EMBEDDIA

Cross-Lingual Embeddings for Less-Represented Languages in European News Media

Research and Innovation Action
Call: H2020-ICT-2018-1
Call topic: ICT-29-2018 A multilingual Next generation Internet
Project start: 1 January 2019 Project duration: 36 months

D7.4: Selected EMBEDDIA components in ClowdFlows (T7.4)

Executive summary
ClowdFlows is an open source online platform for developing and sharing of data mining and machine learning workflows. Its unique feature is its pure online operation in Web browsers, without client-side installation. The aim of ClowdFlows is to foster sharing of workflow solutions and simplify replication of and building upon shared work. It is therefore very suitable for demonstration of new approaches, particularly those that are interesting as starting points for building upon or for exposing solutions to potential users of EMBEDDIA solutions that are not proficient in programming, but would like to experiment with their own datasets and parameters. This deliverable introduces ClowdFlows, describes the methods developed in EMBEDDIA that were implemented in a way to be used in this platform, the software components that were added to ClowdFlows in order to support the EMBEDDIA contributions and some exemplary workflows that employ them.

Partner in charge: JSI

Project co-funded by the European Commission within Horizon 2020 Dissemination Level		
PU	Public	PU
PP	Restricted to other programme participants (including the Commission Services)	–
RE	Restricted to a group specified by the Consortium (including the Commission Services)	–
CO	Confidential, only for members of the Consortium (including the Commission Services)	–

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825153

Deliverable Information

Document administrative information	
Project acronym:	**EMBEDDIA**
Project number:	**825153**
Deliverable number:	**D7.4**
Deliverable full title:	**Selected EMBEDDIA components in ClowdFlows**
Deliverable short title:	**ClowdFlows selected components**
Document identifier:	**EMBEDDIA-D74-ClowdFlowsSelectedComponents-T74-submitted**
Lead partner short name:	**JSI**
Report version:	**submitted**
Report submission date:	**31/12/2020**
Dissemination level:	**PU**
Nature:	**R = Report**
Lead author(s):	**Martin Žnidaršič (JSI), Janez Kranjc (JSI), Vid Podpečan (JSI)**
Co-author(s):	**Andraž Pelicon (JSI), Senja Pollak (JSI)**
Status:	**__ draft, __ final, _x_ submitted**

The EMBEDDIA Consortium partner responsible for this deliverable has addressed all comments received. Changes to this document are detailed in the change log table below.

Change log

Date	Version number	Author/Editor	Summary of changes made
16/10/2020	v0.1	Martin Žnidaršič (JSI), Senja Pollak (JSI)	Initial draft
22/10/2020	v0.2	Vid Podpečan (JSI), Martin Žnidaršič (JSI)	Description of ClowdFlows
03/11/2020	v0.3a	Martin Žnidaršič (JSI)	Draft descriptions of widgets
12/11/2020	v0.3b	Martin Žnidaršič (JSI)	Workflow 1
20/11/2020	v0.3c	Andraž Pelicon (JSI), Vid Podpečan (JSI)	Improvements in some widget descriptions
28/11/2020	v0.4a	Janez Kranjc (JSI)	Section on deployment
28/11/2020	v0.4b	Martin Žnidaršič (JSI), Janez Kranjc (JSI), Senja Pollak (JSI)	Final workflows and their descriptions
30/11/2020	v0.4c	Martin Žnidaršič (JSI)	Introduction, Conclusion
20/12/2020	v0.4d	Antoine Doucet (ULR), Marko Pranjić (TRI)	Internal review
26/12/2020	v0.5	Martin Žnidaršič (JSI), Janez Kranjc (JSI), Vid Podpečan (JSI)	Corrections according to internal reviews
28/12/2020	v1.0	Nada Lavrač (JSI)	Report quality checked and finalised
29/12/2020	final	Martin Žnidaršič (JSI)	Final version
29/12/2020	submitted	Tina Anžič (JSI)	Report submitted

Table of Contents

1 Introduction

The EMBEDDIA project aims to develop an array of methods and software solutions for the development and use of multilingual text embeddings. A selection of these software tools, particularly those that are relevant for the media industry, are to be implemented as Web services in the scope of WP6. While Web services can also be used for demonstration and experimentation purposes, they are primarily intended for use by professionals and are suitable for incorporation in industrial solutions. However, there are several outcomes of EMBEDDIA that are primarily of interest to the research community, also from domains of science in which knowledge of programming is not an essential skill. To these kinds of users, an opportunity to explore research experiments and demonstrations as functional and replicable software solutions is very valuable. The aim of Task T7.4 is to make this possible for a selection of EMBEDDIA outcomes in scope of a platform named ClowdFlows. This online platform is primarily intended for development and sharing of data mining and machine learning solutions that are developed in the form of software components and workflows of these components. Most of the EMBEDDIA methods are of these or similar kinds, but there are some specific characteristics, such as use of data in form of textual corpora, exchange of textual embeddings and some others that require specific software to be developed to support the EMBEDDIA solutions in ClowdFlows and their interoperability with the other relevant components.

There are two main kinds of software results from EMBEDDIA to be shared through ClowdFlows: (i) the individual software components (named widgets) that offer specific functionalities and form building blocks of workflows, and (ii) entire workflows that demonstrate the use of EMBEDDIA components in use cases that are relevant to the aims and purposes of EMBEDDIA.

This deliverable presents an intermediate state of integration of software components and workflows from EMBEDDIA in ClowdFlows. The final selection, implementation and integration of components and workflows will be reported in Deliverable D7.6, which is due in M36. We first introduce the platform itself in Section 2, where we describe ClowdFlows and its features and provide essential instructions for its use. The individual software components that were prepared either to showcase methods that originate from EMBEDDIA or are used in the EMBEDDIA experiments are described in Section 3. The entire exemplary workflows are presented in Section 4. Section 5 concludes the deliverable with a brief summary of presented work and plans for further developments.

2 ClowdFlows

ClowdFlows is an online data processing and machine learning platform that is being developed at the Jožef Stefan Institute since 2010. Development of ClowdFlows resulted in the first stable version in the scope of the PhD thesis of Janez Kranjc (Kranjc, 2017). The platform prototype was used for software integration in the EU project MUSE[1]. A fork of the platform called ConCreTeFlows was developed in the scope of the EU project ConCreTe[2] for the purposes of shared development of computational creativity solutions. This fork received many contributions and extensions from developers in the computational creativity community and served its purpose successfully (Žnidaršič et al., 2016). Another dedicated fork of the platform was TextFlows (Perovšek, Kranjc, Erjavec, Cestnik, & Lavrač, 2016), which was aimed at text-mining and natural language processing tasks. Operations with large textual corpora, however, were difficult to handle efficiently with the original workflow execution engine, which was one of the reasons for re-development of ClowdFlows into its latest version ClowdFlows3 (CF3). The current development of CF3 is to a large extent influenced by the needs of its use in EMBEDDIA, research dissemination and teaching.

The main idea with an explanation of its features is provided in Section 2.1. The reasons for using ClowdFlows in EMBEDDIA are explained in Section 2.2. In Section 2.3 we introduce the latest public version of the platform in a bit more detail.

