# ANGLEr: A General Extensible Data Model and (Web-based) Tool for Text Analysis and Exploration

Slavko Žitnik, Timotej Knez

February 27, 2023

**Abstract**

Natural language processing is used for solving a wide variety of problems. Some scholars and interest groups working with language resources are not well versed in programming, so there is a need for a good graphical framework that allows users to quickly design and test natural language processing pipelines without the need for programming. The existing frameworks do not satisfy all the requirements for such a tool. We therefore propose a new framework that provides a simple way for its users to build language processing pipelines. It also allows a simple programming language agnostic way for adding new modules, which will help the adoption by natural language processing developers and researchers. The main parts of the proposed framework consist of (a) a pluggable Docker-based architecture, (b) general data model, and (c) APIs description along with the graphical user interface. The proposed design is being used for implementation of a new natural language processing framework, that we will call ANGLEr, based on our initial explorations [8].

## 1 Introduction

Within the project "Development of Slovene in a Digital Environment" we planned to create blueprints for a general (Web-based) tool for text analysis. The deliverable focus is on extensibility and a general data model that would support a number of text processing tasks. The concrete goal is as follows: "Izdelali bomo načrt in zasnovo (spletnega) orodja za izvajanje analiz nad besedili po vzoru orodij kot so GATE, UIMA oz. idejno zastavljen model nutIE. Poudarek orodja bo predvsem na razširljivosti in podatkovnem modelu za predstavitev podatkov. Namen orodja bo zagotoviti lažjo uporabo razvitih orodij za netehnične uporabnike. Le ti bodo lahko uvozili lastne podatke oz. korpus, zaporedno izvajali posamezne analize procesiranja naravnega jezika in evalvirali ter predstavili rezultate na grafičen način."

The area of natural language processing (NLP) is used in a wide variety of applications. NLP tools are used by people from various backgrounds. In order for the users outside the computer science area to use the available tools effectively, we have to provide them with an intuitive graphical user interface that allows a simplified construction of text processing pipelines. We also need to enable the NLP researchers to showcase their projects by including them in our framework with very little additional effort. In the past, a number of frameworks for simplifying NLP workflows were designed. However, most of them fall short either in the expandability or the ease of use. In our work we identify the key components of such framework and improve upon the existing frameworks by fixing the identified flaws.

In this report we propose:

1. a pluggable Docker-based architecture along with the APIs descriptions,
2. a general data model, and
3. extensible graphical user interface.

## 2 Existing frameworks review

A number of frameworks for natural language processing already exist out there. In this section we review the most prospective frameworks in order to determine their strengths and weaknesses. This helps us to define a list of features that could be improved by introducing a new framework and justify the need for it. The comparison is summarized in Table 1.

## 2.1 General Architecture for Text Engineering (GATE)

One of the best frameworks for text processing using a graphical interface to our knowledge is GATE [2] (Figure 1). It was designed to feature a unified data model that supports a wide variety of language processing tools. It also features a graphical user interface. While the GATE framework provides a graphical way for pipeline construction, its interface has not been updated "in over 10 years". The program is thus easy to install but seems to be difficult to use for new users. The tool allows for creation of custom additional plugins that need to be written using Java codebase. It already defines initial processors but no recent tools are packaged as GATE plugins.

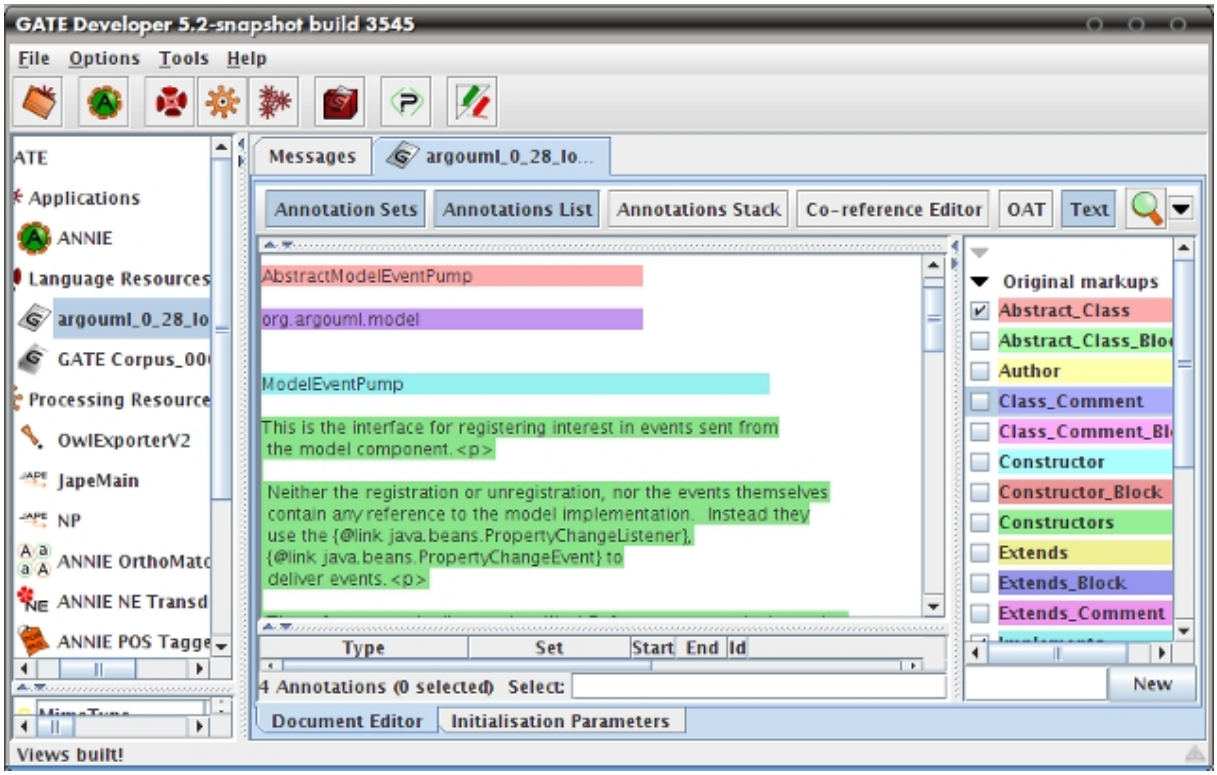


Figure 1: GATE tool user interface.

The tool is actively developed by The University of Sheffield with OntoText as a major contributor. They also offer Cloud service with GATE Mimir search engine in their pricing tier. Their public list of plugins contains some available NLP tools[1].

## 2.2 Unstructured Information Management Applications (UIMA)

UIMA [4] (Figure 2) is a framework for extracting information from unstructured documents like text, images, emails and so on. Apache UIMA is an Apache-licensed open source implementation of the UIMA specification that is being developed concurrently by a technical committee within OASIS (a standards organization). It features a data model that combines extracted information from all previous components. The framework provides some graphical tools for running analysis on the source documents. The graphical tools, however, are not combined into a single application and do not provide an easy way for creating processing pipelines. The primary way of using UIMA still requires the user to edit XML descriptor documents, which limits usability for new users and slows down the development. New components for the framework can be written in the Java or the C++ programming languages.

UIMA enables applications to be decomposed into components, for example "language identification" → "language specific segmentation" → "sentence boundary detection" → "entity detection (person/place names etc.)." Each component implements interfaces defined by the framework and provides self-describing metadata via XML descriptor files. The framework manages these components and the data flow between them. Components are written in Java or C++ and the communication flows between components is designed for efficient mapping between these languages. The components can be defined as Web services and replicated over a cluster of nodes.

[1] https://cloud.gate.ac.uk/shopfront (Accessed: July 18, 2022).

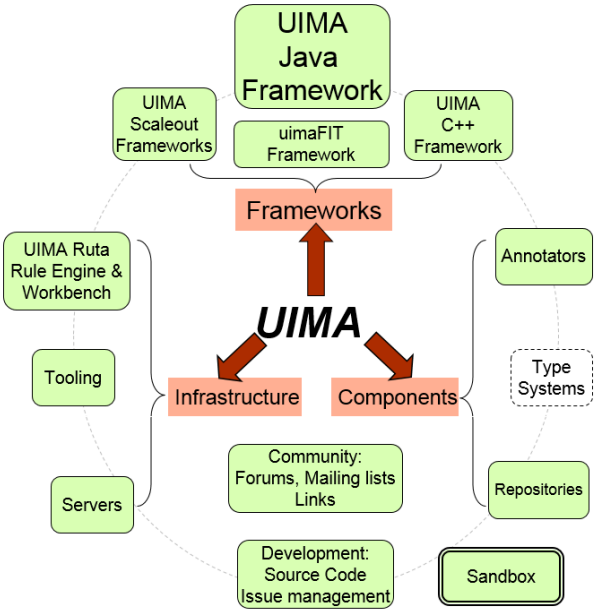


Figure 2: UIMA projects landscape.

The framework support configuring and running pipelines of Annotator components. These components do the actual work of analyzing the unstructured information. Users can write their own annotators, or configure and use pre-existing annotators. Some annotators are available as part of this project; others are contained in various repositories on the internet. GitHub.com lists over 900 repositories that have dependencies on the UIMA Java SDK core. Recent architecture supports REST communication between components.

It is also important to mention that IBM's Watson was built on their implementation of UIMA architecture.

## 2.3 Orange - Data Mining Fruitful and Fun

Orange Data Mining tool [3] (Figure 3), developed at the University of Ljubljana, is one of the best and multiple times awarded tools for building data mining pipelines. It is actively developed and regularly updated. In the future releases a cloud-based version of the tool is expected to be released.

The main feature of the orange framework is a great user interface that is friendly even for non technical users. Orange provides a variety of widgets designed for machine learning tasks and data visualisation. Apart from that, it is possible to use Orange directly using a Python library which makes it very accessible to technical users.

The framework was designed primarily to work with relational data for classic machine learning. It supports two extensions for processing natural language which allow us to use the already existing machine learning tools on text documents. The largest drawback when using Orange for text processing is that the tabular representation, used for representing data, is not well suited for representing information needed for text processing. The two extensions tackle this problem in different ways:

**Orange text mining addon.** (Figure 4) The addon is available to download and enable via Orange user interface[2]. The addon implements *Corpus* and *Documents* type but it seems that tabular format is still main representation, so that data can be compatible with existing widgets. Its biggest limitation is that it only works with features on a document level, but there are options to transform the data.

## 2.4 Textable

Textable[3] (Figure 4) seems to be a fork or Orange with additional implemented specifics for text processing. It is also available as a regular Orange addon. The tool supports text files as text fields and it has separate support for JSONs and URLs. Version 3 update also supports different text segments.

The tool was designed and implemented by LangTech Sarl on behalf of the department of language and information sciences (SLI) at the University of Lausanne. Current it is not regularly updated.

[2] https://orange3-text.readthedocs.io/en/latest (Accessed: July 18, 2022).
[3] http://textable.io (Accessed: July 18, 2022).

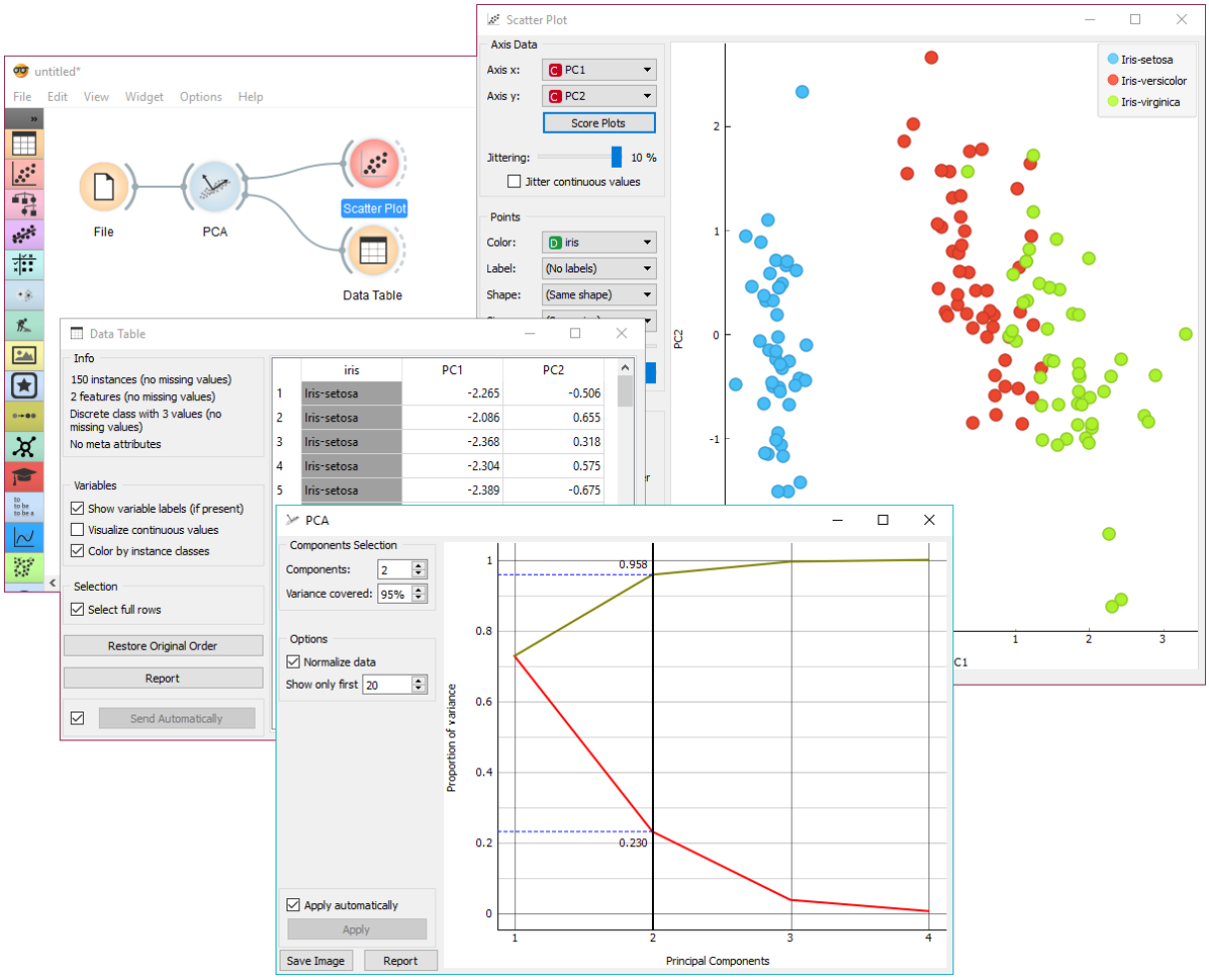


Figure 3: An example of the Orange user interface.

