Why Reinvent the Wheel – Let's Build Question Answering Systems Together

Kuldeep Singh
University of Bonn & Fraunhofer IAIS, Germany
kuldeep.singh@iais.fraunhofer.de

Arun Sethupat Radhakrishna
University of Minnesota, USA
sethu021@umn.edu

Andreas Both
DATEV eG, Germany
contact@andreasboth.de

Saeedeh Shekarpour
University of Dayton, Dayton, USA
sshekarpour1@udayton.edu

Ioanna Lytra
University of Bonn & Fraunhofer IAIS, Germany
lytra@cs.uni-bonn.de

Ricardo Usbeck
University of Paderborn, Germany
ricardo.usbeck@uni-paderborn.de

Akhilesh Vyas
University of Bonn & Fraunhofer IAIS, Germany
akhilesh.vyas@iais.fraunhofer.de

Akmal Khikmatullaev
University of Bonn & Fraunhofer IAIS, Germany
akmal.khikmatullaev@gmail.com

Dharmen Punjani
University of Athens, Greece
dharmen.punjani@gmail.com

Christoph Lange
University of Bonn & Fraunhofer IAIS, Germany
christoph.lange@uni-bonn.de

Maria Esther Vidal
Leibniz Information Centre For Science and Technology University Library & Fraunhofer IAIS, Germany
maria.vidal@tib.eu

Jens Lehmann
University of Bonn & Fraunhofer IAIS, Germany
jens.lehmann@iais.fraunhofer.de

Sören Auer
Leibniz Information Centre For Science and Technology University Library & University of Hannover, Germany
soeren.auer@tib.eu

ABSTRACT

Modern question answering (QA) systems need to flexibly integrate a number of components specialised to fulfil specific tasks in a QA pipeline. Key QA tasks include Named Entity Recognition and Disambiguation, Relation Extraction, and Query Building. Since a number of different software components exist that implement different strategies for each of these tasks, it is a major challenge to select and combine the most suitable components into a QA system, given the characteristics of a question. We study this optimisation problem and train classifiers, which take features of a question as input and have the goal of optimising the selection of QA components based on those features. We then devise a greedy algorithm to identify the pipelines that include the suitable components and can effectively answer the given question. We implement this model within FRANKENSTEIN, a QA framework able to select QA components and compose QA pipelines. We evaluate the effectiveness of the pipelines generated by FRANKENSTEIN using the QALD and LC-QuAD benchmarks. These results not only suggest that FRANKENSTEIN precisely solves the QA optimisation problem but also enables the automatic composition of optimised QA pipelines, which outperform the static Baseline QA pipeline. Thanks to this flexible and fully automated pipeline generation process, new QA components can be easily included in FRANKENSTEIN thus improving the performance of the generated pipelines.

CCS CONCEPTS

• **Computing methodologies → Natural language processing**; **Knowledge representation and reasoning**;

KEYWORDS

Question Answering, Software Reusability, Semantic Web, Semantic Search, QA Framework

ACM ISBN 978-1-4503-5639-8/18/04.
https://doi.org/10.1145/3178876.3186023

1 INTRODUCTION

Answering questions based on information encoded in knowledge graphs has recently received much attention by the research community. Since 2010, more than 62 systems for question answering (QA) over the Web of Data have been developed [12]. These systems typically include components building on Artificial Intelligence,

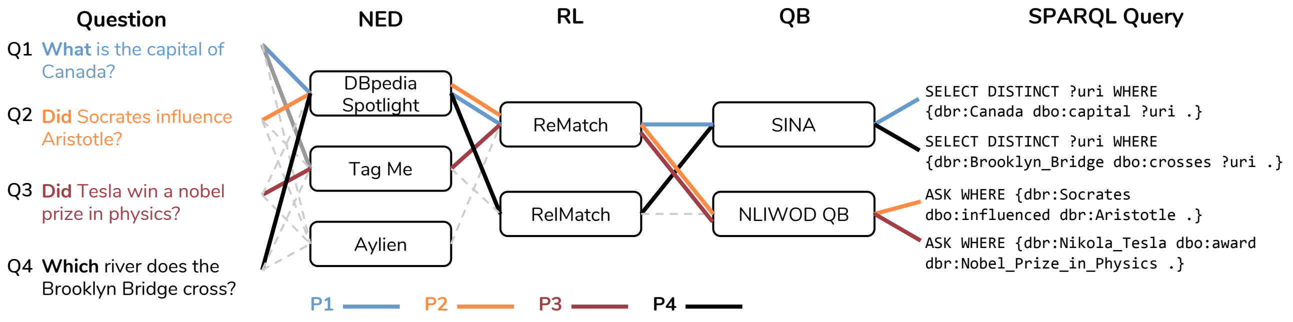
Question
NED
RL
QB
SPARQL Query
Q1 What is the capital of Canada?
Q2 Did Socrates influence Aristotle?
Q3 Did Tesla win a nobel prize in physics?
Q4 Which river does the Brooklyn Bridge cross?
DBpedia Spotlight
Tag Me
Aylien
ReMatch
RelMatch
SINA
NLIWOD QB
SELECT DISTINCT ?uri WHERE {dbr:Canada dbo:capital ?uri .}
SELECT DISTINCT ?uri WHERE {dbr:Brooklyn_Bridge dbo:crosses ?uri .}
ASK WHERE {dbr:Socrates dbo:influenced dbr:Aristotle .}
ASK WHERE {dbr:Nikola_Tesla dbo:award dbr:Nobel_Prize_in_Physics .}
P1
P2
P3
P4

Figure 1: Four natural language questions answered successfully by different pipelines composed of three NED, two RL, and two QB components. The optimal pipelines for each question are highlighted.

Natural Language Processing, and Semantic Technology; they implement common tasks such as Named Entity Recognition and Disambiguation, Relation Extraction, and Query Building. Evaluation studies have shown that there is no best performing QA system for all types of Natural Language (NL) questions; instead, there is evidence that certain systems, implementing different strategies, are more suitable for certain types of questions [22]. Hence, modern QA systems need to flexibly integrate a number of components specialised to fulfil specific tasks in a QA pipeline.

Relying on these observations, we devise FRANKENSTEIN, a framework able to dynamically select QA components in order to exploit the properties of the components to optimise the F-Score. FRANKENSTEIN implements a classification based learning model, which estimates the performance of QA components for a given question, based on its features. Given a question, the FRANKENSTEIN framework implements a greedy algorithm to generate a QA pipeline consisting of the best performing components for this question.

We empirically evaluate the performance of FRANKENSTEIN using two renowned benchmarks from the Question Answering over Linked Data Challenge[1] (QALD) and the Large-Scale Complex Question Answering Dataset[2] (LC-QuAD). We observe that FRANKENSTEIN is able to combine QA components to produce optimised QA pipelines outperforming the static Baseline pipeline.
In summary, we provide the following contributions:

- FRANKENSTEIN framework relying on machine learning techniques for dynamically selecting suitable QA components and composing QA pipelines based on the input question, thus optimising the overall F-Score.
- A collection of 29 reusable QA components that can be combined to generate 360 distinct QA pipelines, integrated in the FRANKENSTEIN framework.
- An in-depth analysis of advantages and disadvantages of QA components in QA pipelines after a thorough benchmarking of the performance of the FRANKENSTEIN pipeline generator using over 3,000 questions from the QALD and LC-QuAD QA benchmarks.

