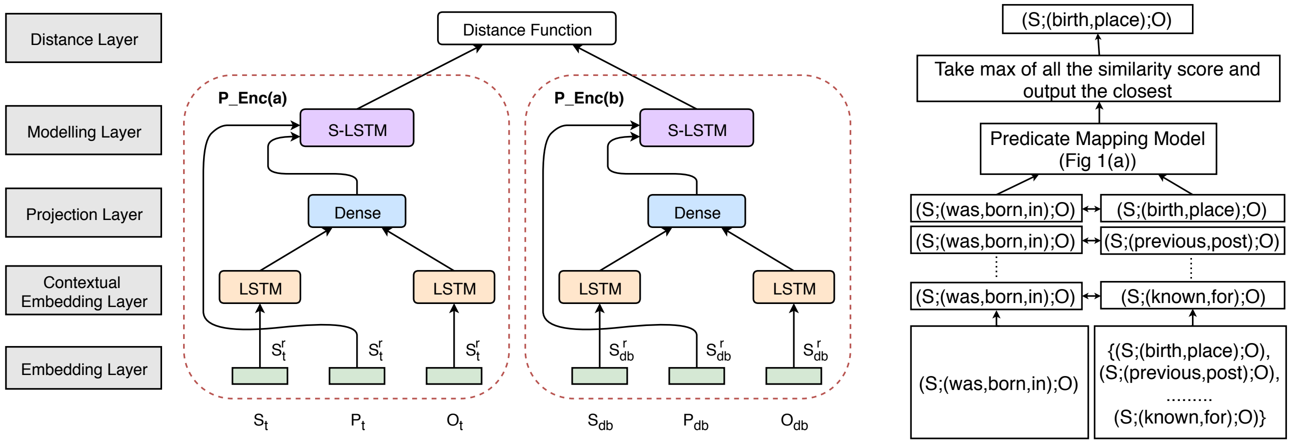
**Sentence Simplification:** English is a difficult language for machines to comprehend. As and when sentences grow complex in nature, extraction of relation triples from unstructured text becomes tougher. Due to complex structure of sentences, the frameworks which try to convert unstructured text into a relation triple, cease to identify correct relationships between the entities at hand. Hence, the aim of this component is to identify complex sentences and convert them into simpler sentence(s), such that the semantics is consistent even after simplification. We implement the approach proposed in [28] to achieve this. Here, *(a)* (table 1) remains unchanged, because it is not a complex English sentence whereas *(b)* and *(c)* change to:
*(b) DBpedia:Barack_Obama was elected in 2009. DBpedia: Barack_Obama was the president of the DBpedia:United_States. He belongs to DBpedia:Honolulu.*
*(c) DBpedia:Barack_Obama served as the 44th president of the DBpedia:United_States. DBpedia:Barack_Obama grew up in DBpedia: Honolulu.*
In section 5.3, we perform a separate study to evaluate the impact of this component in our system.

**Co-reference Resolution:** There are a lot of expressions for a given entity in unstructured text, such as pronouns and abbreviations, which act as a proxy for some real-world entity. And if these expressions are left unannotated, we may lose crucial information. Hence it is important to group all the mentions of an entity and link them to an URI. We use approach proposed by [34] to determine and replace all the linguistic expressions which refer to the same real-world entity and link them to its URI, as determined by the Entity Mapping component. The output of this layer replaces all the co-reference chains by their URI. Only *(b)* changes, since it is the only sentence with a co-reference chain.
*(b) DBpedia:Barack_Obama was elected in 2009. DBpedia:Barack_Obama was the president of the DBpedia:United_States. DBpedia:Barack_Obama belongs to DBpedia:Honolulu.*

**Triple Extraction:** In this component, we extract relation triple from plain text. This is the primary step in extracting information from unstructured text. A relation triple is a data entity composed of a subject-predicate-object. Subject and object are real-world entities and predicate describes the relationship that the subject entity has with object entity. Below we show the triples extracted from the portion of text that talk about the *birth place* of *Obama.*
*(a) {<DBpedia:Barack_Obama; was born in; DBpedia:Honolulu>, <DB-pedia:Barack_Obama; was born at; DBpedia:Honolulu>, <DBpedia:Barack_Obama; was born on; DBpedia:Honolulu>}*
*(b) {<DBpedia:Barack_Obama; belongs to; DBpedia:Honolulu>}*
*(c) {<DBpedia:Barack_Obama; grew up in; DBpedia:Honolulu>}*
The task of extracting a relation triple from plain text can be carried out by any information extraction [1, 8, 12, 24, 39] technique. These techniques extract relation triples from text by identifying relation phrases and associated arguments in a sentence without requiring a pre-specified vocabulary. We use Open Language Learning for Information Extraction (OLLIE) [36] as the triple extractor in our implementation.

