Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

NELLIE: Never-Ending Linking for Linked Open Data

ABDULLAH FATHI AHMED[1,2], MOHAMED AHMED SHERIF[1,2], AND AXEL-CYRILLE NGONGA NGOMO.[1,2],

[1] Data Science, Department of Computer Science, Paderborn University, Warburger Str. 100, 33098 Paderborn, Germany (e-mail: afaahmed@mail.upb.de, mohamed.sherif@upb.de, axel.ngonga@upb.de)
[2]Department of Computer Science, University of Leipzig, 04109 Leipzig, Germany

Corresponding author: Abdullah Fathi Ahmed (e-mail:afaahmed@mail.upb.de).

Acknowledgement:We acknowledge the support of the German Federal Ministry for Economic Affairs and Climate Action (BMWK) within project SPEAKER (grant no 01MK20011A), the German Federal Ministry of Education and Research (BMBF) within project PORQUE (grant no 01QE2056C), the German Research Foundation (DFG) within the project INGRID (grant no NG 105/7-3), the Ministry of Culture and Science of North Rhine-Westphalia (MKW NRW) within the project SAIL (grant no NW21-059D), and the European Union's Horizon Europe research and innovation programme within project ENEXA (grant no 101070305).

ABSTRACT

Knowledge graphs (KGs) that follow the Linked Data principles are created daily. However, there are no holistic models for the Linked Open Data (LOD). Building these models(i.e., engineering a pipeline system) is still a big challenge in order to make the LOD vision comes true. In this paper, we address this challenge by presenting NELLIE, a pipeline architecture to build a chain of modules, in which each of our modules addresses one data augmentation challenge. The ultimate goal of the proposed architecture is to build a single fused knowledge graph out of the LOD. NELLIE starts by crawling the available knowledge graphs in the LOD cloud. It then finds a set of matching KG pairs. NELLIE uses a two-phase linking approach for each pair (first an ontology matching phase, then an instance matching phase). Based on the ontology and instance matching, NELLIE fuses each pair of knowledge graphs into a single knowledge graph. The resulting fused KG is then an ideal data source for knowledge-driven applications such as search engines, question answering, digital assistants and drug discovery. Our evaluation shows an improved $Hit@1$ score of the link prediction task on the resulting fused knowledge graph by NELLIE in up to 94.44% of the cases. Our evaluation also shows a runtime improvement by several orders of magnitude when comparing our two-phases linking approach with the estimated runtime of linking using a naïve approach.

INDEX TERMS Knowledge Graphs, Linked Data, Semantic Web, Data Augmentation, Link Discovery, Data Fusion, Data Integration, Link Prediction, LOD

I. INTRODUCTION

The number of heterogeneous knowledge graphs that obey the principles of linked data rises steadily. These KGs are broadly used in data-driven applications, including information retrieval, Natural Language Processing (NLP), recommendation systems, search engines, conversational agents, e-commerce solutions, and drug discovery. Currently, there are no holistic models for the LOD to build a single fused knowledge graph out of the LOD (i.e., the development of a 24/7 solution (similar to the Never Ending Language Learner) for fusing knowledge graphs on the LOD).

For LOD to have such a complete model, the instances and ontologies in each KG must be linked. Currently, only a small number of such KGs are linked. In particular, the current statistic[1] of LOD[2] shows that there are 1564 KGs with 395.121 billion triples and only 2.72 billion links (0.07%) among them. Therefore, discovering links among these KGs is a major challenge for achieving the vision behind the LOD[3].

Establishing links is a tedious process when performed manually, especially in giant KGs such as DBpedia[4], Linked

[1]Accessed 10.03.2022 https://lod-cloud.net/#about, retrieved using https://github.com/lod-cloud/lod-cloud-draw/blob/master/scripts/count-data.py
[2]https://lod-cloud.net/
[3]https://www.w3.org/DesignIssues/LinkedData.html
[4]https://wiki.DBpedia.org/

Geo Data (LGD) [5], Bio2RDF[6], KEGG [1] and Wikidata[7]. In addition to the ever-increasing number of published KGs, the size of individual KGs increases with each new edition. For example, DBpedia has grown from 103 million triples (DBpedia 2.0), representing 1.95 million entities, to 10.094 billion triples representing more than 8.85 million entities in 2022[8]. Moreover, as the number of independent data providers grows, the simultaneous publication of KGs with the same information is more likely to take place. For instance, DBLP have been published by several bodies,[9] leading to duplicate content in the Data Web. Furthermore, different KGs contain different facets concerning the same data. For example, the drug data within the DrugBank[10] KG are mainly describe the drugs' interactions, pharmacology, chemical structures, targets and metabolism, while The Sider[11] dataset contains data concerning the drugs' side effects. As a result of such huge data expansion as well as multifaceted data publishing, there is a growing demand for the data augmentation tasks such as ontology and instance linking as well as data fusion.

Many frameworks have been developed to address different data augmentation challenges. Prior to fusing KGs, such systems mainly identify semantically equivalent entities in different KGs, where they try to achieve both high effectiveness and efficiency in the linking process. For instance, LogMap [2] and Codi [3] use structural matching based on the ontology structure to discover links between ontologies. Nentwig et al. [4] list many data augmentation systems that have been developed in the last two decades. For example, LIMES [5], [6] and SILK [7] apply matching strategies on instance level for computing the property values. Nentwig et al [4] address many challenges and aspects of the current link discovery frameworks. In a more recent survey [8], Mountantonakis and Tzitzikas presented some linked data integration approaches, including link discovery and KG fusion. For fusing data, the linked data quality assessment and fusion Sieve [9] is proposed, which is integrated into the linked data integration framework (LDIF) [10]. DEER [11] is another data augmentation framework that is capable of performing both links discovery and fusion to produce enriched data.

In this work, we propose NELLIE, a pipeline architecture to build a chain of modules, in which each of our modules addresses one data augmentation challenge. NELLIE first addresses the problem of finding relevant KGs to be integrated. Thereafter, NELLIE tackles the KG data integration task on both the ontology and instance levels. NELLIE then fuses the matched classes and instances to generate a fused KG. Finally, NELLIE carries out KG embedding of the resulting fused KG. The ultimate goal of the proposed architecture is to build a single fused knowledge graph out of the LOD (i.e., the development of a 24/7 solution (similar to the Never Ending Language Learner) for fusing knowledge graphs on the LOD), especially since such a graph does not exist yet.

Our proposed architecture consists of three layers: the core layer, the application layer, and the publication layer. In this paper, we pay more attention to the core layer as it contains the main components and modules of our architecture. In particular, we address the following challenges in our paper: 1) KGs matching (i.e., matching KGs based on their content); 2) KGs linking, including ontology and instance matching; 3) KGs fusion and 4) KGs embedding. Note that, all these challenges are implemented in our core layer. In the application layer, we address the link prediction challenge to evaluate the impact of our KGs fusion on the link prediction task. Figure 1 shows the NELLIE architecture. We summarize our contributions as follows:

- We develop a pipeline modular architecture as a milestone toward the 24/7 linking and fusing the LOD.
- We propose the two-phases linking strategy starting with ontology matching, then instance matching.
- In the KG matching stage, we implemented the three presented methods ourselves.
- In the ontology matching stage, we implemented the content-based class matching ourselves and integrated two state-of-the-art systems.
- For the instance matching stage, we base our implementation on the state-of-the art link discovery framework LIMES [6], where we modified the way of training the WOMBAT [12] to generate link specifications. We then integrated LIMES into NELLIE as listed in Algorithm 1in the paper.
- In the KG fusion stage, we implemented the additive fusion operator with many different fusion strategies. Finally, We study the impact of KGs fusion on the link prediction task.

We evaluated our two-phase linking by computing a pseudo-F-Measure. We also evaluated our approach on the link prediction task and studied the impact of KG fusion on this task. We used different KGs and different link prediction models. Evaluating the efficiency and dependability of NELLIE as a whole is worthy of consideration but is currently too resource-intensive to implement. We used existing benchmarks for the sake of comparability. However, we do agree that the benchmarks we have now are made for specific subtasks like link prediction, ontology matching, and instance matching.

The rest of this paper is structured as follows: Section II introduces the preamble and context of this work. Then, we give an overview of our approach in Section III. We then evaluate and discuss the results of our system in Section IV. After a brief review of related work in Section V, we conclude our work with some final remarks and future work in Section VI.

[5]http://linkedgeodata.org/About
[6]https://download.bio2rdf.org/release/4
[7]https://www.wikidata.org/
[8]Accessed in 10.03.2022 fromhttps://DBpedia.org/sparql
[9]http://dblp.l3s.de/, http://datahub.io/dataset/fu-berlin-dblp and http://dblp.rkbexplorer.com/.
[10]https://go.drugbank.com/
[11]http://sideeffects.embl.de/

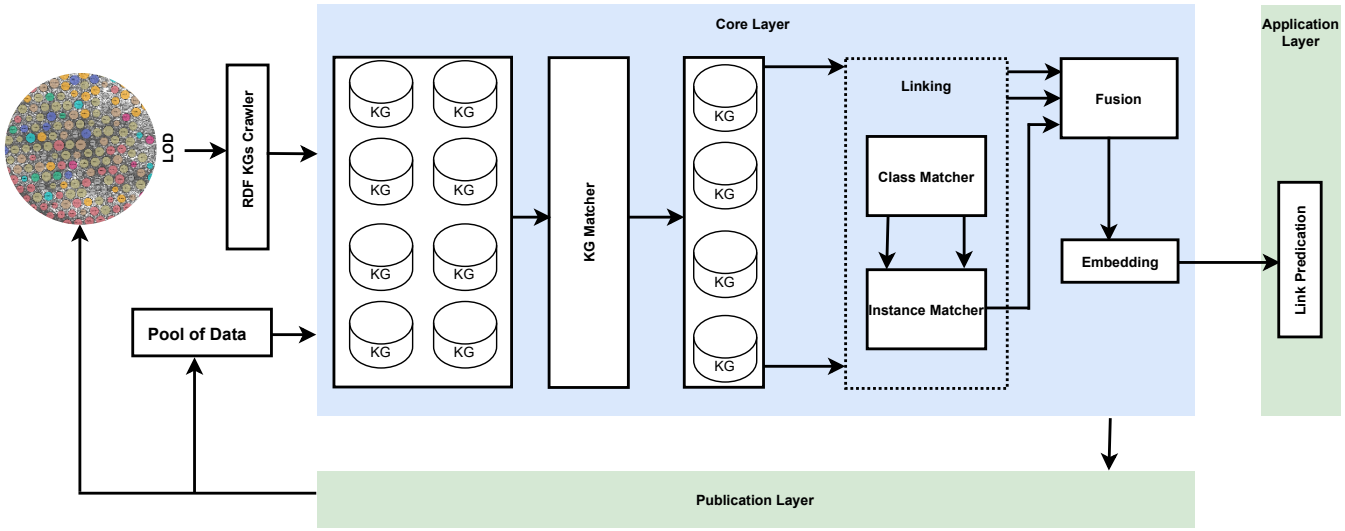

FIGURE 1: The modular pipeline architecture of NELLIE.