As a result of this work, we expect a new class of QA systems to emerge. Currently, QA systems are tailored to a particular domain (mostly common knowledge), a source of background knowledge (most commonly DBpedia [1]) and benchmarking data (most commonly QALD). Based on FRANKENSTEIN, more flexible, domain-agnostic QA systems can be built and quickly adapted to new domains.

The remainder of this article is structured as follows: We introduce the motivation of our work in Section 2. The problem tackled as well as the proposed solution are discussed in Section 3 and the details of FRANKENSTEIN are presented in Section 4. The Section 5 describes the preparation of training datasets followed by performance evaluation of components in Section 6. Section 7 reports the empirical evaluation of FRANKENSTEIN QA pipelines, with subsequent discussions in Section 8. The related work is reviewed in Section 9 and finally, conclusions and directions for future work are discussed in Section 10.

2 MOTIVATING EXAMPLE

A great number of components perform QA tasks – either as part of QA systems or standalone [26]. Table 1 presents several QA components, implementing the QA tasks NED (Named Entity Disambiguation) implemented by (i) DBpedia Spotlight [18], (ii) Aylien API[3], and (iii) Tag Me API [8]), RL (Relation Linking) implemented by (i) ReMatch [20] and (ii) RelMatch [15]), and QB (Query Building) implemented by (i) SINA [23] and (ii) NLIWOD QB[4]).

For example, given the question *"What is the capital of Canada?"*, the ideal NED component is expected to recognise the keyword *"Canada"* as a named entity and map it to the corresponding DBpedia resource, i.e. `dbr:Canada`[5]. Thereafter, a component performing RL finds embedded relations in the given question and links them to appropriate relations of the underlying knowledge graph. In our example, the keyword *"capital"* is mapped to the relation `dbo:capital`[6]. Finally, the QB component generates a formal query (e.g. expressed in SPARQL), which retrieves all answers from the corresponding knowledge graph (i.e. `SELECT ?c {dbr:Canada dbo:capital ?c.}`).

Table 1 presents precision, recall, and F-Score of the listed components for the QALD-5 benchmark (cf. [30] and Section 5.1). We observe that DBpedia Spotlight, ReMatch, and NLIWOD QB achieve the best performance for the tasks NED, RL, and QB, respectively

[1]https://qald.sebastianwalter.org/index.php?x=home&q=5
[2]http://lc-quad.sda.tech/
[3]http://docs.aylien.com/docs/introduction
[4]Component is based on https://github.com/dice-group/NLIWOD and [29].
[5]The prefix dbr is bound to `http://dbpedia.org/resource/`.
[6]The prefix dbo is bound to `http://dbpedia.org/ontology/`.

(cf. Section 6 for details). When QA components are integrated into a QA pipeline, the overall performance of the pipeline depends on the individual performance of each component. The fact that a particular component gives superior performance for a task on a given set of questions does not imply that the component is superior for all types of questions. That is, the performance of components varies depending on the type of question.

Table 1: Performance of QA components implementing various QA tasks evaluated with the QALD-5 benchmark.

QA Component	QA Task	Precision	Recall	F-Score
DBpedia Spotlight	NED	0.67	0.76	0.71
Aylien API	NED	0.60	0.66	0.63
Tag Me API	NED	0.47	0.57	0.52
ReMatch	RL	0.54	0.74	0.62
RelMatch	RL	0.10	0.19	0.13
SINA	QB	0.38	0.41	0.39
NLIWOD QB	QB	0.49	0.50	0.49

The performance values in Table 1 are averaged over the entire query inventory. They are not representative for the specific performance of components for various types of input questions. For example, Figure 1 illustrates the best performing QA pipelines for four exemplary input questions. We observe that Pipeline P1 is the most efficient for answering Question Q_1: *"What is the capital of Canada?"* but it fails to answer Question Q_4: *"Which river does the Brooklyn Bridge cross?"*. This is caused by the fact that the RL component ReMatch in Pipeline P1 does not correctly map the relation `dbo:crosses` in Q_4 for the input keyword *"cross"*, while RelMatch maps this relation correctly. Although the overall precision of ReMatch on QALD-5 is higher than that of RelMatch, for Q_4, the performance of RelMatch is higher. Similarly, for Question Q_2 *"Did Socrates influence Aristotle?"* Pipeline P2 delivers the desired answer, while it fails to answer the similar question Q_3 *"Did Tesla win a nobel prize in physics?"*. Although questions Q_2 and Q_3 have a similar structure (i.e. Boolean answer type), DBpedia Spotlight NED succeeds for Q_2, but on Q_3 it fails to disambiguate the resource `dbr:Nobel_Prize_in_Physics`. At the same time, Tag Me can accomplish the NED task successfully. Although, the optimal pipeline for a given question can be identified experimentally by executing all possible pipelines, this approach is costly and even practically impossible, since covering all potential input questions is not feasible. Therefore, a heuristic approach to identify an optimal pipeline for a given input question is required.

3 PROBLEM STATEMENT

A full QA pipeline is composed of all the necessary tasks to transform a user-supplied Natural Language (NL) question into a query in a formal language (e.g. SPARQL), whose evaluation retrieves the desired answer(s) from an underlying knowledge graph. Correctly answering a given input question q requires a QA pipeline that, ideally, uses those QA components that deliver the best precision and recall for answering q. Identifying the best performing QA pipeline for a given question q requires: (i) a prediction mechanism to predict the performance of a component given a question q, a required task, and a knowledge graph λ; (ii) an approach for composing an optimised pipeline by integrating the most accurate components.

3.1 Predicting Best Performing Components

In this context, we formally define a set of necessary QA tasks as $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ such as NED, RL, and QB. Each task $(t_i : q^* \rightarrow q^+)$ transforms a given representation q^* of a question q into another representation q^+. For example, NED and RL tasks transform the input representation *"What is the capital of Canada?"* into the representation *"What is the* `dbo:capital` *of* `dbr:Canada`*?"*. The entire set of QA components is denoted by $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$. Each component C_j solves one single QA task; $\theta(C_j)$ corresponds to the QA task t_i in $\mathcal{T}$ implemented by C_j. For example, ReMatch implements the relation linking QA task, i.e. $\theta(ReMatch) = RL$. Let $\rho(C_j)$ denote the performance of a QA component, then our first objective is to predict the likelihood of $\rho(C_j)$ for a given representation q^* of q, a task t_i, and an underlying knowledge graph λ. This is denoted as $Pr(\rho(C_j)|q^*, t_i, \lambda)$. In this work, we assume a single knowledge graph (i.e. DBpedia); thus, λ is considered a constant parameter that does not impact the likelihood leading to:

$$Pr(\rho(C_j)|q^*, t_i) = Pr(\rho(C_j)|q^*, t_i, \lambda) \tag{1}$$

Moreover, for each individual task t_i and question representation q^*, we predict the performance of all pertaining components. In other words, for a given task t_i, the set of components that can accomplish t_i is $\mathcal{C}^{t_i} = \{C_j, \ldots, C_k\}$. Thus, we factorise t_i as follows:

$$\forall C_j \in \mathcal{C}^{t_i}, [Pr(\rho(C_j)|q^*) = Pr(\rho(C_j)|q^*, t_i)] \tag{2}$$

Further, we assume that the given representation q^* is equal to the initial input representation q for all the QA components, i.e. $q^* = q$. Finally, the problem of finding the best performing component for accomplishing the task t_i for an input question q, denoted as $\gamma_q^{t_i}$, is formulated as follows:

$$\gamma_q^{t_i} = \arg \max_{C_j \in \mathcal{C}^{t_i}} \{Pr(\rho(C_j)|q)\} \tag{3}$$

Solution. Suppose we are given a set of NL questions $\mathcal{Q}$ with the detailed results of performance for each component per task. We can then model the prediction goal $Pr(\rho(C_j)|q, t_i)$ as a supervised learning problem on a training set, i.e. a set of questions $\mathcal{Q}$ and a set of labels $\mathcal{L}$ representing the performance of C_j for a question q and a task t_i. In other words, for each individual task t_i and component C_j, the purpose is to train a supervised model that predicts the performance of the given component C_j for a given question q and task t_i leveraging the training set. If $|\mathcal{T}| = n$ and each task is performed by m components, then $n \times m$ individual learning models have to be built up. Furthermore, since the input questions $q \in \mathcal{Q}$ have a textual representation, it is necessary to automatically extract suitable features, i.e. $\mathcal{F}(q) = (f_1, \ldots, f_r)$. The details of the feature extraction process are presented in Section 5.2.