**Metadata Processing:** This step is a preprocessing step for our main task of Predicate Mapping (discussed in section 3). The aim of this component is to store all possible DBpedia predicates to which a predicate from text triple can map, by using a pruning strategy. To find such a set of possible DBpedia predicates for a given text predicate, later defined as *Candidate Set*, we define two sub-tasks. Let our text triple be of the form $R_t$ =<*S*;$P_t$;*O*>. At first, we need to find the class to which *S* and *O* entities belong. If the *S* and *O* are mapped to some entities in DBpedia, we extract *S_class* and *O_class* using DBpedia class information. Otherwise, we use a Named Entity Recognizer (NER) [14] to predict it's class and map it to a DBpedia class using NER and Disambiguation (NERD) ontology [35]. For example, an NER class *Person* maps to its equivalent DBpedia class, *DBpedia:Person*. Secondly, after the subject and object class of a text triple are extracted correctly, we prune the DBpedia predicates which can surely not be the mapping for $P_t$. The idea is, if the relation triple in the text $R_t$, is defined between a *Person* class entity and a *Place* class entity, clearly the map of $P_t$ in DBpedia namespace will also be only among the predicates which have domain as *DBpedia:Person* and range as *DBpedia:Place*. So, instead of evaluating its similarity score with all the predicates of DBpedia, we only do it for relationships between a *DBpedia:Person* and a *DBpedia:Place* class. This remaining set, after pruning the unimportant predicates, is defined as the *Candidate Set (CS)* (described in section 3.2) for $P_t$. We explain in detail (section 3.3) the difference in cost of computation of previous similarity-based approach and our approach.

**Predicate Mapping:** The aim of this component is to map predicate of a text triple to its matching predicate in DBpedia. From the previous component we obtained the *Candidate Set (CS)* for $P_t$. Here, we find the similarity of the text predicate with every candidate in the *CS* using the model explained in next section. We finally map $P_t$ to the most similar candidate predicate in DBpedia. Our model is able to map *"was born in"*, *"belongs to"*, *"was born at"*, *"was born on"* and *"grew up in"* correctly to *DBpedia:birthPlace*. Hence, the output for the examples becomes:
*(a) {<DBpedia:Barack_Obama; DBpedia:birthPlace; DBpedia:Honolulu>}*
*(b) {<DBpedia:Barack_Obama; DBpedia:birthPlace; DBpedia:Honolulu>}*
*(c) {<DBpedia:Barack_Obama; DBpedia:birthPlace; DBpedia:Honolulu>}*
Simply extracting relation triples using OLLIE (as shown in fig 1) will give us a graph with four nodes (entities: {*"Barack Obama", "He", "Barack", "Honolulu"*}) and five edges (predicates: {*"was born at", "was born in", "was born on", "belongs to"*}) which clearly holds redundant information. Our approach is able to resolve all the identical mentions of entities and predicates hence forming a more precise graph with two nodes and one edge between them, i.e. *<DBpedia: Barack_Obama;DBpedia: birthPlace;DBpedia: Honolulu>*. The output of the Predicate Mapping step for a particular text triple is a triple that has its entities and predicate linked to DBpedia. The collection of such triples is our constructed KG. Thus, our system has constructed a KG over a specific text article or document. This KG makes the raw text information available for document specific tasks such as question answering and information retrieval. Furthermore, if there is some information present in this text, but not



(a) (b)

**Figure 2: (a) End-to-end Predicate Mapping model, (b) An illustrative example of Predicate Mapping, where S and O are Subject Class and Object Class respectively**

in DBpedia then we could add the missing information to DBpedia, by merging it with our constructed KG.

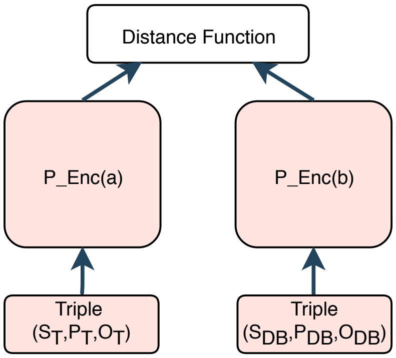

Distance Function
P_Enc(a)
P_Enc(b)
Triple ($S_T$,$P_T$,$O_T$)
Triple ($S_{DB}$,$P_{DB}$,$O_{DB}$)

**Figure 3: Overview of Predicate Mapping Model**

## 3 PREDICATE MAPPING

Let $<S_{db};P_t;O_{db}>$ denote a relation triple extracted from the text using the above pipeline. Our aim is to find a DBpedia predicate (say $P_{db}$) which is semantically similar to $P_t$ and map $P_t$ to $P_{db}$. We use a deep learning based similarity approach to find this $P_{db}$.

The idea is to get a vector representation of the text predicate and compare its similarity with possible DBpedia candidates. The representation of the predicate is formed using the Predicate Encoder ($P\_Enc(a)$ and $P\_Enc(b)$ as shown in fig. 2). The predicate encoder is a multi-stage architecture (described in Section 3.1) which uses subject and object class as initial context to generate the representation since they store valuable information about the predicate.

