

University of West Bohemia  
Faculty of Applied Sciences  
Department of Computer Science and Engineering

**Master's thesis**

**Workflows for  
electrophysiological data  
processing**

Místo této strany bude  
zadání práce.

# Declaration

I hereby declare that this master's thesis is completely my own work and that I used only the cited sources.

Pilsen, 8th August 2020

Filip Jani

## **Abstract**

This master's thesis describes current software and workflows at the neuroinformatics lab KIV/NTIS at the University of West Bohemia utilized for electrophysiological data processing. As the lab's members would like to migrate to an utterly open-source solution, an analysis of existing workflows management systems is done. Each is assessed on several factors to fit the neuroinformatics lab's requirements. For the most appropriate system, a custom library for electrophysiological data processing is implemented. The library's functionality is verified on three existing experiments that are reproduced in the chosen system. In the last part, the results and the implemented library's limitations are discussed.

## **Abstrakt**

Tato diplomová práce popisuje aktuálně používaný software a pracovní postupy neuroinformatické laboratoře KIV/NTIS na Západočeské univerzitě v Plzni používané pro zpracování elektrofyziologických dat. Protože by členové laboratoře chtěli přejít na kompletně open-source řešení, byla provedena analýza existujících systémů pro správu workflow. Každý systém je hodnocen dle několika kritérií a pro nejvhodnější systém je vytvořena knihovna pro zpracování elektrofyziologických dat. Funkčnost knihovny je následně ověřena na třech existujících experimentech, které byly reprodukovány ve vybraném systému za pomoci vytvořené knihovny. V poslední části jsou hodnoceny výsledky a limitace implementované knihovny.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Workflows . . . . .	1
1.2	Scientific workflow systems . . . . .	3
1.2.1	Usability with electrophysiological data . . . . .	3
<b>2</b>	<b>Workflows at neuroinformatics lab KIV/NTIS</b>	<b>5</b>
2.1	Workflows and used analytical methods . . . . .	5
2.2	Tools in use . . . . .	6
2.2.1	EEGLAB . . . . .	7
2.2.2	ERPLAB . . . . .	7
2.2.3	Brain Products tools . . . . .	8
2.3	Experiment examples . . . . .	9
2.3.1	Effects of various music genre on brain activity . . . . .	9
2.3.2	Attention of driver during simulated drive . . . . .	10
2.4	Future vision . . . . .	11
<b>3</b>	<b>Existing workflow management systems</b>	<b>12</b>
3.1	Electrophysiological data standards . . . . .	12
3.2	Orange . . . . .	13
3.3	Snakemake . . . . .	16
3.4	GIN . . . . .	19
3.5	VisTrails . . . . .	21
3.6	Workflow Designer . . . . .	22
3.7	Apache Taverna . . . . .	25
3.8	NeuroPype Suite . . . . .	27
3.9	Other large infrastructures . . . . .	29
<b>4</b>	<b>Workflow management systems requirements</b>	<b>31</b>
4.1	Availability of analytical methods . . . . .	31
4.2	Graphical user interface . . . . .	31
4.3	Syntactical and semantical compatibility . . . . .	31
4.4	Community size . . . . .	32
4.5	User's manual . . . . .	32
4.6	Ease of development . . . . .	32