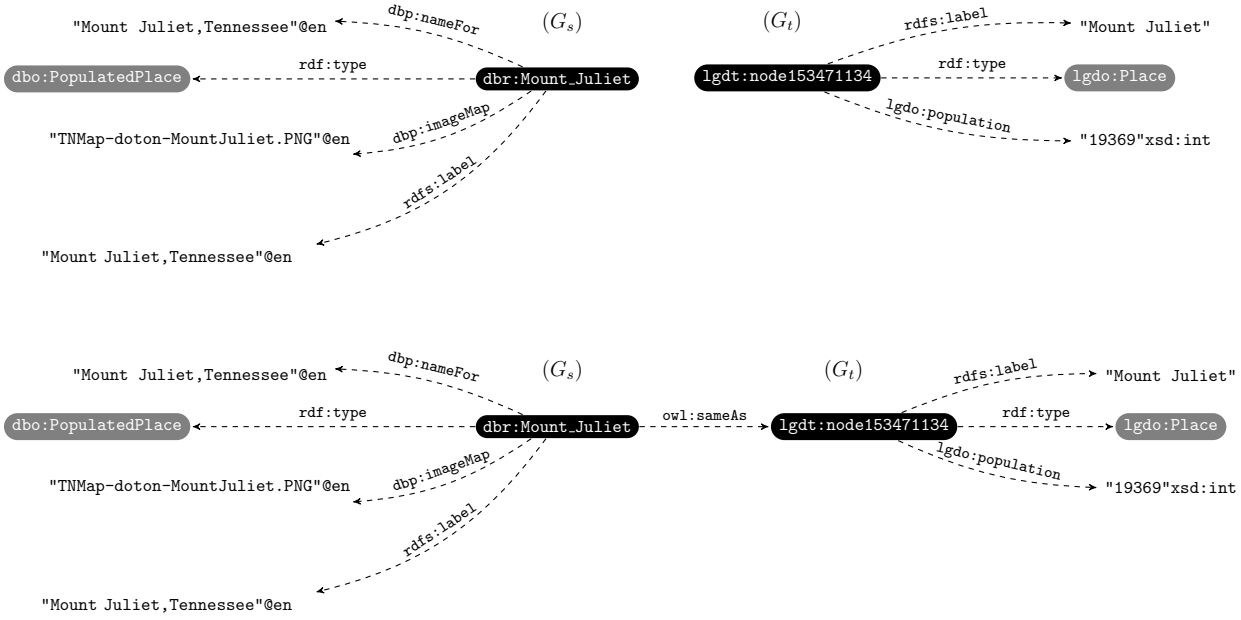

FIGURE 2: Example of linking the `DBpedia` resource `dbr:Mount-Juliet` with the `LinkedGeoData` resource `lgdt:node153471134` using LIMES link discovery framework.

Our source code is available on [12].

II. PRELIMINARY

In this section, we present the core of the formalization and notation necessary to implement NELLIE.

Knowledge Graph. A Knowledge Graph (KG) G is a set of triples $(s, p, o) \in (\mathcal{R} \cup \mathcal{B}) \times \mathcal{P} \times (\mathcal{R} \cup \mathcal{L} \cup \mathcal{B})$, where $\mathcal{R}$ is the set of all resources, $\mathcal{B}$ is the set of all blank nodes, $\mathcal{P}$ is the set of all predicates, and $\mathcal{L}$ the set of all literals.

[12]https://github.com/dice-group/NELLIE

Knowledge Graphs Matching. Given a source KG G and a set of target KGs $\mathcal{T} = \{G_1 \cdots G_n\}$. The goal of KG matching is to rank all KGs within $\mathcal{T}$ based on their likelihood of containing entities that have the potential to be linked to entities in G [13].

Link Specifications.

Our linking is based on the link discovery framework LIMES [6]. LIMES uses link specifications (LSs) to express the conditions necessary for linking resources in input KGs. A LS consists of two types of atomic components: *similarity measures* m and *operators* op.

Similarity measures m is used to compare the property values of input instances. A thresholded similarity measure is an atomic link specification. The *operators* op allow combining link specifications to create complex specifications. In detail, a similarity measure m is defined as a function $m : G_s \times G_t \rightarrow [0, 1]$. A LS is called atomic when it only contains one similarity measure, while a complex specification (complex LS) can be obtained by gluing two specifications L_1 and L_2 through an operator op that combines the results of two LSs L_1 and L_2. Here, we use the operators $\sqcap$, $\sqcup$ and $\setminus$ as they are complete and frequently used to define LS [12]. A LS is also called linkage rule in the literature [7]. Note that a LS can be generated manually or automatically. In NELLIE, we use the state-of-the-art algorithm WOMBAT [12] to automatically generate LS. WOMBAT learns link specifications based on the concept of generalisation in quasi-ordered spaces. We use the unsupervised version of WOMBAT. WOMBAT is integrated to LIMES.

Ontology Matching. Given two sets of classes C_s and C_t and a relation r (e.g., `owl:equivalentClass`), the goal of ontology matching is to find all pairs $(c_i, c_j) \in C_s \times C_t$ such that $r(c_i, c_j)$ holds [2] (e.g., `owl:equivalentClass(City, Town)`.

Instance Matching. The instance matching problem can be expressed as follows: Given two sets of resources G_s and G_t and a relation r (e.g., `owl:sameAs`), the goal of the instance matching is to find all pairs $(s, t) \in G_s \times G_t$ such that $r(s, t)$ holds (e.g., `owl:sameAs(Munich, München)`. The result is produced as a set of links called a mapping: $M = \{(s, r, t)|s \in G_s, t \in G_t\}$. Optionally, a similarity score ($sim \in [0, 1]$) calculated by the instance matching approach can be added to the entries of mappings to express the approach's confidence in the computed links [6].

Knowledge Graphs Fusion. Let G_s be a finite source KG and G_t be a finite target KG. The aim of KG fusion is to find a consolidated KG $G_{s \oplus t}$ that contains a fused version of both related entities from G_s and G_t. We assume that we have a mapping M_{merge} that contains a set of pairs of similar entities among G_s and G_t. A KG fusion approach fuses each pair of similar entities into a single entity applying some predefined fusion strategy operator $\oplus$.

Link Prediction. Given a subset of all true triples, the goal of link prediction is to learn a *scoring function* ϕ for each possible triple (e_s, r, e_o), where e_s is the subject entity e_o is the object entity and r is a relation. In the case of linear models such as TuckER [14], ComplEx [15], and DistMult [16] , the scoring function is a specific form of tensor factorization, while in non-linear models, the scoring function is a more complex (deep) neural network architecture. For a particular triple, a score is either positive in case a true fact is predicted by the model or negative for a false one. Furthermore, logistic sigmoid function is typically applied to the score to return a corresponding probability prediction $p = \sigma(s) \in [0, 1]$ as to if a certain fact is true. Table 1 lists the score functions of three state-of-the-art link prediction models we selected for our experiments. All three models are linear.

III. APPROACH

NELLIE is a modular pipeline architecture consists of three layers: the core layer, the publication layer and the application layer. The NELLIE architecture is depicted in Figure 1. In this paper, we focus on building the core layer as it is the backbone of our system. In the following, we explain the core layer components which are *KG matching, linking, fusion, and embedding*. Given a set of KGs $\{G_1, \ldots, G_n\}$ as input for our system, which are available either from the web of Data (LOD[13]) or stored in a local storage (Pool of Data).

A. KNOWLEDGE GRAPHS MATCHING

For the current version of NELLIE, we implemented three methods for KGs matching:

- *Metadata-Based KGs Matching.* In this method, we first collect KGs' metadata from the LOD Cloud[14]. The KGs' metadata include various features of each KG such as links to other KGs, website, SPARQL endpoint, keywords and domain. We then configure the link discovery framework LIMES [6] to match KGs using the `exact match` string similarity among both `keywords` and `domain` features. We provide the full LIMES configuration file in Listing 3.
- *Content-Based KGs Matching.* For each KG, we retrieve all the text within the literal objects using the SPARQL query in Listing 1. We then concatenate all the literals contained in each KG in order to generate a content document for each input KG. Afterward, we carry out a preprocessing of each KG content document by applying:
  1) *Tokenization.* We perform word tokenization by breaking a raw text into words (tokens) using *White Space Tokenization*[15]
  2) *Stop words removal:* We remove all stop words such as `{a, an, the, in, ···}` to increase the performance during string similarity measure.
  3) *Text Normalization.* We use `Normalization Form KC (NFKC)` [16]Next, we clean the text from numbers and special symbol using regular expressions.

  To this end, we store the set of tokens from the source KG as the first document $A = \{a_1 \ldots a_n\}$ and the set of tokens from the target KG as the second document $B = \{b_1 \ldots b_n\}$. To calculate the similarity between A and B, we use the following similarities `Jaccard`, `Cosine` with TF-IDF document vectors, `weighted Jaccard`, `Dice`, and `BERT` [17].
  Given $A = \{a_1 \ldots a_n\}$ a set of tokens as a first document and $B = \{b_1 \ldots b_n\}$ a set of tokens as a second document [17], we can compute the `weighted`

[13]https://lod-cloud.net/
[14]https://lod-cloud.net/lod-data.json
[15]https://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/process/WhitespaceTokenizer.html
[16]https://docs.oracle.com/javase/tutorial/i18n/text/normalizerapi.html
[17]We trim the longer document to n tokens as the shorter one.

TABLE 1: Scoring functions for three state-of-the-art link prediction models, together with the dimensionality of their relation parameters and major terms of their space complexity. $\overline{\mathbf{e}}_o \in \mathbb{C}^{d_e}$ is the complex conjugate of $\mathbf{e}_o$, $\underline{\mathbf{e}}_s, \underline{\mathbf{w}}_r \in \mathbb{R}^{d_w \times d_h}$ denote a 2D reshaping of $\mathbf{e}_s$ and $\mathbf{w}_r$ respectively, $\mathbf{h}_{e_s}, \mathbf{t}_{e_s} \in \mathbb{R}^{d_e}$ are the head and tail entity embedding of entity e_s, and $\mathbf{w}_{r^{-1}} \in \mathbb{R}^{d_r}$ is the embedding of relation r^{-1} (which is the inverse of relation r). $\langle\cdot\rangle$ denotes the dot product and $\times_n$ denotes the tensor product along the n-th mode, and $\mathcal{W} \in \mathbb{R}^{d_e \times d_e \times d_r}$ is the core tensor of a Tucker decomposition.

Model	Scoring Function	Relation Parameters	Space Complexity
TuckER [14]	$\mathcal{W} \times_1 \mathbf{e}_s \times_2 \mathbf{w}_r \times_3 \mathbf{e}_o$	$\mathbf{w}_r \in \mathbb{R}^{d_r}$	$\mathcal{O}(n_e d_e + n_r d_r)$
DistMult [16]	$\langle \mathbf{e}_s, \mathbf{w}_r, \mathbf{e}_o \rangle$	$\mathbf{w}_r \in \mathbb{R}^{d_e}$	$\mathcal{O}(n_e d_e + n_r d_e)$
ComplEx [15]	$\text{Re}(\langle \mathbf{e}_s, \mathbf{w}_r, \overline{\mathbf{e}}_o \rangle)$	$\mathbf{w}_r \in \mathbb{C}^{d_e}$	$\mathcal{O}(n_e d_e + n_r d_e)$

```
SELECT DISTINCT  ?literal
WHERE {
  ?s ?p ?literal
  FILTER isLiteral(?literal)
}
```

Listing 1: SPARQL query to retrieve literal objects.

```
PREFIX owl:<http://www.w3.org/2002/07/owl#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?c where {
    ?c a owl:Class.
    FILTER NOT exists {[] rdfs:subClassOf ?c}
}
```

Listing 2: SPARQL query for finding leaf classes.

`Jaccard` similarity between the two preprocessed content documents using the formula:

$$J_w = \frac{\sum_i^n \min(a_i, b_i)}{\sum_i^n \max(a_i, b_i)},$$

where a_i is a token in A and b_i is a token in B.
For `Cosine` with TF-IDF document vectors, we used the framework [18] developed by DKPro, where we start by calculating the TF-IDF document vectors for each document, then, applying cosine similarity.

- *Manual KGs Matching.* In order to evaluate the performance of NELLIE within a small set of KGs, we manually select some KGs that belong to the *biology* domain: `Kegg`, `Drugbank`, `Sider`, `Omim`, `Sgd`. We also select to match the two KGs `LGD` and `DBpedia`. Although `LGD` belongs to geographic domain and `DBpedia` belong to general domain, they still have potential to be linked since they have many classes in common (e.g., organization, place, location and city) with many instances that refer to the same physical facts. For instance, `:Mount_Juliet,_Tennessee` is a city located in western Wilson County, Tennessee as described in `DBpedia` and `:node153471134` refers to the same city in `LGD`, which we used as our running example (See Figure 2).

