A Full-Fledged Framework for Combining Entity Linking Systems and Components

Kristian Noullet
kristian.noullet@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Baden-Wurttemberg
Germany

Ayoub Ourgani
ayoub.ourgani@student.kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Baden-Wurttemberg
Germany

Michael Färber
michael.faerber@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Baden-Wurttemberg
Germany

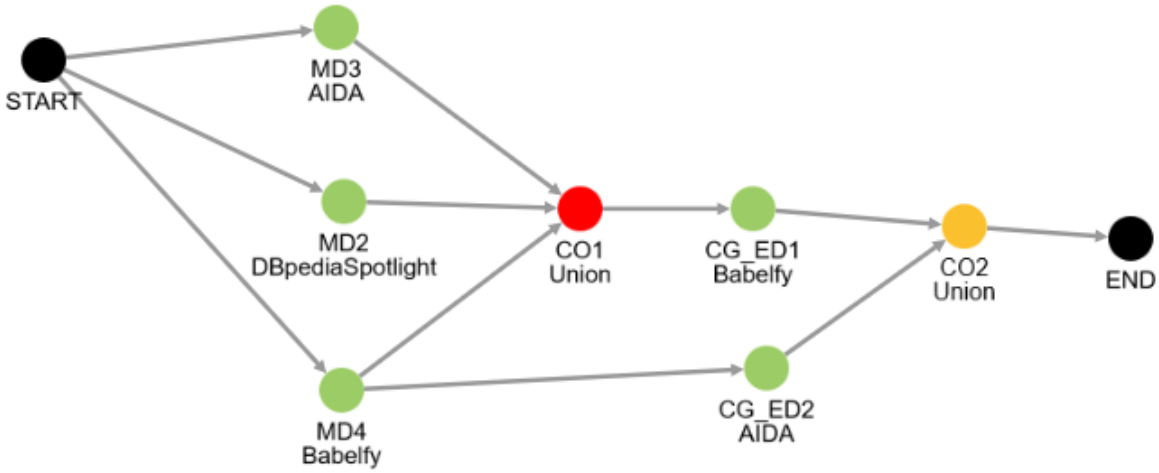


Figure 1: Visualization of a combination of entity linking components.

ABSTRACT

Named entity recognition and disambiguation, often referred to as entity linking systems, refers to the task of automatically identifying knowledge graph entities in text documents. While a variety of entity linking systems based on very different approaches exist, these systems implicitly share certain processing steps in their pipeline. Despite this fact, they have been mainly used as stand-alone solutions. In this paper, we propose a framework for combining entity linking methods. This allows multiple entity linking systems and especially their components to be used in combination to an unlimited extent, thus allowing to achieve the best possible performance. In addition, the framework allows user-developed entity linking systems or components to be easily tested and automatically evaluated against other systems without having to set up other systems first. Essentially, our framework is knowledge graph agnostic and entity linking systems can be compared across knowledge graphs. Furthermore, our framework enables entity linking method or component recommendation, supporting the goal of achieving the best performance in a given context. We demonstrate that non-domain-expert users are able to deploy the framework within minutes and integrate unknown homebrew systems into it in less than an hour. Our framework is fully open source and available on GitHub[1] along with Docker containers and tutorials[2] (incl. Jupyter Notebooks).

KEYWORDS

Entity Linking, Recommender System, Framework, FAIR, NLP, Semantic Web, NERD Orchestration.

[1] https://github.com/kmdn/combining-linking-techniques
[2] https://github.com/kmdn/clit-tutorials

1 INTRODUCTION

The field of entity linking is concerned with linking textual mentions in text-based documents to corresponding *knowledge graph* entities. Although entity linking has been successfully used in a variety of scenarios (e.g., semantic search), the design, analysis, evaluation of entity linking systems can be a tedious process [8, 24, 25].

First, this concerns the analysis of existing systems and their components. Each component (e.g., mention detection, candidate generation, ...) can affect the final outcome in unpredictable ways. For instance, if a *candidate generator* fails to find a particular entity in a knowledge graph, the disambiguation of entities of other mentions may be affected. Consequently, due to the complexity of interactions in processing pipelines, the task of researchers performing ablation studies is often a complex one. Comprehensive comparability with other entity linking systems is virtually impossible without extensive manual efforts to untangle components of one entity linking system and link them to another. Thus, a workflow is needed to allow components of a given entity linking system to be easily exchanged and immediately executed and evaluated.

Second, there is a lack in that creating a custom entity linking system including already-available components (e.g., disambiguation part) is still a tedious task. Until now, developers and researchers had to take care of the individual supported protocols as well as other specifics, such as supported knowledge graphs.

Third, while much research has been performed to improve the effectiveness of entity linking, no framework has yet been proposed to actually *combine* entity linking systems in a multi-subsystem fashion to achieve optimal performance based on output. To date, only individual research papers [12] have been published focusing on "AI methods" for ensemble learning rather than allowing any system to be reused and combined out-of-the-box.Further, these do not take into consideration individual components of entity linking systems. In addition, GERBIL [24] has been proposed as a related framework. However, it focuses on *evaluating* systems as a *whole* and ignores system components and their rich interactions. Moreover, GERBIL does not allow visual analysis of entity linking results or export of entity linking results based on arbitrary inputs; instead, only given benchmark datasets can be used. Thus, while GERBIL focuses on benchmarking, our system aims at (distributed, fine-grained) execution of an orchestra of entity linking systems.