Let the final encoded vector representation of the two predicates obtained from the predicate encoder be $H_t = [x_1, .., x_k]$ and $H_{db} = [y_1, .., y_k]$. The similarity between the two vector representations $H_t$ and $H_{db}$ is given by the following equation:

$$f(H_t, H_{db}) = exp(-||H_t - H_{db}||) \in [0, 1] \qquad (1)$$

In the next section, we define our Predicate Mapping model.

### 3.1 Model

Our Predicate Mapping model (shown in fig. 2(a)) is a hierarchical multi-stage Siamese Network, inspired by [27]. The model consists of five layers. The *Embedding Layer* maps each word to a vector space using a pre-trained embedding model. The *Contextual Embedding Layer* utilizes the vector representations of surrounding words to refine the embeddings of the words. The *Projection Layer* forms the initial state for our Modeling Layer by densely combining subject class and object class embeddings. The *Modeling Layer* employs a Siamese Recurrent Network to create final embedding of the predicate using the combined embedding of subject class and object class as context. The *Distance Layer* computes the final similarity score.

**Embedding Layer:** This is a word embedding layer which maps each word to a high-dimensional vector space. In our model, we use $d$-dimensional pre-trained GloVe word embeddings [32]. Let $\{s_1, s_2, ...s_{k_t}\}$, $\{p_1, p_2, ...p_{l_t}\}$ and $\{o_1, o_2, ...o_{m_t}\}$ denote the sequence of words in subject, predicate and object of a text triple and similarly for a DBpedia triple. The output of this layer consists of three matrices for text triple: $S^r_t \in \mathbb{R}^{d \times k_t}$, $P^r_t \in \mathbb{R}^{d \times l_t}$, $O^r_t \in \mathbb{R}^{d \times m_t}$.

$$S^r_t = [s^r_1 s^r_2 ... s^r_{k_t}]$$
$$P^r_t = [p^r_1 p^r_2 ... p^r_{l_t}]$$
$$O^r_t = [o^r_1 o^r_2 ... o^r_{m_t}]$$

where $s^r_i \in \mathbb{R}^{d \times 1}$, $p^r_i \in \mathbb{R}^{d \times 1}$ and $o^r_i \in \mathbb{R}^{d \times 1}$ represent the embedding of $s_i$, $p_i$ and $o_i$. Similarly for DBpedia triple, we obtain: $S^r_{db}$, $P^r_{db}$ and $O^r_{db}$.

**Contextual Embedding Layer:** In this layer, we use LSTM [17] to output the encoded representation of a given sequence. We use four LSTMs, two for $S_t$ and $S_{db}$ to form subject class encodings and the other two for $O_t$ and $O_{db}$ to form object class encodings.

LSTMs adapt feedforward neural networks for sequence data where at each time-step (say $x$), updates to a hidden state vector $h_x$ are performed. In our case, the input sequence data is, $[s^r_1 s^r_2 ... s^r_{k_t}]$. These updates also rely on a memory cell containing four components (which are real-valued vectors): a memory state $c_x$, an output gate $q_x$ (that determines how the memory state affects other units), an input gate $i_x$ (that controls what gets stored in memory) and a forget gate $f_x$ (that controls what does not get stored in memory). Below are the updates performed (for $S_t$) at each time-step $x \in 1, ..., k_t$:

$$\begin{aligned} i_x &= sigmoid(W_i s^r_x + U_i h_{x-1} + b_i) \\ f_x &= sigmoid(W_f s^r_x + U_f h_{x-1} + b_f) \\ \widetilde{c_x} &= tanh(W_c s^r_x + U_c h_{x-1} + b_c) \\ c_x &= i_x \cdot \widetilde{c_x} + f_x \cdot c_{x-1} \\ q_x &= sigmoid(W_q s^r_x + U_q h_{x-1} + b_q) \\ h_x &= q_x \cdot tanh(c_x) \end{aligned} \tag{2}$$

where $W_i, W_f, W_c, W_q, U_i, U_f, U_c, U_q$ are the trainable weight matrices and $b_i, b_f, b_c, b_q$ are the trainable bias-vectors. We take the last hidden state of the LSTM (i.e. $h_{k_t}$) as our final representation. Hence, we obtain $G_t \in \mathbb{R}^{d \times 1}$, $U_t \in \mathbb{R}^{d \times 1}$ for subject classs ($S_t$) and object class ($O_t$) of text triple and $G_{db} \in \mathbb{R}^{d \times 1}$, $U_{db} \in \mathbb{R}^{d \times 1}$ for subject class ($S_{db}$) and object class ($O_{db}$) of DBpedia triple.

**Projection Layer:** This layer aims to prepare context for Modeling Layer using G and U vectors. We define

$$\begin{aligned} J_t &= \{G_t; U_t\} \\ J_{db} &= \{G_{db}; U_{db}\} \end{aligned}$$

as the concatenation of subject class and object class representation of text triple and DBpedia triple respectively. This layer maps $J_t \in \mathbb{R}^{2d \times 1}$ to $Z_t \in \mathbb{R}^{d \times 1}$ by applying a Dense Layer and similarly $J_{db} \in \mathbb{R}^{2d \times 1}$ to $Z_{db} \in \mathbb{R}^{d \times 1}$. $Z_t$ and $Z_{db}$ store the combined representation of subject-object class of text and DBpedia triple and serve as the context for finding the representation of their respective predicates.

