



Grant Agreement: 101016453

## D5.6 – Deep Learning Toolbox – Version 1

Deliverable D5.6		
Authors and institution	INFAI, AIRC	
Date	M15	
Dissemination level		
PU	Public, fully open, e.g., web	
CO	Confidential, restricted under conditions set out in Model Grant Agreement	CO
CI	Classified, information as referred to in Commission Decision 2001/844/EC	

Document change history			
Date	Version	Authors	Description
10.02.2022	V0.1	Mohnish Dubey (INFAI)	Table of Contents
07.04.2022	V0.2	Mohnish Dubey (INFAI)	Introduction, specifications
25.04.2022	V0.3	Kristiina Jokinen (AIRC)	Rasa
26.04.2022	V0.4	Kristiina Jokinen (AIRC)	Edits
28.04.2022	V0.5	Mohnish Dubey (INFAI)	Language Model tasks, References
29.04.2022	V0.6	Rainer Wieching (USI)	Finalization

## Disclaimer

---

This document contains material, which is the copyright of certain e-VITA consortium parties and may not be reproduced or copied without permission.

The information contained in this document is the proprietary CO information of the e-VITA consortium and may not be disclosed except in accordance with the consortium agreement.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the e-VITA consortium as a whole, nor a certain party of the e-VITA consortium warrant that the information contained in this document is capable of use, or that use of the information is free from risk and accept no liability for loss or damage suffered by any person using this information.

**e-VITA – European-Japanese Virtual Coach for Smart Ageing**

**e-VITA (EU PROJECT NUMBER 101016453)**

Work-package 5 – Trustworthy AI, Data Analytics & NLP

**D5.6 Deep Learning toolbox – Version 1**

Editors: Mohnish Dubey (INFAI) and Kristiina Jokinen (AIRC)

Work-package leader: AIRC, INFAI

### **Copyright notice**

2021-2023 Participants in project e-VITA

## Executive Summary

This document describes the deep learning techniques used in the dialogue components of the Evita prototype. It briefly states the purpose of the methodology of deep learning and language models and also presents a short overview of the technology used in the subtasks of the NLP pipeline. Since the project adopted to use the Rasa Open-Source Conversational AI system, much of the toolbox is already provided within the platform itself.

# Table of Contents

Executive Summary ..... 4

Lists of Figures and Tables ..... 6

Acronyms and Abbreviations ..... 6

1 Introduction ..... 7

2 Language Model as NLP toolbox ..... 7

    2.1 Language model ..... 7

    2.2 T5 Language Model ..... 7

    2.3 NLP Tasks and Multitasking..... 8

        2.3.1 Named Entity Recognition ..... 8

        2.3.2 Entity Linking..... 8

        2.3.3 Relation Linking..... 9

    2.4 Single Model Multitasking..... 9

We have developed a single model to perform the above-mentioned tasks and other NLP tasks. We retrained the pretrained T5 language model which has the capability of perform multitasking from single model. .... 9

3 RASA as a Toolbox for Dialogue System ..... 9

    3.1 Tokenizers..... 10

    3.2 Featurizers ..... 10

    3.3 Intent Classifiers ..... 10

    3.4 Entity Extractors ..... 10

    3.5 NLU Pipeline ..... 11

4 Question Answering over Wikipedia ..... 12

    4.1 Overview..... 12

    4.2 Data ..... 13

    4.3 Indexing ..... 13

    4.4 Deployment Components Summary..... 13

5 Conclusion..... 14

6 References ..... 15

## Lists of Figures and Tables

Figure 1 Example showcasing T5 Language Model .....	7
Figure 2 NER example with Spacy code snippet.....	8
Figure 3 NLU pipeline for RASA.....	10
Figure 4 Haystack architecture reference .....	12

## Acronyms and Abbreviations

Acronym/Abbreviation	Explanation
Conversational AI	Interactive system using AI techniques
Knowledge graph	Representation of knowledge in graph format with nodes and connecting arcs
Rasa	Open-source Conversational AI Framework
Knowledge base	Structured knowledge that encodes information necessary for the system's interaction operation
KG, Knowledge graph	Particular type of data representation consisting of nodes and connecting edges.