<b>5</b>	<b>Comparison of existing workflow management systems</b>	<b>34</b>
5.1	Suitable third-party libraries . . . . .	34
5.1.1	MNE-Python . . . . .	34
5.1.2	MNE-BIDS . . . . .	35
5.1.3	NIX-MNE conversion tool . . . . .	35
5.1.4	Elephant . . . . .	35
5.1.5	Conclusion . . . . .	35
5.2	Summary of analyzed systems . . . . .	36
5.2.1	Orange . . . . .	36
5.2.2	Snakemake . . . . .	36
5.2.3	GIN and gin-proc . . . . .	37
5.2.4	VisTrails . . . . .	38
5.2.5	Workflow Designer . . . . .	38
5.2.6	Apache Taverna . . . . .	39
5.2.7	NeuroPype Suite . . . . .	40
5.3	Conclusion . . . . .	41
<b>6</b>	<b>Implementation</b>	<b>43</b>
6.1	Changes to Orange 3 . . . . .	43
6.2	Implemented Widgets . . . . .	43
6.2.1	Data IO . . . . .	44
6.2.2	Preprocessing . . . . .	46
6.2.3	Feature Extraction . . . . .	52
6.2.4	Classification . . . . .	53
6.2.5	Visualization . . . . .	60
<b>7</b>	<b>Verification of proposed solution</b>	<b>62</b>
7.1	Attention of driver during simulated drive . . . . .	62
7.2	Event-related potential datasets based on a three-stimulus paradigm . . . . .	65
7.3	Evaluation of convolutional neural networks using a large multi-subject P300 dataset . . . . .	71
7.4	Conclusion . . . . .	76
<b>8</b>	<b>Conclusion</b>	<b>78</b>
	<b>List of abbreviations</b>	<b>79</b>
	<b>Bibliography</b>	<b>80</b>
	<b>Appendix A Contents of the attached DVD</b>	<b>86</b>

<b>Appendix B Source code for sample modules</b>	<b>87</b>
B.1 Orange . . . . .	87
B.2 VisTrails . . . . .	88
B.3 Workflow Designer . . . . .	89
B.4 NeuroPype Suite . . . . .	89
<b>Appendix C Verification</b>	<b>91</b>
<b>Appendix D User manual</b>	<b>96</b>

# 1 Introduction

Electrophysiological data are extensive data, which means that it is not easy to process them manually. For example, monitoring patient’s EEG<sup>1</sup> signals from 30–100 channels may result in large data files. Depending on the duration of the recording, the file size may reach up to 5–10 GB (for 24 hours long recording) [1].

The manual processing of such large data files could be a time-consuming process because there are multiple steps involved (for example, data cleaning, filtration, feature extraction, or visualization). Every step of the processing may require different software solutions. Nowadays, there are various formats of input data, and some of them are standardized (e.g., NIX, BIDS, or NWB). Because of this, the user has to verify if the data are in the correct format for each step of the processing, and eventually convert the data to the required format or adapt some of the steps.

We are currently in a state where there are many different methods on various platforms (e.g., libraries for multiple programming languages or data mining tools) used to process scientific data. The problem with the current state is not a lack of methods but the lack of integration between them.

As a solution to this problem and making the processing more straightforward, scientific workflow tools are being developed. Such tools promise to make scientists more productive by automating data-driven and compute-intensive analyses [2].

## 1.1 Workflows

Before we take a look at the scientific workflows, we will look at what generic workflows are, how they could be useful, and how they may be visualized.

In the Cambridge Dictionary, the term workflow is defined as ‘the way that a particular type of work is organized, or the order of the stages in a particular work process’. Generally speaking, a workflow describes how to proceed in a specific step of a process. Workflows are usually visualized as a flowchart diagram. For example, in Figure 1.1, we can see a flowchart diagram of the stages of an EEG signal preprocessing.

---

<sup>1</sup>EEG – Electroencephalography, it is a monitoring of the electrical activity of the brain.



