

NPL STUDIO

Jan Přichystal¹, Roman Valovič¹, Tomáš Jakúbek¹

¹Department of Informatics, Faculty of Business and Economics, Mendel University in Brno, Zemědělská 1, 613 00 Brno, Czech Republic

DOI: https://doi.org/10.11118/ecos2026_030

ABSTRACT

This article deals with the description of NLP Studio software. This web application allows users to prepare their own textmining experiments for the natural language analysis without the need to use the programming language. There is a development environment in which the user can work in the form of visual programming. Preprocessing, training or visualization components are available. The software is designed primarily for teaching natural language processing and aims to help students navigate this area even if they do not have a deep knowledge of programming.

Keywords: natural language processing, web application, textmining, visual programming

JEL: L86

1 INTRODUCTION

The field of natural language processing is very much discussed nowadays and text analysis tools are used in many fields of human activity. However, there are not many experts who are able to perform text analysis using commonly available tools, and often they do not have sufficient domain knowledge related to the area in which the texts are published. On the other hand, domain experts often do not have knowledge of the necessary tools (currently often programming languages) and therefore find in-depth analysis of the texts under study difficult. Thus, the gaze of the professional community is currently turned towards tools that would facilitate the process of analyzing textual data in a simple way. One way may be to provide an accessible environment that allows experiments to be carried out at a higher level of abstraction without knowledge of programming languages. This may be a development environment using visual programming principles.

The idea of visual programming is not new, it was first used in programming teaching. Charntaweechun and Wangsiripitak (2006) found that most students were more willing to accept flowcharts as the carrier of programming instead of writing codes directly. There are tools using the same concept mentioned in Boshernitsan (2004). The closest approach that is proposed in this study was proposed by VisComposer (Mei, 2018) which uses the visual programming interface for data transformations and layout construction. At the same time authors inherited the popular visualization tools disadvantages user interface heterogeneity, big number of additional panels with different settings and interaction models.



This article is licensed under CC BY-SA 4.0 Attribution-ShareAlike 4.0 International license.
<https://creativecommons.org/licenses/by-sa/4.0/deed.en>

Ivanov (2020) presented the concept of visual programming data visualization authoring environment that completely relies on a node-based visual editor and allows the separation or reuse of any part of the workflow graph inside a single or multiple environments.

The main goal of this application is to simplify the process of creating textmining experiments especially for users not familiar with traditional programming languages but wanting to analyze the textual data.

2 SYSTEM DESIGN

The application is designed for creating experiments in natural language processing. A development environment using the concept of visual programming is available. This approach is intended to make it easier for users without knowledge of common programming languages to create their own tasks for analyzing text documents.

The application is divided into several parts. The main part contains the development environment, where the user places individual components (tasks) to solve specific tasks. These tasks are subsequently interconnected with each other through links to create a complex dataflow. Each component has defined inputs, outputs or both and performs a specific activity. From the broadest perspective, components can be divided into three groups:

- input component,
- transformation component,
- output (visualization) component.

Input components allow the user to specify the data source to work with. Examples of these sources include JSON, CSV, or plain text file format. Input data can be imported from existing files, from a document database located on the server where the application runs or defined manually. Currently, the set of input formats is limited, but in the future, it will be possible to connect to, for example, a relational database, a web resource containing text data, etc.

The core of the application is of course the transformation components, which further differ in their purpose depending on the natural language processing tasks they solve. Thus, these are primarily components providing preprocessing, classification, cluster analysis, etc. Each of these components allows the user to configure the parameters of a given task in the form of a clear menu. Each task also has specific requirements for the format of input and output data.

The task of the output components is to clearly display the results of analyses in the form of tables, graphs, maps or other formats. These components are usually the final element in the dataflow and do not allow the linking of other components.

The components are available in the left part of the application in a separate menu from where the user can select them as needed.