**Modeling Layer:** In this layer, we use another pair of LSTMs to create encoded representation of text predicate and candidate predicate. We use the output of the previous layer as context, i.e., $Z_t$ and $Z_{db}$ become the first hidden states ($h_0$ in eq. (2)) for the two LSTMs respectively. The input to this LSTM is $P_t$. We obtain $H_t \in \mathbb{R}^{d \times 1}$ as the last hidden state of this *S-LSTM* (as shown in fig. 2(a)). Similary, we obtain $H_{db} \in \mathbb{R}^{d \times 1}$. $H_t$ and $H_{db}$ represent the final text and DBpedia triple encoding, respectively.

**Distance Layer:** This layer computes the similarity score between $H_t$ and $H_{db}$ using the eq. (1). This score is used to choose the most similar predicate amongst a set of predicates called *Candidate Set* (discussed in detail in the next section).

## 3.2 Candidate Selection

In the previous work, the approach to map predicate of a text triple to DBpedia predicate requires high computation cost. The high computation is because it requires an entire scan of DBpedia[4]. Thus making the computation cost in the order of total number of triples in DBpedia ($1.3 \times 10^9$). This step has many unnecessary comparisons, for example, an extracted predicate "*was born in*" is being compared with DBpedia predicates such as *DBpedia:biggestCity* and *DBpedia:altitude*. To overcome this, we identify the DBpedia predicates which can be the possible mapping for the text triple's predicate.

The idea is as follows: each predicate, say $P_{db}$ in DBpedia has a unique subject class and object class, defined in the RDF-schema as *rdfs:domain* and *rdfs:range* respectively. This defines that, in any relation triple which has $P_{db}$ as it's predicate, say $<S; P_{db}; O>$, all such $S$'s and $O$'s belong to the class defined by *rdfs:domain* and *rdfs:range*, respectively. We exploit this information about a text triple, to narrow down our search space significantly. DBpedia ontology forms a subsumption hierarchy [21], hence our new search space does not lose any potential candidates. Thus, we introduce a notion of *Candidate Set (CS)* for a text triple.

Formally, a *CS* for a text triple, say $R_t = <S; P_t; O>$, is a set of all the DBpedia predicates (stored as $<S\_class; P_{db}; O\_class>$ in the *CS*) whose predicates occur between $S$'s class and $O$'s class. In our experiments, we find that the average size of the *Candidate Set* for a predicate is approximately 200, which reduces the search space by an order of $10^6$ for each text predicate. Now that we have defined *CS* for each extracted predicate, we describe the steps to output the *closest* (most similar) DBpedia predicate for a particular $R_t$. Consider $C_t = <S\_class; P_t; O\_class>$ and the *Candidate Set* for $P_t$,

$$\begin{aligned} CS = \{&<S\_class; P^1_{db}; O\_class>, \\ &...., <S\_class; P^K_{db}; O\_class>\} \end{aligned} \tag{3}$$

where K is the size of the *Candidate Set*.
Let us do a running example for *Candidate Selection*. Consider a $R_t$ and its corresponding $C_t$ and *Candidate Set (CS)* be

$R_t$ = *<DBpedia:Barack_Obama; was born in; DBpedia:Honolulu>*
$C_t$ = *<DBpedia:Person; was born in; DBpedia:Place>*
$CS$ = {*<DBpedia:Person; DBpedia: birthPlace; DBpedia:Place>*,
*<DBpedia:Person; DBpedia: previousPost; DBpedia:Place>*,
... ,*<DBpedia:Person; DBpedia:knownFor; DBpedia:Place>*}

where K = 255 since there are 255 different predicates between *DBpedia:Person* and *DBpedia:Place*[5]. The steps are:

1. We calculate the similarity score for $C_t$ and $i^{th}$ element of the *Candidate Set* ($CS[i]$), $\forall i \in [1, K]$ by feeding $C_t$ and $CS[i]$ to our Predicate Mapping model. That is, we calculate similarity score of *<DBpedia:Person; was born in; DBpedia:Place>* with *<DBpedia:Person; DBpedia:birthPlace; DBpedia:Place>* and so on with all the elements of *CS*.
2. We find the candidate from the *CS* which gives maximum similarity score with $C_t$, say $CS[j]$. In our example, we find that *<DBpedia:Person; DBpedia:birthPlace; DBpedia:Place>* has the maximum similarity with *<DBpedia:Person; was born in; DBpedia:Place>*.
3. Finally, we map $P_t$ with the predicate of $CS[j]$ by replacing all occurrences of $P_t$ with $P^j_{db}$. That is, we update our $R_t$ to *<DBpedia:Barack_Obama; DBpedia:birthPlace; DBpedia:Honolulu>*.