[1] https://cordis.europa.eu/project/id/296703
[2] https://cordis.europa.eu/project/id/611733

2.1 Background

ClowdFlows (Kranjc, Podpečan, & Lavrač, 2012) is an online data processing platform with a graphical user interface that can run in any browser and does not require local installations. It was made to support data analytics and machine learning in the cloud. The user interface allows combination of software components (called widgets) into functional workflows, which can be executed in the cloud, stored, and shared.

Tools for composition of functional components into workflows often use visual programming for their user interface, which makes them suitable also for non-experts by representing complex procedures as sequences of simple processing steps. ClowdFlows also employs such an interface and can be compared to other data mining workflow management tools, such as Weka (Witten & Frank, 2005), Orange[3] (Demšar et al., 2013) and RapidMiner (Mierswa, Wurst, Klinkenberg, Scholz, & Euler, 2006; Hofmann & Klinkenberg, 2016). The ClowdFlows environment, compared to other existing workflow processing tools and platforms has several advantages but its main distinctive feature is its pure Web application design, which allows use in Web browsers without the need for any kind of additional software installation. ClowdFlows is open source software, it allows for workflow sharing at the ClowdFlows instalation site itself thus making them always available (unlike the most widely spread platforms) and makes workflows easy to compose and execute. Sharing of workflows is a feature already implemented at the myExperiment portal (Roure, Goble, & Stevens, 2009), or at the ever more useful and popular OpenML initiative[4] (Vanschoren, Van Rijn, Bischl, & Torgo, 2014), which allow the users to publicly upload and share their workflows. However, the users that wish to execute or edit these workflows must still install specific software in which the workflows were designed. Remote workflow execution is also employed by RapidMiner on the RapidAnalytics server. This allows the execution on servers and data sharing with other users. But also in this case, the client software must be installed on the user's computer.

There are two ways to use ClowdFlows: (i) use of the publicly deployed version and (ii) use and development of the local version. For the first kind of use, it suffices to visit the published URL with a Web browser, while for hosting of a dedicated platform and its development, one must download the code and set the hosting infrastructure. For mere use of the existing components and development and execution of workflows, the publicly deployed version is more appropriate. This instantiation of ClowdFlows runs on a dedicated server and its latest public version is available at:

`http://cf3.ijs.si`

In order to use it, no installation is needed as it runs in a Web browser. Its home page is shown in Figure 1 . This online version is regularly updated from the common code base, whenever its state is considered stable.

2.2 Purpose in EMBEDDIA

The main purpose of employing a visual workflow composition platform like ClowdFlows in EMBEDDIA is to cater to the needs of users that are not familiar with programming, but would still like to use the methods that are developed in EMBEDDIA on their own data or would benefit from having the ability to experiment with parameters and methodological alternatives. Most of the state-of-the-art solutions in domain of text analysis and natural language processing are provided in form of command line tools, programming language libraries or Web services. These kinds of implementations offer flexibility and efficiency of use, which are essential for their application in demanding real-world tasks or their reuse and integration into more complex solutions or experiments. However, while use of such implementations is seamless for people with programming skills, it represents a hurdle for potential users that lack the necessary technical knowledge. There are many users of this kind in domains of research and practice that are relevant for EMBEDDIA: from linguistics to social sciences and journalism. A benefit of visual

[3] Also supports work with textual embeddings: `https://orange.biolab.si/blog/2020/2020-10-15-document-embedders/`
[4] https://www.openml.org

Figure 1: ClowdFlows home page.

workflows is also their representation of processes, which is done in a way that is to some people more understandable than programming code and therefore serves also didactic or explanatory purposes. In case of ClowdFlows, of course, the use is further simplified by the fact that such workflows can be executed, explored and redesigned without any kind of client-side installation.

The second reason for using ClowdFlows in EMBEDDIA is to simplify sharing of developed methods and experiments. Namely, workflows in ClowdFlows can be made public, which allows anyone to inspect, re-run and build upon them. This allows for easy reimplementation, transparency and replicability of experiments, which are all important for dissemination of our work in the research community.

Finally, ClowdFlows was a tool of choice also because of our substantial development experience with the platform, which allows for providing support for specific demands of EMBEDDIA and its solutions.

2.3 ClowdFlows3

The latest generation of the ClowdFlows platform is named *ClowdFlows3*. This is the latest stable version that is currently being developed. There are numerous workflows that were developed with specific components of the previous generation of the platform, so the previous generation platform is still maintained at: `http://clowdflows.org/`

ClowdFlows2 was a short-lived development version, which was never publicly deployed and is not an official version. It was meant to address some inefficiencies of data interchange and some issues of the backend-frontend relation, but with backwards compatibility (to allow all the workflows that were developed for the first version to be transferred seamlessly to the new one), which proved to be a major stumbling block. ClowdFlows3 was therefore developed anew.

As of ClowdFlows3 the platform became modular. The backend and the frontend are now completely separated. They communicate through a well-defined API exposed by the backend. The frontend is written in AngularJS while the backend is written in Python and Django. The backend consists of the main module named *clowdflows-backend* which provides the backbone, and a number of modules that

performs the actual work and reside in separated packages[5] such as:

- *cf_core* which provides core utilities and widgets;
- *cf_text_embeddings* which provides text embeddings componenents;
- *cf_data_mining* which provides data mining components using the scikit-learn library;
- *rdm* which provides components for relational data mining, etc.

ClowdFlows3 has also a new workflow engine, which does not necessarily store data to the database at each step of the execution (at each widget). Which steps get stored and which not can now be decided by the user (by pressing a database sign at the selected widget). These changes to the workflow engine speed-up execution and were mainly made to accommodate execution of EMBEDDIA workflows, which operate with large datasets and exchange a lot of parameters among their components.

In comparison with older versions, ClowdFlows3 supports two modes of deployment: (a) as a Django project and (b) using Docker. The former is suitable for programmers developing new packages or improving existing ones while the later is suitable for end users who just want to run ClowdFlows3 on their own infrastructure with minimal effort.

2.3.1 Brief instructions for using ClowdFlows3

The opening screen of ClowdFlows is shown in Figure 1. If a user is not logged in ClowdFlows already, any action on this screen presents the registration and log-in view as shown in Figure 2. Existing users can log in ClowdFlows there, while new users can register as ClowdFlows users by filling in the registration form.