B. LINKING

For each pair of matched KGs, we carry out our two-phases linking process. In particular, we first perform ontology matching followed by instance matching. In the following, we explain these two linking phases in details.

1) Class Matching

Based on our formal definition of ontology matching in Section II, we implemented *Content-Based Class Matching* ourselves and integrated two state-of-the-art systems:

[18]https://github.com/dkpro/dkpro-similarity

- *Content-Based Class Matching.* We match classes based on the assumption that similar classes describe similar things. Therefore, we measure class similarity based on the overall similarity of the literal objects within those classes.
We start our class matching process by extracting all classes $\mathcal{C}_s$ and $\mathcal{C}_t$ from source KG G_s and target KG G_t, respectively. As shown in Listing 2, we query only for the most specific classes (i.e., the leaf classes) of each KG. However, a more specific list of classes can be provided by the user if necessary. We then rank the properties for each class by calculating their coverage and pick up only the properties with a coverage that exceed a certain propriety-coverage threshold $\beta \in [0, 1]$ defined by the user. The goal of the ranking of properties is to make sure that only the most important properties have been retrieved. For instance, properties such `(label, name, title)` have a high coverage which lead to the retrieval of more information. Formally, we query for proprieties with $coverage(p) \geq \beta$), where the coverage is defined as

$$coverage(p) = \frac{|\{s : (s, p, o) \in c_i\}|}{|\{s : \exists q\ (s, q, o) \in c_i\}|}, \qquad (1)$$

where $c_i \in \mathcal{C}_s$. β is a user defined value between $[0, 1]$. For the target KG, we replace c_i by c_j in Equation 1. After extracting all classes and ranking properties, we retrieve only the objects with literal values using SPARQL queries. The retrieved data (i.e., the literals) require to be pre-processed before it can be used for the class matching task. We used the same preprocessing steps as in the case of content-based KG matching (see Section III for more details). Accordingly, we store the set of the distinct tokens (i.e., words) that belong to each class $c_i \in G_s$ as $\mathcal{L}_i = \{l_1 \cdots l_n\}$. Formally, $(key = c_i, value = \mathcal{L}_i)$. We repeat the same proce-

dures for each target class $c_j \in G_t$. Now, the classes together with their cleaned lateral objects values are ready for matching. Our content-based class matching is formally defined as $ClassMatching(c_i, \mathcal{L}_i, c_j, \mathcal{L}_j)$, where $c_i \in G_s$, $c_j \in G_t$, G_s is the source KG and G_t is the target KG. We define the class similarity threshold $\tau \in [0, 1]$. In case the $StringSimilarity(\mathcal{L}_i, \mathcal{L}_i) \geq \tau$, then c_i is equivalent to c_j. These equivalent pairs of class are stored in a list of $equivalentClasses(c_i, c_j)$. By default, we use the `Jaccard` string similarity to measure the similarity between $\mathcal{L}_i$ and $\mathcal{L}_j$. Still, the user can configure NELLIE to use other string similarities.

- *Class matching using LogMap.* LogMap [2] is a highly scalable ontology matching system with built-in reasoning and diagnostic mechanisms capabilities. LogMap uses highly optimized data structures to index the input ontologies lexically and structurally. LogMap starts by generating an initial set of anchor mappings (with almost exact lexical correspondences) and give each of them a confidence score. The main part of LogMap is an iterative process that starts with the initial anchors and alternates between mapping repair and mapping discovery. LogMap offers a sound and highly scalable (but potentially incomplete) ontology reasoner as well as a greedy diagnosis method to find and correct unsatisfiable classes on the fly during the matching process. Given the ability of LogMap to successfully match semantically rich ontologies of classes as well as its scalability, we embedded LogMap into NELLIE as an external library.
- *Class matching using FCA-Map.* FCA-Map [18] is based on formal concept analysis to find and evaluate mappings across ontologies, including one-to-one mappings, complicated mappings and correspondences between object characteristics. It generates lexical mappings from class names and labels, as well as mappings based on ontology structures. FCA-Map generates three types of formal contexts before extracting mappings from the resultant lattices. To begin, the token-based formal context illustrates how class names, labels, and synonyms all share lexical tokens, leading to lexical mappings (anchors) between ontologies. Second, the relation-based formal context specifies how classes are connected to anchors in taxonomic, partonomic, and disjoint ways, yielding positive and negative structural evidence for lexical matching validation. Third, the positive relation-based context may be leveraged to find more structural mappings once incoherence has been rectified [19]. Thus, we can use FCA-Map to extract lexical and structural mappings of matched classes, objects, and data attributes. As in the case of LogMap, we embedded FCA-Map into NELLIE as an external library.

Note that, this class matching phase reduces the runtime needed for the instance matching phase (our second linking phase) as we only perform instance matching among instances of the matched pairs of classes.

2) Instance Matching

Based on our formal definition of *instance matching* in Section II, we focus only on the `owl:sameAs` as the relation r between s and t. We rely on LIMES [6] as it is a state-of-the-art declarative link discovery framework with open source implementation that can be easily adopted and extended in NELLIE. The Algorithm 1 shows the procedures we follow to establish the linking among the instances of the source KG G_s and target KG G_t. Computing the mapping *M* among all source and target KGs' instances in a trivial way would result on a quadratic complexity (i.e., $O(|G_s| \times |G_t|)$). Therefore, we compute the approximate mapping $M' = \{(s, t) \in G_s \times G_t \ \texttt{owl:sameAs(s,t)} \geq \theta\}$, where θ is a threshold between $[0, 1]$ to filter out all pairs with similarity measures less than θ.

While Lines 1-11 in Algorithm 1 describe the configuration and preparation for instance matching using LIMES, Lines 12-18 describe the preparation of caches to train WOMBAT. The goal of the procedures stated in (Lines 12-18) is to reduce the training time in case there is a large KG. For instance, the idea in Line 12 is to filter out any cache that has a small number of instances (i.e., less than 100). Accordingly, we define the parameter *the minimum cache size* mcs for both source and target caches. For example, setting mcs to 100 means that the caches must contain at least 100 instances. We also define the parameter *minimum sample size* mss. The large size of cache increases the training time of WOMBAT. Therefore, the parameter mss plays an important role in such a case (i.e., cache size $> mss$) by only training WOMBAT on a sample of the cached data. For example, in case we have a source cache of size of 10000 instances, target cache of size 5000 instances and mss with the value of 4000, we then select a sample of size 4000 instances from the smaller cache which is the target cache in our example here. By taking the sample from the smaller cache, we have a better potential to find matches, if such a matches exists. As we can see in (Lines 13-16) in Algorithm 1. We train WOMBAT then by the source and target training caches from previous step in (Line 19) to generate the best link specification. Using LIMES, we generate the mapping among the instances of the pair of the input classes in (Line 20) by applying the best link specification to the original KG.

C. KNOWLEDGE GRAPH FUSION

In order to perform the KGs fusion, we merge all the mappings $M_{set} = \{M_1 \dots M_n\}$ of the matched instances (from the previous linking step) into one universal mapping $M_{merge} = M_1 \cdots \cup M_n$. Accordingly, the KG fusion task uses M_{merge} to fuse G_s and G_t. In the following, we present our fusion operator and strategies implemented so far in NELLIE.

Algorithm 1: Linking of Knowledge Graphs

input : $EQ = \{(c_{s_1}, c_{t_1}) \dots (c_{s_n}, c_{t_n})\}$ is list of equivalent classes
input : mcs is the minimum cache size
input : mss is the minimum saple size
output : *approximate mapping*: $M' = \{(s,t) \in G_s \times G_t : sameAs(s,t) \geq \theta\}$

foreach $(c_{s_i}, c_{t_i}) \in EQ$ **do**
  p_{s_i}=GetPropertiesWitAtLeast Covering(c_{s_i}, β);
  p_{t_i}=GetPropertiesWitAtLeast Covering(c_{t_i}, β);
  // Configure LIMES
  LIMES.sourceKG(G_s);
  LIMES.sourceKGRestriction(c_{s_i});
  LIMES.sourceKGProperties(p_{s_i});
  $souceCache$ = LIMES.fillSourceCache();
  LIMES.targetKG(G_t);
  LIMES.targetKGRestriction(c_{t_i});
  LIMES.targetKGProperties(p_{t_i});
  $targetCache$ = LIMES.fillTargetCache();
  if $souceCache.size() > mcs$ & $targetCache.size() > mcs$ **then**
    if $souceCache.size() > mss$ & $targetCache.size() > mss$ **then**
      $sourceTrainingCache = Max(souceCache, targetCache)$;
      $targetTrainingCache = Sample(Min(souceCache, targetCache))$;
    else
      $sourceTrainingCache = souceCache$;
      $targetTrainingCache = targetCache$;
  $BestLS$ = LIMES.runUnsupervisedWombat($sourceTrainingCache, targetTrainingCache$);
  $M' = LIMES(sourceCache, targetCache, BestLS)$;
return M'

Additive Fusion Operator. Based on the mapping M_{merge} that contains the linked resources from G_s and G_t, we implement our additive fusion operator to combine all resources from G_s and G_t. In particular, our additive fusion operator starts by adding all triples from the source KG G_s to the fused KG $G_{s\oplus t}$. Thereafter, it combines all similar triples from the target KG G_t (i.e., triples which subjects contained in M_{merge}) with the similar triples from G_s. Note that, all the subjects of the fused triples are from G_s. We present our additive fusion operator formally in Algorithm 2. Figure 3 shows an example of fusing one `DBpedia` resource with one `LinkedGeoData` resource using our additive fusion operator. Note that, our operator is additive in the sense that is keep all triples of source KG G_s, even the ones with no similar triples in G_t (See Figure 4 for an example).

Algorithm 2: KGs additive fusion algorithm.

input : Source KG G_s,
Target KG G_t,
Mapping $M_{merg} = \{(x,y)|x \in G_s, y \in G_t\}$
output: Fused KG $G_{s\oplus t}$

/* Add every triple in G_s to $G_{s\oplus t}$ */
foreach $triple(\langle s,p,o\rangle) \in G_s$ **do**
  $G_{s\oplus t} = G_{s\oplus t}.addTriple(\langle s,p,o\rangle)$
/* Fuse only similar triples in G_t into $G_{s\oplus t}$ */
foreach *Mapping pair* $(x,y) \in M_{merg}$ **do**
  foreach $triple(\langle y,p,o\rangle) \in G_t$ **do**
    $G_{s\oplus t} = G_{s\oplus t}.addTriple(\langle x,p,o\rangle)$
return $G_{s\oplus t}$

Fusion Strategies. After applying our additive fusion operator, we define a number of type-based strategies for fusing the literal objects of the same subject and predicate. For example, in our example in Figure 3, we have the triple from `DBpedia <dbr:Mount_Juliet, rdfs:label, "Mount Juliet,Tennessee"@en>` and the triple from `LinkedGeoData <lgdt:node153471134, rdfs:label, "Mount Juliet">`. For the two lateral objects `"Mount Juliet,Tennessee"@en` and `"Mount Juliet"`, we need to decide to keep either one of them, both of them or to combine them somehow. Formally, for any two triples $\langle s,p,\lambda_1 >$ and $\langle s,p,\lambda_2\rangle$, we implement the following type-based fusion strategies:

- **KeepBoth strategy:** We add the two triples $\langle s,p,\lambda_1\rangle$ and $\langle s,p,\lambda_2\rangle$ to the fused KG.
- **PreferSource strategy:** We add only the triples $\langle s,p,\lambda_1\rangle$ from the source KG to the fused KG.
- **PreferTarget strategy:** We add only the triples $\langle s,p,\lambda_2\rangle$ from the target KG to the fused KG.
- **Maximum strategy:** We define the `Maximum` strategy for all numeric object literals such as `xsd:integer`[19] and `xsd:decimal` as well as dates (e.g., `xsd:date`), where we add the triple $\langle s,p,\max(\lambda_1,\lambda_2)\rangle$ to the result fused KG. For object string literals, the `Maximum` strategy selects the longest string. Formally, add the triple $\langle s,p,\arg\max(|\lambda_1|,|\lambda_2|)\rangle$ to the result fused KG, where $|\lambda_1|$ is the string length of the string λ_1. For literal