In the right part of the application there is a menu allowing modification of user data in the basic settings and in case a specific component is selected there is an option to make its settings. As such, the application allows you to manage the user accounts of registered users. The main point is that the user can save the experiments in progress and their results. A registered user has the possibility to work on multiple experiments and save datasets at will. An unregistered user is only allowed to create one experiment and is not allowed to save any data on the server where the application runs.

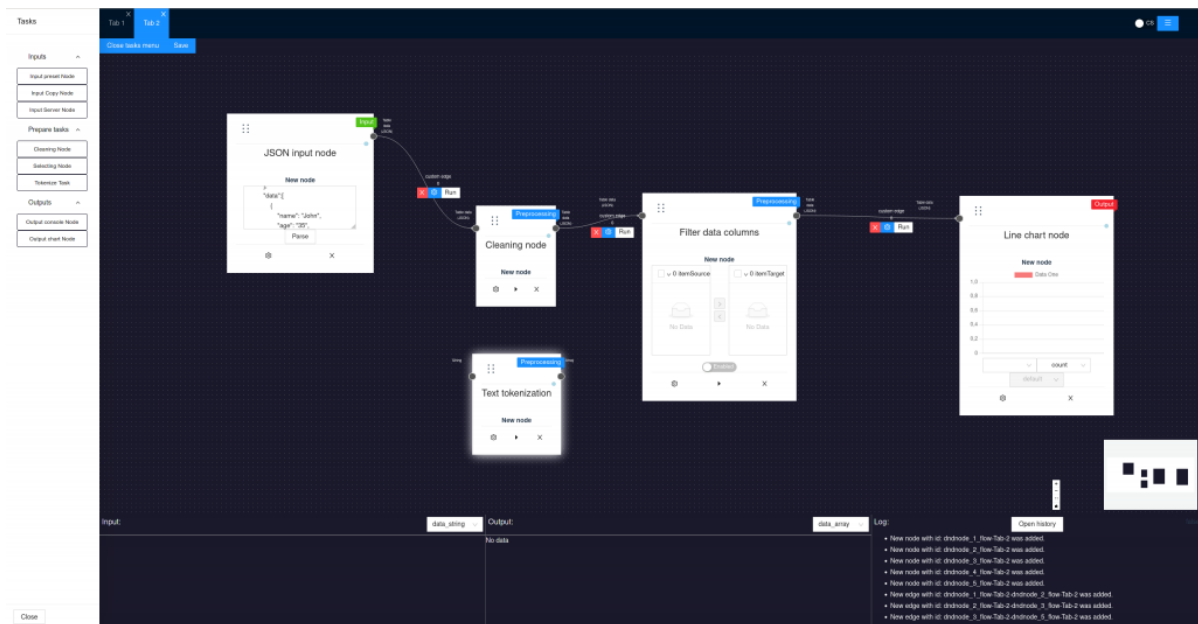


Fig. 1: Development environment

Source: own elaboration

2.1 Frontend implementation

The frontend is implemented as a standard single-page application. For this reason, a whole range of interactions and application logic is performed directly in the browser and the server faces significantly less load. Conversely, a significant load is transferred to the heterogeneously powerful end-user computers, which leads to a visible slowdown and decrease in the user experience if optimization is neglected. In applications with a one multi-interactive page (e.g., photo or text editor), this applies many times more. Therefore, with such an application concept, memoization, optimization of calculations, or avoidance of unnecessary renders play an important role.

This client application belongs to the category of so-called visual programming. The focus is oriented on visual objects and especially various relationships between them. Other features are concreteness, directness, explicitness and immediate visual feedback.

2.1.1 Used technologies

The implementation of the user interface is built on the JavaScript framework VueJS 3, which is used to create web UIs. This is the most up-to-date version that provides improved reactivity and change detection, a typed and better optimized core and a new Composition API syntax.

The key library is the open source vue-flow, which provides an engine for building interactive diagrams and complex node-based editors. It includes a workspace, graph controls and a dynamic viewport with custom elements. In this case, the elements are not rendered into the HTML canvas, but are normal DOM elements positioned dynamically using traditional CSS styles.