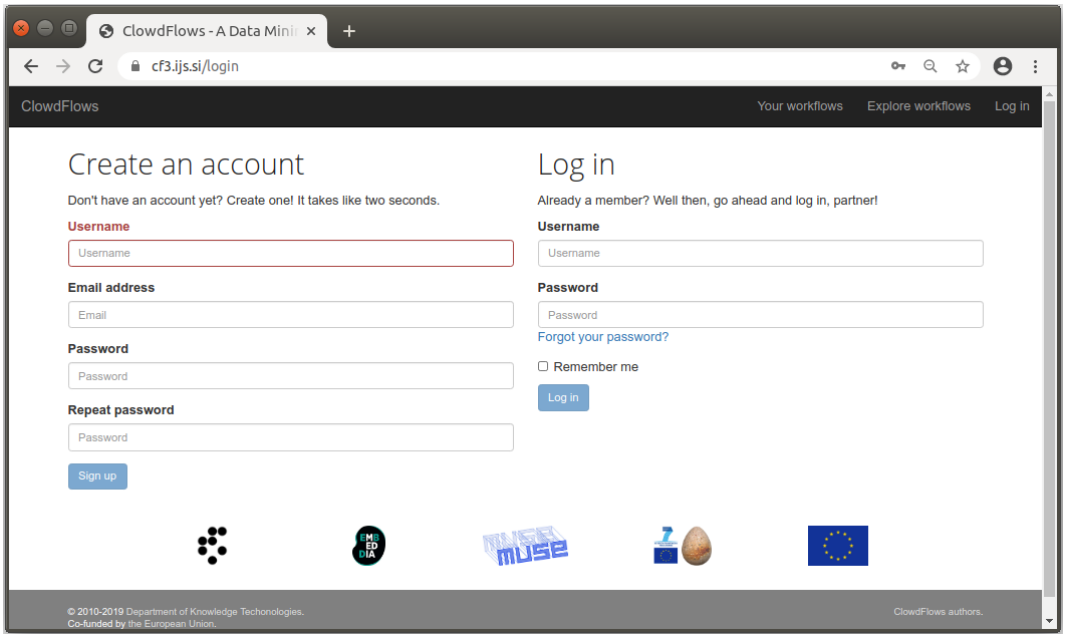

Figure 2: Registration and log in page.

After logging in, one can select to try the tutorial, construct a new workflow from scratch, or explore either the collection of our own workflows or the collection of the publicly available ones.

[5]These are regular Python packages that can be installed using **pip**, the Python package manager. Some of them can also be used outside of ClowdFlows.

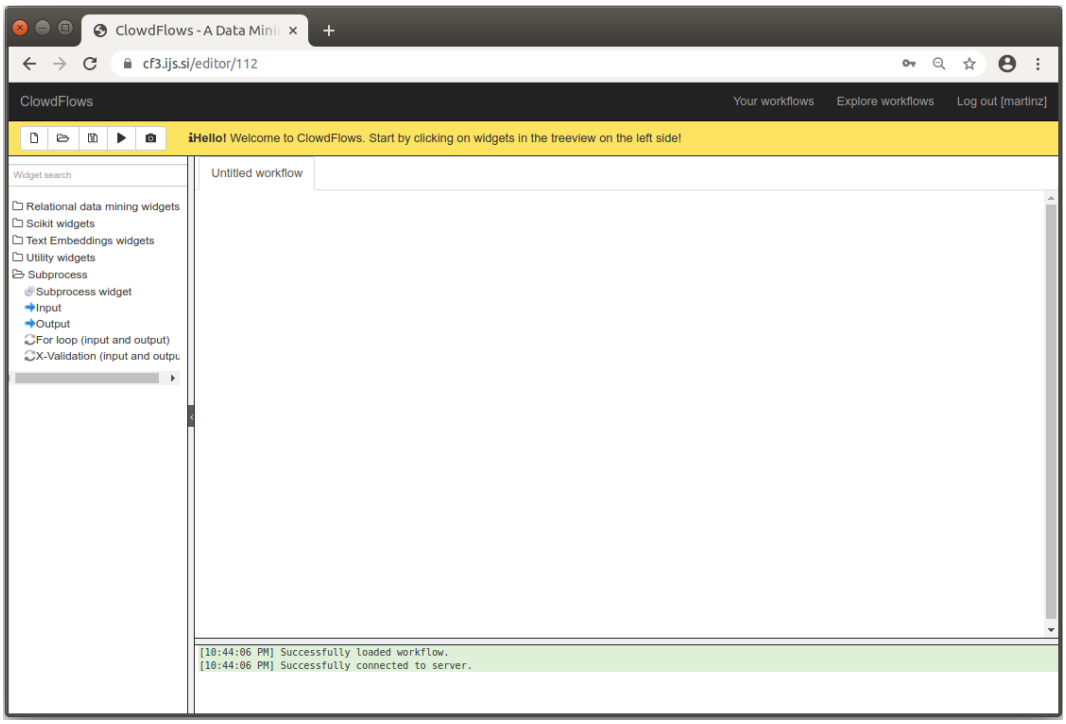

Figure 3: The main screen with workflow development canvas and widget collection.

The workflows get constructed on the main view with the workflow canvas, widget collection and control buttons, which is shown in Figure 3. The widgets get put on the canvas by either selecting them in the collection and dragging them to the canvas or double clicking them in the collection. On the canvas, the outputs of the widgets can get connected with inputs of other (suitable) widget in a way that allows sensible data-flow. A workflow with connected widgets in ClowdFlows is shown for example in Figure 7 on page 15. Connections among the widgets are made by clicking on the output of one widget and input of another. Inputs to a widget are represented visually as light blue rectangles on the left side of the widget and outputs are represented as such rectangles on the right. The rectangles have a short label that indicates the name of the widget's input or output, such as the label *txt* of the *ML BERT for hate speech* widget in Figure 4. Connections can be removed by right-clicking on the links and selecting *Remove*.

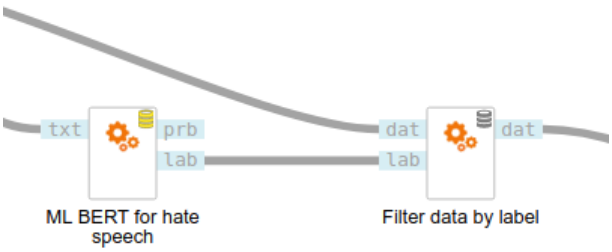

Figure 4: Exemplary connected widgets.

A workflow can be executed in its entirety by clicking the *play* control button. The widgets that are already executed get a green check-mark beneath them (see Figure 6). Such widgets will not get executed again, if the widgets leading to them or their parameters do not change. In such situations,

only the new widgets or widgets that are affected by any change in the workflow get executed. Individual widgets can be marked to re-execute by right-clicking them and selecting *Reset*. Also entire workflow can be marked this way by selecting *Reset workflow* at any widget. Workflows can also be executed only partly. If we right-click on a widget and select *Run*, only the widgets that are necessary for providing the input to this widget get executed.

Widgets can also be set to store their results in a database which is particularly useful when working with lenghty calculations and large corpora, such as the ones often used in EMBEDDIA. Storing to a database is set by clicking on a database cylinder symbol, which becomes yellow to indicate that results of such a widget will be stored (see for example the widget *ML BERT for hate speech* in Figure 4), so the widget will not get executed again in full or partial workflow executions, unless the widgets that provide its inputs were changed.

Workflows are auto-saved, but can be also saved explicitly and named, by pressing the *save* control button. All the user's workflows are available in the view that the user gets upon pressing *Your workflows* in the top right corner of the main view window. This view outlines the user's workflows, with workflow management options as shown in Figure 5. There, the user can open a selected workflow for editing, open its copy, delete a workflow, export it to JSON, or make it public. Public workflows can be shared by distributing their URL, which allows every ClowdFlows user with such an URL to have their own copy of the shared workflow and use it further. Public workflows also appear in the *Explore workflows* view of all the ClowdFlows users.