3.2 Identifying Optimal QA Pipelines

The second problem deals with finding a best performing pipeline of QA components ψ_q^{goal}, for a question q and a set of QA tasks called *goal*. Formally, we define this optimisation problem as follows:

$$\psi_q^{goal} = \arg \max_{\eta \in \mathcal{E}(goal)} \{\Omega(\eta, q)\} \tag{4}$$

where $\mathcal{E}(goal)$ represents the set of pipelines of QA components that implement *goal* and $\Omega(\eta, q)$ corresponds to the estimated performance of the pipeline η on the question q.

Solution. We propose a greedy algorithm that relies on the *optimisation principle* that states that an optimal pipeline for a goal and a question q is composed of the best performing components that implement the tasks of the goal for q. Suppose that $\oplus$ denotes the composition of QA components, then an optimal pipeline ψ_q^{goal} is defined as follows:

$$\psi_q^{goal} := \oplus_{t_i \in goal}\{\gamma_q^{t_i}\} \quad (5)$$

The proposed greedy algorithm works in two steps: *QA Component Selection* and *QA Pipeline Generation*. During the first step of the algorithm, each task t_i in *goal* is considered in isolation to determine the best performing QA components that implement t_i for q, i.e. $\gamma_q^{t_i}$. For each t_i an ordered set of QA components is created based on the performance predicted by the supervised models that learned to solve the problem described in Equation 3. Figure 2 illustrates the QA component selection steps for the question q=*"What is the capital of Canada?"* and $goal = \{NED, RL, QB\}$. The algorithm creates an ordered set OS_{t_i} of QA components for each task t_i in *goal*. Components are ordered in each OS_{t_i} according to the values of the performance function $\rho(.)$ predicted by the supervised method trained for questions with the features $\mathcal{F}(q)$ and task t_i; in our example, $\mathcal{F}(q)$={(QuestionType:What), (AnswerType:String), (#words:6), (#DT:1), (#IN:1), (#WP:1), (#VBZ:1), (#NNP:1), (#NN:1)} indicates that q is a *WHAT* question whose answer is a *String*; further, q has six words and POS tags such as determiner, noun etc. Based on this information, the algorithm creates three ordered sets: OS_{NED}, OS_{RL}, and OS_{QB}. The order in OS_{NED} indicates that Dandelion[7], Tag Me, and DBpedia Spotlight are the top 3 best performing QA components for queries with the features $\mathcal{F}(q)$ in the QA task NED; similarly, for OS_{RL} and OS_{QB}.

In the second step, the algorithm follows the optimisation principle in Equation 5 and combines the top k_i best performing QA components of each ordered set. Values of k_i can be configured; however, we have empirically observed that for all studied types of questions and tasks, only the relation linking (RL) task requires considering the top 3 best performing QA components; for the rest of the tasks, the top 1 best performing QA component is sufficient to identify a best performing pipeline. Once the top k_i QA components have been selected for each ordered set, the algorithm constructs a QA pipeline and checks if the generated pipeline is able to produce a non-empty answer. If so, the generated QA pipeline is added to the algorithm output. In Equation 5, the algorithm finds that only the QA pipeline Dandelion, ReMatch, and SINA produces results; the other two pipelines fail because the QA components RNLIWOD[8] and Spot Property[9] are not able to perform the relation linking task of the question q=*"What is the capital of Canada?"*. The algorithm ends when the top k_i QA components have been combined and checked; the output is the union of the best performing QA pipelines that produce a non-empty answer.

[7]https://dandelion.eu/docs/api/datatxt/nex/getting-started/
[8]Component based on https://github.com/dice-group/NLIWOD.
[9]This component is the combination of the NLIWOD and RL components of [15].

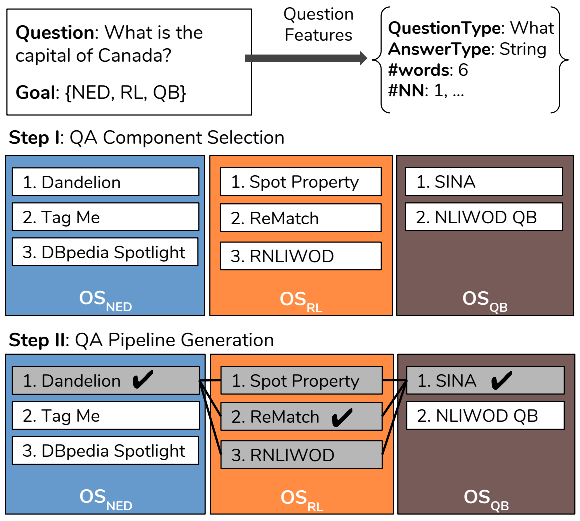

Figure 2: QA Optimisation Pipeline Algorithm. The algorithm performs two steps: First, QA components are considered in isolation; supervised methods are used to predict the top k best performing QA components per task and question features. Second, the QA Pipelines are generated from the best performing QA component of the tasks NED and QB, and the top 3 QA components of RL. The QA pipeline formed of Dandelion, ReMatch, and SINA successfully answers q.

4 FRANKENSTEIN FRAMEWORK

Frankenstein is a framework that implements the QA optimisation pipeline algorithm and generates the best performing QA pipelines based on the input question features and QA goal.

4.1 Frankenstein Architecture

Figure 3 depicts the Frankenstein architecture. Frankenstein receives, as input, a natural language question as well as a goal consisting of the QA tasks to be executed in the QA pipeline. The features of an input question are extracted by the *Feature Extractor*; afterwards the *QA Component Classifiers* predict best performing components per task for the given question; these components are passed to the *Pipeline Generator*, which generates best performing pipelines to be executed, eventually, by the *Pipeline Executor*. The Frankenstein architecture comprises the following modules:

Feature Extractor. This module extracts a set of features from a question. Features include question length, question and answer types, and POS tags. Features are discussed in Section 5.2.