From the analysis perspective it offers quite some basic processors (e.g., basic text analysis, concordances, collocations, document-term matrices, lemmatization, POS tagging, ), connectors to 3rd-party libraries (e.g., NLTK, Pattern, GenSim), and import functionality (e.g., HTML, CSV, XML files).

## 2.5 Natural language processing programming libraries

Multiple librariest that support text processing have been developed. Some of them are still very active or have recently appeared. These libraries are commonly used by experts in the field of natural language processing. They are not well suited for use by other people that might also be interested in language processing. One of such libraries is called OpenNLP[4]. It supports the development of natural language applications in the Java programming language. There are also multiple Python libraries, such as NLTK [1] or Gobbli [11]. Gobbli is a library trying to simplify the use of deep learning in text processing. Another popular library for natural language processing in Python is Stanza [12]. Stanza is being developed at Stanford university and its main goal is to support a large number of languages. Recent very popular library and tools containing many pretrained model is Huggingface[5]. Popular library in industry is Spacy[6]

In order to use any of these libraries, the user is required to have some programming knowledge. Writing a program is also a lot slower than constructing a pipeline through a graphical interface. Because of that, we believe that a good graphical interface is one of the key features of such frameworks.

## 2.6 Side-by-side comparison

In Table 1 we compare reviewed frameworks along selected dimensions. As we can see, NLP libraries do not fit most of the criteria, so we will not take them into account. All other frameworks support some level

[4] https://opennlp.apache.org (Accessed: July 18, 2022).
[5] https://huggingface.co (Accessed: July 18, 2022).
[6] https://spacy.io (Accessed: July 18, 2022).


Figure 4: Orange text mining addon (left) and Textable (right) widgets.

| Framework | Graphical UI | Unified data model | Plugin language |
|---|---|---|---|
| GATE | Yes (Native) | Yes (too general) | Java |
| UIMA | Yes (Limited) | Yes | Java or C++ |
| Orange /w addons | Yes (Native) | Diverse (transformations available) | Python |
| NLP Libraries | No | Diverse | Java |
| **ANGLEr** | **Yes (Web)** | **Yes (versioned)** | **Arbitrary (Docker)** |

Table 1: Comparison of different natural language processing frameworks.

of graphical user interface and also programmatic access to the tools. Orange seems to provide the best user-friendly and intuitive user interface of all. All three frameworks are available as native applications while we propose Web-based framework that would require no installation. GATE and UIMA on the other hand provide cloud-hosted Web services. All three frameworks are also "locked" into a specific programming language. We propose a completely decoupled architecture, interconnected only via the data model, so anyone can provide their functionalities in their own language. The same also holds for new plugins development.

One of important aspects is also that the project offering main framework is alive. It seems that Orange is the only one with a vibrant community, good legacy and therefore has best options to survive longer.

# 3 Towards a common and general data model for natural language processing tools

The data model should be as simple as possible. It seems that GATE and UIMA provide an overwhelming data model that might be hard to understand. While GATE's might be to general, UIMA's is standard-based. On the other hand, Orange provides a simple model but lacks text-based representation as

"first-class citizen," although it allows for simple transformations between data representations. Based on the review we will try to propose as simple and general data model as possible.

Apart from the data models offered by GATE, UIMA and Orange, we will also include Stanza's data model (a NLP library that provides some common structure) and NIF standard. There exist also other proposals such as CDA+GrAF [10] or International Standard for a Linguistic Annotation Framework [7][7] for which we found only a few usages and we do not consider including in the review.

## 3.1 NLP Interchange Format (NIF) data model

NIF [5] offered an RDF/OWL-based format that allows to combine and chain several NLP tools in a flexible, light-weight way.

The core of NIF consists of a vocabulary, which can represent Strings as RDF resources. A special URI Design is used to pinpoint annotations to a part of a document. These URIs can then be used to attach arbitrary annotations to the respective character sequence. Based on these URIs, annotations can be interchanged between different NLP tools.

The project does not seem actively maintained[8] and not many works have applied this data model (Figure 5) to their tools. The model merely focuses on strings and offers a structure of a fixed set of representations, such as title, paragraph, phrase, sentence and word.

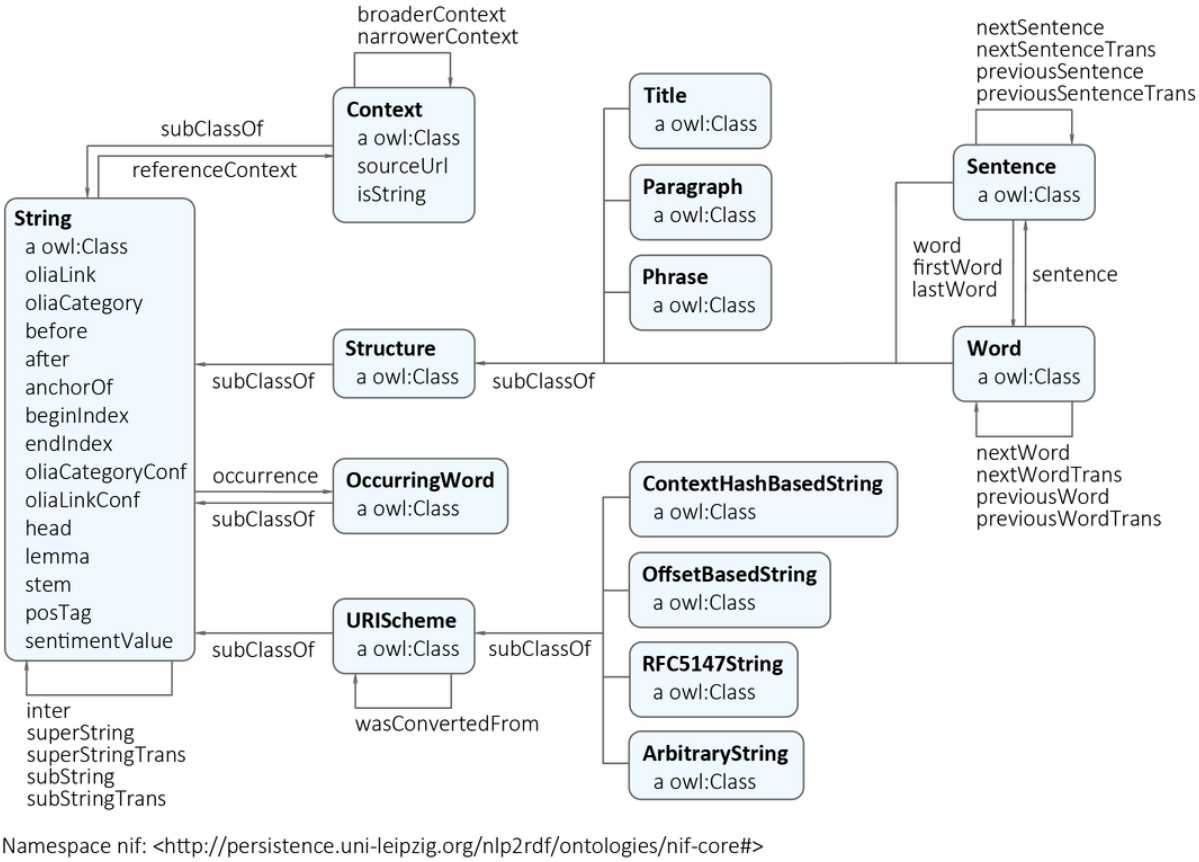


Figure 5: NIF 2.0 ontology schema representation.

We do not know exactly why the model might be obsolete, but there might be multiple reasons: (a) community did not adopt the model, (b) the model is to general and does not support specific tools, or (c) researchers do not want to use RDF overhead. Although we believe Semantic Web functionalities would be an added value to the NLP annotations, it should be used when needed and not forced.

## 3.2 GATE data model

GATE implements multiple document formats in a Corpora Java class[9] package. Examples of documents are TikaFormat, JsonDocuments, EmailDocuments, UIMADocument or XMLDocument. Corpus is rep-

[7] https://www.cs.vassar.edu/~ide/papers/ISO+24612-2012.pdf (Accessed: July 18, 2022).
[8] https://github.com/NLP2RDF (Accessed: July 18, 2022).
[9] https://jenkins.gate.ac.uk/job/gate-core/javadoc/index.html (Accessed: July 19, 2022).

resented as a set of such documents. By default GATE transforms these documents into an internal annotation-based GATE format. Documents are basically (a) content, (b) annotations, and (c) features.

Document defines an interface and each document must support generation of outputs to a GATE XML. Each document can be represented as a list of annotations, where each annotation is defined by start node, end node and type. Node is defined by id and offset.

Based on the above, a user can create its own schema or re-use an existing one[10]. In Table 2 we show extracted annotations in a JSON format. As we can see, the format consists of different annotation types - e.g., location, organization, ... We believe these are the same type of annotations and should have been structured similar/hierarchicaly. Each of the token types we see contain a map for definition of features.


```
{
    "tokens": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "category": "",
            "kind": "",
            "length": 0,
            "orth": "",
            "stirng": ""
        }
    }],
    "location": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "locType": "",
            "matches": [{
                "id": 0
            }],
            "rule": ""
        }
    }],
    "lookup": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "majorType": "",
            "minorType": ""
        }
    }],
    "organization": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "matches": [{
                "id": 0
            }],
            "orgType": "",
            "rule": ""
        }
    }],
    "person": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "matches": [{
                "id": 0
            }],
            "firstName": "",
            "lastName": "",
            "gender": "",
            "rule": ""
        }
    }],
    "split": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "data": {
            "kind": ""
        }
    }],
    "sentence": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "id": 0
        }
    }],
    "address": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "kind": "",
            "rule": "",
            "string": ""
        }
    }],
    "measurement": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "dimension": "",
            "unit": "",
            "value": 0,
            "normalized": 0
        }
    }],
    "coreference": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "matches": [{
                "id": 0
            }],
            "string": "",
            "rule": ""
        }
    }],
    "sentenceSentiment": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "emotion": "",
            "polarity": "",
            "sarcasm": "",
            "sentiment_string": ""
        }
    }],
    "similarityOfDocuments": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "results": 0.0
        }
    }],
    "textSummarization": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "score": 0.0
        }
    }],
    "WordLem": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "originalWord": "",
            "lemmatizedWord": ""
        }
    }],
    "wordStem": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "originalWord": "",
            "stemmedWord": ""
        }
    }],
    "answering": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "score": 0.0,
            "result": ""
        }
    }],
    "spaceToken": [{
        "start": 0,
        "end": 0,
        "id": 0,
        "type": "",
        "data": {
            "length": 0
        }
    }],
    "textClassification": [{
        "data": {
            "matches": [],
            "classification": "",
            "confidence": 0.0
        }
    }]
}
```


Table 2: GATE schema (annotation types) represented in a JSON format. Schema was extracted by Nik Hrovat [6].

## 3.3 UIMA data model

UIMA uses a standardized schema, prepared by OASIS standardization group. The last published version is from 2008 and in Table 3 we show extracted annotations in a JSON format. OASIS published version is UIMA 1.0, dated in 2009[11].

The specification also defines a communication protocol and processes for an NLP system. Annotations are basically still objects that relate to specific types, such as persons or specific systems outputs like summarization.

[10] https://gate.ac.uk/sale/tao/splitch5.html#x8-840005.3 (Accessed: July 19, 2022).
[11] http://docs.oasis-open.org/uima/v1.0/uima-v1.0.html (Accessed: July 19, 2022).

```
{
    "tokens": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "string": ""
        }
    }],
    "name": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "string": ""
        }
    }],
    "personTitle": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "kind": ""
        }
    }],
    "government": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "string": ""
        }
    }],
    "email": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "string": ""
        }
    }],
    "sentence": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "string": ""
        }
    }],
    "location": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "kind": ""
        }
    }],
    "link": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "type": "",
            "string": ""
        }
    }],
    "coreference": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "matches": [{
                "id": 0
            }],
            "score": 0.0
        }
    }],
    "documentsSimilarity": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "documentId": 0,
            "result": 0.0
        }
    }],
    "summarization": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "sentences": []
        }
    }],
    "stemmer": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "end": 0,
        "data": {
            "original": "",
            "stemmed_word": ""
        }
    }],
    "sentenceSentiment": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "emotion_name": "",
            "sentiment_string": ""
        }
    }],
    "classification": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "topic": "",
            "confidance": 0.0
        }
    }],
    "lemma": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "original": "",
            "lemmatized_word": ""
        }
    }],
    "answer": [{
        "id": 0,
        "begin": 0,
        "end": 0,
        "data": {
            "result": 0.0,
            "sentence": ""
        }
    }]
}
```


Table 3: UIMA schema (annotation types) represented in a JSON format. Schema was extracted by Nik Hrovat [6].