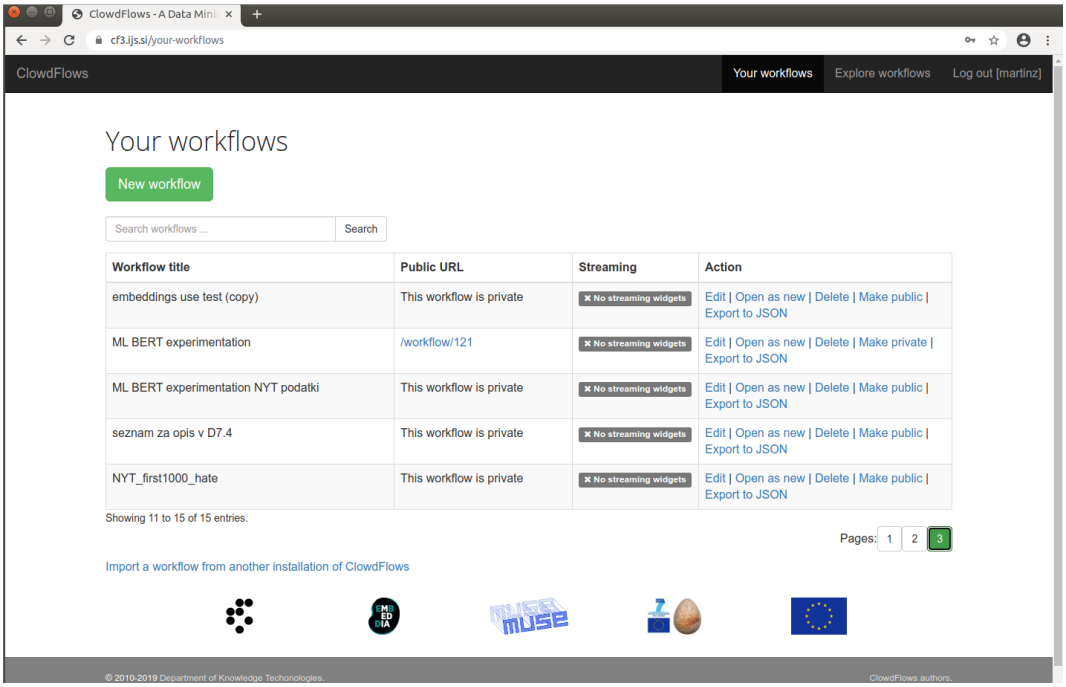

Figure 5: Windows with user's workflows.

2.3.2 Instructions for deployment

The infrastructure of ClowdFlows consists of several services, which need to be installed on a server in order to deploy the application.

These are the services:

ClowdFlows Frontend
The frontend is written in JavaScript using the Angular Framework. The frontend consists of markup

files, stylesheets and script files that all run on the client's browser. These files must be served with a Web server in order to be accessible to end users. The code for the frontend is located on GitHub: https://github.com/xflows/clowdflows-webapp.

ClowdFlows API Server
The API server is the backbone of ClowdFlows and processes all requests made by the user and interacts with the distributed workers and the database. It is written in Python using the Django Framework. The code for the backend is hosted on GitHub: https://github.com/xflows/clowdflows-backend.

WebSocket server
The WebSocket server is required to provide real-time updates to the graphical user interface. These updates include workflow execution status updates and information about the workflows themselves (e.g. positions of the components). The repository for the backend includes the code for running the WebSocket server.

Distributed Workers
These workers execute the workflow components. The scalability of the ClowdFlows platform is directly influenced by the number of distributed workers running concurrently. During larger load times it is sensible to run more workers to ensure platform stability and functionality. The repository for the backend includes the code for running the distributed workers.

Redis
This is an in-memory data store used as a cache and message broker. The workers are using it to communicate with each other.

Relational database
Workflows and the data on them are saved in a relational database. The platform itself is database agnostic and supports the following databases: PostgreSQL, MySQL, Oracle, MariaDB and SQLite. In our deployments we use PostgreSQL because of its DDL (data definition language) transaction support, which is crucial to allow updates of the application while some users might be using it.

Specific deployment instructions
There are two possibilities to deploy and run ClowdFlows: as a Django project or using Docker. The former is suitable for programmers developing new packages or improving existing ones; the latter is suitable for end-users who want to run ClowdFlows on their own infrastructure. Both of these methods require all the previously described services running either on a development computer or on a production server.

The services can be run using Docker so getting a development version up and running is a matter of cloning a repository containing configuration files and running a single command. The configuration files are written for the Docker Compose tool. Compose is a tool for defining and running multi-container Docker applications. We have written a YAML file that configures all ClowdFlows' services and created a repository where it is located (https://github.com/xflows/clowdflows-docker). Then, with a single command, all the services from the configuration are created and started.

These configuration files can be augmented to also include a reverse proxy to expose the client facing services behind a domain name and automatically acquire a SSL certificate for secure communication. Such an example configuration can be found on the GitHub repository for the cf3.ijs.si installation (https://github.com/xflows/cf3.ijs.si), where a registered user can create, run, and share scientific workflows. In order to migrate this installation to another server or server cluster it is only required to change all references to this domain name in the configuration file to a custom domain name where a new installation will be served from. A single command runs all the services necessary to serve a production deployment of ClowdFlows.

3 EMBEDDIA components

The software components that form units of graphical workflows are denoted as widgets. This section outlines the widgets that were either developed in the scope of the work in the EMBEDDIA project, or were developed to support the workflows that are aimed at showcasing the concepts and solutions of EMBEDDIA.

3.1 Document based embedding

The BERT widgets introduced in this section perform tokenization internally, so the inputs to them are documents, which is why they are listed as document embedding widgets.

BERT
A widget that implements the pre-trained BERT : Bidirectional Encoder Representations from Transformers (Devlin, Chang, Lee, & Toutanova, 2018).

BERT Embeddia
A widget that implements a trilingual BERT model, trained on Croatian, Slovenian, and English data. The neural network weights and configuration files in pytorch format (ie. to be used with pytorch library) are available online[6].

ML BERT for hate speech
A widget that implements a multilingual BERT for hate speech classification. The model was trained on the training subset of the OLID dataset. It was evaluated on the test subset of the OLID dataset (Zampieri et al., 2019) using their official gold labels and on the test subset of the GermEval 2018 dataset (Wiegand, Siegel, & Ruppenhofer, 2018). In both datasets, only first level annotations (from the three-level hierarchical annotation scheme), which classify tweets into offensive and not offensive classes, were used. The second level annotations then categorize offensive language into targeted or untargeted, and the third level identifies the target. Input into this widget is a list of documents (strings), such as the one created by the *Load corpus from CSV* widget. There are two outputs: *lab* contains the list of prescribed labels ('hate' or 'normal') for each document, and *prb* contains the list of probabilities associated with the labels for each document. Examples of inputs and outputs for this widget are presented in Figure 6.