To solve the described issues, we propose the full-featured entity linking framework *Combining Linking Techniques (CLiT)*. Specifically, the framework provides the following features to address the issues discussed above:

(1) We enable visualisation of Entity Linking (EL) workflows and certain interactions through a simple front-end demonstrator. It serves as a demo implementing a subset of the provided framework's features.
(2) We introduce novel ways for combining entity linking systems and their components while ensuring backwards compatibility with existing natural language processing annotation paradigms.
(3) We enable full configurability of all components.
(4) We enable down-stream processing of obtained entity linking annotation results rather than pure evaluation metrics.
(5) We enhance the re-usability and functionalities of existing entity linker components and ones to come, increasing the degree of system interoperability in the field.
(6) We support and provide knowledge graph-agnostic and multi-knowledge graph-supporting annotations.
(7) We support configurable metrics, evaluation schemes, and ways of recommending entity linking ensemble settings.

The source code of our framework, as well as detailed instructions, Jupyter Notebooks, tutorials, an experiment, videos, etc. may be found in our linked GitHub repository. Fig. 2 shows a front-end user interface to access functionalities of our framework with Fig. 1 visualising connections between various components on multiple levels. On one hand, the architecture of our framework is *centralised* in terms of workflow management, task distribution, and annotator tracking; on the other, all component- and sub-component-based functions can be executed in a distributed manner, which gives it *decentralised* characteristics, allowing for fine-grained resource management. While this setup allows for effective reusability of existing systems in the spirit of *reuse over redo*, the disadvantage of system decentralization is the framework's explicit reliance and requirement on systems being online and accessible.

The Combining Linking Techniques (CLiT) framework will not only enable higher performance of entity linking systems, but also significantly advance researchers' development and related applications, such as semantic search and recommender systems. A compelling example: our framework enables the setup of state-of-the-art entity linking combination workflows, as van Hulst et al. [26] did, within minutes – instead of estimated *hours* or *days* (see Section 6). An overview of some existing and potential use cases for the presented framework is presented in Section 4.

2 RELATED WORK

A preliminary version of our framework formed the basis for a demonstration system for combining entity linking systems [16]. However, the system presented in this paper differs significantly from the preliminary version in several ways. Among other things, our framework now features: (1) a data model that allows entity linking systems and their subcomponents to be combined (thus ensuring persistence and reproducibility of results), (2) user-defined remote invocation functionalities, (3) machine learning based linker recommendations, (4) novel extensible components (e.g., Evaluator, Explainer; see Sec. 3.3.6), (5) datasets for comparability testing, (6) sample pipelines & template components, (7) visualization front-end module allowing interaction with a subset of features, and (8) integration of other annotators relying on various knowledge graphs (e.g., DBpedia, Wikidata), making our framework fully usable by everyone.

In [18], the authors describe a framework that provides a REST API to easily request multiple entity linking systems. They also propose the NERD ontology to standardize the output format of the entity linking task. While the framework focuses on easily retrieving but not consolidating the results of multiple EL systems, the ontology aims to represent the final output of the entity linking task, not the partial steps. GERBIL [24] provides a benchmarking platform for centralized evaluation of different Natural Language Processing (NLP) mechanisms. We appreciate and support GERBIL

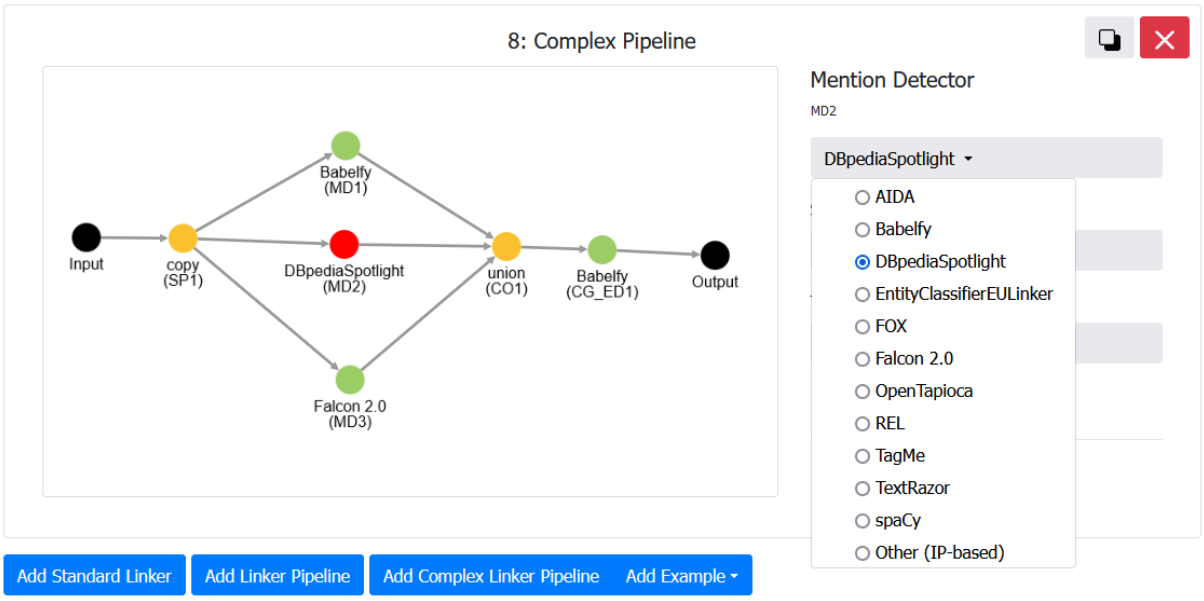


Figure 2: A proposed user interface visualising a combination of entity linking components.

in their efforts to improve the comparability of systems and build on their concepts and the tangential path they have paved that facilitates our research efforts in this regard. To our understanding, their influence has contributed greatly to the development of NIF-based protocols for annotators that we use and extend. They have also indirectly enabled us to further develop our conceptual paradigm and test its generalizability. While GERBIL is primarily for benchmarking NLP systems for comparison purposes, our intention is to support the development of new systems by allowing easy (re)use and combination of existing entity linking systems (see differentiation in the Introduction).