[4] [19] Page 5: eq. (2) and eq. (3)

[5] We use SPARQL endpoint (http://yasgui.org/) to query DBpedia

So effectively, we compare *"was born in"* with only 255 predicates in DBpedia. We input each pair $C_t$ and $CS[i]$ to our model (as shown in fig. 2(b)) and get similarity scores for each of them. The vector representation of *"was born in"* using *Person* and *Place* as context comes out to be closest (using eq. (1)) to the vector representation *DBpedia:birthPlace* calculated using *Person* and *Place* as context. If the size of *CS* for any text predicate is zero, we discard that text triple.

## 3.3 Scalability

Consider a text triple $R_t$ =<$S_t$; $P_t$; $O_t$> and let its mapping be $R_{db}$ =<$S_{db}$; $P_{db}$; $O_{db}$>. In the previous similarity based approach [10], it does an entire scan of DBpedia to find[4] this $P_{db}$. We find that it makes a lot of unnecessary computations. In our approach, we get rid of them by defining our *Candidate Set* using SPARQL queries. *As per the latest release of DBpedia*[6] *, in order to map a single* $P_t$ *to* $P_{db}$*, our system requires approximately 200 computations while T2KG[19] essentially requires computation in the order of size of DBpedia, that is,* $1.3 \times 10^9$.

# 4 TRAINING DETAILS

To train this model in an end-to-end fashion, one would require a dataset consisting of pairs of similar triples. But, to the best of our knowledge, there is no such dataset available. Therefore, we need to train the components independently. We train two different models on two different datasets: SNLI[5] corpus and SemEval STS [18]. The SNLI dataset is a set of human-written English sentence pairs manually labeled for the task of Recognizing Textual Entailment (RTE). While the SemEval STS dataset consists of labeled cross-level semantically similar pairs of a paragraph and a sentence. The Predicate Mapping model is divided into three parts for training.

## 4.1 Contextual Embedding Layer Training

This part aims at obtaining the weights ($W_i, W_f, W_c, W_q, U_i, U_f, U_c, U_q$ in eq. (2)) for the Contextual Embedding Layer. We train the LSTM in this layer similar to a Language Model (LM)[26]. The goal of an LM is to predict the $(k+1)^{th}$ word of sequence, given the previous $k$ words. A huge chunk of text is given as input to the LSTM to train it as an LM. This way, it learns to encode a sequence of words and hence we use this trained LSTM as a sequence encoder in the Contextual Embedding Layer. We transform the sentences in the dataset into a sequence of tokens or words that is used for training the model. The weights obtained by training this model are used in all the four LSTMs in the Contextual Embedding Layer.

## 4.2 Projection Layer Training

This part aims at obtaining the weights for the Projection Layer. Since this layer, at its core, aims to reduce the dimension of the output of the Contextual Embedding Layer from $2d$x1 to $d$x1, we use an autoencoder[16] for this task. The input to the autoencoder is the concatenation of the two $d$x1 outputs of the LSTMs of the previous layer, which gives us a $2d$x1 vector. We store such $2d$x1 vectors for multiple triples and use them to train the autoencoder. The input is first transformed into a $d$x1 vector using a weight matrix $W_1 \in \mathbb{R}^{2d \times d}$, and then this intermediate vector is converted back to a $2d$x1 vector using another weight matrix $W_2 \in \mathbb{R}^{d \times 2d}$. The first part of the autoencoder reduces the input to $d$x1 vector. Hence, we use this $W_1$ matrix, as our weight matrix for this layer.

## 4.3 Modeling and Distance Layer Training:

This part aims at obtaining the weights of the Modeling Layer. We train the LSTMs in Modeling Layer and the Distance Layer as an end-to-end Siamese Network[29]. For a given pair of sentences in the dataset, first and second sentence are respectively fed to the two *S-LSTM*s. In the Distance Layer, similarity score between the last hidden states of the two *S-LSTM*s ($H_t$ and $H_{db}$) is calculated using eq. (1). With similarity label known, the corresponding contrastive loss[15] is back propagated to the *S-LSTM*s according to the equation:

$$L(Y, H_t, H_{db}) = (Y)\frac{1}{2}(D)^2 + (1-Y)\frac{1}{2}\{max(0, 1-D)\} \qquad (4)$$

where $D$ is the similarity score obtained using eq. (1) and Y is the similarity label.

# 5 EXPERIMENTS

The aim of these experiments is to evaluate the performance and impact of different components in our pipeline. We also show how the Sentence Simplification component improves the quality and quantity of triples extracted. We perform following four experiments:

## 5.1 Automatic Evaluation

The aim of this experiment is to evaluate our Predicate Mapping model. Since there is no gold standard dataset for evaluation, we build a testable dataset by setting a ground truth. Consider a text triple, say $R_t$ =<$S_t$; $P_t$; $O_t$>, which after performing Entity Mapping would become $R_t$ =<$S_{db}$; $P_t$; $O_{db}$> where $S_{db}$ and $O_{db}$ are the corresponding DBpedia URIs. Now, if there is only one predicate, say $P_{db}$, between $S_{db}$ and $O_{db}$ in DBpedia then $P_{db}$ is the ground truth for $P$. Such $R_t$ constitute our test dataset.