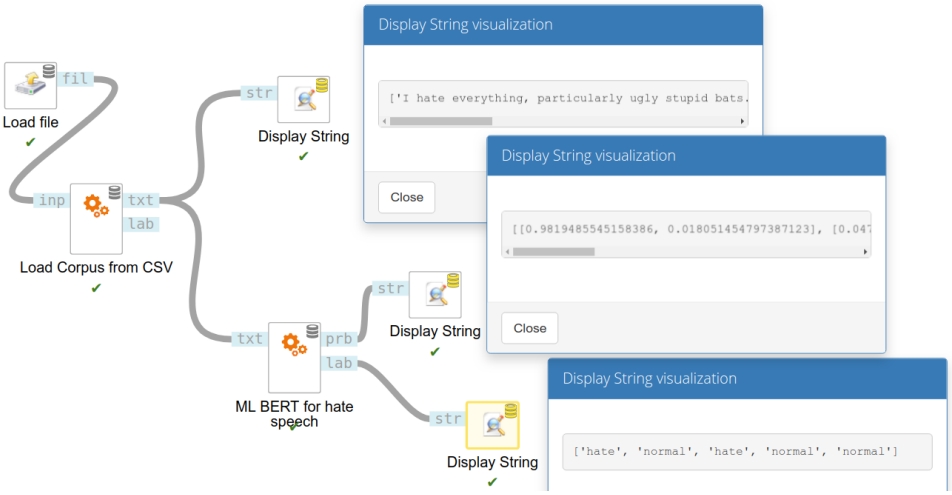

Figure 6: Input and output examples for the *ML BERT for hate speech* widget.

[6] https://www.clarin.si/repository/xmlui/handle/11356/1317

ML BERT for sentiment classification
A widget that implements a multilingual BERT for news sentiment classification. The model was trained on the Slovenian news sentiment dataset (Bučar, Žnidaršič, & Povh, 2018) using a two-step training approach, described in (Pelicon, Pranjić, Miljković, Škrlj, & Pollak, 2020b). The training of the model was done using labels on the document and paragraph levels. The model was subsequently tested on the document-level labels of the Croatian news sentiment dataset (Pelicon, Pranjić, Miljković, Škrlj, & Pollak, 2020a) in a zero-shot setting. The possible labels prescripted to documents are 'positive', 'negative' and 'neutral'. In the deployed implementation, there is a minor difference with respect to the approach described above - shortening of too long documents is done in a way to consider only the first parts of documents, while in the original approach, a portion from the start and the end is retained. This change was done to simplify and speed-up the code. The inputs and outputs of this widget are of the same kind as the inputs and outputs of the *ML BERT for hate speech*.

Doc2Vec
A widget that implements *Doc2vec* - an unsupervised algorithm to generate vectors for sentences, paragraphs and documents. The algorithm is an adaptation of *Word2Vec* which can generate vectors for words.

LSI
A widget implementing a fast truncated SVD (Singular Value Decomposition). The SVD decomposition can be updated with new observations at any time, for an online, incremental, memory-efficient training.

3.2 Sentence based embedding

ELMo
ELMo is a deep contextualized word representation that models complex characteristics of word use (e.g., syntax and semantics) and how these uses vary across linguistic contexts (Peters et al., 2018). The word vectors are based on a deep bidirectional language model, which is pre-trained on a large text corpus. They were successfully used in solutions to various natural language processing problems.

ELMo Embeddia
ELMo language model (https://github.com/allenai/bilm-tf) used to produce contextual word embeddings, trained on large monolingual corpora for 7 languages: Slovenian, Croatian, Finnish, Estonian, Latvian, Lithuanian and Swedish. Each language's model was trained for approximately 10 epochs. Corpora sizes used in training range from over 270 M tokens in Latvian to almost 2 B tokens in Croatian. About 1 million most common tokens were provided as vocabulary during the training for each language model. The model can also infer out-of-vocabulary words, since the neural network input is on the character level.

Universal Sentence Encoder
The Universal Sentence Encoder encodes text into high dimensional vectors for use in text analysis (Cer et al., 2018). The model is trained and optimized for greater-than-word length text, such as sentences, phrases or short paragraphs. It is trained on a variety of data sources and a variety of tasks with the aim of dynamically accommodating a wide variety of natural language understanding tasks.

3.3 Word based embedding

GloVe
GloVe provides vector representations of words with unsupervised learning (Pennington, Socher, & Manning, 2014). Training is performed on global word co-occurrence statistics from a given corpus.

Word2Vec
Word2vec is a group of related models that are used to produce word embeddings. These models are

shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space. Word vectors are positioned in the vector space so that words that share common contexts in the corpus are located closely together in the word space.

fastText
fastText is an open-source library for learning of word embeddings and text classification (Bojanowski, Grave, Joulin, & Mikolov, 2016; Joulin, Grave, Bojanowski, & Mikolov, 2016).

fastText Croatian
CLARIN.SI-embed.hr contains word embeddings induced from a large collection of Croatian texts composed of the Croatian Web corpus hrWaC and a 400-million-token-heavy collection of newspaper texts. The embeddings are based on the skip-gram model of fastText trained on 1,852,631,924 tokens of running text for (1) 1,742,837 lowercased surface forms (e.g., "hrvatske") and (2) 1,404,515 lowercased lemmas with added part-of-speech information (e.g., "hrvatska#Np").

fastText Embeddia
These are fastText embeddings trained on Slovenian Gigafida 2.0 corpus. A skipgram model was trained with default hyperparameters on 8 threads, except for the following two changes: dim parameter was set to 300 and minCount parameter was set to 20. That is, we calculated 300-dimensional word vectors of every word that appears at least 20 times in the corpus. Each line in the .vec file consists of the word, followed by the 300 dimensional vector, all fields are space separated. The first line (642655 300) tells, there are 642655 word vectors of 300 dimensions.

fastText Slovenian
CLARIN.SI-embed.sl contains word embeddings induced from a large collection of Slovene texts composed of existing corpora of Slovene, e.g GigaFida, Janes, KAS, slWaC etc. The embeddings are based on the skip-gram model of fastText trained on 3,557,125,771 tokens of running text for (1) 2,466,596 lowercased surface forms (e.g., "slovenije").

3.4 Tokenizers

Punkt Sentence Tokenizer
This tokenizer divides text into a list of sentences by using an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences. Before use, it must be trained on a text collection in the target language.

Regex Word Tokenizer
The regex word tokenizer is a simple tokenizer that tokenizes a document with '\w+' regex to words. It can also transform document text to lowercase.

Tok Tok Word Tokenizer
The tok-tok tokenizer is a popular simple multilingual tokenizer (Dehdari, 2014).

3.5 Utility widgets for embeddings

Concatenate embeddings
This widget concatenates collections of embeddings. The number of its inputs dynamically increases to allow for an arbitrary number of collections to be concatenated together. For example, in Figure 7 we can see a concatenation of four collections. If a fifth collection would be added, a sixth input to this widget would appear to allow further additions.

Create Dataset
Combines embeddings and labels to a dataset of type *Orange Data Table*.

Create scikit bunch
Creates a scikit bunch. This is a temporary solution because scikit bunch is deprecated as of 0.22.

Export dataset
Exports embeddings to a file.

Import dataset
Import dataset from a NumPy matrix of embeddings and NumPy matrix of labels.