An alternative breakdown of the entity linking pipeline as defined by us is proposed by Sevgili et al. [21]. Their first of two main steps is *entity detection* (equivalent to Mention Detection (MD)), followed by *entity disambiguation*. The latter includes the substeps *candidate generation* and *entity ranking* in combination. João et al. [12] defines *Meta Entity Linking* (*MetaEL*) as the task of combining the results of multiple EL systems to find a good unified set of annotations. When combining entity linking systems, João et al. [12] do not refer to the granular substeps of EL systems, but to end-to-end systems that implement the entire pipeline. Canale et al. [1] treats the step of Named-Entity Recognition (NER) and Named-Entity Disambiguation (NED) separately when it proposes a neural network for each that integrates the results of eight NER/NED systems-without proposing a framework. Like [12], they focus on improving quality by combining existing approaches, but do not attempt to make the underlying entity linking systems comparable or interchangeable. Other approaches that focus on combining multiple *end-to-end* EL systems are [2], which provides rule-based and supervised merging of micropost annotations, and Ruiz et al. [19], which combine the results of five open source annotators through weighted voting. Overall, a framework that focuses on the actual execution of entity linking systems and their components is lacking so far.

3 THE COMBINING LINKING TECHNIQUES FRAMEWORK

In the following we present components and subcomponents of the framework, a subset of which may be accessed through a graphical user interface via the provided demonstrator front-end. Our framework is based on the *Pipes and Filters*[3] design pattern, effectively generating a dependency graph of nodes – so-called *(sub-)components* – and edges representing connections between these for information flow. For a graphical representation of said concepts, we refer to Figures 2 and 3.

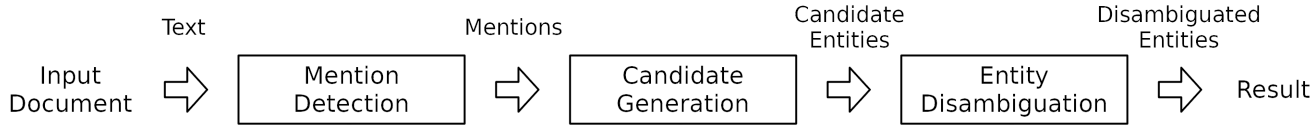
3.1 Classical Pipeline (Standard Linker)

While Named Entity Recognition and Disambiguation (NERD) systems, sometimes referred to as *annotators* or *linkers*, vary in terms of approaches and workflow steps, in our definition of *the classical pipeline* (see Figure 3) we identify the ones most commonly used. Our general conceptualization of *the classical pipeline* is further confirmed by the separation of duties defined by van Hulst et al. [26]. In our case, the classical pipeline consists of an **input document** as well as various mechanisms commonly referred to as **mention detection (MD)**, **candidate generation (CG)**, and **entity disambiguation (ED)**, and an annotated **output document**.

We have used this pipeline as a starting point for our framework to maximising existing-system compatibility. Therefore, we consider the paradigm of classical end-to-end pipeline instances as *standard linkers*. Another common configuration splitting an annotator into mention detection and a simultaneous "entity candidate

[3] https://docs.microsoft.com/en-us/azure/architecture/patterns/pipes-and-filters

Figure 3: Classical Pipeline for an entity linking system, consisting of mention detection (MD), candidate generation (CG) and entity disambiguation (ED).



generation and disambiguation" – these are referred to as *linker pipelines* and can be easily tuned by the framework.

In the following, we characterize the single steps (mention detection, candidate generation, entity disambiguation) in more detail. These are considered principal components within an entity linking pipeline and may be combined, split, filtered, executed sequentially to emulate a classical pipeline as well as executed in parallel with other similar components in order to a create customized annotation experience.

3.1.1 *Mention Detection.*

Preceded by:	Input document.
Succeeded by:	Candidate Generator or any subcomponent.
Commonly:	A combination of tokenization and predefined dictionary matching techniques.

A mention is a literal occurrence of a named entity as may be identified through textual information. A mention detection component is a mechanism taking an input document, generally made up of one or more plaintext sequences and yielding so-called *mentions*.

3.1.2 *Candidate Generation.*

Preceded by:	Mention detection or any subcomponent; technically any (sub)component with mentions to be enriched with entity candidates).
Succeeded by:	Entity disambiguator or any subcomponent.
Commonly:	A mention to entity candidates dictionary, e.g. generated from Wikipedia disambiguation pages.

Candidate generation refers to the fanning of candidates for each detected mention. This results in a number of potential entity candidates for each mention. Since the process is dependent on detected mentions, a chosen mention detection mechanism (or combinations thereof) will directly affect ensuing results.

3.1.3 *Entity Disambiguation.*

Preceded by:	Candidate generator or any subcomponent; technically any (sub)component leaving mentions and related candidates to some degree.
Succeeded by:	Result output possibilities or evaluator.
Commonly:	A coherence ranking mechanism based on global ('general knowledge') and local (context-dependant) features of entities.