## 1 Introduction

The Deep Learning Toolbox is to provide a set of tools and techniques that can be used in the annotation, segmentation, analysis, and processing of language data. In e-VITA project, the tools are related to the Rasa Open-Source Conversational AI platform which offers a wide variety of state-of-the-art tools to experiment with different pipelines and parameters. This deliverable gives as short overview of the available tools.

## 2 Language Model as NLP toolbox

### 2.1 Language model

The use of various statistical and probabilistic techniques to predict the probability of a given sequence of words appearing in a phrase is known as language modeling (LM)[1]. To establish a foundation for their word predictions, language models evaluate large amounts of text data. They're employed in natural language processing (NLP) applications, especially those that output text. Machine translation and question answering are two examples of these applications.

Language models analyze text data to calculate word probability. They use an algorithm to interpret the data, which establishes rules for context in natural language. The model then uses these principles to accurately predict or construct new sentences in language problems. The model basically learns the basic properties and qualities of language and then applies them to new phrases.

There are numerous probabilistic ways to modeling language, which vary based on the language model's aim. Technically, the different varieties differ in terms of the amount of text data they examine and the math they utilize to do so.

### 2.2 T5 Language Model

T5, or Text-to-Text Transfer Transformer [3], is a Transformer[2] based architecture that uses a text-to-text approach. Every task – including translation, question answering, and classification – is cast as feeding the model text as input and training it to generate some target text. This allows for the use of the same model, loss function, hyperparameters, etc. across our diverse set of tasks.

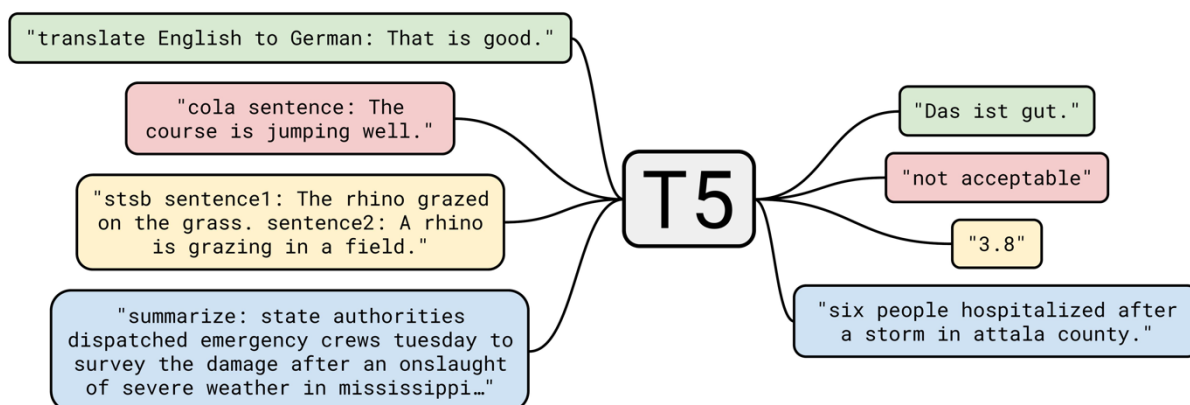


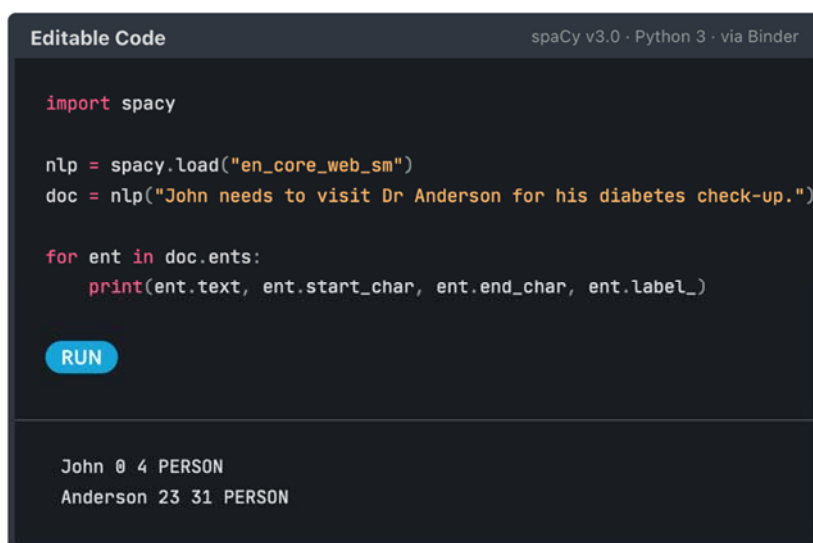
Figure 1 Example showcasing T5 Language Model