Language
This widget can be used to set a language to multiple Text Embeddings widgets.

Load Corpus from CSV
This widget loads a file and extracts the document texts and labels into two lists. You can specify to skip the first header row and specify the delimiter. In the case of tab-separated file, use \t as a delimiter. You can also define the text and label columns (counting of the columns starts with 1).

Token Filtering
Token Filtering widget filters uninformative characters. The default filter filters:

```
!, ", #, $, %, &, "", (, ), *, +, ,', -, ., /, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, :, ;, <, =, >, ?,
@, [, \, ], ^, _, {, |, }.
```

The user can specify custom tokens to filter, which need to be separated by a new line. Custom tokens override the default filter.

3.6 Selection of other utility widgets

In the following we list descriptions of some additional utility widgets, which were not developed for working with embeddings, but they appear in the workflows shown in Section 4. Some of them (*Filter data by label* and *Join list of strings*) were developed specifically to cover the needs of EMBEDDIA workflows.

Load file
Uploads a file to the ClowdFlows server. Outputs the file name on the server.

Display String
A trivial widget that displays a string from its input.

Filter data by label
This widget filters the data according to a given label. The inputs are *dat*: a list of data items, and *lab*: a corresponding list of labels. The label to use for filtering is a parameter of the widget, which can be provided by the user upon double-clicking on the widget. Result is a list of only those data items that have a specified label.

Join list of strings
This widget takes a list of strings from its input and concatenates them into a single string with spaces put between the list items.

Word cloud
Displays a word cloud based on the input string. If the string is a named line documents file (each line starts with !classname) a wordcloud will be displayed for each class. Notice: the *Word cloud* widget has a specific requirement: it only works properly, if the (string) output of the widget that provides it with input is set to store the result. For example, see the store setting (yellow stack) in widgets *Join list of strings* in the workflows in Figures 8 and 9.

4 EMBEDDIA workflows

In this section we present some exemplary workflows, that is, entire experiments or solutions that incorporate and demonstrate the use of several software components. The first presented workflow is primarily a demonstration of experimentation possibilities in ClowdFlows, but it also contains some of the early results of the work in EMBEDDIA. The next two workflows present how a software component from EMBEDDIA can be used to form a ready-made software solution or a basis for a study.

4.1 Embeddings experimentation workflow

The workflow *embedding use test* that is shown in Figure 7 and shared at: `https://cf3.ijs.si/editor/91` demonstrates how ClowdFlows can be used to experiment with variuos combinations of machine learning algorithms and sets of embeddings.

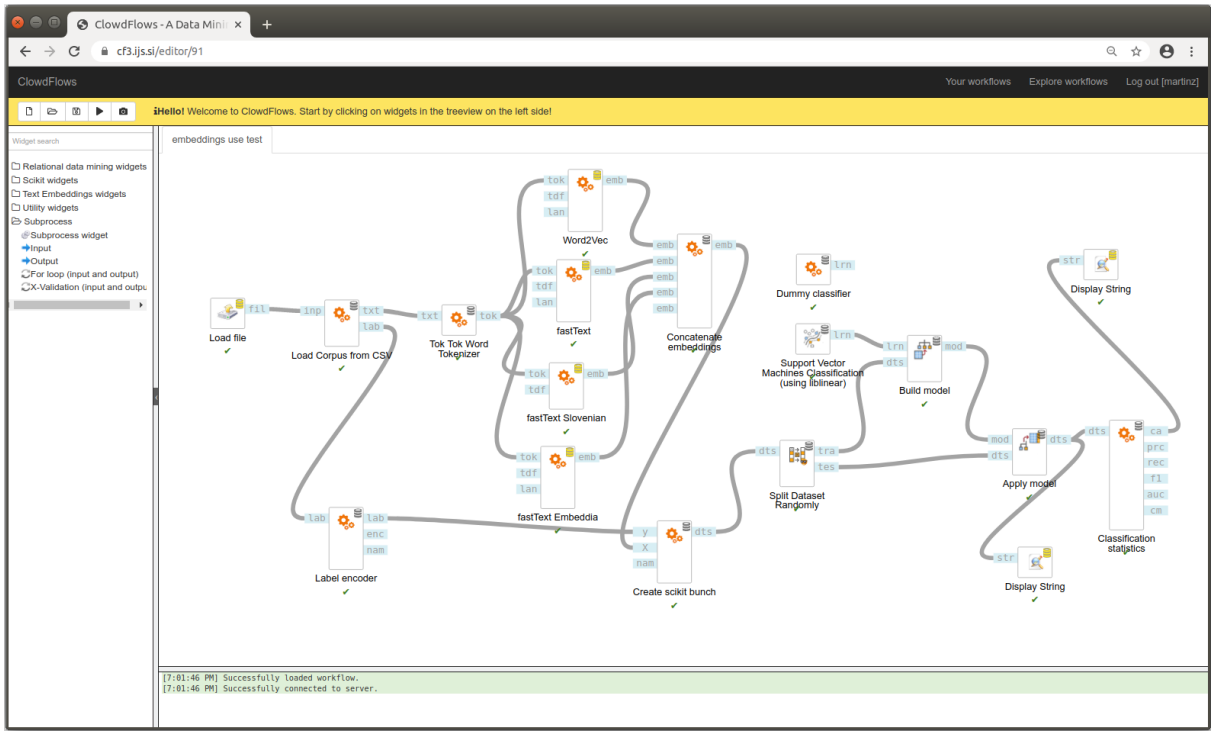

Figure 7: Workflow *embedding use test* that demonstrates experimentation with the use of different embeddings.

In this workflow we first load a file that contains news (in Slovene) and their sentiment as prescribed by the annotators. As we only need the text and sentiment fields from this CSV file that contains also some other fields (such as news title, etc.), the *Load Corpus from CSV* widget is employed, which filters the fields of interest into separate lists of texts and their associated sentiment labels. The text is used as input in a number of methods that create embeddings. These are then concatenated together and represent a feature space for the machine learning problem. These features and the labels (that go also through a suitable transformation widget) are used by the *Create scikit bunch* widget to create a scikit-learn dataset. Namely, the machine learning components that follow are based on the scikit-learn[7] library. The dataset is then split randomly into a training set and test set. The *Build model* widget uses a (training) dataset and a learner algorithm to learn a model - in this case a classifier. The learned model is applied to the test part of the initial dataset in the *Apply model* widget in order to evaluate its performance. The output is used by *Classification statistics*, which provides various performance measures of this combination.

[7] https://scikit-learn.org

By connecting various learners, like the *Dummy classifier* instead of *Support Vector Machines Classification*, one can test performance of various machine learning methods with the provided collection of embeddings. By adding or removing embedding creation methods one can also analyse the effect of various embeddings (individual, or various combinations) on classification performance.

4.2 Multilingual hate speech and sentiment detection workflows

In the following we present two similar workflows that demonstrate the use of the *ML BERT for hate speech* and the *ML BERT for sentiment classification* widgets.