QA Components. Frankenstein currently integrates 29 QA components implementing five QA tasks, namely Named Entity Recognition (NER), Named Entity Disambiguation (NED), Relation Linking (RL), Class Linking (CL), and Query Building (QB). To the best of our knowledge, only two reusable CL and QB components, and five reusable RL components are available, therefore the component distribution among tasks is uneven. In most of the cases NED, RL and QB components are necessary to generate the SPARQL query for a NL question. However, to correctly generate a SPARQL query for certain NL questions, it is sometimes necessary to also disambiguate classes against the ontology. For example, in the question

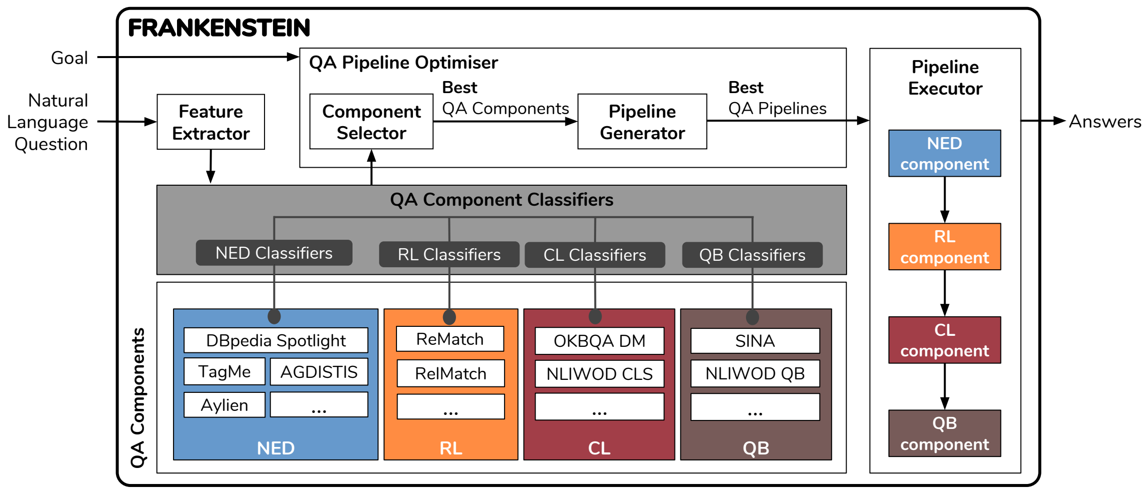

Figure 3: Frankenstein architecture comprising separate modules for question feature extraction, pipeline generation and optimisation, as well as pipeline execution.

"Which comic characters are painted by Bill Finger", *"comic characters"* needs to be mapped to `dbo:ComicsCharacter`[10]. Table 2 provides a list of QA components integrated in Frankenstein. The 11 NER components are used with AGDISTIS to disambiguate entities as AGDISTIS requires the question and spotted position of entities as input [32]. Henceforth, any reference to NER tool, will refer to its combination with AGDISTIS, and we have excluded individual performance analysis of NER components. However, other 7 NED components recognise and disambiguate the entities directly from the input question. Hence, Frankenstein has 18 NED, 5 RL, 2 CL, 2 QB components.

QA Component Classifiers. For each QA component, a separate Classifier is trained; it learns from a set of features of a question and predicts the performance of all pertaining components.

QA Pipeline optimiser. Pipeline optimisation is performed by two modules. The **Component Selector** selects the best performing components for accomplishing a given task based on the input features and the results of the QA Component Classifiers; the selected QA components are afterwards forwarded to the **Pipeline Generator** to dynamically generate the corresponding QA pipelines.

Pipeline Executor. This modules executes the generated pipelines for an input question in order to extract answers from the knowledge base (i.e. DBpedia in our case).

4.2 Implementation Details

The code for Frankenstein including all 29 integrated components and empirical study results can be found in our open source GitHub repository[11]. The integration of QA components within Frankenstein as a loosely coupled architecture is based on the following guiding principles: (a) Reusability (the framework should be available as open source), (b) Interoperability between QA components, (c) Flexibility (easy integration of components at any step of the QA pipeline), and (d) Isolation (components are independent of each other and provide exchangeable interfaces). We studied the implementations of *OKBQA* [15], *openQA* [17], *Qanary* [4, 25] and *QALL-ME* [9]; from these, to the best of our knowledge, only Qanary can fulfil the aforementioned guiding principles. Unlike a monolithic QA system the output of a component is not directly passed to the next component in the QA process and Qanary enhances the knowledge base after each step via the abstract level defined by the qa vocabulary. Therefore, components become independent of each other, and can easily be exchangeable just by configuration. The integration of the 29 new components with the *Qanary* methodology in Frankenstein is implemented in Java 8. Remaining Frankenstein modules are implemented in Python 3.4.

5 CORPUS CREATION

In this section, we describe the datasets used in our study and how we prepare the training dataset for our classification experiments. All experiments were executed on 10 virtual servers, each with 8 cores, 32 GB RAM and the Ubuntu 16.04.3 operating system. It took us 22 days to generate training data by executing questions of considered datasets for all 28 components, as some tools such as ReMatch[20] and RelationMatcher [27] took approximately 120 and 30 seconds, respectively, to process each question.

5.1 Description of Datasets

Throughout our experiment, we employed the Large-Scale Complex Question Answering Dataset[12] (LC-QuAD) [28] as well as the 5th edition of Question Answering over Linked Data Challenge[13] (QALD-5) dataset [30].

LC-QuAD has 5,000 questions expressed in natural language along with their formal representation (i.e. SPARQL query), which is executable on DBpedia. W.r.t. the state of the art, this is the largest available benchmark for the QA community over Linked Data. We

[10] http://dbpedia.org/ontology/ComicsCharacter
[11] https://github.com/WDAqua/Frankenstein

[12] http://lc-quad.sda.tech/
[13] https://qald.sebastianwalter.org/index.php?x=home&q=5

Table 2: 29 QA components integrated in Frankenstein: 8 QA components are not available as open source software, 25 provide a RESTful service API and 19 are accompanied by peer-reviewed publications.

Component/ Tool	QA Task	Year	Open Source	RESTful Service	Publi- cation
Entity Classifier [7]	NER	2013	✗	✓	✓
Stanford NLP [10]	NER	2005	✓	✓	✓
Ambiverse [11][i]	NER/NED	2014	✗	✓	✓
Babelfy [19][ii]	NER/NED	2014	✗	✓	✓
AGDISTIS [32]	NED	2014	✓	✓	✓
MeaningCloud[iii]	NER/NED	2016	✗	✓	✗
DBpedia Spotlight [18]	NER/NED	2011	✓	✓	✓
Tag Me API [8]	NER/NED	2012	✓	✓	✓
Aylien API[iv]	NER/NED	-	✗	✓	✗
TextRazor[v]	NER	-	✗	✓	✗
OntoText [16][vi]	NER/NED	-	✗	✓	✓
Dandelion[vii]	NER/NED	-	✗	✓	✗
RelationMatcher [27]	RL	2017	✓	✓	✓
ReMatch [20]	RL	2017	✓	✓	✓
RelMatch [15]	RL	2017	✓	✓	✓
RNLIWOD[viii]	RL	2016	✓	✗	✗
Spot Property [15][ix]	RL	2017	✓	✓	✓
OKBQA DM CLS[ix]	CL	2017	✓	✓	✓
NLIWOD CLS[viii]	CL	2016	✓	✗	✗
SINA [23]	QB	2013	✓	✗	✓
NLIWOD QB[viii]	QB	2016	✓	✗	✗

[i] https://developer.ambiverse.com/
[ii] https://github.com/dbpedia-spotlight/dbpedia-spotlight
[iii] https://www.meaningcloud.com/developer
[iv] http://docs.aylien.com/docs/introduction
[v] https://www.textrazor.com/docs/rest
[vi] http://docs.s4.ontotext.com/display/S4docs/REST+APIs
[vii] https://dandelion.eu/docs/api/datatxt/nex/getting-started/
[viii] Component is similar to Relation Linker of https://github.com/dice-group/NLIWOD.
[ix] Component is similar to Class Linker of http://repository.okbqa.org/components/7.

ran the entire set of SPARQL queries (on 2017-10-02) over the DBpedia endpoint[14], and found that only 3,252 of them returned an answer. Therefore, we rely on these 3,252 questions throughout our experiment.