[19] `xsd=<http://www.w3.org/2001/XMLSchema#>`

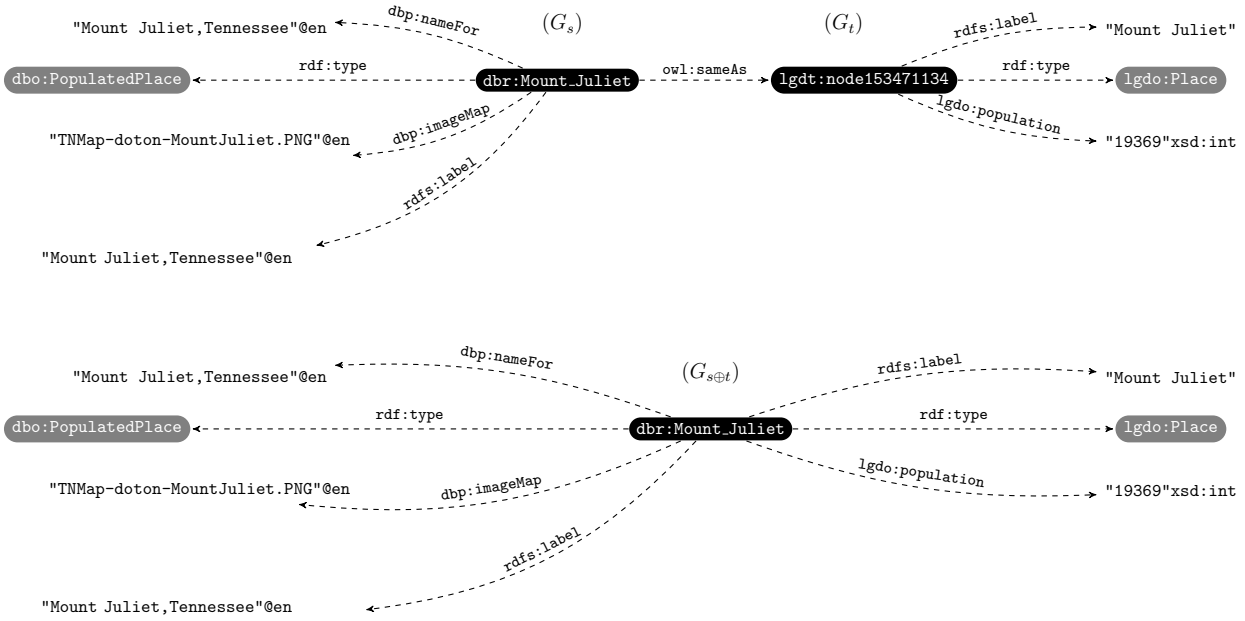

FIGURE 3: Example of fusing the DBpedia resource dbr:Mount-Juliet with the LinkedGeoData resource lgdt:node153471134 using our additive fusion operator from Algorithm 2.

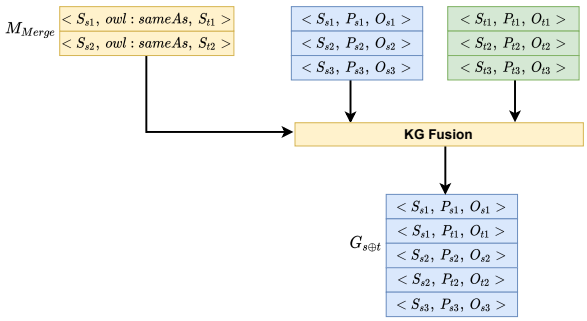

FIGURE 4: An example of fusing knowledge graphs using our additive fusion operator $\oplus$.

objects of type xsd:boolean, the triple $\langle s, p, \lambda_1 || \lambda_2 \rangle$ is added the the fused KG, where $||$ is the logical OR operator.

- **MINIMUM STRATEGY:** Flowing the same manner of the Maximum strategy, we define the Minimum strategy for numeric and date literals literal objects, where we add the triple $\langle s, p, \min(\lambda_1, \lambda_2) \rangle$ to the result fused KG. For object string literals, the Minimum strategy selects the shortest string. i.e., add the triple $\langle s, p, \arg\min(|\lambda_1|, |\lambda_2|) \rangle$ to the result fused KG. For xsd:boolean object literals, the triple $\langle s, p, \lambda_1 \&\& \lambda_2 \rangle$ is added the the fused KG, where $\&\&$ is the logical AND operator.
- **AVERAGE STRATEGY:** We define the Average strategy for numeric and date object literals, where we add the triple $\langle s, p, \frac{1}{2}(\lambda_1 + \lambda_2) \rangle$ to the result fused KG. For string object literals, the Average strategy is not defined. For xsd:boolean object literals, the triple $\langle s, p, \lambda_1 \&\& \lambda_2 \rangle$ is added the the fused KG.
- **UNION STRATEGY:** We define the Union strategy for the object literals of type xsd:boolean, where we add the triple $\langle s, p, \lambda_1 \&\& \lambda_2 \rangle$ to the result fused KG. For object string literals, the Union strategy is the string concatenation operator. i.e., the triple $\langle s, p, \lambda_1 + \lambda_2 \rangle$ to the result fused KG, , where $+$ is the string concatenation operator. Union strategy is not defined for numerical and date data types.

Table 2 lists all the type-based fusion strategies we have implemented so far. For our experiments, we apply the **KEEPBOTH** strategy.

D. KG EMBEDDING & LINK PREDICTION

Although there are dozens of embedding models that can be used to perform our link prediction task, we deploy the three embedding models *TuckER* [14], *ComplEx* [15] and *DistMult* [16] for embedding in our link prediction task. We use these models as they are state-of-the-art linear models for link prediction on knowledge graphs. Based on the NELLIE architecture, any other embedding model could be easily added to it.

- *TuckER.* TuckER is based on Tucker decomposition [20] that factorizes a tensor into a set of matrices and a smaller core tensor. In a three-mode case, given the original tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, Tucker decomposition produces a tensor $\mathcal{Z} \in \mathbb{R}^{P \times Q \times R}$ and three matrices

TABLE 2: Fusion strategies for fusing the two triples $\langle s, p, \lambda_1 \rangle$ and $\langle s, p, \lambda_2 \rangle$.

Method	`xsd:string`	`xsd:integer`	`xsd:date`	`xsd:boolean`
KEEPBOTH	$\langle s, p, \lambda_1 \rangle$, $\langle s, p, \lambda_2 \rangle$	$\langle s, p, \lambda_1 \rangle$, $\langle s, p, \lambda_2 \rangle$	$\langle s, p, \lambda_1 \rangle$, $\langle s, p, \lambda_2 \rangle$	$\langle s, p, \lambda_1 \rangle$, $\langle s, p, \lambda_2 \rangle$
PREFERSOURCE	$\langle s, p, \lambda_1 \rangle$	$\langle s, p, \lambda_1 \rangle$	$\langle s, p, \lambda_1 \rangle$	$\langle s, p, \lambda_1 \rangle$
PREFERTARGET	$\langle s, p, \lambda_2 \rangle$	$\langle s, p, \lambda_2 \rangle$	$\langle s, p, \lambda_2 \rangle$	$\langle s, p, \lambda_2 \rangle$
MAXIMUM	$\langle s, p, \arg\max(\lvert\lambda_1\rvert, \lvert\lambda_2\rvert) \rangle$	$\langle s, p, \max(\lambda_1, \lambda_2) \rangle$	$\langle s, p, \max(\lambda_1, \lambda_2) \rangle$	$\langle s, p, \lambda_1 \lvert\lvert \lambda_2 \rangle$
MINIMUM	$\langle s, p, \arg\min(\lvert\lambda_1\rvert, \lvert\lambda_2\rvert) \rangle$	$\langle s, p, \max(\lambda_1, \lambda_2) \rangle$	$\langle s, p, \max(\lambda_1, \lambda_2) \rangle$	$\langle s, p, \lambda_1 \&\& \lambda_2 \rangle$
AVERAGE	$\langle s, p, \frac{1}{2}(\lambda_1 + \lambda_2) \rangle$	$\langle s, p, \frac{1}{2}(\lambda_1 + \lambda_2) \rangle$	$\langle s, p, \frac{1}{2}(\lambda_1 + \lambda_2) \rangle$	$\langle s, p, \lambda_1 \&\& \lambda_2 \rangle$
UNION	$\langle s, p, \lambda_1 + \lambda_2 \rangle$	-	-	$\langle s, p, \lambda_1 \&\& \lambda_2 \rangle$

$\mathbf{A} \in \mathbb{R}^{I \times P}$, $\mathbf{B} \in \mathbb{R}^{J \times Q}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$:

$$\mathcal{X} \approx \mathcal{Z} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}, \tag{2}$$

And, The score function of TuckER model:

$$\phi(e_s, r, e_o) = \mathcal{W} \times_1 \mathbf{e}_s \times_2 \mathbf{w}_r \times_3 \mathbf{e}_o, \tag{3}$$

For link prediction on a KG's binary tensor representation, TuckER model uses Tucker decomposition by constructing entity embedding matrix $\mathbf{E}$ that is equal for subject and object entities, i.e., $\mathbf{E} = \mathbf{A} = \mathbf{C} \in \mathbb{R}^{n_e \times d_e}$ and relation embedding matrix $\mathbf{R} = \mathbf{B} \in \mathbb{R}^{n_r \times d_r}$, where n_e and n_r denote the number of entities and relations and d_e and d_r the dimensionality of entity and relation embedding vectors. TuckER architecture can be seen in Figure 5, where $\mathbf{e}_s, \mathbf{e}_o \in \mathbb{R}^{d_e}$ are the rows of $\mathbf{E}$ representing the subject and object entity embedding vectors, $\mathbf{w}_r \in \mathbb{R}^{d_r}$ the rows of $\mathbf{R}$ representing the relation embedding vector and $\mathcal{W} \in \mathbb{R}^{d_e \times d_r \times d_e}$ is the core tensor.

- *DistMult.* The scoring function of DistMultin Table 1 can be regarded as equivalent to the scoring function of TuckER in Equation 2. The scoring function consists of a core tensor $\mathcal{Z} \in \mathbb{R}^{P \times Q \times R}$, $P = Q = R = d_e$. The *superdiagonal* of $\mathcal{Z}$ is with 1s (i.e. all elements z_{pqr} with $p = q = r$ are 1 and all the other elements are 0). In *DistMult*, subject and object entity embedding vectors $\mathbf{e}_s, \mathbf{e}_o \in \mathbb{R}^{d_e}$ are represented by rows of $\mathbf{E} = \mathbf{A} = \mathbf{C} \in \mathbb{R}^{n_e \times d_e}$ and rows of $\mathbf{R} = \mathbf{B} \in \mathbb{R}^{n_r \times d_e}$ represent relation embedding vectors $\mathbf{w}_r \in \mathbb{R}^{d_e}$. Given that matrices $\mathbf{A}$ and $\mathbf{C}$ are identical, the TuckER interpretation of the DistMult scoring function can alternatively be interpreted as a special case of CP decomposition [21]. DistMult belongs to the family of bilinear models.
- *ComplEx.* The scoring function of ComplEx in 1 can also be viewed as equivalent to the scoring function of TuckER in Equation 2. The core tensor $\mathcal{Z} \in \mathbb{R}^{P \times Q \times R}$, $P = Q = R = 2d_e$ in which $3d_e$ elements on different tensor diagonals are set to 1 and d_e elements on one tensor diagonal are set to -1 while all other elements are set to 0. [22] explained that ComplEx can be regarded a bilinear model with the real and imaginary part of an embedding for each entity concatenated in a single vector.