Taking a group of mentions, each associated with related candidate entities, entity disambiguation refers to identifying the most likely entity (if any) a mention is referring to.

3.2 Existing Entity Linkers

Our framework supports the combination of several entity linking approaches and their single components. As of writing, a number of end-to-end annotators have been integrated out-of-the-box, namely AIDA [6], Babelfy [10], DBpediaSpotlight [14], EntityClassifierEU [4], Falcon 2.0 [20], FOX [22], OpenTapioca [3], REL [26], spaCy [9], TagMe/WAT [17], TextRazor [23] – with more on the way. For each of these, we have respectively introduced appropriate *mention detector* and *combined candidate generator & disambiguator* components for optimal intercompatibility. Further, for the specific use of mention detection, we have introduced an API template with a spaCy-based [9] detection mechanism and a step-by-step guide [4] to integrate FLAIR [5] (used in our user study). These templates may easily be adapted to accommodate any desired novel component. Please note that some of these online services are publicly accessible and their status are out of our control.

3.3 Novel Components (Complex Pipeline)

The following **subcomponents** ideologically place *in between* (*processing*) components presented within the classical model of a EL pipeline, as well as *before* (*preprocessing*) and *after* (*postprocessing*) of a given pipeline. To allow for individual experiences and configurations, in addition to *classical pipeline* elements and *standard annotators*, we introduce processing capabilities allowing for nearly infinite combinations of system components (see Fig. 2). These are coined *subcomponents*, which, among other things, handle post-processing of the results of certain structures from previous tasks and prepare them for possible further processing by subsequent steps in a configured workflow. We define *7 types of subcomponents*: *splitter*s, *combiner*s, *filter*s, *translator*s, *evaluator*s, *explainer*s and *recommender*s.

The motivation behind our choice of considering these subcomponents is linked to:

(1) Maximising interoperability of linkers and knowledge graphs (see *Translator* below);
(2) Allowing for ever-improving recommendations, learning from pipeline results (see *Recommender*);
(3) Enabling leveraging strengths from multiple approaches simultaneously (see *Splitter* and its counterpart, *Combiner*);
(4) Allowing for pruning of noise that may have been introduced through use of multiple systems (see *Filter*);
(5) Allowing for custom metric evaluation of results (see *Evaluator*);
(6) Explaining results of linkers and their respective evaluation results (see *Explainer*).

[4] https://colab.research.google.com/drive/1D0SgDqMA20w3PKodtOPAXTHsq9OsxkLr

The following paragraphs detail the processing subcomponents and their positioning relative to other pipeline components, their role(s) within a defined pipeline, and their most common uses.

3.3.1 Recommender (Preprocessing).
Preceded by: Input document.
Succeeded by: Output document.
Commonly: By default, a multi-label SVM implementation trained on CoNLL03-based generated labels.

Integrated into the core of our framework as a potential final goal and usage feature, we allow for the definition of custom recommender systems for entity linkers. By default, an internal recommender is applied based on input characteristics, predicting the most suitable ('standard') entity linker available.

3.3.2 Translator (Processing).
Preceded by: Any component or subcomponent.
Succeeded by: Any component or subcomponent.
Commonly: `owl:sameAs` linking across Knowledge Graph (KG)s.

Enabling seamless use of annotation tools regardless of underlying KG, the translator subcomponent is meant as a processing unit capable of *translating* entities and potentially other features, allowing further inter-system compatibility. It may be employed at any level and succeeded by any (sub-)component due to its ubiquitous characteristics and necessity when working with heterogeneous systems. Simple use cases include intercompatibilities of entities e.g. from `DBpedia`, `Wikipedia` and `Wikidata`.

3.3.3 Splitter (Processing).
Preceded by: Any single (sub)component.
Succeeded by: ≥2 (sub)components.
Commonly: Directly passing same information to two (or more) components.

Allowing for processing of items prior to passing them on to a subsequent step, a splitter is utilised in the case of a *single* stream of data being sent to *multiple* components, potentially warranting specific splitting of data streams (e.g. people-related entities being handled by one system, while another processes movies). This step encompasses both a post-processing step for a prior component, as well as a pre-processing step for a following one. A potential post-processing step may be to filter information from a prior step, such as eliminating superfluous candidate entities or unwanted mentions.

3.3.4 Combiner (Processing).
Preceded by: ≥2 (sub)components.
Succeeded by: Any single (sub)component.
Commonly: *Union* operation, *intersection* operation.

As a counterpart to a splitter, a *combiner* subcomponent must be utilised in case multiple components were utilised in a prior step and are meant to be consolidated through a variety of possible combination actions (e.g., union and intersection). It combines results from multiple inputs into a single output, passing merged partial results on to a subsequent component.

3.3.5 Filter (Processing).
Preceded by: Any single (sub)component.
Succeeded by: Any single (sub)component.
Commonly: NER-, Part-of-Speech (POS)-specific or `rdf:type` filtering.

To allow removal of particular sets of items through user-defined rules or dynamic filtering, we introduce a subcomponent capable of processing results on binary classifiers: a *filter*. The truth values evaluated on passed partial results define which further outcomes may be detected by a subsequent component or translator.

3.3.6 Evaluator (Post-processing).
Preceded by: Output document and an annotated NIF-based document.
Succeeded by: Explainer or displaying mechanisms.
Commonly: Base metrics computations, such as precision, recall, accuracy and F1-measure.

The evaluator takes the pipeline results and generates evaluation results based on them. These may be constituted of any custom metrics and displayed accordingly in generic fashion.