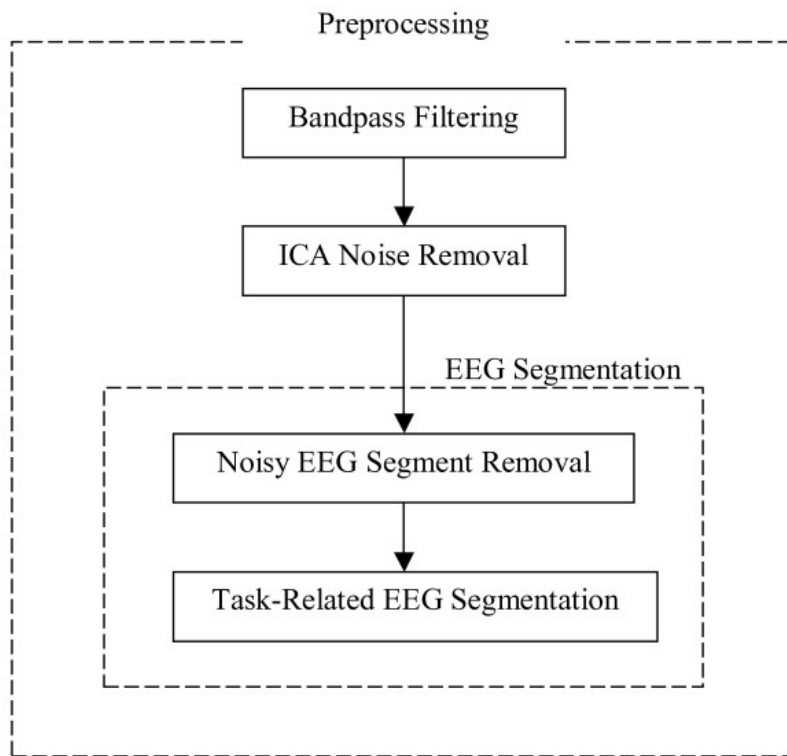


Figure 1.1: An example flowchart diagram of the steps of EEG preprocessing [3]

## 1.2 Scientific workflow systems

New scientific knowledge is increasingly obtained through complex computations and data analysis. As stated before, these computations may contain thousands of steps. Each step may utilize a variety of models, data sources, and even demand a different software or execution environment.

Over recent years, the major issue with these computations is the reproducibility. Even when specific tools were used, a textual description of the methodology was insufficient to reproduce experiments [4]. Some computations could be beneficial for other scientists, but maintaining and distributing them with other scientists is quite challenging [5, 6].

In recent years there has been an increase in research and development of scientific workflow systems. These systems are designed to control the data-flow and execution of computation across the individual steps. Workflows in scientific workflow systems are represented as directed graphs, where node indicates computational step and connections among nodes describes data-flow. One advantage of workflow systems based upon a dataflow approach is the possibility to route produced data by one step to multiple following steps. While it is often difficult to describe and understand the flow of the data in plain text for such workflows, the dataflow approach makes it straightforward. The downside of this approach is that it can become a confusing tangle of nodes and connections for complex workflows [2].

While these systems are a viable alternative to the traditional approach based on scripting languages, there is still considerable room for improvement. Nevertheless, in the next ten years, workflows will continue to play a central role in knowledge discovery in the major research projects [7].

### 1.2.1 Usability with electrophysiological data

The scientific workflow system in the processing of electrophysiological data has its value. Analytical methods often have several input parameters used to control and manage the processing. The wrong configuration of these parameters may result in distorted or degraded results. If there is more than one method applied during the steps of processing, the risk of obtaining the wrong result increases. The experience with measurements is as important as the perfect knowledge of analytical methods to ensure the correct result. It is possible to choose the individual steps of processing and create a workflow from them. As it is required for the scientist to have a considerably broad knowledge of the problem to process measured data correctly, the scientists may share the workflow with the team's less experi-

enced colleagues. This workflow may run repeatedly, and even with different data models, and therefore simplify the work for them [8].

# 2 Workflows at neuroinformatics lab KIV/NTIS

In this chapter, I will focus on workflows currently in use at the neuroinformatics lab KIV/NTIS of the University of West Bohemia. First, I will describe the analytical methods for loading, preprocessing, processing, and visualization of electrophysiological data. Then, I will introduce tools that are in use to make the analysis of electrophysiological data possible. In the next part, I will give some use-case examples from completed experiments. In the last part of the chapter, I will describe the future vision and plans for the workflows at the neuroinformatics lab.