## 3.4 Orange data model

Orange's native model are *.tab* files, which is basically a tabular format for standard machine learning tasks. For the needs of text mining addon, a corpus can be fed to Orange using a *.tab* file, where each line is a document that can contain multiple features in each column, such as fulltext, class, ...

A user also has an option to import documents encoded in predefined formats (i.e., .txt, .docx, .odt, .pdf, .xml, and .conllu) from a folder. Similar functionality is also available in Textable. Corpus can be created also interactively. Apart from these, separate widgets are available to import data from predefined structured sources such as The Guardian, NY Times, Pubmed, Twitter, and Wikipedia.

Results from supported text widgets are returned as additional column features. For example, document embeddings add [embedding-length] columns to each document as an embedding representation. Some widgets introduce their own formats, such as bag of words, which works with a column that contains words and their frequencies, e.g., "word1=3, word2=123, word3=66, ..."

Formats can also be transformed, so that they can be natively used with pre-existing ML stack, and this is a great added value of Orange.

## 3.5 Stanza data model

Stanford NLP Group has already a long tradition in offering NLP tools. Prevously they provided Java-based Stanford Core NLP [9] that provided a *HashMap-based* data model. Similarly, they now provide Stanza library [12] with a simpler model.

The Stanza's model is nicely described on its Web page[12]. Model is key-value based and therefore it supports for adding new features to each object type. In Table 4 we show Stanza's data format in JSON representation.

The model seems easy to understand, is extensible, but is clearly prepared for only a subset of natural language processing tasks, such as named entity recognition or lemmatization.

[12] https://stanfordnlp.github.io/stanza/data_objects.html (Accessed: July 19, 2022).

```
// Document
{
    "text": "The raw text.",
    "sentences": [{Sentence1}, {Sentence2}, ...],
    "entities": [{Span1}, {Span2}, ...],
    "num_tokens": 51,
    "num_words": 23
}

//Sentence
{
    "doc": Document, //back-pointer
    "text":    "The raw text",
    "dependencies": [
        {"head w.": "go", "rel.": "subj", "dep. w.": "home"},
        ...
    ],
    "tokens": [{Token1}, {Token2}, ...],
    "words": [{Word1}, {Word2}, ...],
    "entities": [{Span1}, {Span2}, ...],
    "sentiment": "positive",
    "constituency":    {ParseTree}
}
```

```
//Token
{
    "id": {"start-idx": 1, "end-idx": 4},
    "text":    "The",
    "misc": "Custom annotation value",
    "words": [{Word1}, {Word2}, ...],
    "start_char": 24,
    "end_char": 43,
    "ner": "B-ORG"
}

//Word
{
    "id": 3,
    "text": "The",
    "lemma": "the",
    "upos": "NOUN",
    "xpos": "NNP",
    "feats": "Gender=Fem|Person=3",
    "head": 2, //syntactic head word
    "deprel": "nmod", //relation to head word
    "deps": "head-word + deprel",
    "misc": "custom value",
    "parent": {Token}
}
```

```
//Span
{
    "doc": {Document},
    "text": "The Batman",
    "tokens": [{Token1}, {Token2}, ...],
    "words": [{Word1}, {Word2}, ...],
    "type": "PERSON",
    "start_char": 21,
    "end_char": 43
}

//ParseTree
{
    "label": "Noun",
    "children": [{ParseTree1}, {ParseTree2}, {ParseTree3}, ...]
}
```

Table 4: Stanza schema (annotation types) represented in a JSON format.

# 4 ANGLEr data model (proposal)

According to the data models reviewed above, we believe that they introduce useful features but are not general, extensible or easy to understand. For example: GATE introduces some document types and annotation types but new types would need to be made almost from scratch and encoded in Java. UIMA features very comprehensive ans complex schema which seems is not developing anymore. Orange offers tabular-oriented format with transformations for specific tasks and therefore no general format for text processing representation was made. Stanza presents a clear and understandable format but is not comprehensive.

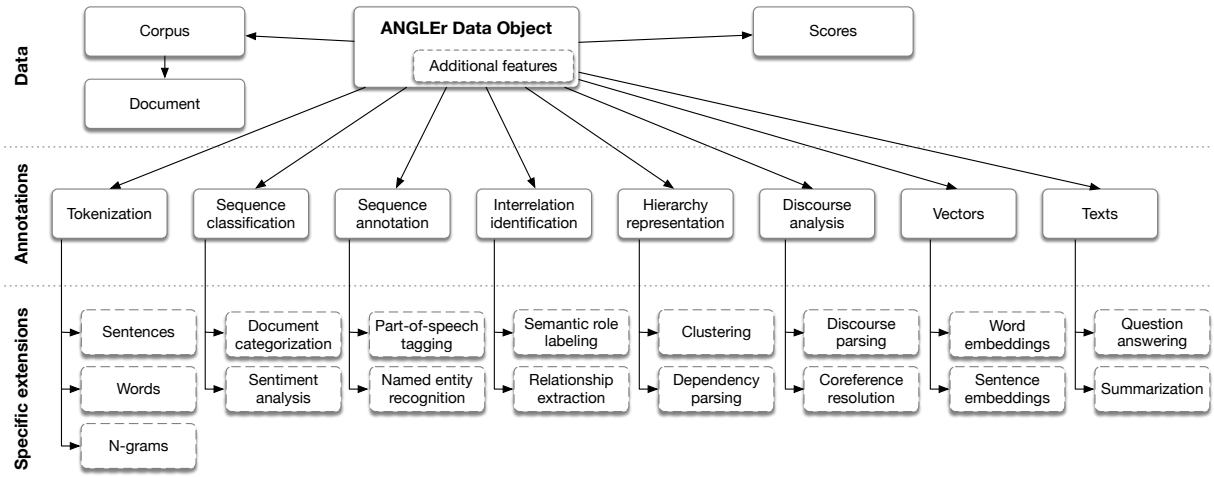


Figure 6: ANGLEr object type hierarchy.

We believe that a data model need to identify general approaches and provide minimum viable set of necessary attributes that can be further extended by specific methods. Also, "hierarchical-"style of model representation would better suit for navigation, extensibility and understanding of the model. Based on our knowledge of the NLP field, we believe the following top-level data model categories need to be supported:

**Corpus** : As in almost most of the models, corpus should consist of documents, which should be a lower-level representation of a document. Documents can be of different types, lengths and should contain text and metadata.

**Tokenization** : Textual data is mostly needed to be represented as sentences, words, multi-word expressions, n-grams, ... This type should therefore allow multiple representations. Pre-existing tokenizations are stored within this field - the same also holds for true annotations for other categories below.

**Sequence classification** : We believe sequence is a general term that represents, tokens, words, sentences, ... Some examples of concrete tasks are sentiment analysis or document categorization. For the result of classification it is important to know the classified label.

**Sequence annotation (i.e., tagging)** : This type should be used to store a set of tags for different sequences of the documents. Some examples of concrete tasks include part-of-speech tagging or named entity recognition.

**Interrelation identification** : This type should be used to model any type of relationships between two (or more) objects. Some examples would be similarity between two documents, semantic role labeling, or semantic relationship between two words in a sentence ("John" → *lives_in* → "Boston").

**Discourse analysis** : This type should be able to address different sequences, for example, sequences of mentions (i.e., subtype fo tokenization) for coreference resolution, or sequences of sentences. Each sequence can have associated labels that identify objects' interrelationships.

**Hierarchy representation** : This type needs to enable a representation of a hierarchical structure between different elements of text. Examples of such tasks are chunking, dependency parsing, or clustering (on a top level we have a root that might form N clusters of documents).

**Texts** : This type is intended to represent textual results of analysis. For example, a document might consist of different components - text, question, answer. For a question answering, this field would contain generated answers; for translation translations; for summarization summaries (of specific document types).

**Vectors** : In the era of deep learning, vectorized representations became even more popular. This field should be able to represent vectors of any other type - e.g., document, token, ...

**Scores** : Evaluation results of specific algorithm. The type includes reference to algorithm annotations and includes metrics with scores.

**Additional features** : Each type should offer an algorithm to store additional features (key/value pairs) to its object. These features might be date and time of annotation, algorithm name, ...

In Figure 6 we present a general ANGLEr data model. The "umbrella object" includes (a) corpus, (b) scores, and (c) high-level annotation objects. The objects are hierachical, so descendants inherit methods and attributs from their parents. Therefore, objects on annotation level define a minimum subset of attributes that can be extended in lower levels. This enables interoperability between different methods. Also, each object includes additional features attribute that contains key/value pairs for metadata of a specific object.

In Table 6 we propose an example structure of proposed data model in a JSON format. Different parts are interconnected using component ids. Some of the components can be used as-is, others need to be specified depending on specific algorithm. For example, a discourse analysis template can be extended for coreference resolution or discourse markers analysis.

### 4.0.1 Versioning

Type hierarchy is proposed to improve backward compatibility when adding new object types. This way each new specific object type contains the attributes of its parent as well as some of its own attributes (each level also allows for key-value metadata storage). We can thus ensure that an algorithm that was created for working with a general object type can also work with all of its descendants. For instance, a tool for computing token frequencies could be created to accept a *token*. This enables it to accept all other specific objects that inherit from the token type - e.g., the Parsing type. In named entity recognition, for example, the program would accept two types of input. Firstly, it would accept the *token* type to get the separation of the documents into tokens. In addition to that, it would also accept the part of speech tags represented in the *sequence tagging* object type. The resulting named entities would be represented by an object with the sequence tagging type.

As new algorithms are developed, additional object types might get defined. We would like to store a list of all available object types in a way where it can get updated later on. It is also important that each data model be marked with a unique version number. The data model will be versioned using *SemVer* guidelines and all the versions should be available in open source code repository.

# 5 ANGLEr framework architecture (proposal)

During the review of existing frameworks we discovered good and bad practices. The idea is to provide an extensible and scalable framework where researchers could easily integrate their tools, and users (e.g., non-technical) could also easily use them. Apart from the versioned data model, we believe it is important to provide (a) an extensible API-based architecture with unified, and (b) pluggable user interface.

```
//ANGLEr Data Object
{
  "corpora": [{Corpus1}, {Corpus2}, ...],
  "scores": [{Scores1}, {Scores2}, ...],
  "annotations": [{Annotations1}, {Annotations2}, ...],
  "features": {"version": "1.0"} //Custom attributes
}

//Corpus
{
  "id": "1",
  "name": "Best NER-KB corpus",
  "documents": [{Document1}, {Document2}, ...],
  "features": {"url": "http://corpus-1.0.si"} //Custom attributes
}

//Document
{
  "id": "1-23", //Corpus id + Document id
  "text": "Best Story! Once upon a time, ...", //text used for analysis
  //Structured text parts that can be used by algorithms by key
  "text-parts": {
    "title": "Best Story!",
    "content": "Once upon a time, ...",
    "likes": 34
  }
  "features": {"length": 320, "separator": " "} //Custom attributes
}

//Scores
{
  "predicted_annotations_id": 42,
  "true_annotations_id": 31,
  "scores": {
    "F1": 0.45,
    "P": 0.88,
    "R": 0.91
  },
  "features": {"time": 2353242362} //Custom attributes
}
//Tokenization template
{
  "id": 33, //Used for algorithm input selection
  "annotation-type": "TOKENIZATION",
  "algorithm": "rule-based-1",
  "type": "SENTENCE", //WORDS, NGRAMS, WORD-PARTS, ...
  "documents": [
    {"doc_id": "1-23",
      "tokens": [{"id": 1,  "start_idx": 0, "end_idx": 342,
      "text": "Tralala, hopsasa."}, ...]
    }, ...
  ],
  "features": {"time": 2353242362} //Custom attributes
}
//Sequence classification template
{
  "id": 36, //Used for algorithm input selection
  "annotation-type": "SEQUENCE-CLASSIFICATION",
  "algorithm": "Best NN categorizer",
  "corpus_id": "1",
  "sequences": [
    {"id": 1, "class": "POSITIVE"}, ...
  ],
  "features": {"time": 2353242362} //Custom attributes
}
//Sequence annotation template
{
  "id": 39, //Used for algorithm input selection
  "annotation-type": "SEQUENCE-ANNOTATION",
  "algorithm": "CRF annotator",
  "tokenization_id": "1",
  "annotations": [
    {"doc_id": "1-23", "tags": ["O", "ORG", "O", ...]}, ...
  ],
  "features": {"annotation_time": "20sec"} //Custom attributes
}
```

```
//Interrelation identification template
{
  "id": 41, //Used for algorithm input selection
  "annotation-type": "INTERRELATION-IDENTIFICATION",
  "algorithm": "Intraverse relation extractor",
  "tokenization_id": "3",
  "relations": [
    {"relation": "employed_at", ...} //This object is specified on lower levels
  ],
  "features": {"annotation_time": "20sec"} //Custom attributes
}
//Hierarchy representation template
{
  "id": 43, //Used for algorithm input selection
  "annotation-type": "HIERARCHY-REPRESENTATION",
  "algorithm": "Agglomerative clusterer",
  "tokenization_id": null, //One of the following set only!
  "corpus_id": "1"
  "hierarchies": [
    {"parent_id": "34", "descendants": [{"id": "44", "relation": "nsubj"}, ...]}
    //This objects are specified on lower levels
  ],
  "features": {"max-hierarchy": 13} //Custom attributes
}
//Discourse analysis template
{
  "id": 45, //Used for algorithm input selection
  "annotation-type": "DISCOURSE-ANALYSIS",
  "algorithm": "Coref resolver SkipCor v2",
  "tokenization_id": 1, //These are sentences, mentions, ...
  "chains": [
    {"id1": "34", "id2": "45", "type": "anaphora"} //This object is specified on lower levels
  ],
  "features": {"singletons-num": 61} //Custom attributes
}
//Vectors template
{
  "id": 47, //Used for algorithm input selection
  "annotation-type": "VECTORS",
  "algorithm": "Doc2Vec",
  "tokenization_id": null, //One of the following set only!
  "corpus_id": "1"
  "vectors": [
    {"id": "78", "vec": [2.000, 3.241, 4.267, 5.987, ...]}
  ],
  "features": {"vector-dim": 350} //Custom attributes
}
//Texts template
{
  "id": 49, //Used for algorithm input selection
  "annotation-type": "TEXTS",
  "algorithm": "AnglerSummarizer",
  "corpus_id": "1",
  "doc-text-part-inputs": {
    "question": "question-part",
    "content": "text-part"
  }, //This object is specified on lower levels
  "texts": [
    {"doc-id": "78", "value": "No."}
  ],
  "features": null //Custom attributes
}
```

Table 5: ANGLEr schema (annotation types) represented in a JSON format.