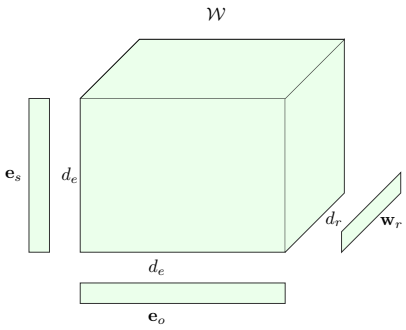

FIGURE 5: TuckER architecture [14].

IV. EVALUATION & DISCUSSION

In this section, we evaluate each of the NELLIE components, i.e., KG matching, linking, fusion, and embedding, where we performed a set of experiments to evaluate the different techniques we implemented for each component.

A. KNOWLEDGE GRAPHS MATCHING EVALUATION

Metadata-based KGs Matching. We start by retrieving the global metadata of all knowledge graphs available on the LOD[20]. In total, we get the metadata of 1118 knowledge graphs. We convert the metadata into RDF format[21]. We then use the link discovery framework LIMES for matching KGs based on their metadata. The LIMES configuration file we used to match the KGs' metadata is presented in Listing 3, also publically available from the project web site[22]. In particular, we configure LIMES to compute the *exact match* string similarity between the `keywords` and `domain` properties. With this configuration, LIMES generated 186452 links.

Content-based KGs Matching. To the best of our knowledge, there is no benchmark for the KGs matching task, therefore we had to create our own benchmark. In particular, we chose a list of 8 KGs and manually annotated them by three annotators into either *matched* (✓) or *not matched* (✗). We then computed the mutual agreement (*MA*) of our three annotators as listed in Table 3. To this end,

[20]https://lod-cloud.net/lod-data.json, accessed in October 2022

[21]https://git.cs.uni-paderborn.de/kgfusionpg/kgfusion/-/blob/main/lod_metadata_11_2022.ttl

[22]https://git.cs.uni-paderborn.de/kgfusionpg/kgfusion/-/blob/main/LIMESConfig.xml

TABLE 3: Manually annotation results of our KGs matching. A_1 to A_3 are our 3 annotators, MA is the mutual agreement.

KG_s	KG_t	A_1	A_2	A_3	MA
harvard_eagle-i_net_sparqler.nt	onto_fel_cvut_cz_rdf4j-server_repositories.nt	✗	✗	✗	✗
harvard_eagle-i_net_sparqler.nt	ldf_fi_ww1lod.nt	✓	✗	✓	✓
harvard_eagle-i_net_sparqler.nt	lov_linkeddata_es_dataset_lov.nt	✓	✓	✓	✓
harvard_eagle-i_net_sparqler.nt	dbtune_org_bbc_peel_sparql.nt	✓	✓	✓	✓
harvard_eagle-i_net_sparqler.nt	www_imagesnippets_com_sparql.nt	✓	✓	✓	✓
harvard_eagle-i_net_sparqler.nt	dbtune_org_magnatune_sparql.nt	✓	✗	✓	✓
harvard_eagle-i_net_sparqler.nt	data_nobelprize_org.nt	✓	✓	✓	✓
onto_fel_cvut_cz_rdf4j-server_repositories.nt	ldf_fi_ww1lod.nt	✗	✗	✗	✗
onto_fel_cvut_cz_rdf4j-server_repositories.nt	lov_linkeddata_es_dataset_lov.nt	✓	✓	✓	✓
onto_fel_cvut_cz_rdf4j-server_repositories.nt	dbtune_org_bbc_peel_sparql.nt	✗	✓	✗	✗
onto_fel_cvut_cz_rdf4j-server_repositories.nt	www_imagesnippets_com_sparql.nt	✗	✓	✗	✗
onto_fel_cvut_cz_rdf4j-server_repositories.nt	dbtune_org_magnatune_sparql.nt	✗	✓	✗	✗
onto_fel_cvut_cz_rdf4j-server_repositories.nt	data_nobelprize_org.nt	✗	✗	✗	✗
ldf_fi_ww1lod.nt	lov_linkeddata_es_dataset_lov.nt	✓	✗	✓	✓
ldf_fi_ww1lod.nt	dbtune_org_bbc_peel_sparql.nt	✓	✓	✓	✓
ldf_fi_ww1lod.nt	www_imagesnippets_com_sparql.nt	✓	✓	✗	✓
ldf_fi_ww1lod.nt	dbtune_org_magnatune_sparql.nt	✓	✓	✓	✓
ldf_fi_ww1lod.nt	data_nobelprize_org.nt	✓	✓	✓	✓
lov_linkeddata_es_dataset_lov.nt	dbtune_org_bbc_peel_sparql.nt	✓	✓	✓	✓
lov_linkeddata_es_dataset_lov.nt	www_imagesnippets_com_sparql.nt	✓	✓	✓	✓
lov_linkeddata_es_dataset_lov.nt	dbtune_org_magnatune_sparql.nt	✓	✓	✓	✓
lov_linkeddata_es_dataset_lov.nt	data_nobelprize_org.nt	✓	✓	✓	✓
dbtune_org_bbc_peel_sparql.nt	www_imagesnippets_com_sparql.nt	✓	✓	✗	✓
dbtune_org_bbc_peel_sparql.nt	dbtune_org_magnatune_sparql.nt	✓	✓	✓	✓
dbtune_org_bbc_peel_sparql.nt	data_nobelprize_org.nt	✓	✓	✓	✓
www_imagesnippets_com_sparql.nt	dbtune_org_magnatune_sparql.nt	✓	✓	✓	✓
www_imagesnippets_com_sparql.nt	data_nobelprize_org.nt	✓	✓	✓	✓
dbtune_org_magnatune_sparql.nt	data_nobelprize_org.nt	✓	✓	✓	✓

we applied our content-based KGs matching as described in Section III-A. We set the threshold of similarity between the generated documents of KGs to be ≥ 0.1. Using the mutual agreement among our annotators (*MA*) as the ground truth, we compute precision, recall and F-Measure among the KG content documents using the `Cosine`, `Jaccard`, `Weighted Jaccard`, `Dice` and `BERT` similarity measures. The results are listed in Table 4. In particular, we achieve an F-Measure of 1.0 using `Jaccard` similarity and 0.95 using `Cosine-TF-IDF` similarity. On the other hand, using the `BERT` similarity resulted in an F-Measure of only 0.88. However, the *document similarity scores* resulted using the `BERT` similarity is in general higher than the other similarity measures such as `Jaccard`. The reason is that `BERT` is an advanced language model that takes into account semantic, contextual and relation between words in its word representation vectors. For computing `Dice` similarity, we used the open-source Java library `SimMetrics`[23]. We use the pre-trained BERT from `HuggingFace`[24] for the embedding of the preprocessed documents, where we calculate the similarity of vectors using the cosine similarity.

Manual KGs matching. For *Manual KGs matching*, we select the following KGs from the biological domain: `Kegg`, `Drugbank`, `Sider`, `Omim` and `Sgd`. We manually match these KGs to ensure the best matching of them, as we used them to evaluate all the next components of NELLIE (i.e., ontology matching, instance matching, fusion and link predication). Table 5 provides the characteristics of these KGs.

[23]https://github.com/Simmetrics/simmetrics
[24]https://huggingface.co/sentence-transformers/all-mpnet-base-v2

TABLE 4: Content-based KGs matching.

Similarity Method	`Cosine TF-IDF`	`Jaccard`	`Weighted Jaccard`	`DICE`	`BERT`
Precision	0.95	1.0	1.0	1.0	0.79
Recall	0.95	1.0	0.68	0.86	1.0
F-Measure	0.95	1.0	0.81	0.93	0.88

TABLE 5: Evaluation KGs characteristics.

#	`Kegg`	`Omim`	`Sider`	`Drugbank`	`Sgd`
Classes	71	34	12	98	85
Entities	8.627 M	1.127 M	0.479 M	0.421 M	0.991 M
Triples	67.89 M	9.68 M	5.57 M	5.5 M	12.95 M

B. LINKING EVALUATION

Setup. For the linking task, we set the configuration of NELLIE as follows: We set the *propriety-coverage* threshold β to 0.5 and the *minimum threshold for string similarities* when computing equivalent classes τ to 0.2. We also configured the parameter of Wombat as follows: We set the string similarity measures to `{jaccard, cosine, qgrams, levenshtein}`, the maximum iteration number to 10, the maximum execution time to 200 minutes and minimum the properties coverage to 0.9.

Content-based Class Matching. In Table 6, we present the results of applying the content-based class matching to the classes within the manually matched KGs from our

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE LIMES SYSTEM "limes.dtd">
<LIMES>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/2000/01/rdf-schema#</
     NAMESPACE>
    <LABEL>rdfs</LABEL>
  </PREFIX>
  <PREFIX>
    <NAMESPACE>http://www.w3.org/2002/07/owl#</NAMESPACE>
    <LABEL>owl</LABEL>
  </PREFIX>

  <SOURCE>
    <ID>S</ID
    <ENDPOINT>lod_metadata_11_2022.ttl</ENDPOINT>
    <VAR>?x</VAR>
    <PAGESIZE>-1</PAGESIZE>
    <RESTRICTION> </RESTRICTION>
    <PROPERTY>rdfs:keywords</PROPERTY>
    <PROPERTY>rdfs:domain</PROPERTY>
    <TYPE>NT</TYPE>
  </SOURCE>

  <TARGET>
    <ID>T</ID>
    <ENDPOINT>lod_metadata_11_2022.ttl</ENDPOINT>
    <VAR>?y</VAR>
    <PAGESIZE>-1</PAGESIZE>
    <RESTRICTION> </RESTRICTION>
    <PROPERTY>rdfs:keywords</PROPERTY>
    <PROPERTY>rdfs:domain</PROPERTY>
    <TYPE>NT</TYPE>
  </TARGET>
  <METRIC>AND(exactmatch(x.rdfs:keywords,y.rdfs:keywords)
     |0.9, exactmatch(x.rdfs:domain,y.rdfs:domain)|0.9)
  </METRIC>

  <ACCEPTANCE>
    <THRESHOLD>0.98</THRESHOLD>
    <FILE>accepted.ttl</FILE>
    <RELATION>owl:sameAs</RELATION>
  </ACCEPTANCE>

  <REVIEW>
    <THRESHOLD>0.80</THRESHOLD>
    <FILE>review.ttl</FILE>
    <RELATION>owl:sameAs</RELATION>
  </REVIEW>

  <EXECUTION>
    <REWRITER>default</REWRITER>
    <PLANNER>default</PLANNER>
    <ENGINE>default</ENGINE>
  </EXECUTION>