## 2.1 Workflows and used analytical methods

There are five categories of methods that are in use in the neuroinformatics lab. There are methods for loading/saving, preprocessing methods, feature extraction methods, classification methods, and visualization methods. I will briefly introduce them in this section.

### Methods for loading/saving of data

These methods are used to load/save data in various formats. Some of those formats are:

- BrainVision files
- LabStreamingLayer streams
- LabStreamingLayer files

### Preprocessing methods

Preprocessing methods are used to prepare raw data for processing (e.g., data cleaning). Selected methods are entirely dependant on the analysis type. Some of those methods are listed below:

- Epoch extraction
- Averaging
- Segmentation
- Artifact rejection
- ICA

### **Feature extraction methods**

Large datasets contain a large number of variables that require a lot of computation resources. The goal of feature extraction methods is to reduce the number of variables, thus reducing computation resources requirements while accurately describing the original data set.

- Subsampling
- Windowed Means
- Common spatial patterns

### **Classification methods**

The purpose of the classification methods is to find a structure based on mutual relations in data. These methods are usually based on clustering or machine learning.

- Linear Discriminant Analysis
- Multi-layer perceptron
- Support Vector Machines
- Convolutional neural network

### **Visualization methods**

These methods are useful to understand the results of the analysis better, as pictures or graphs are more easily understandable for humans than a table of numbers, for example.

- Time series
- Scalp map
- Time-frequency color-coded maps

## **2.2 Tools in use**

The most used tools at the neuroinformatics lab are EEGLAB and ERPLAB, followed by BrainVision Analyzer 2 and Brain Vision Recorder. The first two tools are open-source software, while the BrainVision products are proprietary software.

In this section, I will briefly introduce these tools.