In Figure 7) we show a high-level architecture of the proposed ANGLEr framework which consists of (a) ANGLEr modules, (b) ANGLEr Core, (c) ANGLEr UI, and (d) ANGLEr data models. From the terminology point of view, it is important to understand the following:

**Processor** : A processor is an algorithm that expects to get some data and return news results following the ANGLEr data model. A processor corresponds to an algorithm/step in a workflow.

**Module** : A module is a (Docker-based) package that can contain multiple processors and all the functionalty to be integrated into ANGLEr Core (i.e., UIs, widget definitions, settings, ...). It communicates with the ANGLEr Core using a Module API (see Section 6). A Module can also be used independently by a third-party application without other ANGLEr framework parts.

ANGLEr Core is the main building block, which provides all the framework functionalities to glue different components together. It provides functionality to manage workflows, to install/remove modules, to work with the Docker subsystem, saves the data in its own internal database, and provides APIs for communication. ANGLEr user interface is able to communicate with ANGLEr core and includes essential functionality for working with Core, such as support for settings, menus, Core functions, and workflow features (i.e., running, stopping, saving, opening existing workflows, ...). Most of the specific UIs (e.g., widget icons, widget settings, processor visualizations) are directly embedded via specific Module APIs. In that sense, each ANGLEr module developer has its own options how to visualize his own part

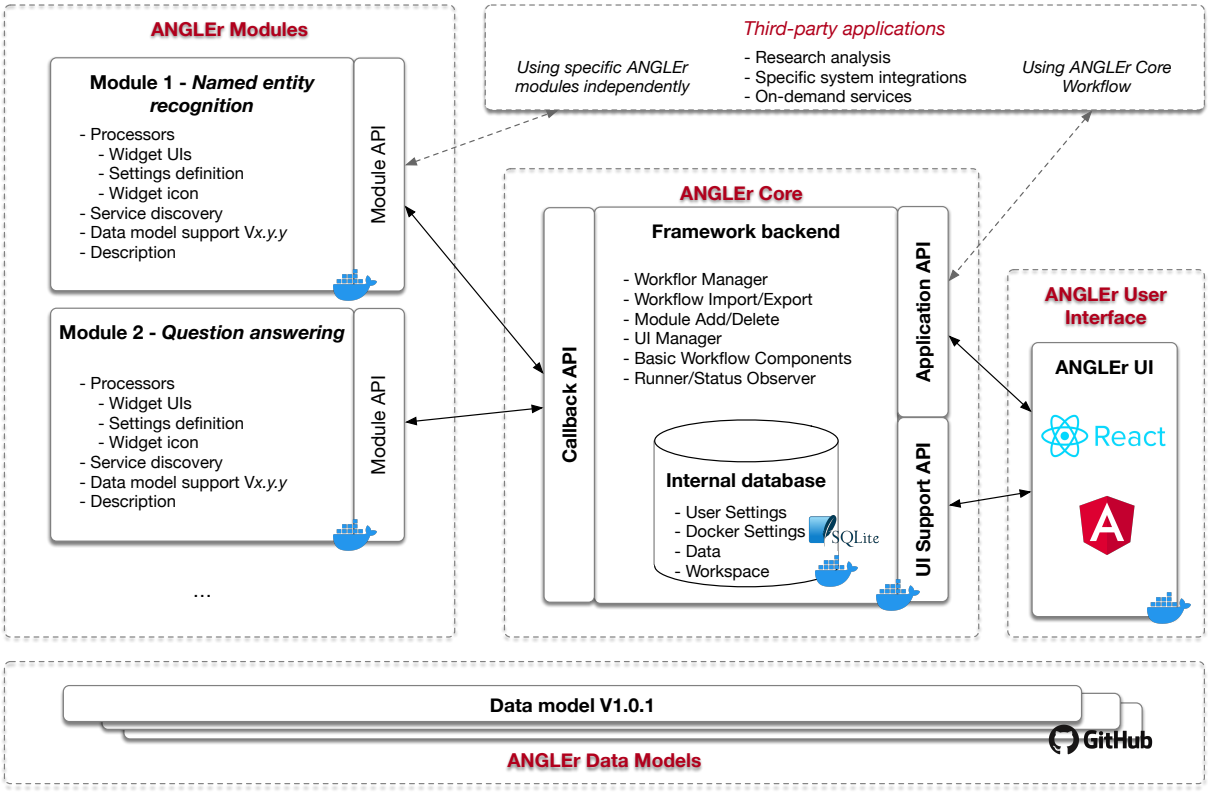


Figure 7: ANGLEr high-level system architecture proposal.

of functionality. The underlying glue component is the ANGLEr data model, which enables seamless communication.

As all the APIs should follow guidelines, the system is loosely-coupled. Therefore, third-party systems will be able to use modules as standalone components. Also, in case someone will first prepare a workflow using ANGLEr, he will be able to programmatically integrate with ANGLEr Core and directly use results in their applications (without using ANGLEr UI).

# 6 The module API

```
// Endpoint: /about GET
{
  "UUID": "81a03d1a-0844-11ed-861d", //Identifier of the module.
  "name": "Best NER processors", //Name of the module.
  "version": "v1.2.9", //Module version.
  "data_model": "v1.0.1", //Version of the data model used.
  "desc": null, //(optional) Module description.
  "authors": "Jane Doe", //(optional) List of authors.
  "organisation": "University of Ljubljana", //(optional) Organisation of the authors.
  "url": "best-ner.github.io", //(optional) URL address of the page about the module.
  "docs_url": //(optional) A web page containing the documentation.
}

// Endpoint: /processors GET
[
  {
    "name": "CRF-based NER tagger" //Name of a processor.
    "short_name": "CRF-NER", //(optional) Short version of the name.
    "settings_endpoint": "/crf-ner/settings", //(optional) A module-relative address of a page containing processor settings.
    "ui_endpoint": "/ner-visualizations", //(optional) A module-relative address of a page for visualization.
    "icon": "/crf-ner/icon.ico", //A module-relative address of the icon to be used to represent the  processor.
    "category": "Semantic analysis", //A category of the processors menu that should  contain this processor (i.e., widget).
    "run_endpoint":  "/crf-ner/run", //Starting a processor. Data (i.e., current ANGLEr data object) is sent as payload.
    "status_endpoint":  "/crf-ner/status", //(optional) Returning statuses READY, PROCESSING (XX \%)
    "stop_endpoint":  "/crf-ner/stop", //Stopping/cancelling current processing
    "inputs": ["TOKENIZATION", "WORDS"], //A list of ANGLEr data object types (from a hierarchy) that need to exist in the current data object for running this processor.
    "outputs": ["NER-ANNOTATION"], //A list of ANGLEr data object types (from a hierarchy, as high-level as possible) representing the processor's outputs.
    "docs_url": //(optional) A web page containing the documentation.
  },
  ... // Other available processors
]
```

Table 6: ANGLEr Module APIs definition (proposal).

ANGLEr is working with a mutable ANGLEr Data Object that processors can change (mutable object). All of the communication between the parts of the architecture is done over REST application programming interfaces (API). In Table 6 we define the necessary module API that ANGLEr uses in

the "service discovery" part. The two obligatory endpoints provide information about the module and processors. With the results of */processors* endpoint, ANGLEr can show the processor, provide access to its UI (settings, visualizations), and communcate with it (i.e., running, stopping, ...).

As presented, UI endpoint should provide Web-based interface that would be integrated into the ANGLEr's Core. Apart from numerous options of Web programming languages to prepare a new UI, developers might reach out to simple prepared frameworks that provide UIs for their use case. An example of such a framework is Gradio[13]

## 6.1 Docker-based architecture

The use of Docker-based implementations is actually not needed, but to provide better interoperability between systems, easy installation, and running, we provide all components packages as Docker images.

Using this approach a Docker image can contain all dependencies, each processor can be implemented in its own programming language, distribution of updates is easier, and also enables scalability for users that would like to run modules on different physical machines or in cloud (multiple instances). Still, the designed system is intended to be single-user only.

## 6.2 ANGLEr graphical user interface (proposal)

The graphical user interface allows for custom acyclic widget-graph creation for running workflows. It is important to note that running the components will be done in a strictly sequential manner (e.g., different branches do not mean execution in parallel), and all the processors will have access to the same object that was altered (i.e., mostly with just additional annotations added) during the processing workflow.

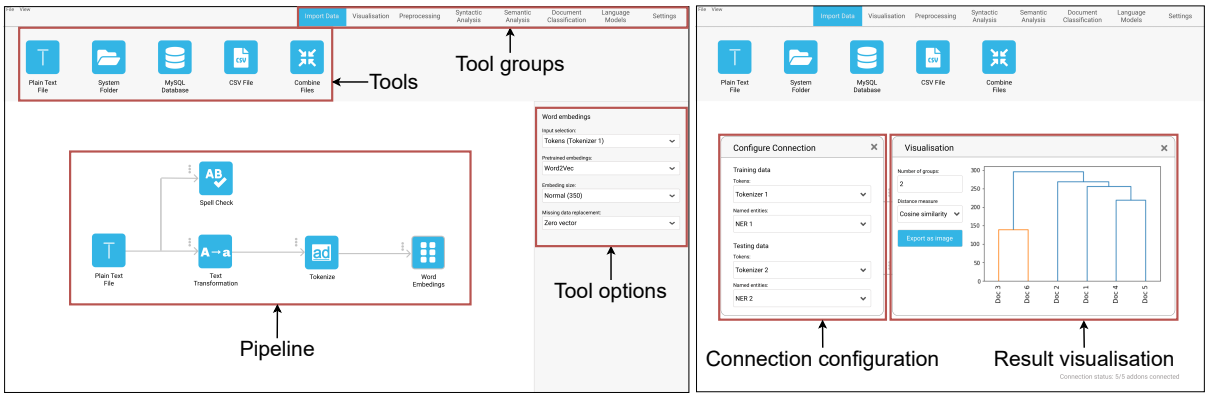


Figure 8: A general ANGLEr user interface (proposal).

In Figure 8 we show a general ANGLEr user interface. The upper part features categories of different processors. A user can drag-and-drop a specific processor to the canvas to see its settings, connect it to other processors, and then observe results. We show additional user interface actions in Figures 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, and 26.

## 6.3 A use case scenario

ANGLEr needs to have some imported modules that enable processing. In this use case we will briefly describe how we imagine working with ANGLEr.

Once the framework is started, ANGLEr backend docker starts running and runs also other Docker images if not already running. If the database container was not removed, settings from the previous system run remained the same. From a user perspective, all work is done via the Web user interface.

First, when adding a new module to the framework, ANGLEr backend creates a request to the */about* endpoint URL we provided to get the basic information about the entire module. After that it requests the */processors* endpoint to include all the available processors into the framework (i.e., internal database and graphical user interface).

A user can now drag-and-drop some widgets to the canvas and connect them between each other. Probably, first widget will import data, while other widgets will process/manipulate or visualize it. A

[13] https://gradio.app (Accessed: July 25, 2022).