Communication between remote parts of the application is managed by the Pinia library, an alternative to the traditional Vuex. It is a global state manager that centralizes data, serves as the main source of truth and solves a programming anti-pattern called *props drilling*. The Ant Design framework provides the initial design of the component. Type checking from TypeScript makes the source code of higher quality.

2.1.2 User interface

The user interface is based on the *Canvas plus* palette design pattern. Its principle consists of a canvas on which nodes and edges are added, manipulated and their state changed in real time. A similar Microsoft Azure Machine Learning Designer application was designed with this idea in mind.

Nodes represent an input dataset, a processing task, or an output point. The user can define his own dataset or use one of the prepared ones. The growing list of supported tasks includes e.g. tokenization, lemmatization, POS tagger or stop-words removal. Edge between nodes means a connection enabling mutual communication and data transfer through the universal interface of input and output points.

The user can create multiple canvases within one workspace and solve different tasks at the same time. In the lower panel there is a visualization of the data of the active node and a log with last events. The user interface is currently localized into English and Czech languages and the new visitor is guided through it using quick onboarding. The layout of the entire flow is continuously saved in the *localStorage* of the browser, so the user does not lose his progress even after closing.

2.2 Application backend

The server part of the system is responsible for solving various tasks required by the user. From this perspective, the backend architecture can be divided into two levels.

The first level is the communication interface with the frontend and the database. Basically, it is a conventional web server that communicates using the REST API. At this level, the application receives and responds to requests, authenticates the user but also queues required tasks.

The second level is made up of a more complex, analytical part. The system is used for solving various NLP tasks and it is oriented towards processing large amounts of data, which is usually both time-consuming and computationally complex. Therefore, we implemented this part of the application as independent software. The analytic part picks up tasks from a queue and solves them based on the specified parameters in the background. It stores the intermediate state as well as the final result in a database. Thus, the first level is practically not involved in the actual task solving. If it did, the API would respond to requests with high latency. Additionally, such an arrangement does not allow for efficient task management or running in the background.

2.3 Communication interface

The Rest API is implemented in Python language in the modern FastAPI framework. One of the main reasons for choosing FastAPI is that it allows asynchronous handling of HTTP requests. Normally, Python executes code sequentially. For example, in the case of HTTP, this means that the server will only accept a new request when the previous request has been completely handled. However, asynchronous methods can be temporarily suspended (e.g. while waiting for data from the database), thus they do not block other, so-called competing requests waiting in the queue to run (Hattingh, 2020). Thus, the asynchronous paradigm significantly reduces the time required to handle a request and is well suited to a concurrent task environment such as a web server. The tasks are handled by the analytical part of the application at the second level, as we mentioned in the introduction of this chapter.

The communication is initiated by the client, by sending a POST request to the API endpoint/task. The endpoint accepts requests for all task types. The specific specification and parameters are sent in the request body in JSON format. For example, the following request illustrated in Figure 2 invokes the “Word tokenize” task over the dataset ID 622, which is stored in the database. A request body can contain any number of parameters depending on the type of task.

```

{
  "input_data_id": "622",
  "task": "word_tokenizer",
  "lang": "english"
}