This experiment is performed using 140,000 randomly selected Wikipedia articles. A total of 3,271,660 triples were extracted and the ground truth was established for 58,842 triples. We compare our system's results with a baseline model defined below[7].

In the baseline model, we use cosine distance metric to calculate the similarity score between a text predicate and a DBpedia predicate. Let a text predicate ($P_t$) be a sequence of $\{t_1, t_2, ..., t_n\}$ words. We encode $P_t$ as the sum average of GLoVE vector embeddings of the $t_i's$. Similarly a DBpedia predicate ($P_{db}$) is encoded as a sum average of GLoVE embeddings of $\{d_1, d_2, ..., d_m\}$ words. The cosine similarity score between these two encodings - $P_t$ and $P_{db}$ is defined as:

$$CosineSimilarityScore = \frac{P_t \cdot P_{db}}{||P_t|| \cdot ||P_{db}||}$$

Table 2 shows comparative results using cosine metric baseline, the rule-based approach [10], and our model trained on SNLI dataset

[6] https://wiki.dbpedia.org/develop/datasets/dbpedia-version-2016-10

[7] We could not compare our results with the state-of-the-art paper ([19]) due to ambiguities in the implementation details of their paper

| | Recall | Precision | F1-score |
|---|---|---|---|
| Cosine | 0.2561 | 0.3478 | 0.2949 |
| Rule-based | 0.3514 | 0.4627 | 0.3994 |
| SNLI trained - Our approach | **0.6069** | **0.7684** | **0.6781** |
| STS trained - Our approach | 0.5382 | 0.6762 | 0.5994 |

**Table 2: Automatic Evaluation results**

[5] and SemEval STS dataset [18]. We observe that training our model on SNLI dataset gives the high *F1 score of 0.678*. It is interesting to note that the model is trained on text similarity dataset to identify the similar DBpedia triples (essentially verb phrases).

It took approximately *35 hours* to complete the steps before Predicate Mapping for 140,000 Wikipedia articles, a majority of which is network latency time due to large number of SPARQL [33] queries in defining *Candidate Set* for each triple. For the final step of Predicate Mapping, we were able to generate the mappings in approximately *six hours using the pre-trained model for 58,842 triples.*

## 5.2 Manual Evaluation

Since there is no standard evaluation metric to evaluate the quality of the Predicate Mapping step, it is necessary to perform a manual evaluation since in Experiment 1, we are able to consider a special case - when only one predicate exists between a particular subject and object. In this experiment, we randomly select 200 sentences and were able to extract 416 triples using our triple extraction step. We then map the predicates of these triples manually to DBpedia predicates. We feed this set of triples to our Predicate Mapping model and observe that we are able to map predicates with a *recall of 0.633* and with a *precision of 0.771.*

Based on our error analysis of the extracted triples, results show that the most errors i.e. *35.2% are caused due to triple extractor.* While *30.7% errors are caused from co-reference resolution step, 18.3% due entities* and *15.8% due predicates.* The rule-based approach [10] indicates that the most errors (*46%* approximately) were caused due to wrong Predicate Mapping, making this step a severe drawback. *Our Predicate Mapping along with the Sentence Simplifier is able to reduce this bottleneck leading to improvement of 0.22 in F1 score.*

## 5.3 Sentence Simplification Evaluation

Previous studies show that significant amount of errors are due to limitations of triple extraction, especially, when dealing with complex sentences. Hence in this experiment, we aim to quantify the extent to which the Sentence Simplification model improves the triple extraction performance, when dealt with complex sentences. We analyze OLLIE information extractor performance when it is fed with i) complex sentences only ii) simplified sentences using the Sentence-Simplification idea [28]. We randomly choose 100 complex sentences from Zhu dataset [40] and obtain their corresponding simple sentences and run OLLIE triple extractor on these simple sentences. We then manually classify each triple in three of the following categories: (a) *Correct triple* is a relation triple that can be justified as true, given the sentence, (b) *Incorrect triple* is a relation triple that can not be justified as true, given the sentence, and (c) *Misleading triple* is a triple which is neither correct nor incorrect but possess incomplete information, and may lead to incorrect relation if added in the KG.

| Sentence Type | Correct | Incorrect | Misleading | Total |
|---|---|---|---|---|
| Complex | 109 | 40 | 64 | 213 |
| Simple | 171 | 33 | 54 | 258 |

**Table 3: Sentence Simplification Evaluation results**

| | Accurate | Additional | Total Triples | Accuracy |
|---|---|---|---|---|
| History | 1895 | 278 | 2173 | 0.872 |
| Science | 948 | 217 | 1165 | 0.813 |

**Table 4: Redundancy Reduction Evaluation results**

As shown in table 3, there is not only increase in total number of relation triples extracted, but also a *36.25% increase in the number of correct triples* obtained and a *17.5% decrease in the number of incorrect triples* from OLLIE when fed with simplified sentences vs complex sentences. This entails that the simplification component helps construct larger KG with less ambiguity and less incorrect relationships.