## 2.3 NLP Tasks and Multitasking

### 2.3.1 Named Entity Recognition

The term "named entity" refers to a group of elements crucial to comprehending text. Parts of speech are the source of some familiar entities (like nouns, verbs, adjectives, etc.). Nouns, in particular, are critical to comprehend the finer points of a phrase. Humans pay greater attention to nouns than other parts of speech when dealing with named entity recognition (NER) [4].

For example, we have a sentence "John needs to visit Dr Anderson for his diabetes check-up.", an NER machine learning (ML) model might detect the word "John" and "Dr Anderson" in a text and classify it as a "Person". One can follow the same example at spacy.io [5]



```

Editable Code spaCy v3.0 · Python 3 · via Binder

import spacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("John needs to visit Dr Anderson for his diabetes check-up.")

for ent in doc.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)

RUN

John 0 4 PERSON
Anderson 23 31 PERSON

```

Figure 2 NER example with Spacy code snippet

### 2.3.2 Entity Linking

Named Entity Recognition is the function of assigning a unique identity-class (such as a person, location, things or events) to the entities mentioned in the text. Entity Linking is the process to spot a mentioned entity in the text (question in our use-case) and connect it to the entity representing the span-entity in the knowledge graph. With Entity Linking, we are able to use a large amount of information on the real-world individuals and their relationships in the publicly available knowledge graphs, such as Wikipedia, DBpedia or Wikidata. Entity Linking connects text (unstructured) data to a structured database, providing a base to use the knowledge stored in structured data. This gives more semantic information to understand the texts better.

For example, in the sentence "Manhattan is the birthplace for Robert Downey Jr", by Entity Linking we can link "Robert Downey Jr" to <https://www.wikidata.org/wiki/Q165219> and "Manhattan" to <https://www.wikidata.org/wiki/Q11299>. These linkings shows that the sentence is regarding movie actor.

Entity Linking [6] is generally a three-step procedure involving Entity Span(Mention) Detection, Candidate Generation and Candidate Disambiguation. The first task is to spot the span of the entity in the sentence which is also called named entity recognition. Once the span of the entity is detected,



the EL system applies string similarity to generate a set of Entity Candidates for the span. The final task is to re-rank the candidate list such that the correct candidate is the top of the list.

### 2.3.3 Relation Linking

A knowledge graph has facts stored in terms of a triple format that is  $\langle Subject, Predicate, Object \rangle$  ( $\langle S, P, O \rangle$ ). The process of annotating given natural language text with KG predicate is termed Relation Linking. The generic approach to Relation Linking [7] is often similar to Entity Linking. First, find the key phrase in the sentence; second, generate a candidate list based on the previous step; third, re-rank the list to get the correct annotation.

In many cases, this approach is not ideal as text spans might not be sufficient for correct candidate space, as the expressivity of a predicate could vary more than that of a named entity.

For better understanding, let's look few examples:

Consider we have triple about diabetes and its precaution as:

$\langle \text{subject} = \text{"diabetes"}, \text{predicate} = \text{"precautions"}, \text{object} = \text{"Exercise regularly"} \rangle$

#### Example 1:

text: What are the precautions for diabetes?

relation span: "precautions".

correct predicate: "precautions"

#### Example 2:

text: How to safeguard me from diabetes?

relation span: "safeguard".

correct predicate: "precautions"

#### Example 3:

text: What things should I follow not to have diabetes?

relation span: *unclear*

correct predicate: "precautions"

## 2.4 Single Model Multitasking

We have developed a single model to perform the above-mentioned tasks and other NLP tasks. We retrained the pretrained T5 language model which has the capability of perform multitasking from single model.

We used Huggingface and SimpleTransformer library for the retaining. We used several exiting datasets KGQA and dialogue dataset as mentioned in Dubey et al [7].