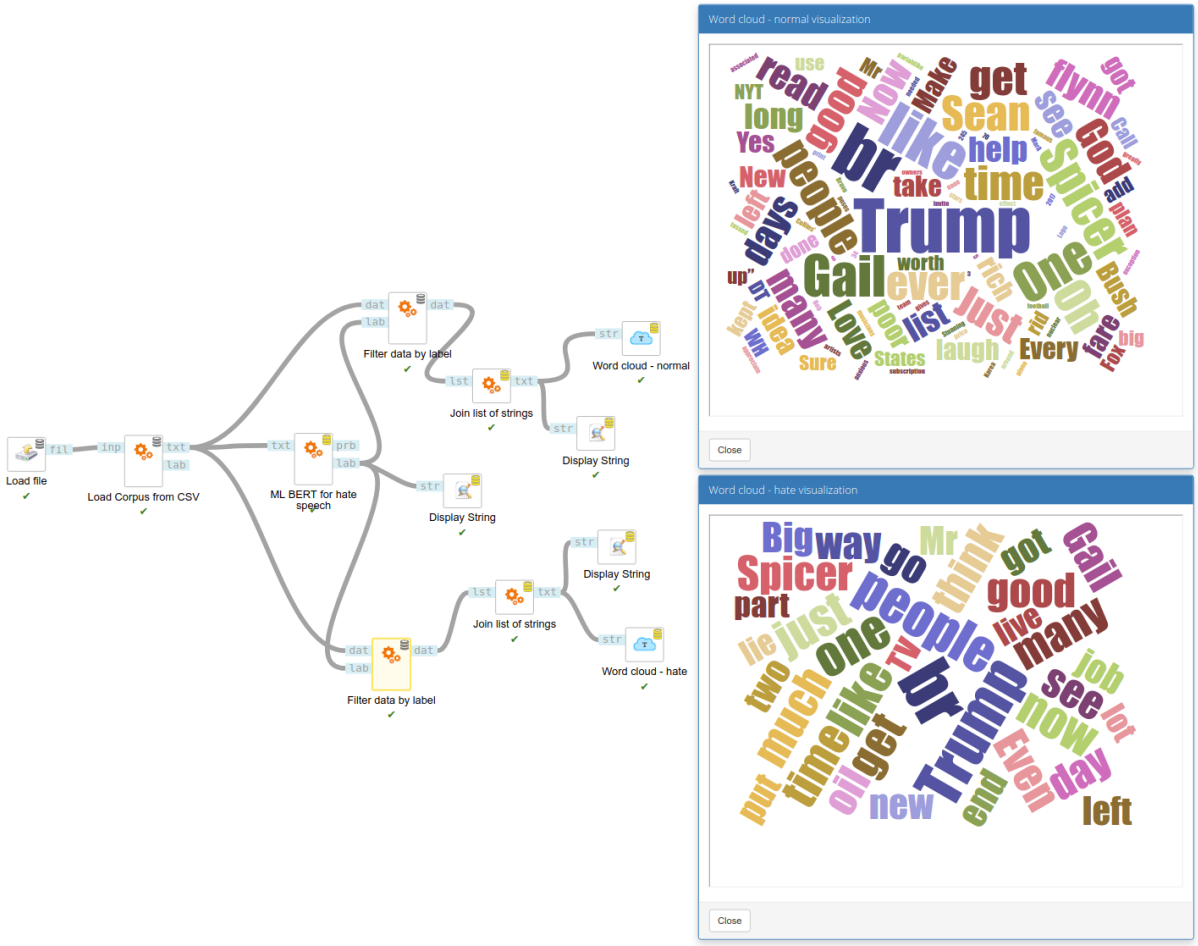

Figure 8: Workflow that shows how the *ML BERT for hate speech* widget and some supporting widgets can be used to classify and study hate speech in text. An example of use with the first 1000 comments from the April 2017 NYT comments dataset is shown. The top word cloud is done from comments that were classified as 'normal' and the bottom word cloud is made from comments that were classified as 'hate'.

The hate speech detection workflow[8] from Figure 8 first loads a file of news comments, specifically the first 1000 comments from April 2017 from the New York Times Comments dataset[9]. This data is not labeled, so the *Load Corpus from CSV* widget is only used to filter out the comment texts. The labels are then provided by the *ML BERT for hate speech*. The original texts are then filtered according to the labels provided, in order to obtain separate collections of either *hate* and *normal* speech. Finally, the

[8] https://cf3.ijs.si/workflow/133

[9] This dataset is available at: `https://www.kaggle.com/aashita/nyt-comments` under CC BY-NC-SA 4.0 license which applies also to the sample used here.

collections (lists) of comments are joined together into simple text which is represented in form of a word cloud.

The sentiment classification workflow[10] from Figure 9 uses a dataset of news articles from the *24sata*[11] news site. This dataset contains also sentiment labels, but instead of these, we use the ones provided by the *ML BERT for sentiment classification* widget. Similarly as in the hate speech detection workflow, the data is filtered according to these labels and the word clouds are created. The word clouds of the articles that were classified as expressing a *positive* or a *negative* sentiment are shown in the middle of Figure 10. The word clouds point to an issue in their input data processing - namely they are not very indicative of the sentiment and mainly contain the so-called *stop-words*, words which are very common and do not contain much meaning. While removal of such words might not be wise before the classification step, it makes sense before the word cloud visualization. There is, however, no stop-word removal widget in ClowdFlows yet. That is why we also added the display of the classified article texts. The positive ones are on the let part of Figure 10 and indeed talk about a famous person enjoying on the beach, some amazing achievements of Croatian sportsmen[12] and similar topics. The articles classified as negative are on the opposite side of the picture and talk about casualties of bad weater in Asia, drunk driving, etc. The results indicate that the *ML BERT for sentiment classification* widget classifies the texts well, although it was tuned on Slovenian news and is employed in this workflow on texts in Croatian language.