```

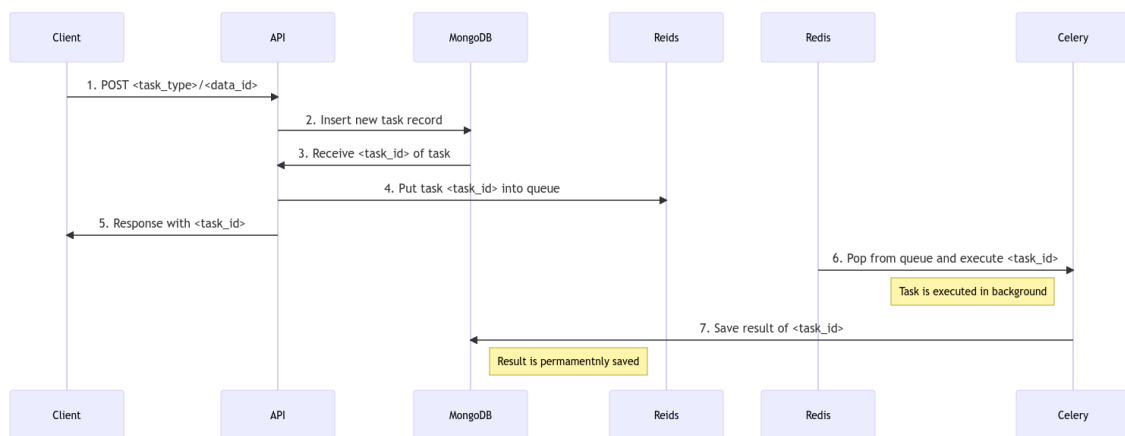
Fig. 2: Example of task request data

Source: own elaboration

Instead of executing the request directly, i.e., running the corresponding method, the API creates a new object for the task and inserts a new record in the Mongo database containing the necessary metadata. The API returns the ID of the created task to the user, but the result is processed asynchronously in the background.

The running of the task itself is handled by Celery, an implementation of a queue of tasks which asynchronously executed in the background. However, Celery requires a message transport system to send and receive messages between the client and the so-called worker (the thread that executes the code itself). For these purposes, we used Redis, in which the API stores the job record. This is an important part of the system since the API and the analytics layer works independently. Redis, thus, acts as a communication interface between them. The data flow is illustrated in Figure 3.

Once the job is completed, Celery writes the result to MongoDB. The client can retrieve the result by addressing the API at the appropriate endpoint, as it is shown in Figure 3. In this case, we no longer require Celery since it is a simple process, which only reads records from the database.

**Fig. 3:** Flowchart for processing a task request

Source: own elaboration

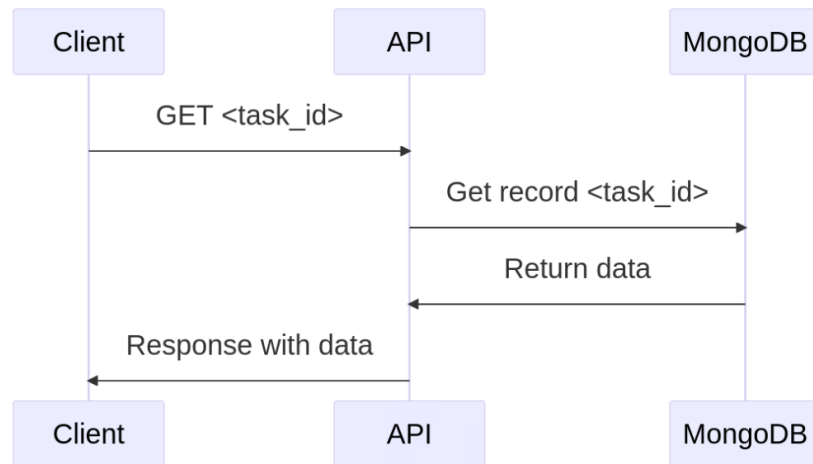


Fig. 4: Flowchart for processing a task request

Source: own elaboration

2.4 Data storage

Similarly, to the application layer, the data layer can be divided into two levels. Permanent data storage is provided by the non-relational Mongo database.

Temporary data, related to the task passing between the API and Celery is handled by Redis technology.

One of the key aspects of the system is the processing of data based on the user's needs. In order to provide the best user experience, we do not require any data structure that the user must follow in order to interact with the system. The user can upload their data stored in a text file in txt (plain text), JSON, or CSV format. The API recognizes the uploaded file's format and automatically executes the appropriate parser. To be able to work with the data regardless of its input format, the parser converts content of each file into a uniform internal format that the various system components are prepared for. An example of converting a CSV file to an internal format is illustrated in Figure 5. Content of each uploaded file is labeled by an ID and name as user's reference.