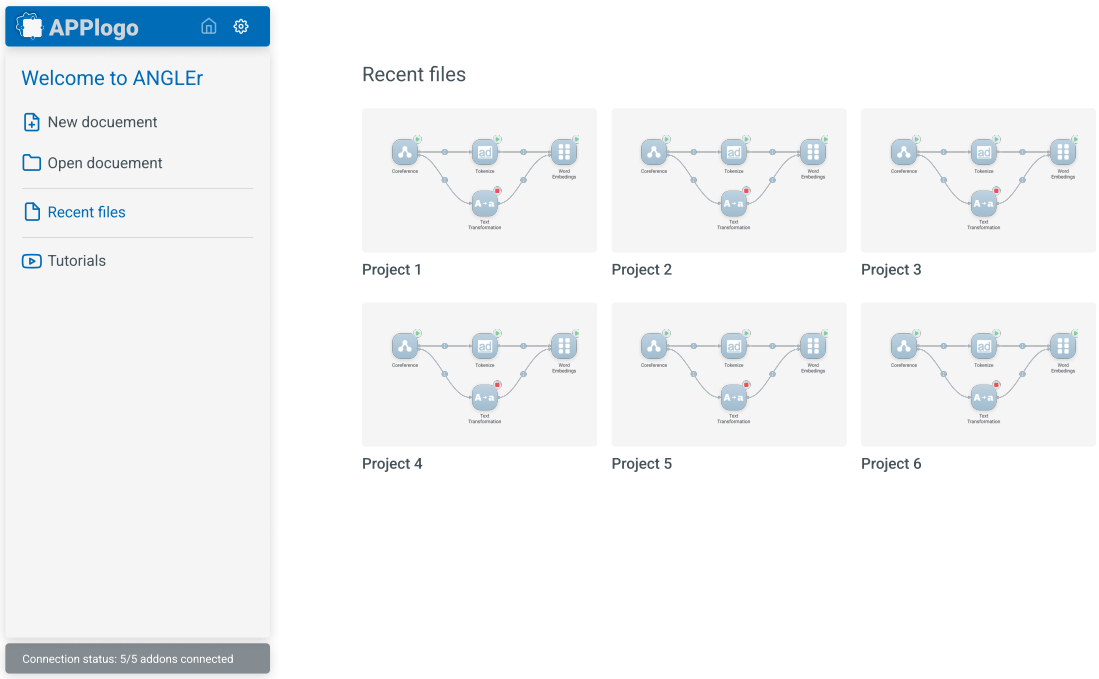


Figure 9: ANGLEr showcase UI (proposal).

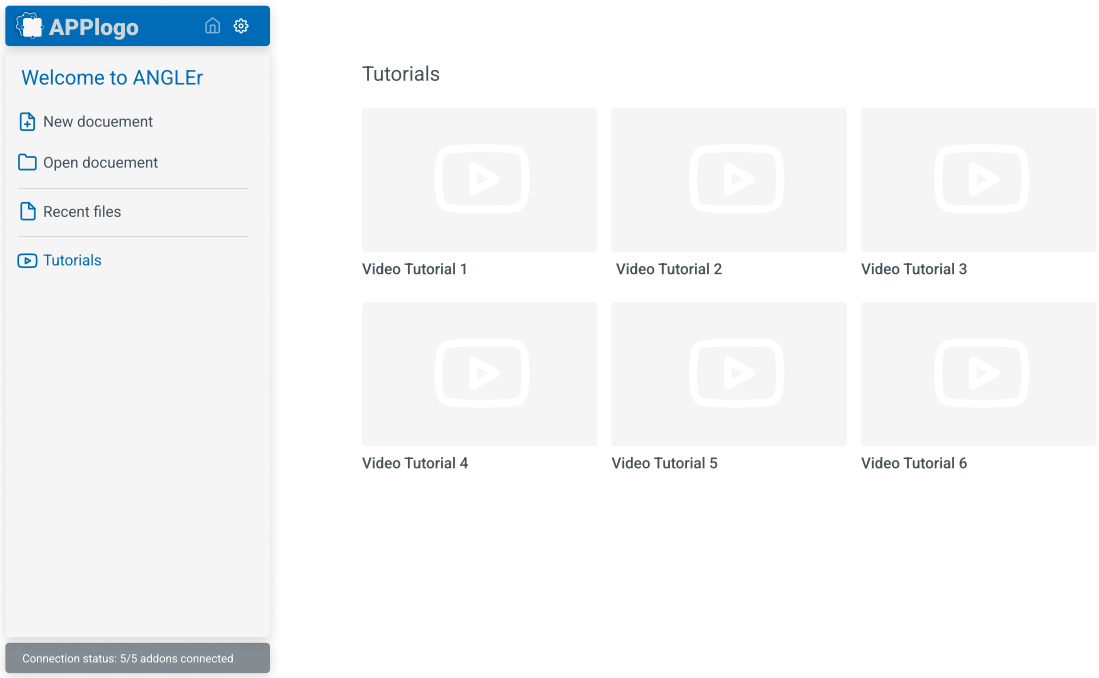


Figure 10: ANGLEr showcase UI (proposal).

user can also define specific settings that are provided for each processor separately[14]. After a user runs the workflow, he can observe results or use visualization widgets to browse results.

# 7 Final thoughts

In this report we described the high-level specifics and provided initial blueprints for the implementation of a general extensible data model and (Web-based) tool for text analysis and exploration (ANGLEr).

[14]It is not yet defined how specific processor settings would be stored when exporting workflow. Also, one processor can be used multiple times with different settings, so they should be somehow monitored by the ANGLEr.

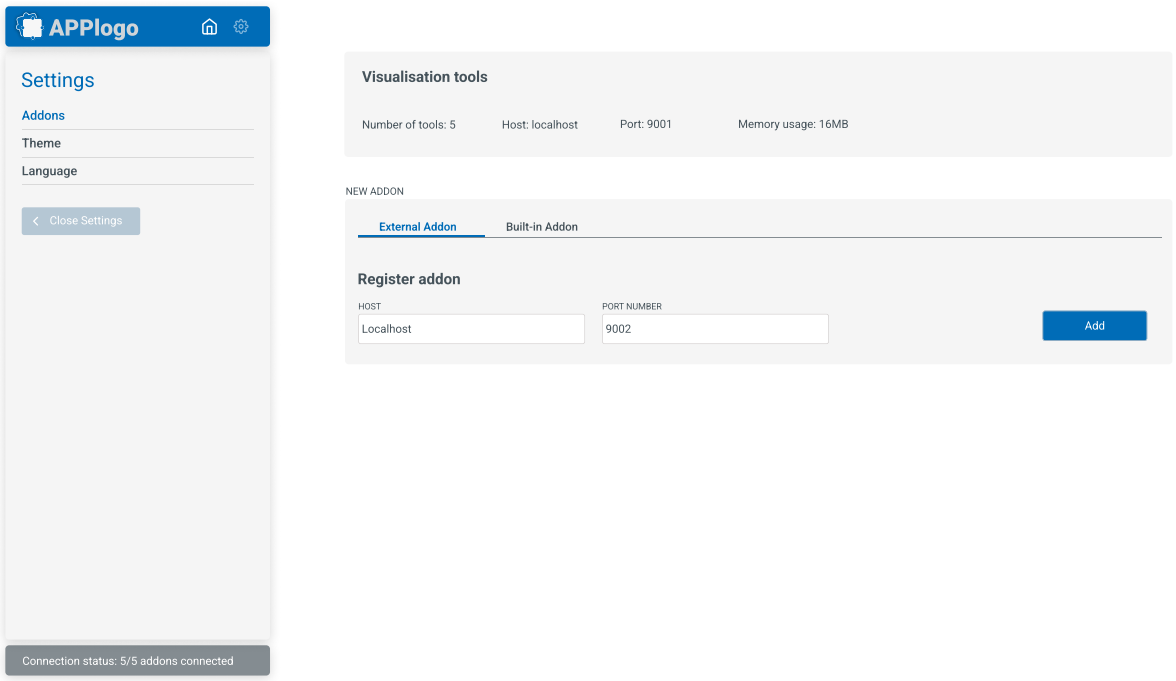


Figure 11: ANGLEr showcase UI (proposal).

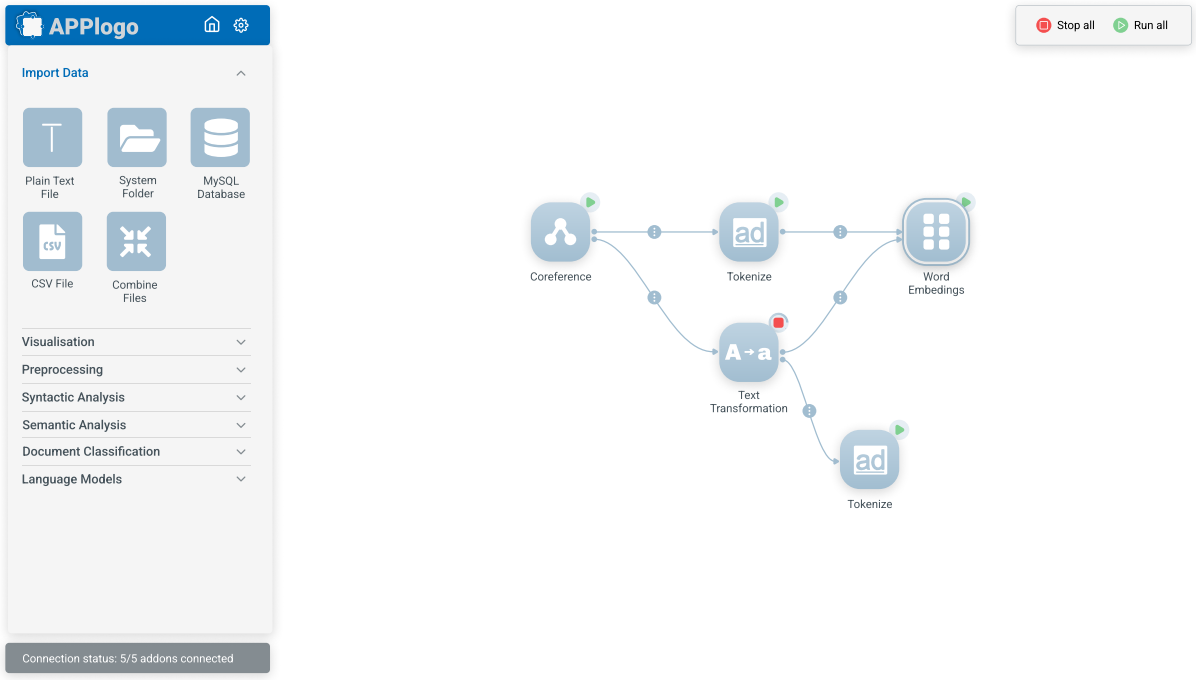


Figure 12: ANGLEr showcase UI (proposal).

The research community and other interested parties would benefit from a tool like this. The tool would allow to easily run specific state-of-the-art methods and share them to use by non-technical users.

We have described already existing approaches that solve this problem. For those that are mostly used, it is important to provide a continual improvements and vibrant community (and employed programmers) that updates the system. The main thoughts from our journey are as follows:

We describe the main components for implementing a new natural language processing framework - ANGLEr. We believe that a new framework based on our proposal would provide a large improvement over the existing frameworks and would greatly benefit users that are working with natural language processing. The framework would provide a fast way for prototyping when developing text processing pipelines. It would also allow users with no programming knowledge to build advanced nlp pipelines. In

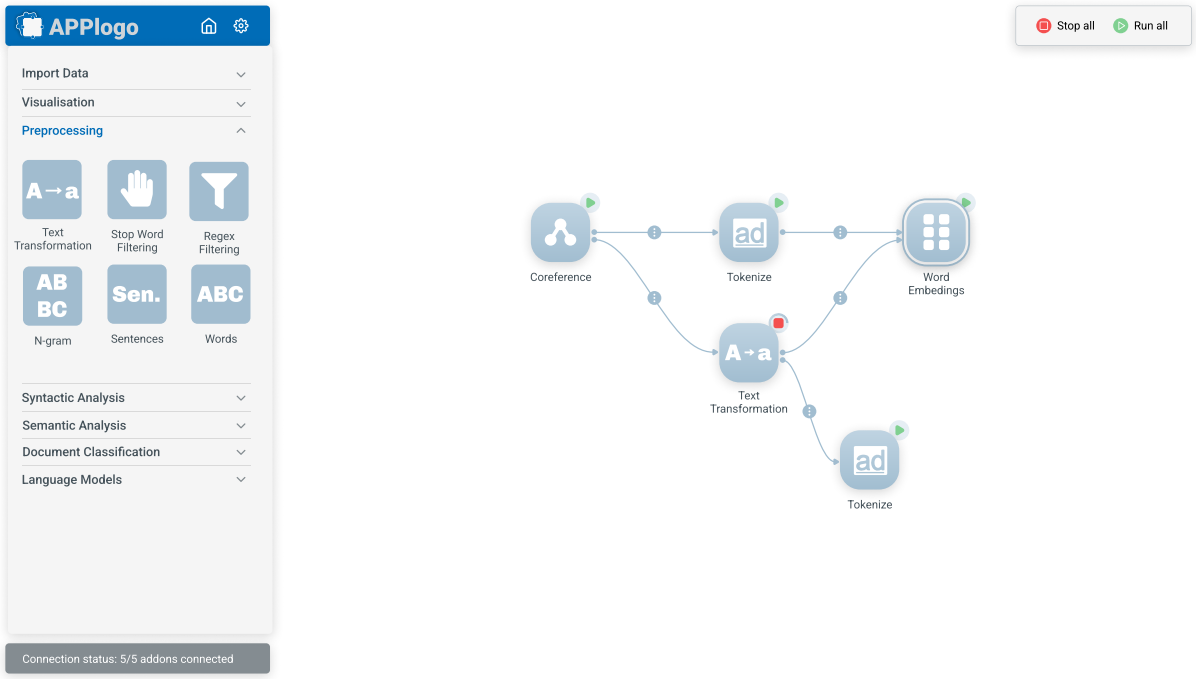


Figure 13: ANGLEr showcase UI (proposal).