## 3 RASA as a Toolbox for Dialogue System

Rasa Open-source Conversational AI platform is a versatile platform which supports various NLP tools and also allows the user to use their own tailor-made tools as necessary [8]. The Rasa platform organises the tools roughly into two groups: those for natural language processing and those for dialogue management. The former set is realised in the so called NLU pipeline, while the latter one uses machine-learning techniques, notably transformers to implement state tracking and dialogue action decisions. In this deliverable, we follow closely the description in [Intents & Entities: Understanding the Rasa NLU Pipeline | The Rasa Blog | Rasa](#)

The NLU pipeline in Rasa looks as in Figure 3 (from the above-mentioned blog). It consists of four basic components which are: 1) Tokenizers, 2) Featurizers, 3) Intent Classifiers, and 4) Entity Extractors.

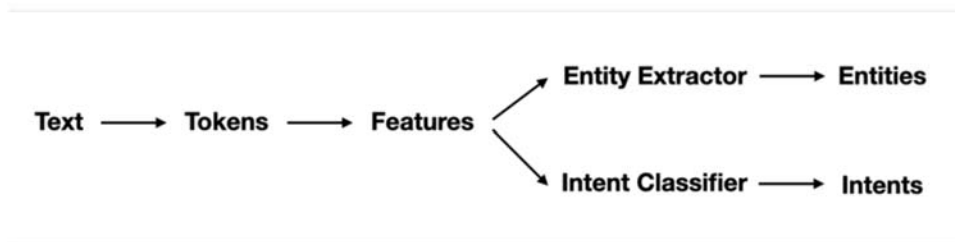


Figure 3 NLU pipeline for RASA

### 3.1 Tokenizers

Tokenizers take the text input and segment it into suitable chunks. In natural dialogue processing the suitable chunks are usually words, and the output is a list of words. The Tokenizer also provides separate tokens for punctuation marks. The default tokenizer for English is the [WhiteSpaceTokenizer](#) but for languages other than English it is possible to use other ones.

### 3.2 Featurizers

Featurizers generate numeric features for the machine learning models. Two different types of feature sets can be used: sparse feature vectors, which represent the words as one token among the other zero-valued vector slots, and dense features, which contain many pretrained embeddings. Rasa uses [CountVectorizer](#) to generate sparse features, and it also uses [LexicalSyntacticFeaturizer](#) that generates window-based features useful for entity recognition. In order to include part of speech features, the [LexicalSyntacticFeaturizer](#) can be combined with spaCy ([spaCy · Industrial-strength Natural Language Processing in Python](#)), a comprehensive state-of-the-art NLP library and NLP pipeline system for language processing in python.

Dense Features are commonly generated from [SpaCyFeaturizers](#). Also, the machine learning libraries from [Huggingface](#) via [LanguageModelFeaturizers](#) can be used. More details of the dense features can be obtained from the Rasa documentation.

### 3.3 Intent Classifiers

The important part of dialogue modelling is to correctly classify the user's intent. For intent classification, Rasa uses its own DIET model (Dual Intent and Entity Transformer) which handles both intent classification and entity extraction. For the description of how DIET works, we refer to the relevant papers and Rasa documentation [Intents & Entities: Understanding the Rasa NLU Pipeline | The Rasa Blog | Rasa](#)

### 3.4 Entity Extractors

The DIET model learns how to extract entities simultaneously with the intent recognition, this is not optimal and not recommended for every type of entity. For instance, structured patterns such as phone numbers, can be extracted directly using [RegexEntityExtractor](#) and Named Entities with SpaCy language models: [ner\\_spacy](#). Usually there are more than one type of entity extractor in the Rasa pipeline. Entities can also be mapped to synonyms.

### 3.5 NLU Pipeline

The NLU pipeline defines the tools used in the Rasa language processing. The pipeline is configured in the ‘config.yml’ file. There is a default pipeline that is used unless the user defines specific modules to be used in the dialogue model.

The pipeline used in the Prototype1 is given below:

```
pipeline:
- name: WhitespaceTokenizer
- name: RegexFeaturizer
- name: LexicalSyntacticFeaturizer
- name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: char_wb
  min_ngram: 1
  max_ngram: 4
- name: DIETClassifier
  epochs: 100
  constrain_similarities: true
- name: EntitySynonymMapper
- name: ResponseSelector
  epochs: 100
  constrain_similarities: true
- name: FallbackClassifier
  threshold: 0.3
  ambiguity_threshold: 0.1
```

For the dialogue policy, i.e., determining how the system will respond once it has recognized a particular user intent, Rasa uses different types of policies, combining machine-learning policies and rule-based policies.