  <OUTPUT>TTL</OUTPUT>
</LIMES>
```

Listing 3: LIMES configuration for metadata-based KGs matching.

previous step. In particular, we computed the `Jaccard` string similarity between the cleaned laterals of class pairs within each pair of KGs as described in Section III-B1. Accordingly, a pair of classes is matched if the similarity between their respective cleaned laterals ≥ 0.2. In the next step, we run LIMES on each pair of matched classes to link the instances with these classes following the procedures defined in Algorithm 1. In Table 6, we list the pairs of matched classes for each of the manually matched KGs (from Section 5). Algorithm 1 then runs LIMES on each pair of the matched classes. We also calculate the Pseudo-F-Measure $\mathcal{F}$ [23] for the instances with each paired of classes. The basic assumption behind the pseudo-F-Measure is that symmetrical one-to-one links exist between the resources in source and target datasets. For example, $\mathcal{F} = 0.89$ for the matched classes (`Settlement, Place`) of *DBpedia* and *LGD*. We used unsupervised version of the WOMBAT algorithm [12] to calculate $\mathcal{F}$. We also computed average macro Pseudo-F-Measure in case there is more than one pair of matched classes such as (`DBpedia, LGD`), (`Kegg, Omim`), (`Kegg, Drugbank`) and (`Kegg, Sgd`). The average macro Pseudo-F-Measure is also listed in Table 6 with bold font. We also report the time required to link the instances with in each pair of matched classes.

LogMap & FCA Based Class Matching. We embed the ontology matching components of *LogMap* and *FCA* into the NELLIE ontology matching phase as external libraries. For the evaluation of each of the two systems, please refer to the original papers of the systems [18], [24].

Performance gain using our two-phase linking. By applying the ontology matching phase prior to the instance matching phase, we aim to reduce the overall runtime of the linking procedure. In particular, when to apply a single phase linking of all-against-all instances directly, we would need $\left(\sum_{i=1}^{n}|C_i|\right)\left(\sum_{j=1}^{n}|D_j|\right)$ comparisons between the instances of the leaf classes C_i of the source KG G_s and the leaf classes D_j of the target KG G_t. Note that, we assume that both G_s and G_t have the same number of classes n without loss of generality. W.l.o.g, we will also assume that the classes are ordered in such as manner that the first $k \leq n$ classes match. On the other side, using our two-phases linking, we need n^2 comparisons for the first linking phase (i.e., class matching). Note that, in general, the average class in a knowledge graph has a magnitude larger than the total number of classes. Hence, $n << \sum_{i=1}^{n}\left(\frac{|C_i|}{n}\right)$. The analog holds for G_t. For the second linking phase (i.e., instance matching), we need $\sum_{i=1}^{k}|C_i||D_i|$ for the k pairs of the matched leaf classes from G_s and G_t. This gives us a total cost of $\left(\sum_{i=1}^{k}|C_i||D_i| + n^2\right)$ comparisons of our 2-phase linking. Our gain is then the difference between the number of comparisons of all-against-all instance linking and our 2-phase linking. Given that $n << \sum_{i=1}^{n}\left(\frac{|C_i|}{n}\right)$, the expected value of this difference is positive for $k < n$.

Empirically, we can compute the speedup achieved by our two-phase linking as the number of comparisons using all-against-all instance linking divided by the number of comparisons of our two-phase linking. For example, if we carry out the all-against-all for the KGs `Drugbank` and `Sgd`, we get $0.421 \times 10^6 \times 0.991 \times 10^6 = 41.211 \times 10^{10}$ comparisons. On the other hand, the number of comparisons using our two-phases linking needs only $1819 \times 3488 + n^2 = 6.3446 \times 10^6 + 8330$ given that $k = 1$, where $|C_1| = 1819$ and $|D_1| = 3488$, and n^2 is 8330. Accordingly, our the speedup

TABLE 6: Class matching results. $|c|$ is the number of instances of a class c. Time is in milliseconds. $\mathcal{F}$ is Pseudo-F-Measure.

G_s	G_t	**Source class** (c_s)	$\lvert c_s\rvert$	**Target class** (c_t)	$\lvert c_t\rvert$	**Time**	$\mathcal{F}$
Kegg	Omim	uniprot_vocabulary:Resource	25009	uniprot_vocabulary:Resource	14259	180686	0.82
		ncbigene_vocabulary:Resource	39726	ncbigene_vocabulary:Resource	16080	227034	0.68
		ensembl_vocabulary:Resource	22171	ensembl_vocabulary:Resource	29006	586008	0.67
						Average macro Pseudo-F-Measure	**0.72**
Kegg	Drugbank	cas_vocabulary:Resource	21271	cas_vocabulary:Resource	3167	17261	0.35
		atc_vocabulary:Resource	3775	atc_vocabulary:Resource	1739	5	0.77
						Average macro Pseudo-F-Measure	**0.56**
Kegg	Sgd	go_vocabulary:Resource	3665	go_vocabulary:Resource	5555	6094	0.38
		ec_vocabulary:Resource	6172	ec_vocabulary:Resource	444	570	0.27
						Average macro Pseudo-F-Measure	**0.32**
Drugbank	Sider	pubchem.compound_vocabulary:Resource	6111	pubchem.compound_vocabulary:Resource	2097	8376	0.31
Drugbank	Omim	uniprot_vocabulary:Resource	8392	uniprot_vocabulary:Resource	14259	62370	0.34
Drugbank	Sgd	pfam_vocabulary:Resource	1819	pfam_vocabulary:Resource	3488	2291	0.46
DBpedia	LGD	Settlement	11705	Place	11485	22516	0.89
		Settlement	11705	Village	8443	15756	0.83
		PopulatedPlace	11318	Place	11485	16709	0.89
		PopulatedPlace	11318	Village	8443	11966	0.83
		Place	11367	Place	11485	17717	0.89
		Place	11367	Village	8443	15174	0.83
						Average macro Pseudo-F-Measure	**0.86**

here is

$$\frac{41.211 \times 10^{10}}{6.3446 \times 10^{6} + 8330} = 6.48695 \times 10^{4}$$

which is 4 orders of magnitude.

C. THE FUSION OF KNOWLEDGE GRAPHS & LINK PREDICTION TASK

In the third task of our evaluation, we study the impact of fusion on the link prediction task. Since the cost of computing and resources allocation of the link prediction task on KGs that contain millions of triples is very high, we made two data augmentation scenarios for conducting experiments on data fusion and link prediction tasks:

- *Scenario A:* The idea of this scenario is to study the impact of fused KG on the quality of the link prediction task. Thus, we used augmented versions of the source KG G_s and target KG G_t by only filtering them to the entities within the mapping: $M = \{(s,t)|s \in G_s, t \in G_t\}$. Formally, we augmented our data as follows: Given the mapping M, we retrieve the sub-KG $G'_s = \{\langle s,r,o\rangle \subseteq G_s \forall s \in M\}$ and the sub-KG $G'_t = \{\langle t,r,o\rangle \subseteq G_t \forall t \in M\}$. To this end, we run Algorithm 2 to fuse the triples of G'_s and G'_t into the fused KG $G_{s\oplus t}$. For evaluating the link prediction task, we compute Hit@1, Hit@3, Hit@10 and MRR for the source KGs G'_s and fused KGs $G_{s\oplus t}$ The results are in Tables 8, 9, 10, 11, 12, and 13
- *Scenario B:* The idea of this scenario is similar to the one of *Scenario A*. However, we modify *Scenario A* by adding a random subset X of resources from G_s, which are not included within the mapping M, to G'_s. The goal here is to perform the link prediction task on KGs that contains some enriched entities and some non-enriched ones. Formally, $X = \{\langle s,r,o\rangle \subseteq G_s \quad \forall s \in G_s \wedge s \notin M\}$ and $|X| = |M|$. Note that, we limit the size of X to the the size of M to keep the balance between the enriched and non-enriched resources. This results in $G'_s = \{\langle s,r,o\rangle \subseteq G_s \quad \forall s \in M \cup X\}$. To this end, we repeat the procedures done in *Scenario A*. See the results in Tables 14, 15, 16, 17, 18, and 19

TABLE 7: Hyper-parameter values for TuckER, DistMult, and ComplEx across all datasets, where lr denotes learning rate, dr decay rate, ls label smoothing.

Model	lr	dr	d_e	d_r	ls
TuckER	0.005	1.0	200	200	0.1
DistMult	0.001	0.99	200	200	0.1
ComplEx	0.001	0.99	200	200	0.1

Setup. We set the hyper-parameters as listed in the Table 7. We followed the same setting, training and evaluation procedures introduced in the TuckER model [14] and the codebase[25] for TuckER, ComplEx and DistMult.

Results. We calculate the improvement percentage of each source KG and its fused KG using the formula: $(Hit@k_{fusedKG} - Hit@k_{sourceKG}) * 100$. For MRR, we apply a similar formula. For example, The improvement of fused KG between KEGG and Omim in Table 12 for $Hit@1$ is $(0.5008 - 0.3808) * 100 = 12.0\%$ using DistMulti model. By repeating this procedure, we calculate all values for all KGs. In particular, we found that the fused KG shows

[25]https://github.com/ibalazevic/TuckER

TABLE 8: Link Prediction for Drugbank, Omim and the fused data in *Scenario A*

Model	Data	Hit@1	Hit@3	Hit@10	MRR	$\|Data\|$
TuckER	Source	0.6580	0.6581	0.7223	0.6741	18.545
	Fused	0.5735	0.6209	0.6506	0.6034	20.891
	Improve	-8.45	-3.73	-7.17	-7.07	
DistMulti	Source	0.5458	0.6378	0.6858	0.5997	18.545
	Fused	0.5919	0.6279	0.6750	0.6197	20.891
	Improve	4.61	-0.98	-1.08	2.00	
ComplEx	Source	0.6887	0.7486	0.7704	0.7236	18.545
	Fused	0.5768	0.6468	0.6800	0.6179	20.891
	Improve	-11.19	-10.17	-9.04	-10.58	

TABLE 9: Link Prediction for Kegg, Drugbank and the fused data in *Scenario A*

Model	Data	Hit@1	Hit@3	Hit@10	MRR	$\|Data\|$
TuckER	Source	0.5326	0.5540	0.5999	0.5523	30.351
	Fused	0.4351	0.5207	0.5625	0.4830	34.731
	Improve	-9.75	-3.34	-3.74	-6.92	
DistMulti	Source	0.4172	0.4596	0.4936	0.4459	30.351
	Fused	0.4160	0.4727	0.5043	0.4517	34.731
	Improve	-0.12	1.31	1.07	0.58	
ComplEx	Source	0.4713	0.6075	0.6336	0.5445	30.351
	Fused	0.4057	0.4479	0.4646	0.4311	34.731
	Improve	-6.56	-15.96	-16.90	-11.33	

TABLE 10: Link Prediction for Drugbank, Sider and the fused data in *Scenario A*

Model	Data	Hit@1	Hit@3	Hit@10	MRR	$\|Data\|$
TuckER	Source	0.6460	0.6735	0.723	0.6668	11.511
	Fused	0.6426	0.7669	0.8132	0.7076	25.491
	Improve	-0.34%	9.33%	8.99%	4.08%	
DistMulti	Source	0.6086	0.668766	0.7040	0.6446	11.511
	Fused	0.6215	0.7297	0.7964	0.6839	25.491
	Improve	1.29%	6.09%	9.24%	3.93%	
ComplEx	Source	0.7040	0.7995	0.8249	0.7569	11.511
	Fused	0.6003	0.7066	0.7727	0.6613	25.491
	Improve	-10.37%	-9.29%	-5.23%	-9.56%	

TABLE 11: Link Prediction for Kegg, Sgd and the fused data in *Scenario A*

Model	Data	Hit@1	Hit@3	Hit@10	MRR	$\|Data\|$
TuckER	Source	0.551	0.5982	0.632	0.5809	12.641
	Fused	0.5628	0.6357	0.6761	0.604	21.215
	Improve	1.18	3.75	4.41	2.31	
DistMulti	Source	0.4527	0.5249	0.5649	0.4962	12.641
	Fused	0.5132	0.5861	0.6264	0.5568	21.215
	Improve	6.05	6.12	6.16	6.07	
ComplEx	Source	0.5279	0.6665	0.6928	0.6026	12.641
	Fused	0.5244	0.6	0.6409	0.5679	21.215
	Improve	-0.35	-6.65	-5.19	-3.47	

improvement of $Hit@1$ compared to the $Hit@1$ of source KG. To the best of our knowledge, there is no scientific evidence about the impact of KG alignment or KG fusion on link prediction task. However, the results show improvement in all metrics $Hit@1$, $Hit@3$, $Hit@10$, and (MRR). For example, in *Scenario A* KG fusion improved 10 $Hit@1$ out of 18 $Hit@1$ which is 55.55% of cases.While in *Scenario B* KG fusion improved 17 $Hit@1$ out of 18 $Hit@1$ which is 94.44% of cases.

The results of *Scenario A* for the KG KEGG and KG Omim (Table 12) shows that KG fusion improves $Hit@1$, $Hit@3$, $Hit@10$ and (MRR) by up to 12%, 8.31%, 7.66% and 10.10% respectively. Hence, we could see that KG fusion plays an important role in improving the link prediction task.