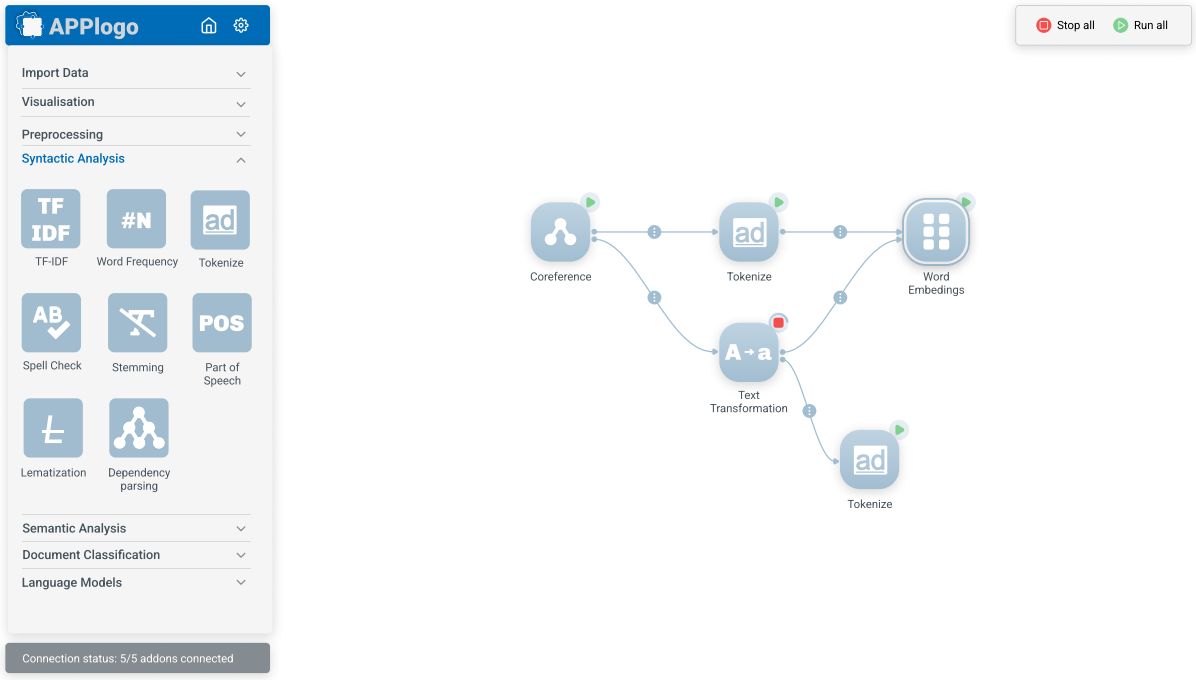


Figure 14: ANGLEr showcase UI (proposal).

addition to that, the new framework would provide the researchers with a great way for showcasing their work in the nlp area. Since the tools can be implemented in any programming language, their inclusion is much less complicated than with existing frameworks.

**Is there a need to represent all the NLP tasks in one comprehensive data model?** We have multiple types of dataset annotations and a lot of efforts go into writing scripts to wrangle these data. The same holds for data modeling of results of different NLP tasks. There have been many different initiatives and it seems none of them survived. We can thus try to prepare a simple, hierarchical and as much comprehensive model as possible.

**Should we develop Yet Another NLP tool/framework?** Technology is constantly changing. Of

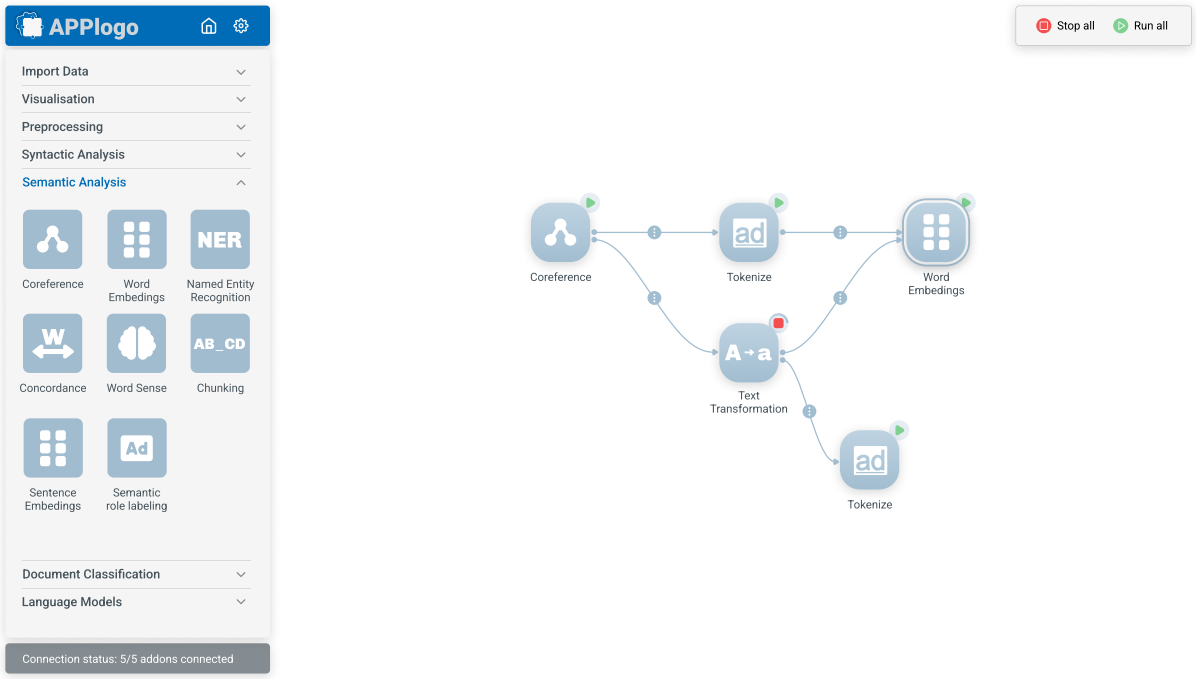


Figure 15: ANGLEr showcase UI (proposal).

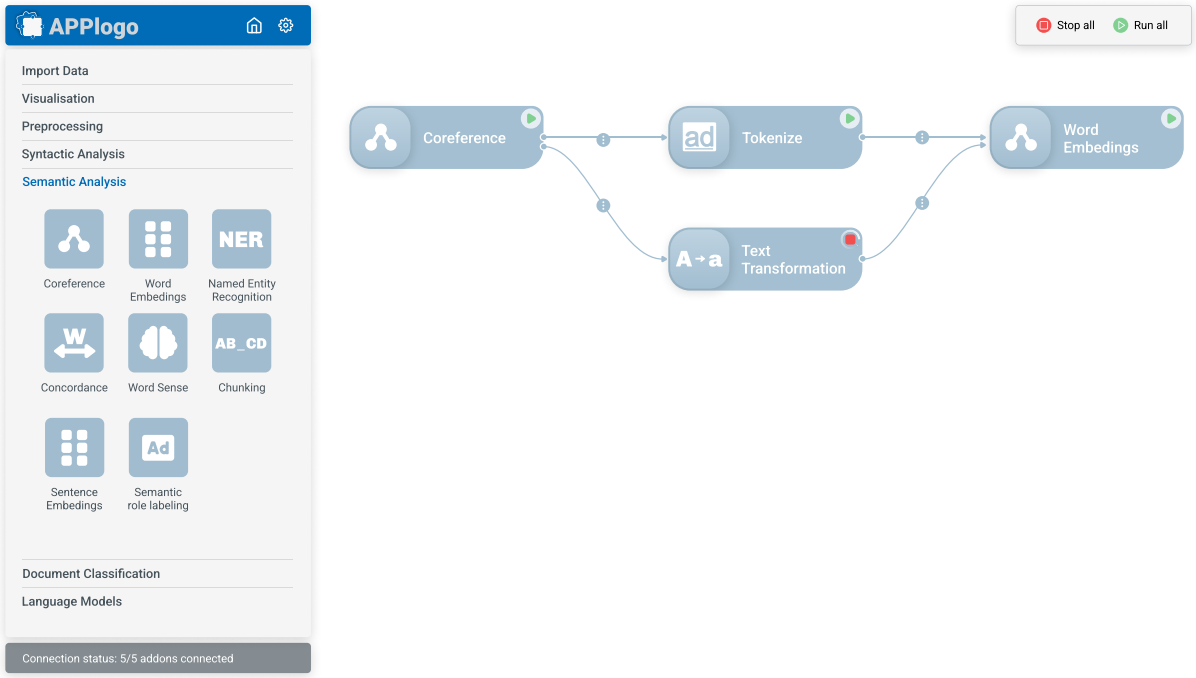


Figure 16: ANGLEr showcase UI (proposal).

course the right direction seems to have loosly-coupled components as possible, but we also need to regularly promote and maintain the system. Currently, Docker-based and Web-based implementation seems the smartest choice, which might change in the future. Half-way alternative would be to integrate into some existing framework[15] and develop only processors (the approach might not be feasible as it would still be an effort for 3rd parties to add new widgets/tools; data processing is different as we expect to have a mutable object; running should be different as should be more controlled and not immediate; tool would not be model agnostic, ...). Nevertheless, continual maintenance would still be needed, but maybe less frequent.

[15] For example, Orange3 addons - https://github.com/biolab/orange3-example-addon (Accessed: July 20, 2022).

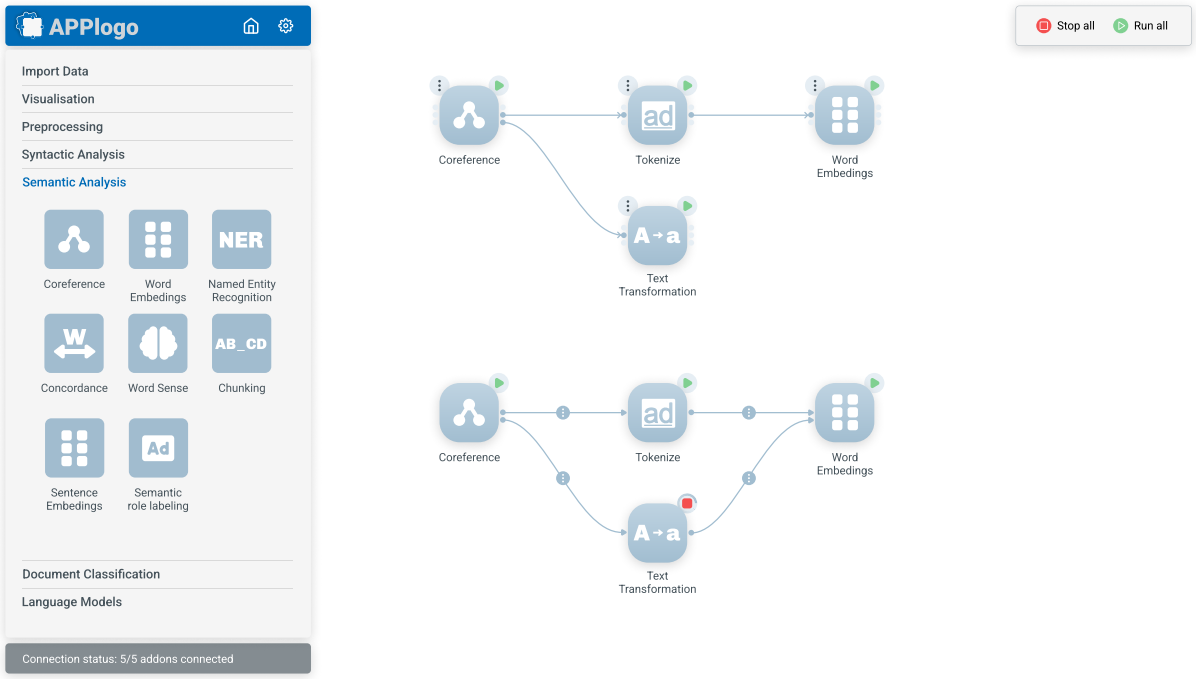


Figure 17: ANGLEr showcase UI (proposal).