QALD-5. Out of the QALD challenge series, we chose the 5th version (QALD-5) because it provides the largest number of questions (350 questions). However, during the experimental phase the remote Web service of the ReMatch component went down and we were only able to obtain proper results for 204 of the 350 questions. Therefore, we took these 204 questions into account to provide a fair and comparable setting (although, we obtained the results for all 350 questions for all other components).

5.2 Preparing Training Datasets

Since we have to build an individual classifier for each component in order to predict the performance of that component, it is required to prepare a single training dataset per component. The whole sample set within the training dataset was formed by using the NL questions included from the datasets described previously (from both QALD and LC-QuAD). In order to obtain an abstract and concrete representation of NL questions, we extracted major features enumerated below.

(1) *Question Length:* The length of a question w.r.t. the number of words has been introduced as a lexical feature by Blunsom et al. [3] in 2006. In our running example *"What is the capital of Canada?"*, this feature has the numeric value 6.
(2) *Question Word:* Huang et al. [13, 14] considered the question word ("wh-head word") as a separate lexical feature for question classification. If a specific question word is present in the question, we assign the value 1, and 0 to the rest of the question words. We adapted 7 Wh-words: "what", "which", "when", "where", "who", "how" and, "why". In *"What is the capital of Canada?"*, *"What"* is assigned the value 1, and all the other words are assigned 0.
(3) *Answer Type:* This feature set has three dimensions, namely "Boolean", "List/Resource", and "Number". These dimensions determine the category of the expected answer [22]. In our running example, we assign "List/Resource" for this dimension because the expected answer is the resource `dbr:Ottawa`.
(4) *POS Tags:* Part of Speech (POS) tags are considered an independent syntactical question feature that can affect the overall performance of a QA system [3]. We used the Stanford Parser[15] to identify the POS tags, where the number of occurrences is considered as a separate dimension in the question feature extraction.

We prepared two separate datasets from LC-QuAD and QALD. We adopted the methodology presented in [6] and [27] for the benchmark creation of the subsequent steps of the QA pipelines. Furthermore, the accuracy metrics are *micro F-Score (F-Score)* as a harmonic mean of micro precision and micro recall. Thus, the label set of the training datasets for a given component was set up by measuring the micro F-Score (F-Score) of every given question.

6 EVALUATING COMPONENT PERFORMANCE

The aim of this experiment is to evaluate the performance of components on the micro and macro levels and then train a classifier to accurately predict the performance of each component.

Metrics. *i*) `Answered Questions`: The number of questions for which the QA pipeline returns an answer. *ii*) `Micro Precision (MP)`: The ratio of correct answers vs. total number of answers retrieved by a component for a particular question. *iii*) `Precision (P)`: For a given component, the average of the Micro Precision over all questions. *iv*) `Micro Recall (MR)`: For each question, the number of correct answers retrieved by a component vs. gold standard answers for the given question. *v*) `Recall (R)`: For a given component, the average of Micro Recall over all questions. *vi*) `Micro F-Score (F-Score)`: For each question, the harmonic mean of MP and MR. *vii*) `Macro F-Score (F)`: For each component, harmonic mean of P and R.

[14] https://dbpedia.org/sparql

[15] http://nlp.stanford.edu:8080/parser/

6.1 Macro-level Performance of Components

In this experiment, we measured the performance of the reusable components from the QA community that are part of Frankenstein. We executed each component for each individual query from both LC-QuAD and QALD datasets. Then, for each dataset we calculated the macro accuracy per component and selected those representing highest macro performance. The performance results of the best components are shown in Table 3. For brevity, detailed results for each component are placed in our GitHub repository[16].

Table 3: The macro accuracy of the best components for each task on the QALD and LC-QuAD corpora.

QA Task	Dataset	Best Component	P	R	F
QB	LC-QuAD	NLIWOD QB	0.48	0.49	0.48
	QALD-5	NLIWOD QB	0.49	0.50	0.49
CL	LC-QuAD	OKBQA DM CLS	0.47	0.59	0.52
	QALD-5	OKBQA DM CLS	0.58	0.64	0.61
NED	LC-QuAD	Tag Me	0.69	0.66	0.67
	QALD-5	DBpedia Spotlight	0.67	0.75	0.71
RL	LC-QuAD	RNLIWOD	0.25	0.22	0.23
	QALD-5	ReMatch	0.54	0.74	0.62

Key Observation: Dependency on Quality of Input Question. From Table 3, it is clear that the performance considerably varies per dataset. This is because the quality of questions differs across datasets. Quality has various dimensions, such as complexity or expressiveness. For example, only 728 (22 %) questions of LC-QuAD are simple (i.e. with single relation, single entity), compared to 108 questions (53 %) of QALD. The average length of a question in LC-QuAD is 10.63, compared to 7.41 in QALD. Therefore, components that perform well for identifying an entity in a simple question may not perform equally well on LC-QuAD, which is also evident from Table 3 considering the NED task. The same holds for RL components. ReMatch, which is the clear winner on QALD, is outperformed by RNLIWOD on LC-QuAD. Hence, there is no overall best performing QA component for these two tasks, and the definition of the best performing QA component differs across datasets. However, this does holds true neither for CL components nor for QB components (note, these two tasks only have two components each), even though the Macro F-Score values on both datasets have significant differences.

6.2 Training the Classifiers

The aim of this part is to build up classifiers which efficiently predict the performance of a given component for a given question w.r.t. a particular task. As observed in the micro F-Score values of the components, these values are not continuous but usually discrete, e.g., 0.0, 0.33, 0.5, 0.66 or 1. Hence, we adopted five classification algorithms (treating it as a classification problem) namely 1) Support Vector Machines (SVM), 2) Gaussian Naive Bayes, 3) Decision Tree, 4) Random Forest, and 5) Logistic Regression. During the training phase, each classifier was tuned with a range of regularisation parameters to optimise the performance of the classifier on the available datasets. We used the cross-validation approach with 10 folds on the LC-QuAD dataset. Figure 4 illustrates the details of our experiment for training classifiers. Predominantly, Logistic Regression and Support Vector Machines expose higher accuracy as illustrated in Figure 4.

[16]https://github.com/WDAqua/Frankenstein

7 EVALUATING PIPELINE PERFORMANCE

In this experiment, we pursue the evaluation question *"Can an approach that dynamically combines different QA components taking the question type into account (such as Frankenstein) take advantage of the multitude of components available for specific tasks?"* To answer this question, we measure the Frankenstein performance on the (i) task level and (ii) pipeline level. Throughout our experiment, we adopt a component selector strategy as follows:

(1) *Baseline-LC-QuAD*: The best component for each task in terms of Macro F-Score on the LC-QuAD dataset (cf. Section 6.1).
(2) *Baseline-QALD*: The best component for each task in terms of Macro F-Score on the QALD dataset (cf. Section 6.1).
(3) *Frankenstein-Static*: The QA pipeline consisting of the best performing components for each task on the QALD dataset (cf. Section 6.1).
(4) *Frankenstein-Dynamic*: The QA pipeline consisting of the top performing components from the learning approach.
(5) *Frankenstein-Improved*: Similar to the dynamic Frankenstein pipeline with a different setting.

7.1 Task-level Experiment

Our major goal is to examine whether or not we can identify the N-best components for each QA task. Accordingly, we utilised the following metrics for evaluation *i*) *Total Questions*: the average number of questions in the underlying test dataset. *ii*) *Answerable*: the average number of questions for which at least one of the components has an F-Score greater than 0.5. *iii*) *Top N*: the average number of questions for which at least one of the Top N components selected by the Classifier has an F-Score greater than 0.5. Furthermore, we rely on a top-N approach for choosing the best performing component during our judgement.