The trademark of Rasa is the TED (Transformer Embedded Dialogue) policy. This consists of several transformer encoders which are shared for both next-action prediction and entity recognition. See more in the paper Vlasov, Mosig, Nichol (2019) Dialogue Transformers [\[1910.00486\] Dialogue Transformers \(arxiv.org\)](#)

The UnexpectTEDIntentPolicy is an auxiliary policy that shares the same model architecture as TEDPolicy. It allows the system to react to unlikely user turns, see more in Rasa documentation.

The Memoization Policy is based on stories (dialogue scripts) which are saved in the stories.yml -file. If the current conversation matches with one of the stories defined, the next action is predicted based on the story with confidence 1.0, otherwise “none” is predicted with confidence 0.0.

The Rule policy takes care of the conversation which are fixed, such as greetings, which always follow the same pattern. It makes predictions based on any rules in the training data. However, Rule Policy should be used with care as it fixes the dialogue logic and many rules can easily interfere with each other. Moreover, the main point of conversational AI is to use machine-learning techniques.

The following snippet shows a prototype1 policies that determine the next action.

```

policies:
- name: MemoizationPolicy
- name: RulePolicy
- name: UnexpectTEDIntentPolicy
  max_history: 5
  epochs: 100
- name: TEDPolicy
  max_history: 5
  epochs: 100
  constrain_similarities: true

```

## 4 Question Answering over Wikipedia

### 4.1 Overview

“Haystack is an **open-source framework** for building **search systems** that work intelligently over large document collections. Recent advances in NLP have enabled the application of question answering, retrieval and summarization to real world settings and Haystack is designed to be the bridge between research and industry.”

The main components and core concepts of a general Haystack [9] setup can be depicted as follows:

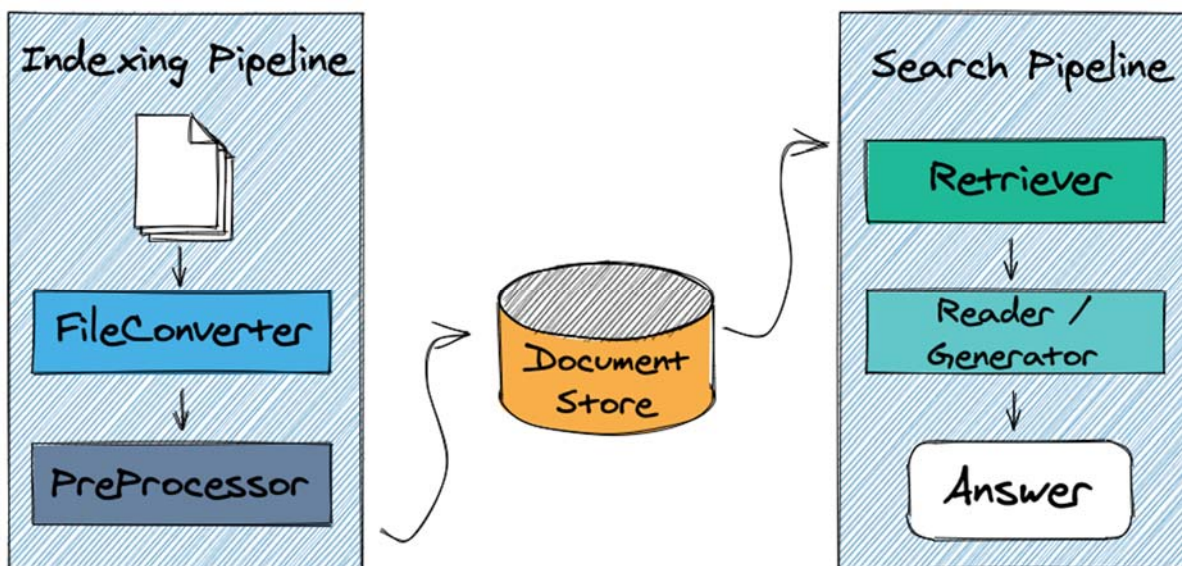


Figure 4 Haystack architecture reference

To build a QA system on Wikipedia we mainly follow this structure and make use of different components provided by the Haystack framework. We’re deploying an **extractive QA system**, i.e. given a corpus of articles and a question in natural language, the goal is to return a text phrase in a particular passage contained in one or more articles.