In *Scenario B*, using TuckER embedding for the KG Drugbank and Omim (Table 16), the results show the improvement in $Hit@1$, $Hit@3$, $Hit@10$, and (MRR) by up to 1.7%, 1.49%, 2.22% and 1.79%, respectively. Another example is the KGs Kegg and Sgd (Scenario B) using TuckER, the improvement is up to 9.34%, 8.94%, 7.05%, and 8.77% for $Hit@1$, $Hit@3$, $Hit@10$, and (MRR), respectively (see, Table 15).

From the results, we also can see in some cases that KG fusion reduces the performance of link prediction task. For instance, KG Drugbank and KG Omim in *Scenario A* using TuckER. See Table 8. However, because of the absence of a benchmark KGs for such task (i.e. the impact of KGs fusion on the quality of the link prediction task), it is difficult to say that this improvement is caused by the models used or the KGs. Fundamentally, the performance of TuckER, DistMulti, and ComplEx in litterateur is evaluated on a benchmark KGs (see, [14]). These benchmark KGs are tailored to evaluate KGE. DistMulti is limited to symmetric relations, while ComplEx is able to capture antisymmetric relations. Thus, if a KG contains many antisymmetric relations, DistMulti may perform poorly. ComplEx can handle antisymmetric relations, but its parameter number grows quadratically with the number of relations, which frequently leads to overfitting, especially for connections with a limited number of training triples. TuckER gets around this problem by modeling relations as vectors w_r, so the number of parameters scales linearly with the number of relations. Another reason that can affect the results and the performance is the selection of hyper-parameters. In our current experiments, we did not conduct any sort of hyper-parameters optimizations. We used hyper-parameters from [14]. Based on these observations, the impact of KGs fusion on the quality of the link prediction task is still an open question and it needs thorough investigation from benchmarking KGs to the hyper-parameters optimization.

V. RELATED WORK

To the best of our knowledge, there are no previous attempts to carry out the 24/7 never ending linking over RDF KG

TABLE 12: Link Prediction for Kegg, Omim and the fused data in *Scenario A*

Model	Data	Hit@1	Hit@3	Hit@10	MRR	$\|Data\|$
TuckER	Source	0.5020	0.5350	0.5541	0.5212	307.375
	Fused	0.5023	0.5605	0.5856	0.5350	496.462
	Improve	0.03	2.55	3.14	1.38	
DistMulti	Source	0.3808	0.4773	0.5090	0.4322	307.375
	Fused	0.5008	0.5604	0.5856	0.5332	496.462
	Improve	12.00	8.31	7.66	10.10	
ComplEx	Source	0.4421	0.5481	0.5703	0.4976	307.375
	Fused	0.4959	0.5668	0.5855	0.5323	496.462
	Improve	5.38	1.87	1.52	3.48	

TABLE 13: Link Prediction for DBpedia, `LinkedGeoData` and the fused data in *Scenario A*

Model	Data	Hit@1	Hit@3	Hit@10	MRR	$\|Data\|$
TuckER	Source	0.4504	0.5326	0.5997	0.5033	271.706
	Fused	0.4802	0.5598	0.6088	0.5278	326.792
	Improve	2.98	2.72	0.91	2.45	
DistMulti	Source	0.3601	0.4548	0.5249	0.4187	271.706
	Fused	0.3772	0.4515	0.5224	0.4277	326.792
	Improve	1.71	-0.33	-0.25	0.89	
ComplEx	Source	0.3350	0.4501	0.5127	0.4022	271.706
	Fused	0.3725	0.4507	0.5361	0.4275	326.792
	Improve	3.76	0.06	2.34	2.52	

TABLE 14: Link Prediction for Drugbank, Sider and the fused data in *Scenario B*

Model	Data	Hit@1	Hit@3	Hit@10	MRR	$\|Data\|$
TuckER	Source	0.4210	0.4567	0.5047	0.4485	29.02
	Fused	0.4571	0.5248	0.5940	0.5037	36.222
	Improve	3.60	6.81	8.94	5.52	
DistMulti	Source	0.3332	0.4233	0.4854	0.3885	29.02
	Fused	0.6943	0.8016	0.8303	0.7485	36.222
	Improve	36.11	37.82	34.49	36.01	
ComplEx	Source	0.4061	0.4842	0.5380	0.4551	29.02
	Fused	0.4189	0.53	0.5941	0.4846	36.222
	Improve	1.28	4.58	5.61	2.95	

TABLE 15: Link Prediction for Kegg, Sgd and the fused data in *Scenario B*

Model	Data	Hit@1	Hit@3	Hit@10	MRR	$\|Data\|$
TuckER	Source	0.4445	0.4840	0.5366	0.4747	23.712
	Fused	0.5379	0.5733	0.6071	0.5624	31.337
	Improve	9.34	8.94	7.05	8.77	
DistMulti	Source	0.3123	0.3792	0.4283	0.3549	23.712
	Fused	0.3998	0.4736	0.5373	0.4482	31.337
	Improve	8.75	9.44	10.90	9.33	
ComplEx	Source	0.3687	0.5128	0.5614	0.4489	23.712
	Fused	0.4580	0.5598	0.5874	0.5150	31.337
	Improve	8.93	4.70	2.60	6.61	

to enable building a holistic model for the LOD. Accordingly the related work comes from four different research area: knowledge graph matching, ontology/instance matching, data fusion and Knowledge graph embedding, therefore we brief some related works for each research area.

A. KNOWLEDGE GRAPHS MATCHING

The use of topic-modeling-based document similarities is well recognized and has been extensively researched in earlier papers, such as [25]. Topic modelling has been utilized for texts with natural language, particularly for applications involving information retrieval. To return documents that are thematically linked to a particular query, Buntime et al. [26] created an information retrieval system based on a hierarchical topic modelling method. In NELLIE, we adopted KG matching techniques based on document similarity, where we generated one document per KG. Sleeman et al. [27] developed a method for using topic modeling with RDF data. This method produces a separate document for each entity described in a KG, whereas our method generates documents that describe an entire KG. Tapioca [13] is based on Latent dirichlet allocation (LDA) [28] to identify the topics of RDF KGs.

B. ONTOLOGY AND INSTANCE MATCHING

Ontology Matching. Knowledge integration depends heavily on ontology alignment, which has been the subject of extensive research in recent years. We list here some state-of-the-art systems. LogMap [2] is a highly scalable ontology matching system that provides reasoning and diagnosis capabilities. Codi [3] is another ontology matching framework relying on Markov logic. The system defines the syntax and semantics and formalizes the ontology matching problem. Chen et al. [24] introduced a machine learning algorithm that makes advantage of distant supervision and semantic embedding to improve the conventional ontology alignment systems. To put it simply, it first generates high precision seed mappings using the original ontology alignment system and class disjointness constraints (as heuristic rules), it then uses these mappings to train a Siamese Neural Network (SiamNN) for predicting cross-ontology class mappings via semantic embeddings in OWL2Vec [29]. VeeAlign [30], a deep learning-based model, employs a new dual-attention technique to compute the contextualized representation of a notion, which is then utilized to find alignments. By doing this, VeeAlign is able to take advantage of the syntactic and semantic data embedded in the ontologies. OntoConnect [31] is a recent domain-independent, non-human intervention ontology alignment approach that uses graph embedding with negative sampling.

Instance Matching. Declarative link discovery frameworks build complex link specifications to specify the conditions necessary for linking resources between RDF knowledge graphs. The SILK [7] and LIMES [5], [6] frameworks, for

TABLE 16: Link Prediction for Drugbank, Omim and the fused data in *Scenario B*

Model	Data	Hit@1	Hit@3	Hit@10	MRR	$\|Data\|$
TuckER	Source	0.4158	0.4537	0.4936	0.4428	46.981
	Fused	0.4328	0.4686	0.5157	0.4607	49.308
	Improve	1.70	1.49	2.22	1.79	
DistMulti	Source	0.3169	0.3906	0.4627	0.3671	46.981
	Fused	0.3297	0.3930	0.4585	0.3737	49.308
	Improve	1.28	0.24	-0.42	0.66	
ComplEx	Source	0.3921	0.4644	0.5054	0.4363	46.981
	Fused	0.4038	0.4765	0.5188	0.4476	49.308
	Improve	1.17	1.21	1.34	1.14	

TABLE 17: Link Prediction for Kegg, Omim and the fused data in *Scenario B*

Model	Data	Hit@1	Hit@3	Hit@10	MRR	$\|Data\|$
TuckER	Source	0.4085	0.4397	0.4642	0.4285	591.5
	Fused	0.4410	0.4856	0.5133	0.4674	690.224
	Improve	3.26	4.59	4.92	3.89	
. DistMulti	Source	0.2883	0.3284	0.3483	0.3116	591.5
	Fused	0.3480	0.4091	0.4452	0.3839	690.224
	Improve	5.97	8.07	9.68	7.23	
ComplEx	Source	0.3760	0.4555	0.4766	0.4192	591.5
	Fused	0.4243	0.4950	0.5261	0.4651	690.224
	Improve	4.83	3.95	4.95	4.59	

TABLE 18: Link Prediction for Kegg, Drugbank and the fused data in *Scenario B*

Model	Data	Hit@1	Hit@3	Hit@10	MRR	$\|Data\|$
TuckER	Source	0.4451	0.4718	0.4921	0.4636	56.970
	Fused	0.4530	0.4795	0.5046	0.4736	61.332
	Improve	0.79	0.76	1.26	1.00	
DistMulti	Source	0.3146	0.3749	0.4150	0.3526	56.970
	Fused	0.5325	0.5540	0.6122	0.5548	61.332
	Improve	21.79	17.90	19.72	20.22	
ComplEx	Source	0.3591	0.4869	0.5105	0.4245	56.970
	Fused	0.3743	0.4878	0.5115	0.4344	61.332
	Improve	1.52	0.09	0.10	1.00	

TABLE 19: Link Prediction for DBpedia, `LinkedGeoData` and the fused data in *Scenario B*

Model	Data	Hit@1	Hit@3	Hit@10	MRR	$\|Data\|$
TuckER	Source	0.3662	0.4313	0.4906	0.4107	511.294
	Fused	0.3728	0.4348	0.4969	0.4161	566.380
	Improve	0.66	0.35	0.63	0.54	
DistMulti	Source	0.2684	0.3645	0.4275	0.3273	511.294
	Fused	0.2637	0.3406	0.4113	0.3152	566.380
	Improve	-0.48	-2.39	-1.62	-1.22	
ComplEx	Source	0.2598	0.3149	0.3924	0.3034	511.294
	Fused	0.2813	0.3349	0.3966	0.3217	566.380
	Improve	2.16	2.00	0.42	1.83	

instance, employ a property-based methods for the computation of links between instances. Such link specifications can be generated using various machine learning approaches such as Wombat [12] and Eagle [32]. Serimi [33] is an automatic interlinking method and matches instances between a source and a target knowledge bases. Niu et al. [34] introduce a a semi-supervised learning algorithm for automatically discovering dataset-specific instance matching rules. Slint [35] uses an approach for schema-independent interlinking. In particular, Slint starts by automatically selecting important RDF predicates using the coverage and discriminability, then it uses the weighted co-occurrence and adaptive filtering for carrying out the instance matching.

C. DATA FUSION

Data fusion is one of the key goals of data integration. Data fusion increases the conciseness through fusing duplicate entries and merging common attributes together. The work [36] defines the goals of data fusion as to achieve more data completeness and conciseness. The main challenges of data fusion are uncertainty due to conflicting data values. In [36], the authors survey different ways of fusing data and present several methods. In the systematic survey [37], the authors introduce the challenges of the knowledge graph fusion, where they discuss advanced techniques for handling knowledge graph fusion. The linked data quality assessment and fusion framework Sieve [9] is based on the linked data integration framework (LDIF) [10]. LDIF is an open-source framework that provides data translation and identity resolution while keeping track of data provenance.

D. KNOWLEDGE GRAPH EMBEDDING

In recent years, dozens of Knowledge graph embedding (KGE) techniques have been developed to address tasks such as graph completion, question answering, and link prediction ([15], [38]–[41]. For instance, RESCAL [41] computes a three-way factorization of an adjacency tensor representing the input KG. The adjacency tensor is decomposed into a product of a core tensor and embedding matrices. HolE [40] uses circular correlation as its compositional operator. On the another hand, TransE [42] is an energy-based KGE model in which a relation r between entities h and t corresponds to a translation of their embeddings, i.e., $h + r \approx t$ provided that (h, r, t) exists in the KG. More details can concerning knowledge graph embedding approaches and applications can be found in [43].

VI. CONCLUSION & FUTURE WORK