Experiments on LC-QuAD. This experiment was run on the questions from the LC-QuAD dataset by applying a cross-validation approach. We compare the component selector approach in (i) learning-based manner – called Frankenstein, and (ii) Baseline-LC-QuAD manner – called Baseline. Table 4 shows the results of our experiment. Frankenstein's learning-based approach selects the top-N components with the highest predicted performance values for a given input question. Obviously, this approach outperforms the Baseline approach for the NED, RL, and QB tasks and equals the Baseline for CL task. When we select the top-2 or top-3 best performing components, Frankenstein's performance improves further.

Cross Training Experiment. The purpose of this experiment is similar to the previous experiment but in order to verify the credibility of our approach, we extended our dataset by including questions from QALD. In fact, questions from QALD are utilised as the test dataset. The results of this experiment are shown in Table 5. We observe that Frankenstein significantly outperforms the LC-QuAD

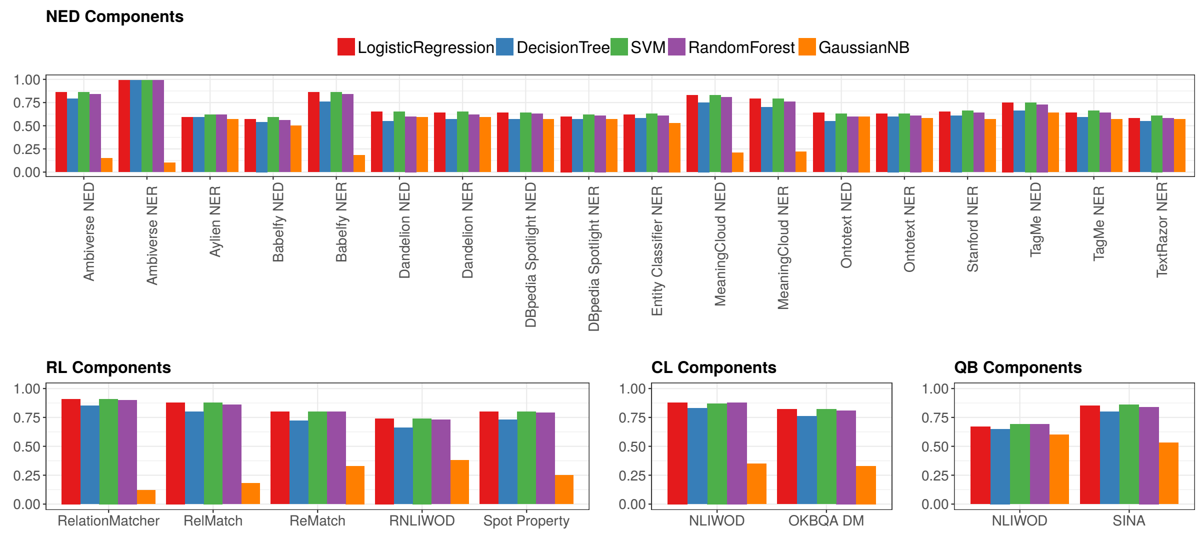

Figure 4: Comparison of Classifiers for all QA Components. Five Classifiers, namely Logistic Regression, Support Vector Machines, Decision Tree, Gaussian Naive Bayes, and Random Forest are compared with respect to accuracy.

Table 4: 10-fold Validation on LC-QuAD

QA Task	Total Questions	Answer-able	FRANKENSTEIN Top1	FRANKENSTEIN Top2	FRANKENSTEIN Top3	Baseline
QB	324.3	175.4	162.7	175.4	–	159.6
CL	324.3	76	68.1	76	–	68.2
NED	324.3	294.2	245.2	270.9	284.3	236.3
RL	324.3	153.1	90.3	118.9	134.4	84.2

Table 5: Performance comparison on task level using LC-QuAD as training and QALD as test dataset (204 questions)

QA Task	Answer-able	FRANKENSTEIN Top1	FRANKENSTEIN Top2	FRANKENSTEIN Top3	Baseline QALD	Baseline LC-QuAD
QB	119	91	119	–	102	102
CL	55	52	55	–	52	52
NED	168	132	153	163	144	109
RL	138	83	107	121	105	46

Baseline components for the NED (i.e. Tag Me) and RL (i.e. RNLIWOD) tasks while it achieves comparable results for the CL task.

7.2 Pipeline-level Experiment

In this experiment, we greedily arranged a pipeline by choosing the best performing components per task from three strategies. We use the same settings as cross training experiments by utilising QALD questions as test dataset. The first one is the FRANKENSTEIN-Static pipeline composed of Baseline components driven by QALD (i.e. DBpedia Spotlight for NED, ReMatch for RL, OKBQA DM for CL, and NLIWOD QB for QB). The other two strategies are the FRANKENSTEIN-Dynamic and FRANKENSTEIN-Improved pipelines composed by the learning-based component selector with top-1 setting (top-3 for RL of the improved strategy). The results of the comparison are demonstrated in Table 6. We conclude that the accuracy metrics for the dynamic pipeline are lower than for the static pipeline. (Note: As performance metrics for state-of-the-art QA systems on the same set of questions in QALD were not available, we excluded this comparison in the table.)

Table 6: Comparison with the Baseline Pipeline

FRANKENSTEIN-Pipeline	Answered Questions	P	R	Macro F-Score
Static	37	0.17	0.19	0.18
Dynamic	29	0.14	0.14	0.14
Improved	41	0.20	0.21	0.20

We noticed that the failure of the RL component significantly affects the total performance of both static or dynamic pipelines. Thus, we selected the top-3 components for the RL task to compensate for this deficiency. This setting yields the FRANKENSTEIN-Improved pipeline, where we ran three dynamically composed pipelines out of the 360 possible ones per question. Although this strategy is expected to affect the total accuracy negatively, this did not happen in practice; we even observed an increase in the overall precision, recall, and F-Score. Typically, the available RL and QB components have a full accomplishment or full failure for the input question. For example, considering the question *"What is the capital of Canada?"*, two of the top-3 selected RL components do not process the question. Hence, eventually, only one of the three pipelines returns a final answer. Thus, this simple modification in the setting significantly improves overall pipeline performance and the number of answered questions. With the static pipeline, the number of answered questions is fixed, however, with dynamic pipelines, the number of answered questions can be increased.

8 DISCUSSION

Despite the significant overall performance achieved by Frankenstein, we investigated erroneous cases in performance specifically w.r.t. classifiers. For instance, in our exemplary question *"What is the capital of Canada?"*, the top-1 component predicted by the Classifier for the RL task fails to map *"capital of"* to `dbo:capital`. Similarly, for the question (*"Who is the mayor of Berlin?"*), the predicted Dandelion NED component can not recognise and disambiguate *"Berlin"*. One of the reasons is related to the primary features extracted from questions. We plan to extend the feature set especially using the recent embedding models and also using different features per task as we have currently used the same features for all tasks. This can be done by associating features with component performance and calculating Cramér's V-coefficient for each feature and a component's ability to answer the given question [31]. One more extension is about switching to more sophisticated learning approaches like HMM, or deep learning approaches which require significantly larger datasets. Another observation is that the existing RL and QB components generally result in poor performance. The QB components need improvement in cases where previous tasks yield a low F-Score for a given question (i.e. returning more than one DBpedia URL as an answer). Hence, QB components should intuitively learn based on available URLs of entities and relations, and then form the right query. The current QB components fail to do so, which severely affected the overall performance of the complete QA pipelines (cf. Table 6). Further, RL and QB components need significant improvements in runtime efficiency and performance on complex questions. Thus, the QA community has to pay more attention to improve the components accomplishing these two tasks. To the best of our knowledge, currently very few independent components are available for these tasks (also for Class Linking) and the QA community can contribute building more independent components for these tasks. The Frankenstein architecture is not rigid and not restricted to the tasks considered in this paper. With the availability of more components performing new QA tasks (e.g., answer type prediction, syntactic parsing, query re-ranking, etc.), just by extending concepts of the `qa` vocabulary, new components can be added to the platform. Furthermore, in real world settings, a greedy approach may negatively affect the runtime of the pipeline. Hence, to provide a more efficient framework for creating QA pipelines, we plan to replace the greedy approach with concepts similar to web service composition [2], where we assign cost metrics (e.g., precision, runtime, or memory consumption) to select components using a pipeline optimiser in an automatic way.