## 5.4 Redundancy Reduction Evaluation

The aim of this experiment is to evaluate how well our system is able to match similar relationships and entities into a homogeneous set of URIs in order to reduce redundancies in the KG. We take a Wikipedia article, say $A$, and paraphrase it to $A'$. We combine $A$ and $A'$ into a single document, say $B$. The idea is, this $B$ will give out redundant triples since same information is paraphrased and added to it (in the form of $A$ and $A'$) and we want to evaluate how well these redundant triples are resolved by our system. Ideally, the triples generated from $A$ should be identical to the triples generated from $B$. We perform our end-to-end KG generation system over $A$ and $B$ independently and study the (a) number of *accurate triples* - triples which are identical across $A$ and $B$, and (b) number of *additional triples* - which are present in $B$ but are absent in $A$.

We use an online paraphrasing tool called spinbot[8] to paraphrase an article. We collect $1,000$ such $A$'s and build their corresponding $B$'s separately for two different topics of Wikipedia - Science and History. Some examples of Science topic are Chlorophyll, Photosynthesis, etc and History topic are French Revolution, World War, etc. We obtain a total of 2173 triples from History articles and 1165 triples from Science articles. We define accuracy for this experiment in the following manner:

$$Accuracy \quad = \quad \frac{n(a)}{n(a) + n(b)}$$

where $n(a)$ and $n(b)$ represent the number of *accurate triples* and *additional triples*, respectively. This accuracy will evaluate to 1.0 when the paraphrasing module and our system perform perfectly. The results show that, the average accuracy over Science and History articles is 84.3%. As shown in table 4, we observe that the

[8] www.spinbot.com

*accuracy for History articles is 87.24%*, slightly higher than *81.37% for Science articles*. We randomly sample 50 articles each of Science and History to qualitatively analyze the difference in accuracy and following are the observations:

- Science articles have a lesser share of facts but a lot of complex definitions which our Sentence Simplifier is often not able to correctly simplify. This could be improved by using a labeled training dataset consisting of complex scientific definitions and their corresponding simplified definitions for learning.
- History articles are almost entirely fact driven. This leads to high number of total triples as well as less ambiguity due to paraphrasing.

## 6 CONCLUSION

Most organizations have high number of textual documents that need to be processed. But domain specific document processing has not seen a lot of work and different kinds of metric are needed to evaluate such a system. In this work, we present an end-to-end system for construction of Knowledge Graph from unstructured text. Our major focus lies in:

(1) Improving the Predicate Mapping step for greater searchability in applications such as question answering and information retrieval.
(2) Developing a pruning strategy to make our KG generation system scalable.
(3) Studying the impact of the Sentence Simplification component in improving the quality of open Information Extraction techniques and evaluating the redundancy reduction in different domains of Wikipedia. Furthermore, these two experiments are novel and reproducible.
(4) Evaluating the quality of constructed KG by performing automatic and manual experiments

Even though text data from different sources have vocabulary gap, our system is able to construct a Knowledge Graph with a *F1-score of* 0.678 as shown in table 2.

As part of ongoing work, we will construct domain specific Knowledge Graphs for health and election. We will also develop a Question Answering system based on Knowledge Graphs.

## REFERENCES

[1] Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D Manning. 2015. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Vol. 1. 344–354.

[2] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*. Springer, 722–735.

[3] Isabelle Augenstein, Sebastian Padó, and Sebastian Rudolph. 2012. Lodifier: Generating linked data from unstructured text. In *Extended Semantic Web Conference*. Springer, 210–224.

[4] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. AcM, 1247–1250.

[5] Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326* (2015).

[6] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka Jr, and Tom M Mitchell. 2010. Toward an architecture for never-ending language learning.. In *AAAI*, Vol. 5. Atlanta, 3.

[7] Roldano Cattoni, Francesco Corcoglioniti, Christian Girardi, Bernardo Magnini, Luciano Serafini, and Roberto Zanoli. 2012. The KnowledgeStore: an Entity-Based Storage System.. In *LREC*. Citeseer, 3639–3646.

[8] Luciano Del Corro and Rainer Gemulla. 2013. Clausie: clause-based open information extraction. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, 355–366.

[9] Amira Abd El-atey and Sherif El-etriby. 1911. Semantic Data Extraction from Infobox Wikipedia Template. *Coordinates* 30 (1911), 261.

[10] Peter Exner and Pierre Nugues. 2012. Entity extraction: From unstructured text to dbpedia rdf triples. In *The Web of Linked Entities Workshop (WoLE 2012)*. CEUR, 58–69.

[11] Anthony Fader, Stephen Soderland, and Oren Etzioni. 2011. Identifying relations for open information extraction. In *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics, 1535–1545.