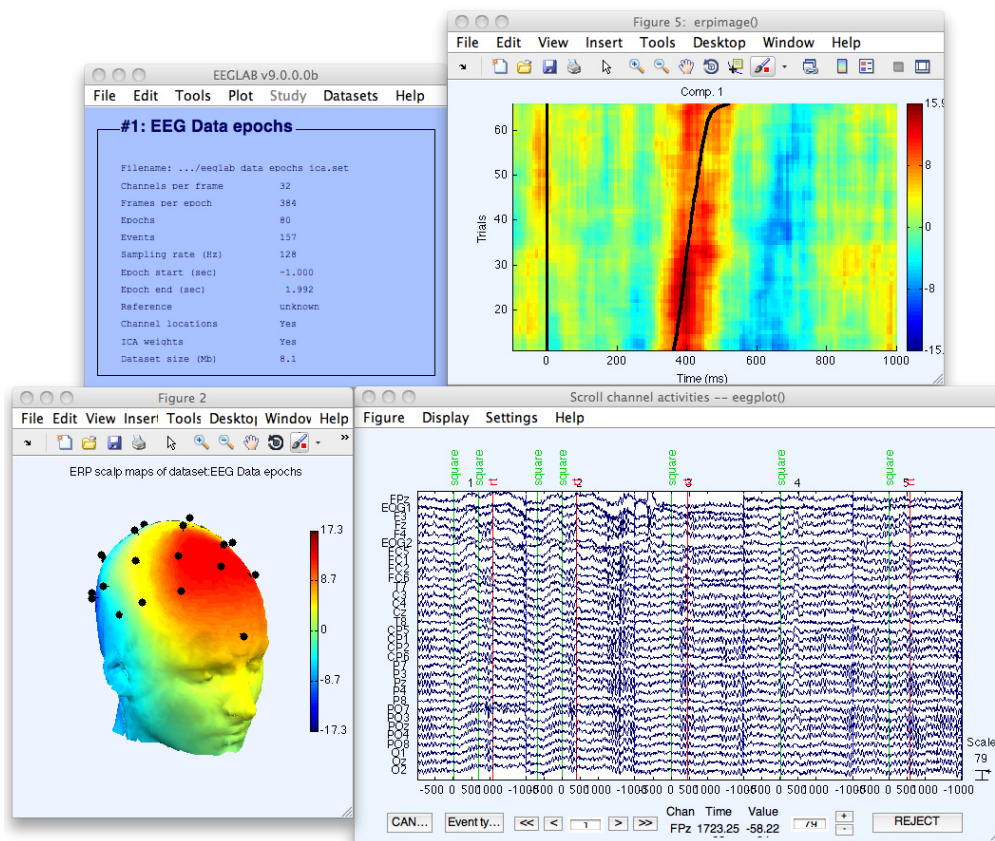


Figure 2.1: Graphical user interface of EEGLAB [10]

## 2.2.1 EEGLAB

EEGLAB is an open-source Matlab toolbox for the analysis of electrophysiological data. Using EEGLAB, users can import various data formats, preprocess data, visualize data, perform independent component analysis, or use multiple time/frequency analysis methods. Third parties can contribute with additional functionality as EEGLAB has an extensible plugin architecture [9].

We can see the graphical user interface of EEGLAB in Figure 2.1.

## 2.2.2 ERPLAB

ERPLAB is an open-source toolbox for processing and analyzing event-related potential (ERP) data in the Matlab. ERPLAB is tightly integrated with EEGLAB and provides additional tools (e.g., filtering, artifact detection, or sorting of events) to EEGLAB. Users can use ERPLAB's tools from a graphical user interface or Matlab scripts.

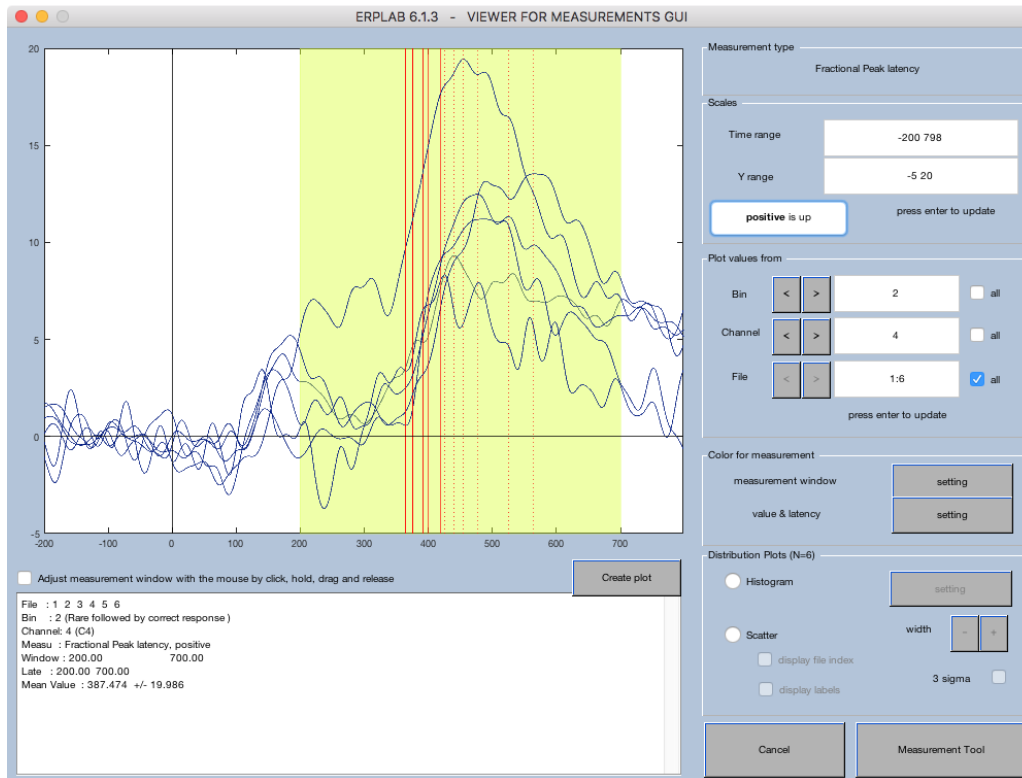


Figure 2.2: Graphical user interface of ERPLAB [11]

In Figure 2.2, we can see the GUI of the ERPLAB.