9 RELATED WORK

Since 2011, 38 QA systems have participated in the eight editions of the Question Answering of Linked Data (QALD) challenge [33]. However, most participants started building their QA system from scratch and did not reuse the plethora of already available components for the QA subtasks that had to be addressed. Nevertheless, some early approaches have already tried to solve this problem; examples include openQA [17], a modular open-source framework for implementing QA systems from components having rigid but standardised Java interfaces. Also, QALL-ME [9] provides a SOA-based architecture skeleton for building components for multilingual QA systems over structured data. Moreover, OAQA [34] follows several architectural commitments to components to enable interchangeability. QANUS [21] also provides capabilities for the rapid development of information retrieval based QA systems. In 2014, Both et al. [5] presented a first semantic approach for integrating components, where modules that carry semantic self-descriptions in RDF are collected for a particular query until the search intent can be served. A more recent approach in this direction is Qanary [4], which generates QA systems from semantically described components wired together manually. Using the QA ontology [24] provided by Qanary, modules can be exchanged, e.g. various versions of NER tools, to benchmark various pipelines and choose the best performing one. In 2017, QA4ML [31] described a similar approach to Frankenstein, where a QA system is selected out of 6 QA systems based on the question type. To the best of our knowledge there is no question answering framework that enables the automatic and dynamic composition of QA components to generate optimised QA pipelines based on question features.

10 CONCLUSIONS AND FUTURE WORK

Frankenstein is the first framework of its kind for integrating all state-of-the-art QA components to build more powerful QA systems with collaborative efforts. It simplifies the integration of emerging components and is sensitive to the input question. The rationale was not to build a QA system from scratch but instead to reuse the currently existing 29 major components being available today to the QA community. Furthermore, our effort demonstrates the ability to integrate the components released by the research community in a single platform. Frankenstein's loosely coupled architecture enables easy integration of newer components in the framework and implements a model that learns from the features extracted from the input questions to direct the user's question to the best performing component per QA task. Also, Frankenstein's design is component agnostic; therefore, Frankenstein can easily be applied to new knowledge bases and domains. Thus, it is a framework for automatically producing intelligent QA pipelines. Our experimental study provides a holistic overview on the performance of state-of-the-art QA components for various QA tasks and pipelines. In addition, it measures and compares the performance of Frankenstein from several perspectives in multiple settings and demonstrates the beneficial characteristics of the approach.

We plan to extend our work in the following directions: (i) improving quality as well as quantity of extracted features, (ii) improving the learning algorithm, (iii) extending our training datasets, and (iv) including more emerging QA components. In conclusion, our component-oriented approach enables the research community to further improve the performance of state-of-the-art QA systems by adding new components to the ecosystem, or to extend the available data (i.e. gold standard) to adapt the training process.

ACKNOWLEDGMENT

This work has received funding from the EU H2020 R&I programme for the Marie Skłodowska-Curie action *WDAqua* (GA No 642795), Eurostars project *QAMEL* (E!9725), and EU H2020 R&I *HOBBIT* (GA 688227). We thank Yakun Li, Osmar Zaiane, and Anant Gupta for their useful suggestions.

REFERENCES

[1] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. 2007. DBpedia: A Nucleus for a Web of Open Data. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007.*

[2] Rainer Berbner, Michael Spahn, Nicolas Repp, Oliver Heckmann, and Ralf Steinmetz. 2006. Heuristics for QoS-aware Web Service Composition. In *2006 IEEE International Conference on Web Services (ICWS 2006), 18-22 September 2006, Chicago, Illinois, USA.* 72–82. https://doi.org/10.1109/ICWS.2006.69

[3] Phil Blunsom, Krystle Kocik, and James R. Curran. 2006. Question classification with log-linear models. In *SIGIR 2006: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, Washington, USA, August 6-11, 2006.* ACM, 615–616.

[4] Andreas Both, Dennis Diefenbach, Kuldeep Singh, Saeedeh Shekarpour, Didier Cherix, and Christoph Lange. 2016. Qanary - A Methodology for Vocabulary-Driven Open Question Answering Systems. In *The Semantic Web. Latest Advances and New Domains - 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 - June 2, 2016, Proceedings.* Springer, 625–641.

[5] Andreas Both, Axel-Cyrille Ngonga Ngomo, Ricardo Usbeck, Denis Lukovnikov, Christiane Lemke, and Maximilian Speicher. 2014. A service-oriented search framework for full text, geospatial and semantic search. In *Proceedings of the 10th International Conference on Semantic Systems, SEMANTICS 2014, Leipzig, Germany, September 4-5, 2014.* ACM, 65–72.

[6] Dennis Diefenbach, Kuldeep Singh, Andreas Both, Didier Cherix, Christoph Lange, and Sören Auer. 2017. The Qanary Ecosystem: Getting New Insights by Composing Question Answering Pipelines. In *Web Engineering - 17th International Conference, ICWE 2017, Rome, Italy, June 5-8, 2017, Proceedings.* Springer, 171–189.

[7] Milan Dojchinovski and Tomas Kliegr. 2013. Entityclassifier.eu: Real-time Classification of Entities in Text with Wikipedia. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD'13).* Springer-Verlag, 654–658.

[8] Paolo Ferragina and Ugo Scaiella. 2010. TAGME: on-the-fly annotation of short text fragments (by wikipedia entities). In *Proceedings of the 19th ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October 26-30, 2010.* 1625–1628. https://doi.org/10.1145/1871437.1871689

[9] Óscar Ferrández, Christian Spurk, Milen Kouylekov, Iustin Dornescu, Sergio Ferrández, Matteo Negri, Rubén Izquierdo, David Tomás, Constantin Orasan, Guenter Neumann, Bernardo Magnini, and José Luis Vicedo González. 2011. The QALL-ME Framework: A specifiable-domain multilingual Question Answering architecture. *J. Web Sem.* 9, 2 (2011), 137–145.

[10] Jenny Rose Finkel, Trond Grenager, and Christopher D. Manning. 2005. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In *ACL 2005, 43rd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, 25-30 June 2005, University of Michigan, USA.* The Association for Computer Linguistics, 363–370.

[11] Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen Fürstenau, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. 2011. Robust Disambiguation of Named Entities in Text. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John McIntyre Conference Centre, Edinburgh, UK, A meeting of SIGDAT, a Special Interest Group of the ACL.* 782–792. http://www.aclweb.org/anthology/D11-1072

[12] Konrad Höffner, Sebastian Walter, Edgard Marx, Ricardo Usbeck, Jens Lehmann, and Axel-Cyrille Ngonga Ngomo. 2017. Survey on challenges of Question Answering in the Semantic Web. *Semantic Web* (2017).