3.3.7 Explainer (Post-processing).
Preceded by: Evaluator, input document and annotation pipeline.
Succeeded by: Displaying mechanisms.
Commonly: Explanations based on result groupings relying on predefined evaluator metrics.

Takes evaluation results, as well as pipeline information in order to explain potential reasons for specific results with the goal of identifying patterns among results of qualitative differences.

3.4 Non-Integrated Custom Components

For novel system integrations that do not require programming skills, we enable on-the-fly component customization by allowing the definition of endpoints that can be integrated into our pipeline. In practice, such components can be added via their IP address (and port)through our REST API protocol definitions (see Sec. 3.5.2). Therefore, extensive custom component interactions are even possible without requiring knowledge of the specific implementation, lowering the barrier for groups of people who are not proficient programmers.

3.5 Technical Usage

To build on a solid foundation, rather than reinventing the wheel, we rely on as many existing entity linking systems and their components as possible. We leverage existing annotator APIs, as this allows for the most up-to-date results in case of system changes and minimizes future effort. CLiT's *generic* (JSON- / NIF-based) API and associated metadata constitute a novel protocol, used by our templates, guides, experiments and internal mechanisms. To enable usage of existing systems, we have internally defined custom adapter structures to easily integrate implementations – either locally or on remote services. To this end, interfaces[5], abstract classes,

[5] https://github.com/kmdn/clit_backend/tree/main/src/structure/interfaces

and common protocol implementations (e.g., GERBIL's NIF[6] annotator protocol[7]) have been defined in Java. We refer to our GitHub repository for more relevant information.

3.5.1 Setup. We have streamlined the setup process by defining Docker[8] images which may be run or built with a single line independently of any underlying files, dependencies or system-specific settings. Specific instructions provided on GitHub.

3.5.2 Data Structure. To ensure reliable and diverse data enrichment throughout our framework, we define a data structure in JSON format. It is passed from one (sub)component to the next, and in turn modified to meet the expectations of a subsequent (sub)component – meaning that input and output correspond to the same structure, with parts being modified or enriched at each step. For details, we refer to our GitHub page.[9] Said structure represents our main supported result format and contains component-enriched information as well as metadata be usable by components for dynamic customization.

Our general structure lists information crucial for result persistence, e.g., an experiment identifier (`experimentId`) and a collection of experiment tasks (`experimentTasks`). The latter consists of (1) the currently active component (`currentComponent`), (2) pipeline configuration information (`pipelineConfig`, see Fig. 3, GitHub README and our Jupyter Notebook tutorials[10]), (3) type of pipeline executed (`pipelineType`, e.g. complex), (4) task identifier (`taskId`), and (5) documents to be processed containing mentions and (optionally) entity annotations. Pipeline configurations contain information about the nature of connections between components, constituting a dependency graph with nodes representing (sub)components. For instance, in Fig. 3, we see a complex pipeline configuration utilising Babelfy as a mention detector component and Babelfy as a combined *candidate generator and entity disambiguation* component.

4 USE CASES

Due to the high degree of modularity, extension, ease of execution and setup, our framework may be utilised for a variety of use cases – we list a few of them.

4.1 Use Case 1: Entity Linking Orchestration for Best Performance

As seen in van Hulst et al. [26], a state-of-the-art entity linker can be created by *standing on the shoulders of giants*. The authors concatenated individual approaches for mention detection, candidate generation, and entity disambiguation. Specifically, for mention detection, they used FLAIR [5]. Their candidates for entity disambiguation are produced through approaches based on [11, 27]. Finally, disambiguation takes place with help of Le and Titov's approach [13]. CLiT allows for these types of combinations to be produced within minutes.

4.2 Use Case 2: Component Development

One of our main uses includes the development of novel components. Presumably, the most efficient way to develop novel components and gain insightful knowledge is to compare with existing ones. Through the simplified execution of techniques, NLP developers may notice patterns (e.g., domain-relatedness) relating to specific similar components and easily extend these. Furthermore, they may improve upon developed components by leveraging existing approaches to counteract their own limitations, for instance by merging newly-generated intermediary results with existing ones. Proper interoperability and comparability of developed techniques may be further tested and executed on multiple knowledge graphs through *translator* subcomponents, among others.

4.3 Use Case 3: Analysis and Evaluation of Entity Linking Components

By running fine-grained experiments, results may be analyzed with our framework and the systems strengths and limitations can be evaluated with a single execution. Moreover, by running the same pipeline with a single component replaced, its quality and complex ensuing effects may be evaluated in an objective, repeatable, and easily comparable environment. For instance, a researcher may switch out a mention detection technique and notice different spans, directly affecting results in positive or negative ways.

4.4 Use Case 4: Contextual Linker Recommendation

A relatively novel area of research is the recommendation [12] and combination [15, 16, 19] of entity annotators. So far, no *framework* has been proposed to combine entity linking systems or specific system components in an orchestra to achieve the best performance. By performing complex interactions of possible workflows, documents can be annotated through a variety of pipelines with differing results. These resulting outputs may be used for the development of AI models predicting metadata-dependent workflows. In addition, explanations of the results can serve as labels that allow entity linking approaches to be classified into specific categories, generating rich information on which document-based recommendations for linker predictions can be based. Our framework's front-end provides a proof-of-concept implementation of linker recommendation.