## 4.2 Data

Wikipedia dumps are usually provided in the form of wikitext source and metadata embedded in XML. While indeed possible to work on this directly while indexing the corpus for Haystack, we decided to convert it to a simpler JSON format in advance by using the 3rd party project Wikiextractor. Most importantly, this tool allows us to strip the Wikipedia Markdown and provide a light-weight JSON format with just title and plain text of each article.

## 4.3 Indexing

Given the converted Wikipedia corpus, we have to first (pre-)process each Wikipedia article. This does not only comprise some more cleansing and normalization tasks like removal of white spaces, but also splitting each article into smaller pieces to optimize retrieval. Following the suggestions in the Haystack guide, we decided to split by words respecting sentence boundaries and used a split length of max. 100 words. Clearly, this leads to multiple documents for the same article, but makes indexing as well as retrieval more efficient.

The next step of the indexing phase is two-fold: first we're computing the embeddings for each text using one of the provided language models of Haystack and then we're putting the data into some document store. We choose the vector database Milvus for storing the embeddings and use SQLite to store the corresponding documents.

### Search

For setting up the search we make use of a so-called **pipeline**. In our case, we decided to use an extractive QA mechanism which comprises the task of searching through a large collection of documents for a span of text that answers a question. The *ExtractiveQAPipeline* combines the *Retriever* and the *Reader* such that:

- The *Retriever* combs through a database and returns only the documents that it deems to be the most relevant to the query.
- The *Reader* accepts the documents returned by the *Retriever* and selects a text span as the answer to the query.

The **output** of the pipeline is a Python dictionary with a list of *Answer* objects stored under the `answers` key. These provide additional information such as the context from which the answer was extracted and the model's confidence in the accuracy of the extracted answer.

## 4.4 Deployment Components Summary

We use the following components and language models:

Haystack framework:

- v1.1.0

Language Models:

- facebook/dpr-question\_encoder-single-nq-base
- facebook/dpr-ctx\_encoder-single-nq-base
- deepset/roberta-base-squad2

Document store:

- Milvus v1.1.1 for storage and retrieval of the passage embedding vectors
- SQLite for storage of the passages

PreProcessor:

- clean\_empty\_lines=True,
- clean\_whitespace=True,
- clean\_header\_footer=True,
- split\_by="word",
- split\_length=100,
- split\_respect\_sentence\_boundary=True,
- split\_overlap=0,

Retriever:

- DensePassageRetriever
  - query\_embedding\_model="facebook/dpr-question\_encoder-single-nq-base"
  - passage\_embedding\_model="facebook/dpr-ctx\_encoder-single-nq-base"

Reader:

- FARMReader
  - model="deepset/roberta-base-squad2"
  - use\_gpu=True

## 5 Conclusion and Outlook

This document briefly described the tools used for the e-VITA prototype 1. The tools consist of state-of-the-art machine-learning tools for the annotation, segmentation, analysis, and processing of language data, as well as to determine the next action in the dialogue context. There will be an update of this deliverable in M27 after the re-design phase.

## 6 References

- [1] Li, Yang, and Tao Yang. "Word embedding for understanding natural language: a survey." Guide to big data applications. Springer, Cham, 2018. 83-104.
- [2] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017)
- [3] Raffel, Colin, et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer." Journal of Machine Learning Research 21 (2020): 1-67.
- [4] Nadeau, David, and Satoshi Sekine. "A survey of named entity recognition and classification." Lingvisticae Investigationes 30.1 (2007): 3-26.
- [5] <https://spacy.io/usage/linguistic-features>
- [6] Dubey, Mohnish, et al. "EARL: joint entity and relation linking for question answering over knowledge graphs." International Semantic Web Conference. Springer, Cham, 2018.
- [7] Dubey, Mohnish. "Towards Complex Question Answering over Knowledge Graphs." (2021).
- [8] Bocklisch, Tom, et al. "Rasa: Open source language understanding and dialogue management." arXiv preprint arXiv:1712.05181 (2017).
- [9] <https://github.com/deepset-ai/haystack>