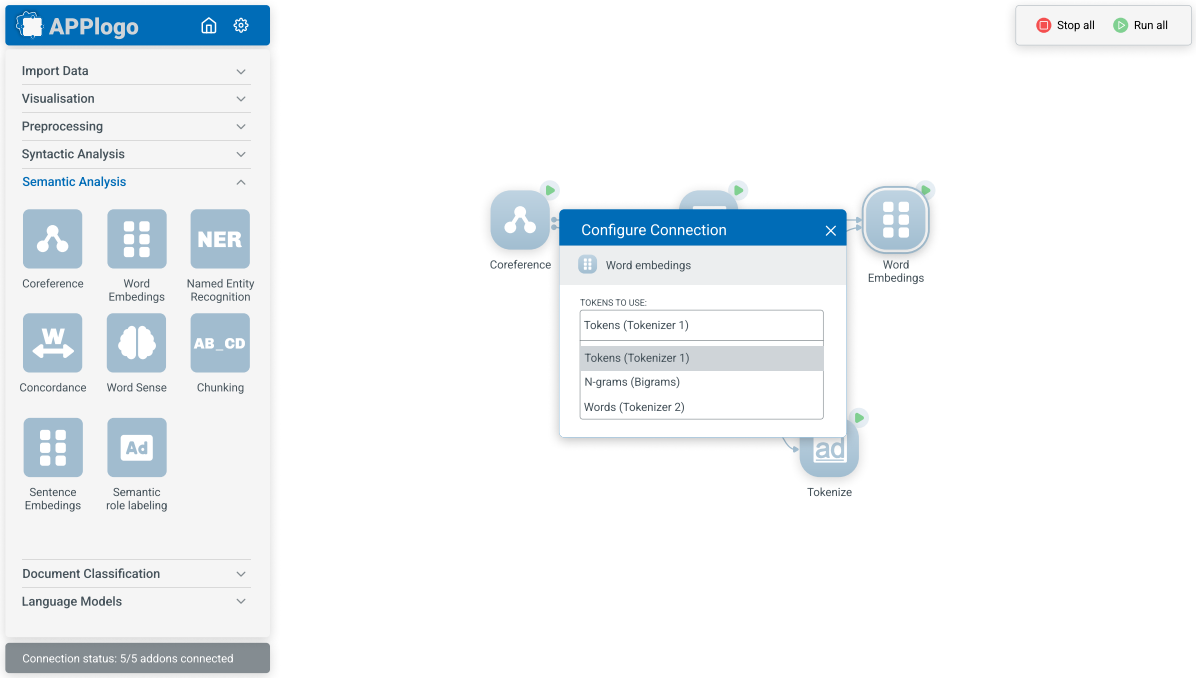


Figure 18: ANGLEr showcase UI (proposal).

**Who are users and who would pay for it?** There exist a number of (data science) linguists who would like to process text with simple techniques. The best would be to employ similar model to SketchEngine[16] or GATE. So starting with some special grants that would fund the design and development. After that some commercial activities would be needed to get maintenance funding.

[16] https://www.sketchengine.eu/ (Accessed: July 20, 2022)

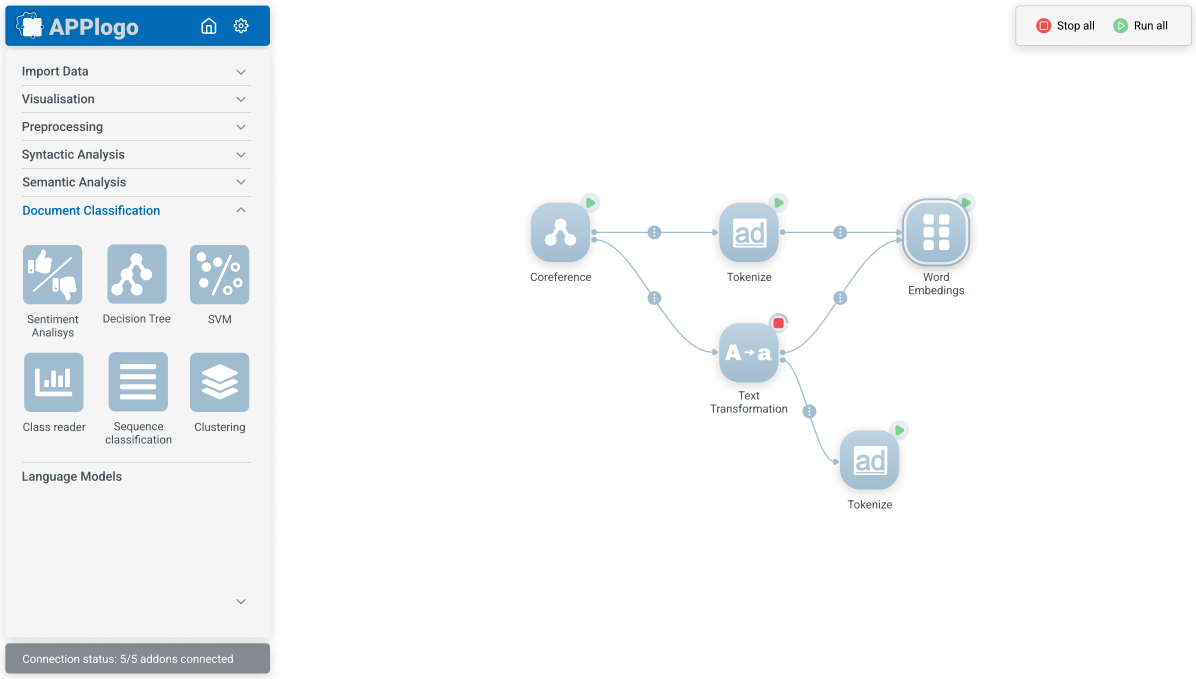


Figure 19: ANGLEr showcase UI (proposal).

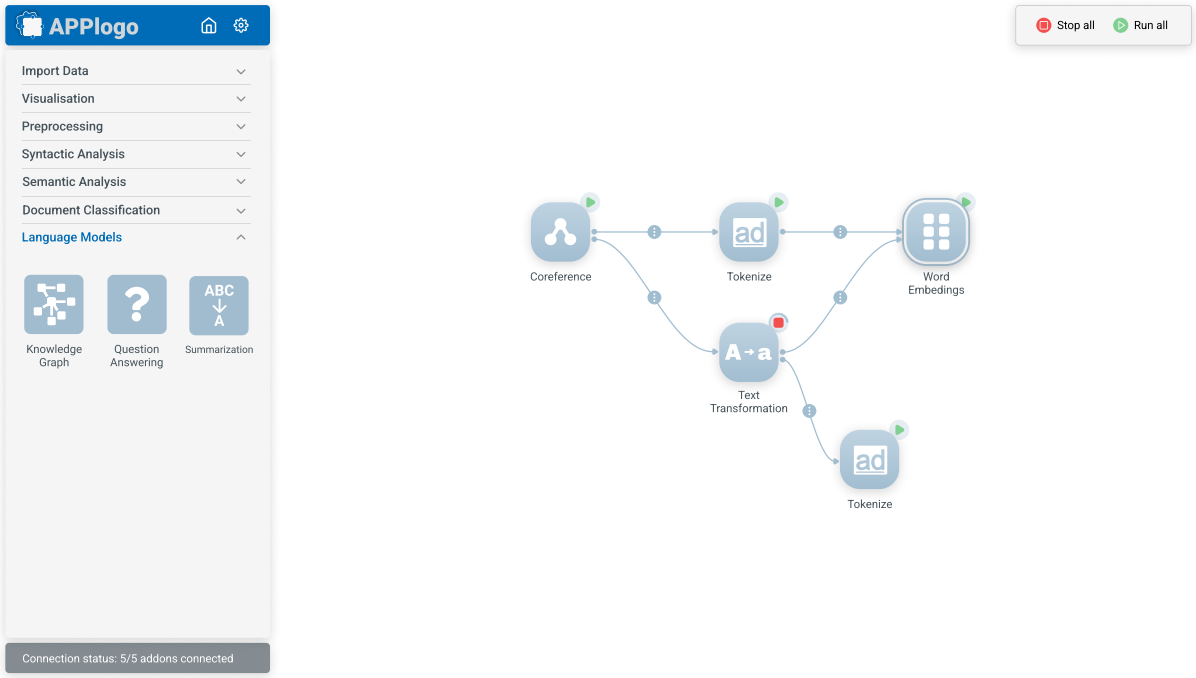


Figure 20: ANGLEr showcase UI (proposal).

# 8 Acknowledgements

Project Development of Slovene in a Digital Environment (RSDO) is financed bt the Slovene Ministry of Culture and the European Regional Development Fund.

EVROPSKA UNIJA
EVROPSKI SKLAD ZA REGIONALNI RAZVOJ
NALOŽBA V VAŠO PRIHODNOST

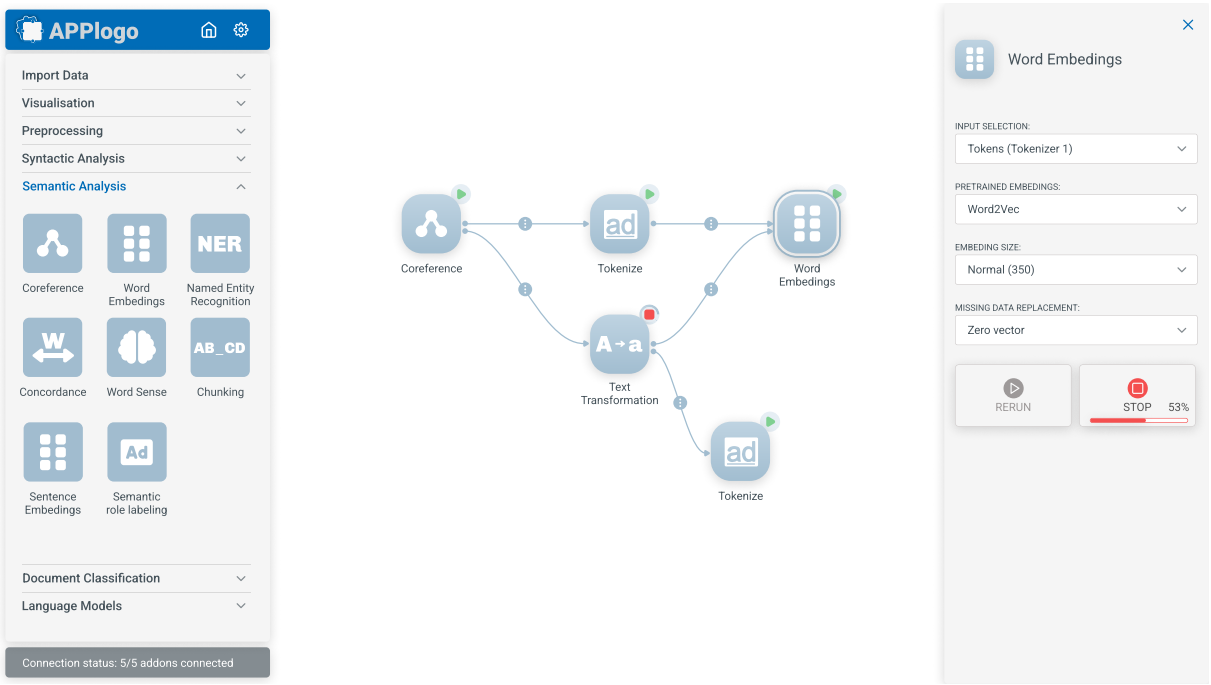


Figure 21: ANGLEr showcase UI (proposal).

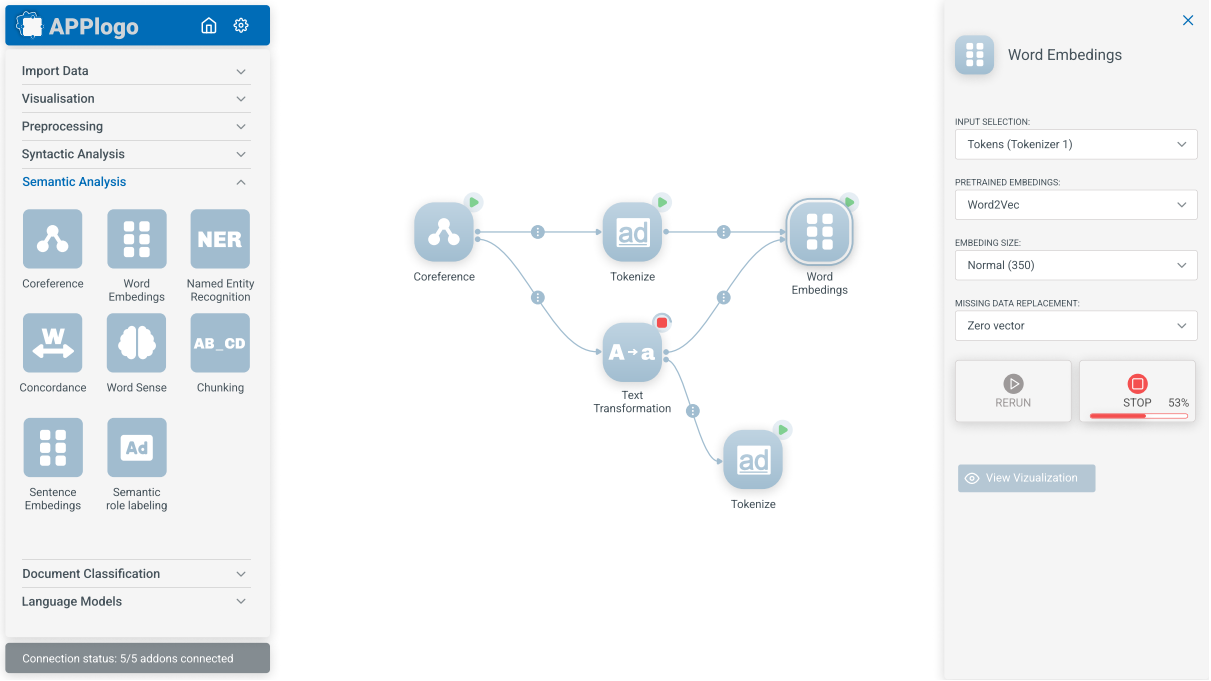


Figure 22: ANGLEr showcase UI (proposal).