[6] NLP Interchange Format definition: https://persistence.uni-leipzig.org/nlp2rdf/
[7] https://github.com/dice-group/gerbil/wiki/How-to-create-a-NIF-based-web-service
[8] https://www.docker.com/
[9] https://github.com/kmdn/combining-linking-techniques#json-result-structure
[10] https://github.com/kmdn/clit-tutorials/tree/main/2.%20Use%20Own%20Pipeline%20Components

5 CORE VALUES AND PRINCIPLES

Impact. We target researchers working in areas such as Semantic Web (SW), KG, and NLP, including NER and NERD, benefitting from high-quality annotations and techniques enabling in-depth, and tailored analysis of entity linking results, provided by CLiT. In particular, Semantic Web-related areas relying on entity linking, such as semantic search, recommender systems, knowledge graph creation and enrichment, and relation extraction, can particularly benefit and be influenced by our efforts and novel conceptual paradigms. Not only can experts in related fields leverage our framework, but many functionalities are accessible without prior domain-specific knowledge through a single line of execution.

For intercomponent communication we relied on common best practices for APIs and utilised state-of-the-art RESTful principles to execute multi-component workflows. Where applicable, we relied on Semantic Web (SW)-based NLP Interchange Format (NIF) endpoints and included them (rather than reimplementing them) to maximize resource reuse. Unfortunately, not all annotator endpoints support the aforementioned protocol. Therefore, JSON and custom communication formats were used in our adapters when needed to unify their execution modes with standardised API calls. To validate the generalizability of our paradigms, we implemented annotators based on a variety of knowledge graphs and different design philosophies, additionally to (sub)components enabling rich system interactions. Throughout execution workflows, a single data structure enriched with relevant (meta)data is employed. Thus far, we have not been subject to limitations thereof, seemingly strengthening its notion of generalizability.

Our framework follows the FAIR principles:

(1) **Findable**: All experimental (meta)data is available in *persistent* JSON and NLP interchange formats (NIF) with *persistent identifiers*.
(2) **Accessible**: The framework including source code, templates, Docker (configurations, build files) and Maven generation of executables, along with all relevant files allowing for setup and execution, are provided on our GitHub repository.
(3) **Interoperable**: We ensure the interoperability of our framework on several levels. First, all result (meta)data is provided in machine-processable JSON documents, providing an easily customizable experience based on user needs, readily interoperable with respect to results from other systems and systems themselves. Second, each component may be accessed separately, making them completely interoperable with existing paradigms. More specifically, the same JSON format is used between all components, maximizing flexible on-the-fly interoperability. Finally, we provide interoperability between systems of different knowledge graphs through easily extensible *translator* components (see our guides for customizations).
(4) **Re-Usable**: Re-usability as well as reproducibility are maximized through a multitude of factors, namely: (1) provided Docker containers; (2) provided modular and extensible source code; (3) templates for easy re-implementation and execution of integrated approaches; (4) videos and guides to enable interoperability with existing systems; (5) all data sets may be easily used or added to the framework (e.g. NIF format).

Availability. Our code and necessary file dependencies for single-line execution (via Docker) and build are published as permanent URLs in the form of DOI[11], W3ID[12], posted on our corresponding GitHub pages[13] under the MIT license with a demo[14] allowing certain limited front-end interactions. CLiT is in development at the authors' research institute and actively utilised in two multi-year projects and related doctoral theses – with plans to include further areas of NLP research in order to maximise its sustainability and longevity.

6 EVALUATION

To facilitate rich and complex entity linking system interactions in our domain, CLiT simplifies processes and provides tools for easy integration of novel systems. To evaluate CLiT, we designed an experiment to test user experience for common scenarios and allow participants to familiarize themselves with a running system. We also conducted two standardized usability assessment questionnaires.

Experiment. We designed an experiment[15] comprising a series of tasks introducing testers to the *Combining Linking Techniques* ecosystem. Testers were guided through a series of tasks via Jupyter Notebook [7] (on Google Colab) utilising code snippets based on our tutorials [16]. The experiment consists of setting up an API for a custom system, allowing for smooth integration into the entity linking workflow, execution thereof, a series of pipeline re-configurations and adaptations, as well as combination of multiple systems within workflows. Further, testers played around with the system, creating complex workflows making use of a multitude of entity linking (sub-)systems.

User Study. We assessed the usability of our system by having 13 testers fill out established tests for software usability assessment, namely User Experience Questionnaire (UEQ)[17] and System Usability Scale (SUS)[18].

Based on our study, the average expertise level of testers in the domains of semantic web (SW), natural language processing (NLP), and machine learning (ML) was **3.23** out of **5**. Additionally, testers were found to be reasonably knowledgeable with programming and the uses of APIs, with an average expertise level of **3.69** out of **5**. During the experiment, most users took between 10 to 20 minutes to complete it, while the majority claimed they would need over 2 to 4 hours or more to complete it without CLiT. It is worth noting that our testing subjects included both experts and non-experts, and we observed a higher level of overall satisfaction among more expert testers. Our study also found that it took testers less than 20 minutes to integrate a novel system into our framework, indicating that we have greatly lowered the barrier of entry for developers to add

[11] Zenodo DOI: https://doi.org/10.5281/zenodo.8318674
[12] W3ID: https://w3id.org/combining-linking-techniques/
[13] GitHub: https://github.com/kmdn/combining-linking-techniques
[14] http://clit.tech
[15] https://colab.research.google.com/drive/1D0SgDqMA20w3PKodtOPAXTHsq9OsxkLr
[16] https://github.com/kmdn/clit-tutorials
[17] https://www.ueq-online.org/
[18] https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

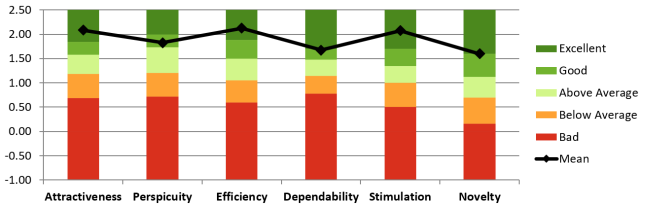


Figure 4: User Experience Questionnaire results

their own systems. This finding is encouraging and suggests that our framework can be easily adopted by developers with varying levels of expertise in the domains of SW, NLP, and ML. As scores for both standardised questionnaires average to above standard mean values (exclusively *good*, *very good* or *excellent*), we consider CLiT to have an above average usability and assume it could be of great use in the domain of EL.

