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

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,
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 \textsc{Frankenstein}, a framework able to dynamically select QA components in order to exploit the properties of the components to optimise the F-Score. \textsc{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 \textsc{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 \textsc{Frankenstein} using two renowned benchmarks from the Question Answering over Linked Data Challenge\footnote{https://qald.sebastianwalter.org/index.php?x=home&q=5} (QALD) and the Large-Scale Complex Question Answering Dataset\footnote{http://lc-quad.sda.tech/} (LC-QuAD). We observe that \textsc{Frankenstein} is able to combine QA components to produce optimised QA pipelines outperforming the static Baseline pipeline.

In summary, we provide the following contributions:

- \textsc{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 \textsc{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 \textsc{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 \textsc{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 \textsc{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 \textsc{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\footnote{Component is based on https://github.com/dice-group/NLIWOD and [29].}, 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\footnote{The prefix \texttt{dbr} is bound to http://dbpedia.org/resource/}. 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. \texttt{dbr:Canada}. 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 \texttt{dbo:capital}\footnote{The prefix \texttt{dbo} is bound to http://dbpedia.org/ontology/}. Finally, the QB component generates a formal query (e.g. expressed in SPARQL), which retrieves all answers from the corresponding knowledge graph (i.e. \texttt{SELECT \texttt{?c} \{\texttt{dbr:Canada dbo:capital \texttt{?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.

\footnote{http://docs.aylien.com/docs/introduction}
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 Q1: “What is the capital of Canada?” but it fails to answer Question Q4: “Which river does the Brooklyn Bridge cross?” This is caused by the fact that the RL component RelMatch in Pipeline P1 does not correctly map the relation dbo:crosses in Q4 for the input keyword "cross", while RelMatch maps this relation correctly. Although the overall precision of RelMatch on QALD-5 is higher than that of RelMatch, for Q4, the performance of RelMatch is higher. Similarly, for Question Q2, “Did Socrates influence Aristotle?” Pipeline P2 delivers the desired answer, while it fails to answer the similar question Q3 “Did Tesla win a nobel prize in physics?”. Although questions Q2 and Q3 have a similar structure (i.e. Boolean answer type), DBpedia Spotlight NED succeeds for Q2, but on Q3 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.

### 3.1 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:capital1 of dbr:Canada?”. The entire set of QA components is denoted by $C = \{ C_1, C_2, \ldots, C_m \}$. Each component $C_j$ solves one single QA task; $\vartheta(C_j)$ corresponds to the QA task $t_j$ in $\mathcal{T}$ implemented by $C_j$. For example, ReMatch implements the relation linking QA task, i.e. $\vartheta(ReMatch) = RL$. 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 a question $t_j$ and an underlying knowledge graph $\lambda$. This is denoted as $Pr(\rho(C_j)|q^*, t_j, \lambda)$. In this work, we assume a single knowledge graph (i.e. DBpedia); thus, $\lambda$ is considered a constant parameter that does not impact the likelihood leading to:

$$ Pr(\rho(C_j)|q^*, t_j) = Pr(\rho(C_j)|q^*, t_j, \lambda) $$

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

$$ \forall C_j \in C^{t_j}, [Pr(\rho(C_j)|q^*) = Pr(\rho(C_j)|q^+, t_j)] $$

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 $t_j$ for an input question $q$, denoted as $y^*_q$, is formulated as follows:

$$ y^*_q = \arg \max_{C_j \in C^{t_j}} \{ Pr(\rho(C_j)|q) \} $$

Solution. Suppose we are given a set of NL questions $Q$ with the detailed results of performance for each component per task. We can then model the prediction goal $Pr(\rho(C_j)|q, t_j)$ as a supervised learning problem on a training set, i.e. a set of questions $Q$ and a set of labels $\mathcal{L}$ representing the performance of $C_j$ for a question $q$ and a task $t_j$. In other words, for each individual task $t_j$ 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 $t_j$ 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 $q \in Q$ 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 $\psi^*_q$, for a question $q$ and a set of QA tasks called goal. Formally, we define this optimisation problem as follows:

$$ \psi^*_q = \arg \max_{\eta \in \mathcal{E}(goal)} \{ \Omega(\eta, q) \} $$

where $\Omega(\eta, q)$ is an objective function that evaluates the performance of a given QA pipeline $\psi^*_q$ for a given question $q$.
where $E(\text{goal})$ represents the set of pipelines of QA components that implement goal and $\Omega(\eta, q)$ corresponds to the estimated performance of the pipeline $\eta$ 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 $\psi_{\text{goal}}^q$ is defined as follows:

$$
\psi_{\text{goal}}^q := \oplus_{i \in \text{goal}} \{ Y_i \}
$$

(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 $i$ in goal is considered in isolation to determine the best performing QA components that implement $t_i$ for $q$, i.e. $Y_i^q$. 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 $\mathcal{OS}_t_i$ of QA components for each task $t_i$ in goal. Components are ordered in each $\mathcal{OS}_t_i$ 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, $\mathcal{F}(q) = \{(\text{QuestionType:What}), (\text{AnswerType:String}), (#\text{words}:6), (#\text{DT}:1), (#\text{IN}:1), (#\text{WP}:1), (#\text{VBZ}:1), (#\text{NNP}:1), (#\text{NN}:1)\}$ 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: $\mathcal{OS}_{\text{NED}}, \mathcal{OS}_{\text{RL}},$ and $\mathcal{OS}_{\text{QB}}$. The order in $\mathcal{OS}_{\text{NED}}$ indicates that Dandelion $\checkmark$, 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 $\mathcal{OS}_{\text{RL}}$, and $\mathcal{OS}_{\text{QB}}$.

In the second step, the algorithm follows the optimisation principle in Equation 5 and combines the top $k$ 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 $k$ 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 $k$ QA components have been combined and checked; the output is the union of the best performing QA pipelines that produce a non-empty answer.

$^8$hhttps://dandelion.eu/docs/api/datatxt/nex/getting-started/
$^9$Component based on https://github.com/dice-group/RNLIWOD.

---

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...
“Which comic characters are painted by Bill Finger”, “comic characters” needs to be mapped to \texttt{dbo:ComicsCharacter}\textsuperscript{10}. Table 2 provides a list of QA components integrated in \textsc{Frankenstein}. The 11 NER components are used with AGDISTIS to disambiguate entities as AGDISTIS requires the question and spotted position of entities as input \cite{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, \textsc{Frankenstein} has 18 NED, 5 RL, 2 CL, 2 QB components.

\textbf{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.

\textbf{QA Pipeline Optimiser}. Pipeline optimisation is performed by two modules. The \textbf{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 \textbf{Pipeline Generator} to dynamically generate the corresponding QA pipelines.

\textbf{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 \textsc{Frankenstein} including all 29 integrated components and empirical study results can be found in our open source GitHub repository\textsuperscript{11}. The integration of QA components within \textsc{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 \cite{15}, openQA \cite{17}, Qanary \cite{4, 25} and QALL-ME \cite{9}; from these, to the best of our knowledge, only Qanary can fulfill 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 \textsc{Frankenstein} is implemented in Java 8. Remaining \textsc{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\cite{20} and RelationMatcher \cite{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\textsuperscript{12} (LC-QuAD) \cite{28} as well as the 3\textsuperscript{rd} edition of Question Answering over Linked Data Challenge\textsuperscript{13} (QALD-5) dataset \cite{30}.

\textbf{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

\textsuperscript{10}http://dbpedia.org/ontology/ComicsCharacter

\textsuperscript{11}https://github.com/WDAqua/Frankenstein

\textsuperscript{12}http://lc-quad.sda.tech/

\textsuperscript{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.

<table>
<thead>
<tr>
<th>Component/Tool</th>
<th>QA Task</th>
<th>Year</th>
<th>Open Source</th>
<th>RESTful Service</th>
<th>Publication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Babelfy [19]</td>
<td>NER/NED</td>
<td>2014</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AGDISTIS [32]</td>
<td>NED</td>
<td>2014</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MeaningCloud [i]</td>
<td>NER/NED</td>
<td>2016</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DBpedia Spotlight [18]</td>
<td>NER/NED</td>
<td>2011</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Tag Me API [8]</td>
<td>NER/NED</td>
<td>2012</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Aylien API [iv]</td>
<td>NER/NED</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TextRazor [v]</td>
<td>NER</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>OntoText [vi]</td>
<td>NER/NED</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Dandelion [vii]</td>
<td>NER/NED</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>RelationMatcher [27]</td>
<td>RL</td>
<td>2017</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ReMatch [20]</td>
<td>RL</td>
<td>2017</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RNLWOD [viii]</td>
<td>RL</td>
<td>2016</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OKBQA DM CLS [ix]</td>
<td>CL</td>
<td>2017</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLIWOD CLS [viii]</td>
<td>CL</td>
<td>2016</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SINA [23]</td>
<td>QB</td>
<td>2013</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NLIWOD QB [viii]</td>
<td>QB</td>
<td>2016</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

1https://developer.ambiverse.com/
2https://github.com/dbpedia-spotlight/dbpedia-spotlight
3https://www.meaningcloud.com/developer
4http://docs.aylien.com/docs/introduction
5https://www.textrazor.com/docs/rest
6http://docs.s4.ontotext.com/display/S4docs/REST+APIs
7https://dandelion.eu/docs/api/datatxt/nex/getting-started/
8Component is similar to Class Linker of https://github.com/dice-group/ NLIWOD.
9Component is similar to Class Linker of http://repository.okbqa.org/ components/7.