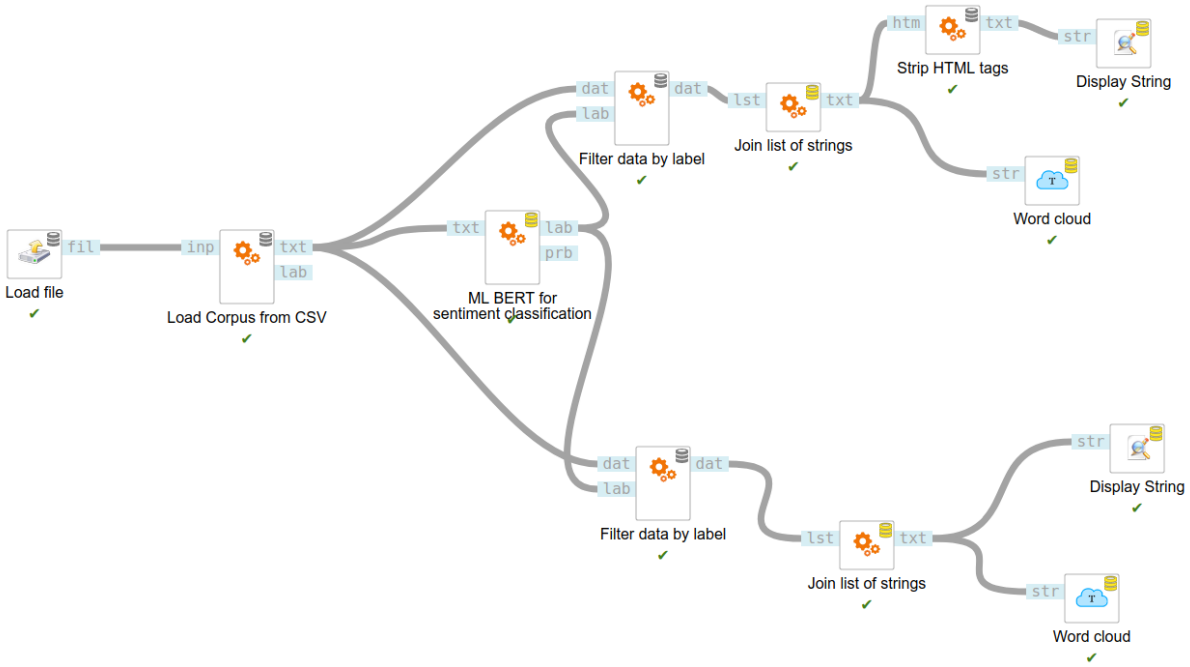

Figure 9: Workflow that shows how the *ML BERT for sentiment classification* widget can be used to classify text with respect to sentiment. An example of use with a collection of news articles from the Croatian news site *24sata* is shown. Some results of this workflow are shown in Figure 10.

These two workflows can be altered and extended in various ways to study other features of the classified collections of comments and news, to test other classification approaches or to simply run the workflows on other datasets.

[10] https://cf3.ijs.si/workflow/137
[11] https://www.24sata.hr/
[12] One might argue that due to frequency of such events these texts should be considered normal or neutral in Croatia.

Figure 10: Results of the sentiment detection workflow shown in Figure 9. Articles that were classified as expressing positive sentiment are shown in display on the left and are represented by the top word cloud. Texts classified as negative are in the right display and have a corresponding word cloud on the bottom.

5 Conclusions and further work

This document introduced the online platform that is used for dissemination of EMBEDDIA methods and solutions among the research community and other interested public prone to experimentation. We outlined the currently integrated software components from EMBEDDIA and presented illustrative workflow examples to showcase the potential of this kind of presentation of results.

To further improve the suitability of ClowdFlows for dissemination of methods that are developed in EMBEDDIA and similar projects, we will first improve its documentation and develop more ready-made solutions. Some additional common utility widgets also need to be added, such as stop-word filters. Afterwards, we plan to engage interested target users to provide feedback, needs and preferences to which the platform should be adapted.

In further work on task T7.4 we will keep adding relevant outcomes from EMBEDDIA as widgets to ClowdFlows. A lot more focus will be on development of workflows, which will be made to serve two main purposes: to demonstrate the results of the work in the scope of the EMBEDDIA project, and to offer interesting and valuable solutions to researchers and practitioners in fields of research that are to be supported by EMBEDDIA. The final collection of EMBEDDIA components and workflows to be integrated in ClowdFlows in the scope of the project will be reported in Deliverable D7.6, but if the aims of this task will be reached, new workflows based on these components should be actively developed also long after that.

6 Associated outputs

The work described in this deliverable has resulted in the following resources:

Description	URL	Availability
ClowdFlows3 (online Web application)	https://cf3.ijs.si/	Public
ClowdFlows3 (server-side code)		
backend	https://github.com/xflows/clowdflows-backend	Public (MIT)
frontend	https://github.com/xflows/clowdflows-webapp	Public (MIT)
Docker version	https://github.com/xflows/clowdflows-docker	Public (MIT)

References

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2016). Enriching word vectors with subword information. *CoRR*, *abs/1607.04606*. Retrieved from http://arxiv.org/abs/1607.04606

Bučar, J., Žnidaršič, M., & Povh, J. (2018). Annotated news corpora and a lexicon for sentiment analysis in slovene. *Language Resources and Evaluation*, *52*(3), 895–919.

Cer, D., Yang, Y., Kong, S.-y., Hua, N., Limtiaco, N., John, R. S., ... others (2018). Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Dehdari, J. (2014). *A neurophysiologically-inspired statistical language model* (Unpublished doctoral dissertation). The Ohio State University.

Demšar, J., Curk, T., Erjavec, A., Gorup, Č., Hočevar, T., Milutinovič, M., ... others (2013). Orange: data mining toolbox in python. *the Journal of machine Learning research*, *14*(1), 2349–2353.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Hofmann, M., & Klinkenberg, R. (2016). *Rapidminer: Data mining use cases and business analytics applications*. CRC Press.

Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). Bag of tricks for efficient text classification. *CoRR*, *abs/1607.01759*. Retrieved from http://arxiv.org/abs/1607.01759

Kranjc, J. (2017). *Web workflows for data mining in the cloud: Doctoral dissertation* (Unpublished doctoral dissertation). Jožef Stefan International Postgraduate School.

Kranjc, J., Podpečan, V., & Lavrač, N. (2012). ClowdFlows: A cloud based scientific workflow platform. In P. A. Flach, T. Bie, & N. Cristianini (Eds.), *Machine learning and knowledge discovery in databases* (Vol. 7524, pp. 816–819). Springer Berlin Heidelberg.

Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., & Euler, T. (2006). YALE: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th acm sigkdd international conference on knowledge discovery and data mining* (pp. 935–940). New York, NY, USA: ACM.

Pelicon, A., Pranjić, M., Miljković, D., Škrlj, B., & Pollak, S. (2020a). Sentiment annotated dataset of croatian news.

Pelicon, A., Pranjić, M., Miljković, D., Škrlj, B., & Pollak, S. (2020b). Zero-shot learning for cross-lingual news sentiment classification. *Applied Sciences*, *10*(17), 5993.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical methods in natural language processing (emnlp)* (pp. 1532–1543). Retrieved from http://www.aclweb.org/anthology/D14-1162

Perovšek, M., Kranjc, J., Erjavec, T., Cestnik, B., & Lavrač, N. (2016). Textflows: A visual programming platform for text mining and natural language processing. *Science of Computer Programming*, *121*, 128–152.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Roure, D. D., Goble, C., & Stevens, R. (2009, February). The Design and Realisation of the myExperiment Virtual Research Environment for Social Sharing of Workflows. *Future Generation Computer Systems*, *25*, 561–567. Retrieved from `http://eprints.soton.ac.uk/265709/`

Vanschoren, J., Van Rijn, J. N., Bischl, B., & Torgo, L. (2014). Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, *15*(2), 49–60.

Žnidaršič, M., Cardoso, A., Gervás, P., Martins, P., Hervás, R., Alves, A. O., ... others (2016). Computational creativity infrastructure for online software composition: A conceptual blending use case. In *International conference on computational creativity* (pp. 371–379).

Wiegand, M., Siegel, M., & Ruppenhofer, J. (2018). Overview of the germeval 2018 shared task on the identification of offensive language.

Witten, I. H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

Zampieri, M., Malmasi, S., Nakov, P., Rosenthal, S., Farra, N., & Kumar, R. (2019). Predicting the Type and Target of Offensive Posts in Social Media. In *Proceedings of naacl.*