# References

[1] Bird, S., Loper, E.: Nltk: the natural language toolkit. Association for Computational Linguistics (2004)

[2] Cunningham, H.: Gate, a general architecture for text engineering. Computers and the Humanities **36**(2), 223–254 (2002)

[3] Demšar, J., Curk, T., Erjavec, A., Gorup, Č., Hočevar, T., Milutinovič, M., Možina, M., Polajnar, M., Toplak, M., Starič, A., et al.: Orange: data mining toolbox in python. the Journal of machine Learning research **14**(1), 2349–2353 (2013)

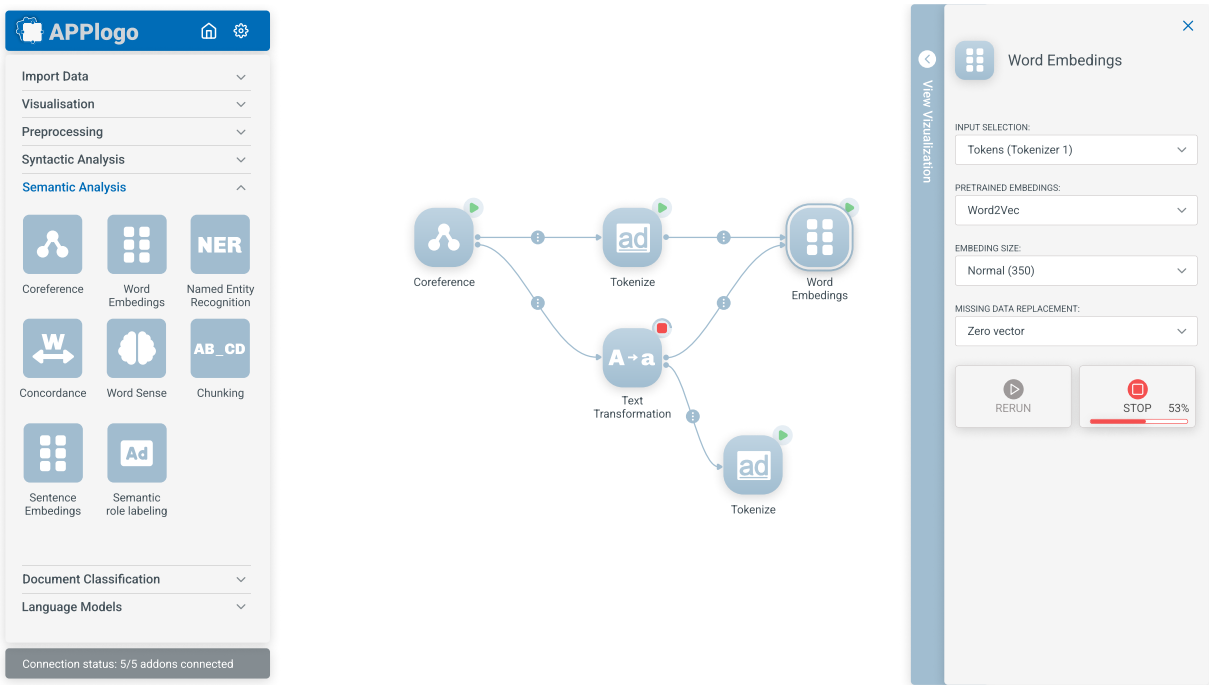


Figure 23: ANGLEr showcase UI (proposal).

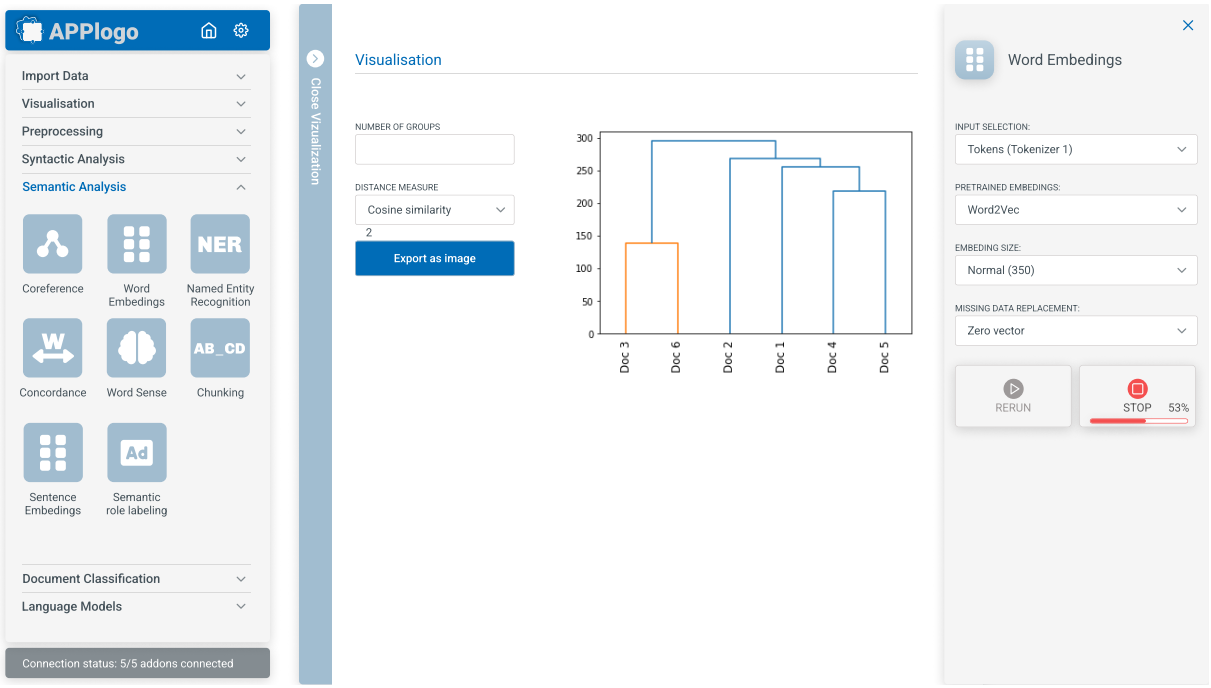


Figure 24: ANGLEr showcase UI (proposal).

[4] Ferrucci, D., Lally, A.: Uima: an architectural approach to unstructured information processing in the corporate research environment. Natural Language Engineering **10**(3-4), 327–348 (2004)

[5] Hellmann, S., Lehmann, J., Auer, S.: Linked-data aware uri schemes for referencing text fragments. In: EKAW 2012. Lecture Notes in Computer Science (LNCS) 7603, Springer (2012). https://doi.org/doi:10.1007/978-3-642-16438-5_10

[6] Hrovat, N.: Zasnova ogrodja za izvajanje metod za procesiranje naravnega jezika. Ph.D. thesis, Univerza v Ljubljani, Fakulteta za računalništvo in informatiko (2022)

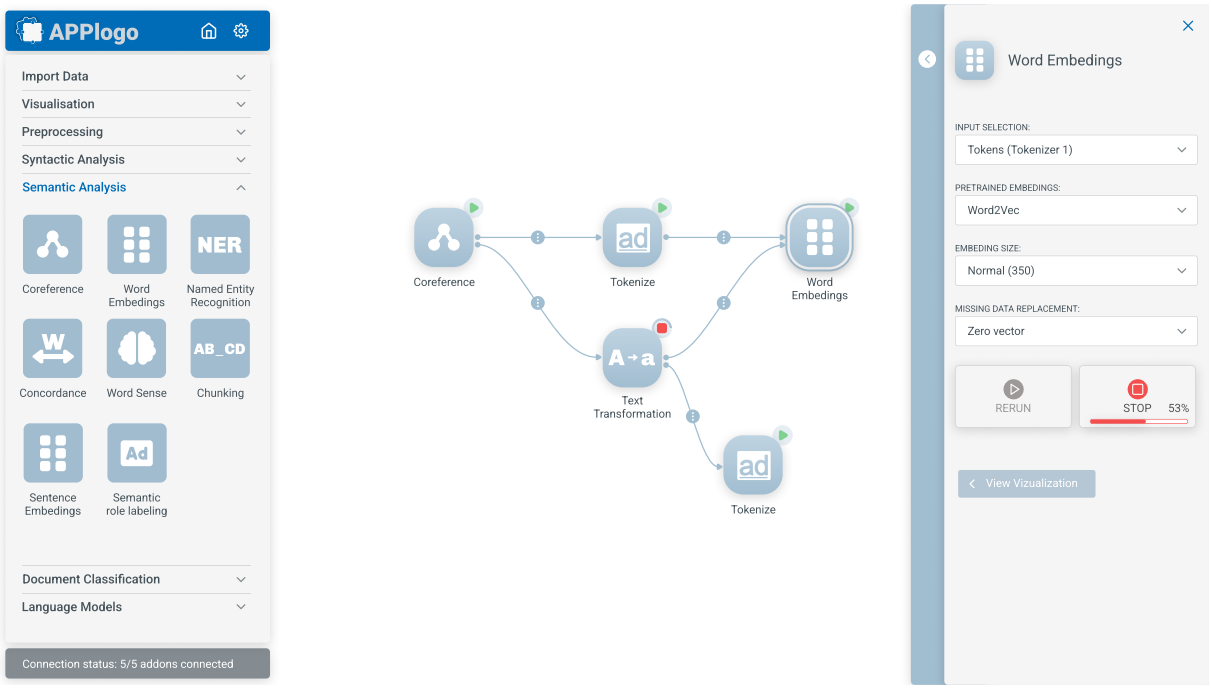


Figure 25: ANGLEr showcase UI (proposal).

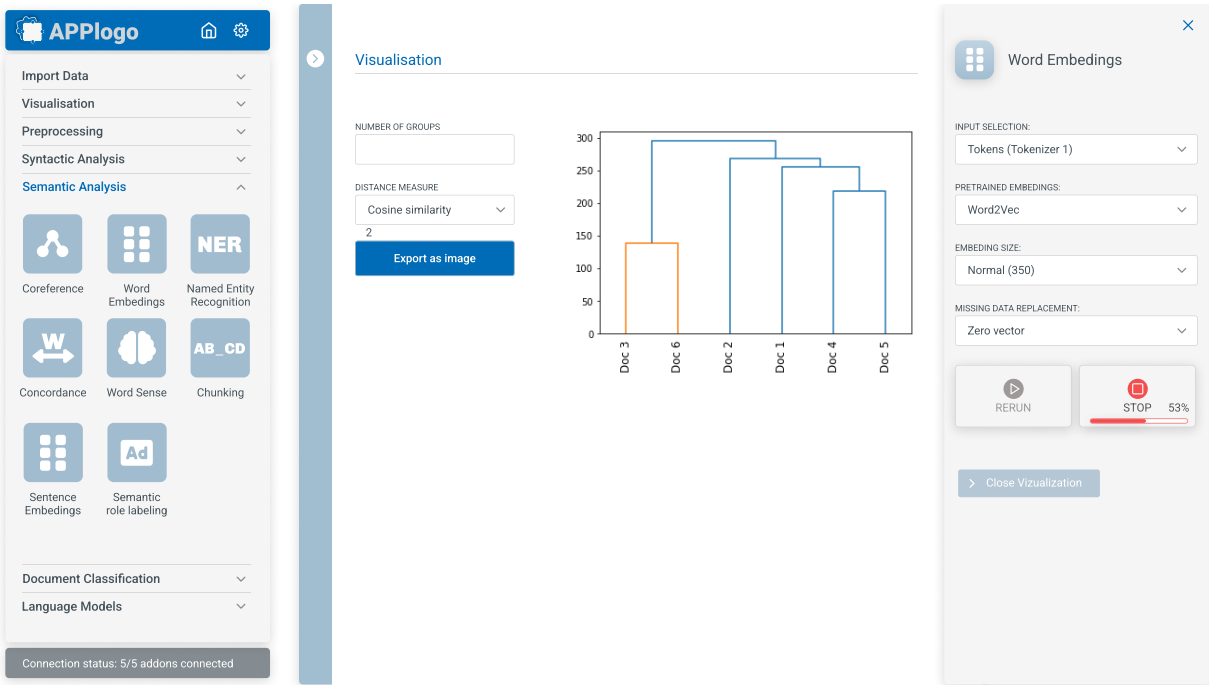


Figure 26: ANGLEr showcase UI (proposal).

[7] Ide, N., Romary, L.: International standard for a linguistic annotation framework. Natural language engineering **10**(3-4), 211–225 (2004)

[8] Knez, T., Bajec, M., Žitnik, S.: Angler: A next-generation natural language exploratory framework. In: International Conference on Research Challenges in Information Science. pp. 761–768. Springer (2022)

[9] Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J.R., Bethard, S., McClosky, D.: The stanford corenlp natural language processing toolkit. In: Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations. pp. 55–60 (2014)

[10] Meystre, S.M., Lee, S., Jung, C.Y., Chevrier, R.D.: Common data model for natural language processing based on two existing standard information models: Cda+graf. Journal of Biomedical Informatics **45**(4), 703–710 (2012). https://doi.org/https://doi.org/10.1016/j.jbi.2011.11.018, translating Standards into Practice: Experiences and Lessons Learned in Biomedicine and Health Care

[11] Nance, J., Baumgartner, P.: gobbli: A uniform interface to deep learning for text in python. Journal of Open Source Software **6**(62), 2395 (2021)

[12] Qi, P., Zhang, Y., Zhang, Y., Bolton, J., Manning, C.D.: Stanza: A python natural language processing toolkit for many human languages. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations. pp. 101–108 (2020)