System Usability Score (SUS). The *System Usability Score* questionnaire is a simple 10-question questionnaire, evaluating usability, outputting a single score out of 100. CLiT scored **77** out of **100** points. For this questionnaire, 68 is considered the mean threshold and our score translates to a *very good*[19]. Below 68 signifies *below average usability*, above 68 implies a *above average usability*.

User Experience Questionnaire. This questionnaire comprises a comprehensive impression of user experience and measures 6 dimensions, each on a scale from -3 to +3. These dimensions are (1) attractiveness, (2) perspicuity, (3) efficiency, (4) dependability, (5) stimulation and (6) novelty. Results for this questionnaire are illustrated in Fig. 4. CLiT scores exclusively in the *Excellent* and *Good* categories of the questionnaire. Specifically, in the domains of *Attractiveness* (2.09), *Efficiency* (2.13) and *Stimulation* (2.08), our framework reached mean values within the defined *Excellent* range, whereas *Novelty* (1.60) and *Dependability* (1.65) reach means in the upper segment of the *Good* range - visually on the threshold between *Excellent* and *Good*. Only *Perspicuity* (1.81) reaches a mean in the lower half of *Good*.

7 CONCLUSION & FUTURE WORK

In this paper, we introduced a decentralized execution framework called *CLiT* that allows for rich combination and execution of multiple entity linking approaches. It supports a variety of end-to-end entity linking systems based on different principles, technologies, and knowledge graphs. We demonstrate how the systems can be leveraged, extended, and modified through sub-components, including interactions for data enrichment, noise reduction, and preprocessing, as well as simultaneous use of multiple techniques. Additionally, we introduce post-processing steps for evaluation and result explanation, along with a use case for recommending entity linkers. We evaluated CLiT by conducting a user study, putting the overall usability of the framework around *good* to *excellent*. In the future, we plan working on semi-automated deep analysis capabilities that enable collaborative evaluation and lead to more fine-grained evaluations of both annotators and datasets.

[19] https://measuringu.com/sus/

ACKNOWLEDGMENTS

This work was partially supported by the German Federal Ministry of Education and Research (BMBF) via *Find What Matters* (FWM), a Software Campus project (01IS17042).

REFERENCES

[1] Lorenzo Canale, Pasquale Lisena, and Raphaël Troncy. 2018. A Novel Ensemble Method for Named Entity Recognition and Disambiguation Based on Neural Network. In *The Semantic Web – ISWC 2018*, Denny Vrandečić, Kalina Bontcheva, Mari Carmen Suárez-Figueroa, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee, and Elena Simperl (Eds.). Springer International Publishing, Cham, 91–107.

[2] Francesco Corcoglioniti, Alessio Palmero Aprosio, Yaroslav Nechaev, and C. Giuliano. 2016. MicroNeel: Combining NLP Tools to Perform Named Entity Detection and Linking on Microposts. In *CLiC-it/EVALITA*.

[3] Antonin Delpeuch. 2020. OpenTapioca: Lightweight Entity Linking for Wikidata. In *Proceedings of the 1st Wikidata Workshop co-located with the 19th International Semantic Web Conference* (Virtual Event) *(Wikidata'20, Vol. 2773)*. CEUR-WS.org.

[4] Milan Dojchinovski and Tomás Kliegr. 2013. Entityclassifier.eu: Real-Time Classification of Entities in Text with Wikipedia. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases* (Prague, Czech Republic) *(ECML-PKDD'13)*. Springer, 654–658.

[5] Akbik et al. 2019. FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP. 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, Demonstrations*, Waleed Ammar, Annie Louis, and Nasrin Mostafazadeh (Eds.). Association for Computational Linguistics, 54–59. https://doi.org/10.18653/v1/n19-4010

[6] Hoffart et al. 2011. Robust Disambiguation of Named Entities in Text. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing* (Edinburgh, UK) *(EMNLP'11)*. ACL, 782–792.

[7] Kluyver et al. 2016. Jupyter Notebooks - a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas, 20th International Conference on Electronic Publishing, Göttingen, Germany, June 7-9, 2016*. IOS Press, 87–90. https://doi.org/10.3233/978-1-61499-649-1-87

[8] Shen et al. 2015. Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions. *IEEE Trans. Knowl. Data Eng.* 27, 2 (2015), 443–460. https://doi.org/10.1109/TKDE.2014.2327028

[9] Explosion. 2021. *spaCy, Industrial-Strength Natural Language Processing*. https://spacy.io/

[10] Tiziano Flati and Roberto Navigli. 2014. Three Birds (in the LLOD Cloud) with One Stone: BabelNet, Babelfy and the Wikipedia Bitaxonomy. In *Proceedings of the Posters and Demos Track of 10th International Conference on Semantic Systems* (Leipzig, Germany) *(SEMANTiCS'14, Vol. 1224)*. CEUR-WS.org, 10–13.