The internal format is designed so that the data is structured and stored in a table. This makes it possible to use popular libraries for working with tabular data such as Pandas or Numpy. Moreover, it gives the user the ability to use features that he is used to from common office applications – column removal, numeric operations on column values, row filtering, transformation of data types, etc.

The API always returns the processed data in JSON format, however, in the future we plan to extend the output to other formats.

```
Survived,Pclass,Name,Sex,Age,Siblings/Spouses Aboard,Parents/Children Aboard,Fare
0,3,Mr. Owen Harris Braund,male,22,1,0,7.25
1,1,Mrs. John Bradley (Florence Briggs Thayer) Cumings,female,38,1,0,71.2833
1,3,Miss. Laina Heikkinen,female,26,0,0,7.925
1,1,Mrs. Jacques Heath (Lily May Peel) Futrelle,female,35,1,0,53.1
```

```
{
  "_id": "63d7d2960b49cf2b9925e446",
  "name": "titanic_passengers",
  "payload": [
    {
      "Survived": 0,
      "Pclass": 3,
      "Name": "Mr. Owen Harris Braund",
      "Sex": "male",
      "Age": 22,
      "Siblings/Spouses Aboard": 1,
      "Parents/Children Aboard": 0,
      "Fare": 7.25
    },
    {
      "Survived": 1,
      "Pclass": 1,
      "Name": "Mrs. John Bradley (Florence Briggs Thayer) Cumings",
      "Sex": "female",
      "Age": 38,
      "Siblings/Spouses Aboard": 1,
      "Parents/Children Aboard": 0,
      "Fare": 71.2833
    }
  ]
}
```

Fig. 5: Transformations of CSV (upper) to internal format (below)

Source: own elaboration

3 DISCUSSION AND CONCLUSIONS

Of course, the presented application is not the only one currently available. There are similar projects available on the Internet that offer visual programming environments for machine learning or similar projects. The closest representatives of this type of applications are e.g. Azure ML Studio (2023), BotX (2023) or Datablocks (2023).

Azure Machine Learning Studio (ML Studio) is a graphical, collaborative, drag-and-drop web interface from Microsoft for building machine learning models. It does not require coding knowledge of R or Python, and the cloud interface can be used to build models through ML Studio. ML Studio lets user to train machine learning models, evaluate trained model performance, launch new experiment and view the results of past experiments, deploy trained models, monitor deployed models, track datasets and models, provision and manage compute resources, conduct data science experiments in online notebooks. The most important part of ML Studio is the Designer a drag and drop tool to create ML workflow from the input of a dataset to the evaluation. It is easy to use and has many features which can be searched via text or found by clicking on the arrows and captions on the left. For the Designer, there are currently more than 60 Modules that can be divided into data

transformation, data input and output, feature selection, statistical functions, regression, clustering, classification, model training, and model evaluation/scoring. The whole machine learning set is covered by the designer modules, so the user can build models without coding. He can simply choose the modules and connect them and then get the pipeline ready to be tested and interfered. The designer is attractive to people who would like to think at a higher level and don't want to worry about specific Python or R code for executing standard tasks. However, users still need to know a lot of machine learning concepts as far as normalizing data, feature engineering, splitting data for training and validation, and various machine learning algorithms. Additionally, it's worth noting that there are code nodes for both Python and R code that users can include into a machine learning pipeline. These nodes allow user to execute custom code in a machine learning pipeline for certain steps while still taking advantage of the drag and drop and visual nature of the designer (Hucker, 2021), (Eland, 2022).

The Datablocks is a node based editor with several blocks for processing data. It's based on our open source library React Flow. The editor is divided into four parts. On the left user has the sidebar with the different blocks. In the center there is the editor pane and at the bottom there is the data output view and a terminal.