[13] Zhiheng Huang, Marcus Thint, and Asli Çelikyilmaz. 2009. Investigation of Question Classifier in Question Answering. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, EMNLP 2009, 6-7 August 2009, Singapore, A meeting of SIGDAT, a Special Interest Group of the ACL.*

[14] Zhiheng Huang, Marcus Thint, and Zengchang Qin. 2008. Question Classification using Head Words and their Hypernyms. In *2008 Conference on Empirical Methods in Natural Language Processing, EMNLP 2008, Proceedings of the Conference, 25-27 October 2008, Honolulu, Hawaii, USA, A meeting of SIGDAT, a Special Interest Group of the ACL.* ACL, 927–936.

[15] Jin-Dong Kim, Christina Unger, Axel-Cyrille Ngonga Ngomo, André Freitas, Young-gyun Hahm, Jiseong Kim, Sangha Nam, Gyu-Hyun Choi, Jeong-uk Kim, Ricardo Usbeck, et al. 2017. OKBQA Framework for collaboration on developing natural language question answering systems. (2017). http://sigir2017.okbqa.org/papers/OKBQA2017_paper_9.pdf

[16] Atanas Kiryakov, Borislav Popov, Ivan Terziev, Dimitar Manov, and Damyan Ognyanoff. 2004. Semantic annotation, indexing, and retrieval. *J. Web Sem.* (2004), 49–79. https://doi.org/10.1016/j.websem.2004.07.005

[17] Edgard Marx, Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Konrad Höffner, Jens Lehmann, and Sören Auer. 2014. Towards an open question answering architecture. In *Proceedings of the 10th International Conference on Semantic Systems, SEMANTICS 2014, Leipzig, Germany, September 4-5, 2014.* ACM, 57–60. https://doi.org/10.1145/2660517.2660519

[18] 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, I-SEMANTICS 2011, Graz, Austria, September 7-9, 2011.* ACM, 1–8. https://doi.org/10.1145/2063518.2063519

[19] Andrea Moro, Alessandro Raganato, and Roberto Navigli. 2014. Entity Linking meets Word Sense Disambiguation: a Unified Approach. *TACL* 2 (2014), 231–244. https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/291

[20] Isaiah Onando Mulang, Kuldeep Singh, and Fabrizio Orlandi. 2017. Matching Natural Language Relations to Knowledge Graph Properties for Question Answering. In *Semantics 2017.*

[21] Jun-Ping Ng and Min-Yen Kan. 2015. QANUS: An Open-source Question-Answering Platform. *CoRR* abs/1501.00311 (2015). http://arxiv.org/abs/1501.00311

[22] Muhammad Saleem, Samaneh Nazari Dastjerdi, Ricardo Usbeck, and Axel-Cyrille Ngonga Ngomo. 2017. Question Answering Over Linked Data: What is Difficult to Answer? What Affects the F scores?. In *Joint Proceedings of BLINK2017: 2nd International Workshop on Benchmarking Linked Data and NLIWoD3: Natural Language Interfaces for the Web of Data co-located with 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 21st - to - 22nd, 2017.* CEUR-WS.org. http://ceur-ws.org/Vol-1932/paper-02.pdf

[23] Saeedeh Shekarpour, Edgard Marx, Axel-Cyrille Ngonga Ngomo, and Sören Auer. 2015. SINA: Semantic interpretation of user queries for question answering on interlinked data. *J. Web Sem.* 30 (2015), 39–51. https://doi.org/10.1016/j.websem.2014.06.002

[24] Kuldeep Singh, Andreas Both, Dennis Diefenbach, and Saeedeh Shekarpour. 2016. Towards a Message-Driven Vocabulary for Promoting the Interoperability of Question Answering Systems. In *Tenth IEEE International Conference on Semantic Computing, ICSC 2016, Laguna Hills, CA, USA, February 4-6, 2016.* IEEE Computer Society, 386–389. https://doi.org/10.1109/ICSC.2016.59

[25] Kuldeep Singh, Andreas Both, Dennis Diefenbach, Saeedeh Shekarpour, Didier Cherix, and Christoph Lange. 2016. Qanary - The Fast Track to Creating a Question Answering System with Linked Data Technology. In *The Semantic Web - ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 - June 2, 2016, Revised Selected Papers.* 183–188. https://doi.org/10.1007/978-3-319-47602-5_36

[26] Kuldeep Singh, Ioanna Lytra, Maria-Esther Vidal, Dharmen Punjani, Harsh Thakkar, Christoph Lange, and Sören Auer. 2017. QAestro - Semantic-Based Composition of Question Answering Pipelines. In *Database and Expert Systems Applications - 28th International Conference, DEXA 2017, Lyon, France, August 28-31, 2017, Proceedings, Part I.* Springer, 19–34. https://doi.org/10.1007/978-3-319-64468-4_2

[27] Kuldeep Singh, Isaiah Onando Mulang, Ioanna Lytra, Mohamad Yaser Jaradeh, Ahmad Sakor, Maria-Esther Vidal, Christoph Lange, and Sören Auer. 2017. Capturing Knowledge in Semantically-typed Relational Patterns to Enhance Relation Linking. In *Proceedings of the Knowledge Capture Conference, K-CAP 2017, Austin, TX, USA, December 4-6, 2017.* 31:1–31:8. https://doi.org/10.1145/3148011.3148031

[28] Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. 2017. LC-QuAD: A Corpus for Complex Question Answering over Knowledge Graphs. In *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II.* Springer, 210–218. https://doi.org/10.1007/978-3-319-68204-4_22

[29] Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. 2012. Template-based question answering over RDF data. In *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012.* ACM, 639–648. https://doi.org/10.1145/2187836.2187923

[30] Christina Unger, Corina Forascu, Vanessa López, Axel-Cyrille Ngonga Ngomo, Elena Cabrio, Philipp Cimiano, and Sebastian Walter. 2015. Question Answering over Linked Data (QALD-5). In *Working Notes of CLEF 2015 - Conference and Labs of the Evaluation forum, Toulouse, France, September 8-11, 2015.* CEUR-WS.org. http://ceur-ws.org/Vol-1391/173-CR.pdf

[31] Ricardo Usbeck, Michael Hoffmann, Michael Röder, Jens Lehmann, and Axel-Cyrille Ngonga Ngomo. 2017. Using Multi-Label Classification for Improved Question Answering. *CoRR (submitted)* (2017). https://arxiv.org/abs/1710.08634

[32] Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Michael Röder, Daniel Gerber, Sandro Athaide Coelho, Sören Auer, and Andreas Both. 2014. AGDISTIS - Graph-Based Disambiguation of Named Entities Using Linked Data. In *The Semantic Web - ISWC 2014.* Springer, 457–471. https://doi.org/10.1007/978-3-319-11964-9_29

[33] Ricardo Usbeck, Michael Röder, Michael Hoffmann, Felix Conrads, Jonathan Huthmann, Axel-Cyrille Ngonga Ngomo, Christian Demmler, and Christina Unger. [n. d.]. Benchmarking Question Answering Systems. *Semantic Web Journal (to be published)* ([n. d.]). http://www.semantic-web-journal.net/content/benchmarking-question-answering-systems

[34] Zi Yang, Elmer Garduño, Yan Fang, Avner Maiberg, Collin McCormack, and Eric Nyberg. 2013. Building optimal information systems automatically: configuration space exploration for biomedical information systems. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, USA.* ACM, 1421–1430.