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.
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 Frankenstei. 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 repository16.

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

<table>
<thead>
<tr>
<th>QA Task</th>
<th>Dataset</th>
<th>Best Component</th>
<th>P</th>
<th>R</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>QB</td>
<td>LC-QuAD</td>
<td>NLIWOD QB</td>
<td>0.48</td>
<td>0.49</td>
<td>0.48</td>
</tr>
<tr>
<td></td>
<td>QALD-5</td>
<td>NLIWOD QB</td>
<td>0.49</td>
<td>0.50</td>
<td>0.49</td>
</tr>
<tr>
<td>CL</td>
<td>LC-QuAD</td>
<td>OKBQA DM CLS</td>
<td>0.47</td>
<td>0.59</td>
<td>0.52</td>
</tr>
<tr>
<td></td>
<td>QALD-5</td>
<td>OKBQA DM CLS</td>
<td>0.58</td>
<td>0.64</td>
<td>0.61</td>
</tr>
<tr>
<td>NED</td>
<td>LC-QuAD</td>
<td>Tag Me</td>
<td>0.69</td>
<td>0.66</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>QALD-5</td>
<td>Dbpedia Spotlight</td>
<td>0.67</td>
<td>0.75</td>
<td>0.71</td>
</tr>
<tr>
<td>RL</td>
<td>LC-QuAD</td>
<td>RNLJWOD</td>
<td>0.25</td>
<td>0.22</td>
<td>0.23</td>
</tr>
<tr>
<td></td>
<td>QALD-5</td>
<td>ReMatch</td>
<td>0.54</td>
<td>0.74</td>
<td>0.62</td>
</tr>
</tbody>
</table>

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 RNLJWOD 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.

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 Frankenstei) take advantage of the multitude of components available for specific tasks?” To answer this question, we measure the Frankenstei 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. Frankenstei-Static: The QA pipeline consisting of the best performing components for each task on the QALD dataset (cf. Section 6.1).
4. Frankenstei-Dynamic: The QA pipeline consisting of the top performing components from the learning approach.
5. Frankenstei-Improved: Similar to the dynamic Frankenstei 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 Frankenstei, and (ii) Baseline-LC-QuAD manner – called Baseline. Table 4 shows the results of our experiment. Frankenstei’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, Frankenstei’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 Frankenstei significantly outperforms the LC-QuAD

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

<table>
<thead>
<tr>
<th>QA Task</th>
<th>Total Questions</th>
<th>Answerable</th>
<th>Frankenstein Top1</th>
<th>Frankenstein Top2</th>
<th>Frankenstein Top3</th>
<th>Baseline</th>
<th>Baseline</th>
<th>Baseline</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>QB</td>
<td>324.3</td>
<td>175.4</td>
<td>162.7</td>
<td>175.4</td>
<td>-</td>
<td>159.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CL</td>
<td>324.3</td>
<td>76</td>
<td>68.1</td>
<td>76</td>
<td>-</td>
<td>68.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NED</td>
<td>324.3</td>
<td>294.2</td>
<td>245.2</td>
<td>270.9</td>
<td>284.3</td>
<td>236.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RL</td>
<td>324.3</td>
<td>153.1</td>
<td>90.3</td>
<td>118.9</td>
<td>134.4</td>
<td>84.2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>QA Task</th>
<th>Answerable</th>
<th>Frankenstein Top1</th>
<th>Frankenstein Top2</th>
<th>Frankenstein Top3</th>
<th>Baseline QALD</th>
<th>Baseline LC-QuAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>QB</td>
<td>119</td>
<td>91</td>
<td>119</td>
<td>-</td>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td>CL</td>
<td>55</td>
<td>55</td>
<td>55</td>
<td>-</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>NED</td>
<td>168</td>
<td>132</td>
<td>153</td>
<td>163</td>
<td>144</td>
<td>109</td>
</tr>
<tr>
<td>RL</td>
<td>138</td>
<td>83</td>
<td>107</td>
<td>121</td>
<td>105</td>
<td>46</td>
</tr>
</tbody>
</table>

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 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.)

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 QA4ML [31], 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 Sklodowska-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.