The blocks are divided into the groups Input, Transform, Geo, Visualization and Misc. The input group gives user blocks to read in data. Either by pasting text content or by file upload. Currently, datablocks can handle json, csv, topojson and geojson files. There is also an example data block for testing and playing around. The transform blocks handle tabular data. Users can filter data entries or merge different data sets for example. The geo blocks help them to simplify data or convert geo formats. With the visualization blocks you can create simple visualizations like a histogram or a bar chart. The misc section contains markdown and a simple statistics block for now.

The editor pane is the place where users are working most of the time. On the editor pane user can connect different blocks to create a data flow and you can configure the blocks. In the current version all configuration happens in the blocks. At the bottom the user can find the data output view. This view shows the output of the selected block. It can display tabular data, JSON objects and geojson data. Users can also export data as csv, json or geojson. In the right bottom corner is the terminal. It is used to display application errors or logs from the Javascript block (Datablocks, 2023).

This overview reveals the development of experiments in the area of machine learning and AI without coding is a very important challenge. This and similar areas related to data analysis are being addressed by several commercial companies that see an opportunity to offer solutions for ordinary users on the market. As we know from discussions with our students, the small knowledge of programming languages limits their ability to analyze the various types of data including the textual. The applications discussed above offer only limited functionalities for the free users. This is an opportunity to apply our application, which is free and offers all functions without limitations. Now, there is a noticeable difference in the number of functions offered between the application we have developed and commercial solutions. However, intensive development and communication with potential users will allow us to offer a competitive solution that will stand up to comparison with others in the foreseeable future.

The prototype of our application is available at <https://mta.pef.mendelu.cz/lab/>.

Acknowledgements

This paper was supported by the project CZ.02.1.01/0.0/0.0/16_017/0002334 Research Infrastructure for Young Scientists, this is co-financed from Operational Programme Research, Development and Education and the project EG17_176/0015722 – Data processing software (2020–2023, MPO/EG).

REFERENCES

- Azure ML Studio. 2023. <https://ml.azure.com>
- BOSHERNITSAN M., DOWNES M. S. 2024. *Visual programming languages: A survey*. Computer Science Division, University of California.
- BotX. 2023. <https://www.botx.cloud/>.
- CHARNTAWEEKHUN, K., WANGSIRIPITAK, S. 2006. Visual programming using flowchart. In: *2006 International Symposium on Communications and Information Technologies*, IEEE, pp. 1062-1065.
- Datablocks. 2023. <https://datablocks.pro/>.
- ELAND, M. 2022. *Azure ML Studio: Getting Oriented with Azure Machine Learning*. <https://dev.to/integerman/azure-ml-studio-getting-oriented-with-azure-machine-learning-5cog>
- Hattingh, C. (2020). *Using Asyncio in Python: Understanding Python's Asynchronous Programming Features*. O'Reilly Media Inc., 2020, ISBN 978-1492075332.
- HUCKER, M. 2021. *A Brief Introduction to Azure Machine Learning Studio*. <https://towardsdatascience.com/a-brief-introduction-to-azure-machine-learning-studio-9bbf41800a60>.
- IVANOV, E., KHOROSHAVIN, A., KARSAKOV, A. 2020. Visual programming environment based on data visualization grammar specification. *Procedia Computer Science*. 178, 434–439.
- MEI, H., CHEN, W., MA, Y., GUAN, H., HU, W. 2018. Viscomposer: A visual programmable composition environment for information visualization. *Visual Informatics*. 2(1), 71–81.
- RAY, P. P. 2017. *A survey on visual programming languages in internet of things/Scientific Programming*.
- SAABITH, A. S., FAREEZ, M. M. M., VINOTHRAJ, T. 2019. Python current trend applications an-overview. *International Journal of Advance Engineering and Research Development*. 6(10).

Contact information

Jan Přichystal: e-mail: jan.prichystal@mendelu.cz