We introduce NELLIE, a pipeline architecture to create a series of modules, each of which handles a different problem involving data augmentation. NELLIE starts by dealing with the problem of identifying relevant KGs for integration. NELLIE then takes over the KG data integration effort at both the ontology and instance levels. Then, NELLIE fuses

the matching classes and instances to make a fused KG. The last step is NELLIE's KG embedding of the fused KG.The ultimate goal of the proposed architecture is to create a single fused knowledge graph from the LOD (i.e., the development of a 24/7 solution for fusing knowledge graphs on the LOD, akin to the Never Ending Language Learner). In the KG matching stage, we implemented the three presented methods ourselves. In the ontology matching stage, we implemented the content-based class matching ourselves and integrated two state-of-the-art systems. For the instance matching stage, we base our implementation on the state-of-the-art link discovery framework LIMES [6], where we modified the way of training the WOMBAT [12] to generate link specifications. We then integrated LIMES into NELLIE as listed in Algorithm 1 in the paper. In the KG fusion stage, we implemented the additive fusion operator with many different fusion strategies. Finally, we studied the impact of KGs fusion on the link prediction task. In the paper, an ultimate goal would make it more comprehensive when we run it 24/7 efficiently, reliably, and fully automatically. However, in this paper, we present the architecture as a milestone toward this ultimate goal. Before that, we need a benchmark to test the efficiency and dependability of NELLIE as a whole, since the benchmarks we have now are made for specific subtasks like link prediction, ontology matching, and instance matching. In future work, we plan to apply more approaches for each task. For instance, we aim to apply automatic KGs matching approaches such as Tapioca [13] in order to make NELLIE fully automated 24/7 never ending linking framework. In addition, We plan to integrate fact checking to NELLIE. In the future, we aim to implement a benchmark for evaluating the efficiency and dependability of NELLIE as a whole.

REFERENCES

[1] M. Kanehisa, M. Furumichi, M. Tanabe, Y. Sato, and K. Morishima, "KEGG: new perspectives on genomes, pathways, diseases and drugs," Nucleic Acids Research, vol. 45, no. D1, pp. D353–D361, 11 2016. [Online]. Available: https://doi.org/10.1093/nar/gkw1092

[2] E. Jiménez-Ruiz and B. Cuenca Grau, "Logmap: Logic-based and scalable ontology matching," in The Semantic Web – ISWC 2011, L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. Noy, and E. Blomqvist, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 273–288.

[3] M. Niepert, C. Meilicke, and H. Stuckenschmidt, "A probabilistic-logical framework for ontology matching." in AAAI. Citeseer, 2010, pp. 1413–1418.

[4] M. Nentwig, M. Hartung, A.-C. Ngonga Ngomo, and E. Rahm, "A survey of current link discovery frameworks," Semantic Web, vol. 8, no. 3, pp. 419–436, 2017.

[5] A. N. Ngomo and S. Auer, "LIMES - A time-efficient approach for large-scale link discovery on the web of data," in IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011, T. Walsh, Ed. IJCAI/AAAI, 2011, pp. 2312–2317. [Online]. Available: https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-385

[6] A.-C. Ngonga Ngomo, M. A. Sherif, K. Georgala, M. Hassan, K. Dreßler, K. Lyko, D. Obraczka, and T. Soru, "LIMES - A Framework for Link Discovery on the Semantic Web," KI-Künstliche Intelligenz, German Journal of Artificial Intelligence - Organ des Fachbereichs "Künstliche Intelligenz" der Gesellschaft für Informatik e.V., 2021. [Online]. Available: https://papers.dice-research.org/2021/KI_LIMES/public.pdf

[7] R. Isele, A. Jentzsch, and C. Bizer, "Efficient Multidimensional Blocking for Link Discovery without losing Recall," in WebDB, 2011.

[8] M. Mountantonakis and Y. Tzitzikas, "Large-scale semantic integration of linked data: A survey," ACM Comput. Surv., vol. 52, no. 5, Sep. 2019. [Online]. Available: https://doi.org/10.1145/3345551

[9] P. N. Mendes, H. Mühleisen, and C. Bizer, "Sieve: Linked data quality assessment and fusion," in Proceedings of the 2012 Joint EDBT/ICDT Workshops, ser. EDBT-ICDT '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 116–123. [Online]. Available: https://doi.org/10.1145/2320765.2320803

[10] A. Schultz, A. Matteini, R. Isele, C. Bizer, and C. Becker, "Ldif-linked data integration framework," in Proceedings of the Second International Conference on Consuming Linked Data-Volume 782. CEUR-WS. org, 2011, pp. 125–130.

[11] M. A. Sherif, A. N. Ngomo, and J. Lehmann, "Automating RDF dataset transformation and enrichment," in The Semantic Web. Latest Advances and New Domains - 12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, May 31 - June 4, 2015. Proceedings, ser. Lecture Notes in Computer Science, F. Gandon, M. Sabou, H. Sack, C. d'Amato, P. Cudré-Mauroux, and A. Zimmermann, Eds., vol. 9088. Springer, 2015, pp. 371–387. [Online]. Available: https://doi.org/10.1007/978-3-319-18818-8_23

[12] M. A. Sherif, A. C. N. Ngomo, and J. Lehmann, "Wombat - A generalization approach for automatic link discovery," in The Semantic Web - 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 - June 1, 2017, Proceedings, Part I, ser. Lecture Notes in Computer Science, E. Blomqvist, D. Maynard, A. Gangemi, R. Hoekstra, P. Hitzler, and O. Hartig, Eds., vol. 10249, 2017, pp. 103–119. [Online]. Available: https://doi.org/10.1007/978-3-319-58068-5_7

[13] M. Röder, A.-C. Ngonga Ngomo, I. Ermilov, and A. Both, "Detecting similar linked datasets using topic modelling," in The Semantic Web. Latest Advances and New Domains, H. Sack, E. Blomqvist, M. d'Aquin, C. Ghidini, S. P. Ponzetto, and C. Lange, Eds. Cham: Springer International Publishing, 2016, pp. 3–19.

[14] I. Balazevic, C. Allen, and T. Hospedales, "TuckER: Tensor factorization for knowledge graph completion," in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 5185–5194. [Online]. Available: https://aclanthology.org/D19-1522

[15] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard, "Complex embeddings for simple link prediction," in International conference on machine learning. PMLR, 2016, pp. 2071–2080.

[16] B. Yang, W. Yih, X. He, J. Gao, and L. Deng, "Embedding entities and relations for learning and inference in knowledge bases," in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6575

[17] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, 2019, pp. 4171–4186. [Online]. Available: https://doi.org/10.18653/v1/n19-1423

[18] M. Zhao, S. Zhang, W. Li, and G. Chen, "Matching biomedical ontologies based on formal concept analysis," Journal of biomedical semantics, vol. 9, no. 1, pp. 1–27, 2018.

[19] M. Zhao and S. Zhang, "Identifying and validating ontology mappings by formal concept analysis." in OM@ ISWC, 2016, pp. 61–72.

[20] L. R. Tucker, "The Extension of Factor Analysis to Three-Dimensional Matrices," Contributions to Mathematical Psychology, vol. 110119, 1964.

[21] F. L. Hitchcock, "The expression of a tensor or a polyadic as a sum of products," Journal of Mathematics and Physics, vol. 6, no. 1-4, pp. 164–189, 1927.

[22] S. M. Kazemi and D. Poole, "Simple embedding for link prediction in knowledge graphs," in Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 4289–4300. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/b2ab001909a8a6f04b51920306046ce5-Abstract.html

[23] A.-C. N. Ngomo and K. Lyko, "Unsupervised learning of link specifications: deterministic vs. non-deterministic," in Proceedings of the Ontology Matching Workshop, 2013.
[24] J. Chen, E. Jiménez-Ruiz, I. Horrocks, D. Antonyrajah, A. Hadian, and J. Lee, "Augmenting ontology alignment by semantic embedding and distant supervision," in European Semantic Web Conference. Springer, 2021, pp. 392–408.
[25] M. Steyvers and T. Griffiths, "Latent semantic analysis: a road to meaning, chapter probabilistic topic models," Laurence Erlbaum, 2007.
[26] W. Buntine, J. Lofstrom, J. Perkio, S. Perttu, V. Poroshin, T. Silander, H. Tirri, A. Tuominen, and V. Tuulos, "A scalable topic-based open source search engine," in IEEE/WIC/ACM International Conference on Web Intelligence (WI'04). IEEE, 2004, pp. 228–234.
[27] J. Sleeman, T. Finin, A. Joshi et al., "Topic modeling for rdf graphs," in 3rd International Workshop on Linked Data for Information Extraction, 14th International Semantic Web Conference, vol. 1267, 2015, pp. 48–62.
[28] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," J. Mach. Learn. Res., vol. 3, no. null, p. 993–1022, Mar. 2003.
[29] J. Chen, P. Hu, E. Jimenez-Ruiz, O. M. Holter, D. Antonyrajah, and I. Horrocks, "Owl2vec*: Embedding of owl ontologies," Machine Learning, vol. 110, no. 7, pp. 1813–1845, 2021.
[30] V. Iyer, A. Agarwal, and H. Kumar, "VeeAlign: Multifaceted context representation using dual attention for ontology alignment," in Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 10 780–10 792. [Online]. Available: https://aclanthology.org/2021.emnlp-main.842
[31] J. Chakraborty, H. M. Zahera, M. A. Sherif, and S. K. Bansal, "Ontoconnect: Domain-agnostic ontology alignment using graph embedding with negative sampling," in 2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA), 2021, pp. 942–945.
[32] A.-C. Ngonga Ngomo and K. Lyko, "Eagle: Efficient active learning of link specifications using genetic programming." Springer Berlin Heidelberg, 2012.
[33] S. Araujo, J. Hidders, D. Schwabe, and A. P. De Vries, "Serimi-resource description similarity, rdf instance matching and interlinking," arXiv preprint arXiv:1107.1104, 2011.
[34] X. Niu, S. Rong, H. Wang, and Y. Yu, "An effective rule miner for instance matching in a web of data," in Proceedings of the 21st ACM International Conference on Information and Knowledge Management, ser. CIKM '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 1085–1094. [Online]. Available: https://doi.org/10.1145/2396761.2398406
[35] K. Nguyen, R. Ichise, and B. Le, "Slint: A schema-independent linked data interlinking system," in Proceedings of the 7th International Conference on Ontology Matching - Volume 946, ser. OM'12. Aachen, DEU: CEUR-WS.org, 2012, p. 1–12.
[36] J. Bleiholder and F. Naumann, "Data fusion," ACM Comput. Surv., vol. 41, no. 1, Jan. 2009. [Online]. Available: https://doi.org/10.1145/1456650.1456651
[37] H. L. Nguyen, D. T. Vu, and J. J. Jung, "Knowledge graph fusion for smart systems: A survey," Information Fusion, vol. 61, pp. 56 – 70, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1566253519307729
[38] X. Huang, J. Zhang, D. Li, and P. Li, "Knowledge graph embedding based question answering," in Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining. ACM, 2019.
[39] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," in Twenty-ninth AAAI conference on artificial intelligence, 2015.
[40] M. Nickel, L. Rosasco, and T. Poggio, "Holographic embeddings of knowledge graphs," in Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, ser. AAAI'16, pp. 1955–1961.
[41] M. Nickel, V. Tresp, and H.-P. Kriegel, "A three-way model for collective learning on multi-relational data." in ICML, vol. 11, 2011.
[42] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data." Curran Associates, Inc., 2013.
[43] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," IEEE Transactions on Knowledge and Data Engineering, vol. 29, no. 12, pp. 2724–2743, 2017.