[11] Ganea and Hofmann. 2017. Deep Joint Entity Disambiguation with Local Neural Attention. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*. Association for Computational Linguistics, 2619–2629. https://doi.org/10.18653/v1/d17-1277

[12] Renato Stoffalette João, Pavlos Fafalios, and Stefan Dietze. 2020. Better Together: An Ensemble Learner for Combining the Results of Ready-Made Entity Linking Systems. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing* (Brno, Czech Republic) *(SAC '20)*. Association for Computing Machinery, New York, NY, USA, 851–858. https://doi.org/10.1145/3341105.3373883

[13] Phong Le and Ivan Titov. 2018. Improving Entity Linking by Modeling Latent Relations between Mentions. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, Iryna Gurevych and Yusuke Miyao (Eds.). Association for Computational Linguistics, 1595–1604. https://doi.org/10.18653/v1/P18-1148

[14] Pablo N. Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. 2011. DBpedia Spotlight: Shedding Light on the Web of Documents. In *Proceedings the 7th International Conference on Semantic Systems* (Graz, Austria) *(I-SEMANTICS'11)*. ACM, 1–8.

[15] Kristian Noullet. 2020. KG-Agnostic Entity Linking Orchestration. In *Proceedings of the Doctoral Consortium at ISWC 2020 co-located with 19th International Semantic Web Conference (ISWC 2020), Athens, Greece, November 3rd, 2020 (CEUR Workshop Proceedings, Vol. 2798)*. CEUR-WS.org, 41–48. http://ceur-ws.org/Vol-2798/paper6.pdf

[16] Kristian Noullet, Samuel Printz, and Michael Färber. 2021. CLiT: Combining Linking Techniques for Everyone. In *The Semantic Web: ESWC 2021 Satellite Events - Virtual Event, June 6-10, 2021, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 12739)*. Springer, 88–92. https://doi.org/10.1007/978-3-030-80418-3_16

[17] Francesco Piccinno and Paolo Ferragina. 2014. From TagME to WAT: a new entity annotator. In *ERD'14, Proceedings of the First ACM International Workshop on Entity Recognition & Disambiguation, July 11, 2014, Gold Coast, Queensland, Australia*. ACM, 55–62. https://doi.org/10.1145/2633211.2634350

[18] Giuseppe Rizzo et al. 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 Association for Computational Linguistics* (Avignon, France) *(EACL '12)*. Association for Computational Linguistics, USA, 73–76.

[19] Pablo Ruiz and Thierry Poibeau. 2015. Combining Open Source Annotators for Entity Linking through Weighted Voting. In *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics*. Association for Computational Linguistics, Denver, Colorado, 211–215. https://doi.org/10.18653/v1/S15-1025

[20] Ahmad Sakor, Kuldeep Singh, Anery Patel, and Maria-Esther Vidal. 2020. Falcon 2.0: An Entity and Relation Linking Tool over Wikidata. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, Mathieu d'Aquin, Stefan Dietze, Claudia Hauff, Edward Curry, and Philippe Cudré-Mauroux (Eds.). ACM, 3141–3148. https://doi.org/10.1145/3340531.3412777

[21] Ozge Sevgili, Artem Shelmanov, Mikhail Arkhipov, Alexander Panchenko, and Chris Biemann. 2020. Neural Entity Linking: A Survey of Models based on Deep Learning. arXiv:2006.00575 [cs.CL]

[22] René Speck and Axel-Cyrille Ngonga Ngomo. 2014. Named Entity Recognition using FOX. In *Proceedings of the ISWC 2014 Posters & Demonstrations Track of the 13th International Semantic Web Conference* (Riva del Garda, Italy) *(ISWC'14, Vol. 1272)*. CEUR-WS.org, 85–88.

[23] TextRazor Ltd. 2023. *TextRazor, Extract Meaning from your Text.* https://www.textrazor.com/

[24] Ricardo Usbeck, Michael Röder, Axel-Cyrille Ngonga Ngomo, Ciro Baron, Andreas Both, Martin Brümmer, Diego Ceccarelli, Marco Cornolti, Didier Cherix, Bernd Eickmann, Paolo Ferragina, Christiane Lemke, Andrea Moro, Roberto Navigli, Francesco Piccinno, Giuseppe Rizzo, Harald Sack, René Speck, Raphaël Troncy, Jörg Waitelonis, and Lars Wesemann. 2015. GERBIL: General Entity Annotator Benchmarking Framework. In *Proceedings of the 24th International Conference on World Wide Web* (Florence, Italy) *(WWW'15)*. ACM, 1133–1143.

[25] van Erp et al. 2016. Evaluating Entity Linking: An Analysis of Current Benchmark Datasets and a Roadmap for Doing a Better Job. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation LREC 2016, Portorož, Slovenia, May 23-28, 2016*. European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2016/summaries/926.html

[26] Johannes M. van Hulst, Faegheh Hasibi, Koen Dercksen, Krisztian Balog, and Arjen P. de Vries. 2020. REL: An Entity Linker Standing on the Shoulders of Giants. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20)*. ACM.

[27] Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yoshiyasu Takefuji. 2018. Wikipedia2Vec: An Optimized Tool for Learning Embeddings of Words and Entities from Wikipedia. *CoRR* abs/1812.06280 (2018). arXiv:1812.06280 http://arxiv.org/abs/1812.06280