[12] Anthony Fader, Stephen Soderland, and Oren Etzioni. 2011. Identifying Relations for Open Information Extraction. In *Proceedings of the Conference of Empirical Methods in Natural Language Processing (EMNLP '11)*. Edinburgh, Scotland, UK.

[13] James Fan et al. 2010. Prismatic: Inducing knowledge from a large scale lexicalized relation resource. In *Proceedings of the NAACL HLT 2010*. ACL, 122–127.

[14] Jenny Rose Finkel et al. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*. ACL, 363–370.

[15] Raia Hadsell, Sumit Chopra, and Yann LeCun. 2006. Dimensionality reduction by learning an invariant mapping. In *null*. IEEE, 1735–1742.

[16] Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *science* 313, 5786 (2006), 504–507.

[17] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[18] David Jurgens, Mohammad Taher Pilehvar, and Roberto Navigli. 2014. Semeval-2014 task 3: Cross-level semantic similarity. In *Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)*. 17–26.

[19] N Kertkeidkachorn and R Ichise. 2017. T2KG: An end-to-end system for creating knowledge graph from unstructured text. In *Proceedings of AAAI Workshop on Knowledge-based Techniques for Problem Solving and Reasoning*. 743–749.

[20] Vincent Kríž, Barbora Hladká, Martin Nečaskỳ, and Tomáš Knap. 2014. Data extraction using NLP techniques and its transformation to linked data. In *Mexican International Conference on Artificial Intelligence*. Springer, 113–124.

[21] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. 2015. DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web* 6, 2 (2015), 167–195.

[22] Yeqing Li. 2017. Research and Analysis of Semantic Search Technology Based on Knowledge Graph. In *Computational Science and Engineering (CSE) and Embedded and Ubiquitous Computing (EUC), 2017 IEEE International Conference on*, Vol. 1. IEEE, 887–890.

[23] Zhenghao Liu, Chenyan Xiong, Maosong Sun, and Zhiyuan Liu. 2018. Entity-Duet Neural Ranking: Understanding the Role of Knowledge Graph Semantics in Neural Information Retrieval. *arXiv preprint arXiv:1805.07591* (2018).

[24] Mausam, Michael Schmitz, Robert Bart, Stephen Soderland, and Oren Etzioni. 2012. Open Language Learning for Information Extraction. In *Proceedings of Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CONLL)*.

[25] Pablo N Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. 2011. DBpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th international conference on semantic systems*. ACM, 1–8.

[26] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.

[27] Jonas Mueller and Aditya Thyagarajan. 2016. Siamese Recurrent Architectures for Learning Sentence Similarity.. In *AAAI*, Vol. 16. 2786–2792.

[28] Shashi Narayan and Claire Gardent. 2014. Hybrid simplification using deep semantics and machine translation. In *Proceedings of the 52nd Annual Meeting of the ACL (Volume 1: Long Papers)*, Vol. 1. 435–445.

[29] Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru. 2016. Learning text similarity with siamese recurrent networks. In *Proceedings of the 1st Workshop on Representation Learning for NLP*. 148–157.

[30] Feng Niu, Ce Zhang, Christopher Ré, and Jude W Shavlik. 2012. DeepDive: Web-scale Knowledge-base Construction using Statistical Learning and Inference. *VLDS* 12 (2012), 25–28.

[31] Rohan Padhye, Debdoot Mukherjee, and Vibha Singhal Sinha. 2014. Api as a social glue. In *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM, 516–519.

[32] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[33] Eric Prud, Andy Seaborne, et al. 2006. SPARQL query language for RDF. (2006).

[34] Karthik Raghunathan, Heeyoung Lee, Sudarshan Rangarajan, Nathanael Chambers, Mihai Surdeanu, Dan Jurafsky, and Christopher Manning. 2010. A Multi-Pass Sieve for Coreference Resolution. In *Empirical Methods in Natural Language Processing (EMNLP)*.

[35] Giuseppe Rizzo and Raphaël Troncy. 2012. NERD: a framework for unifying named entity recognition and disambiguation extraction tools. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the ACL*. ACL, 73–76.

[36] Michael Schmitz et al. 2012. Open language learning for information extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. ACL, 523–534.

[37] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*. ACM, 697–706.

[38] Zhouxia Wang, Tianshui Chen, Jimmy Ren, Weihao Yu, Hui Cheng, and Liang Lin. 2018. Deep reasoning with knowledge graph for social relationship understanding. *arXiv preprint arXiv:1807.00504* (2018).

[39] Aaron Steven White and Reisinger. 2016. Universal Decompositional Semantics on Universal Dependencies. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, 1713–1723. https://aclweb.org/anthology/D16-1177

[40] Zhemin Zhu, Delphine Bernhard, and Iryna Gurevych. 2010. A monolingual tree-based translation model for sentence simplification. In *Proceedings of the 23rd international conference on computational linguistics*. Association for Computational Linguistics, 1353–1361.