### 2.2.3 Brain Products tools

Brain Products GmbH is a company that provides a variety of proprietary software tools for electrophysiological data recording, analysis, and processing. The tools used at the neuroinformatics lab are described below.

#### BrainVision Recorder

BrainVision Recorder is a software developed for recording and visualization of electrophysiological signals. Furthermore, it is also possible to analyze evoked potentials in real-time.

#### BrainVision Analyzer 2

BrainVision Analyzer 2 is a software tool used to process a variety of electrophysiological signals (e.g., EEG, EOG, ECG, EMG, etc.). The Analyzer has a modular structure, so it is possible to expand the Analyzer's functionality

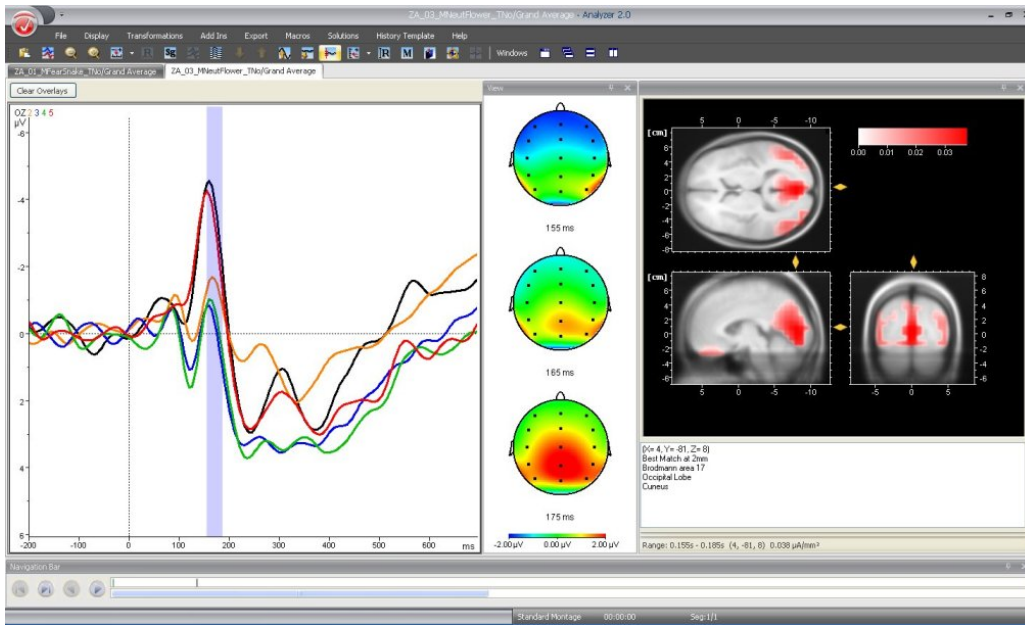


Figure 2.3: Graphical user interface of BrainVision Analyzer 2 [13]

dynamically. It also has a Matlab interface, which ensures efficient data interchange between the Analyzer and Matlab/EEGLAB [12].

We can see the graphical user interface of the Analyzer in Figure 2.3.

## 2.3 Experiment examples

In this section, I will show some examples of completed experiments and describe how the data are processed in each experiment.

### 2.3.1 Effects of various music genre on brain activity

This experiment examined the impact of different music genres on human brain activity during a psychological burden. The main point of the research was a Concentration game (also known as Pexeso, or Pick a pair). During the experiment, the subjects played four games of Concentration with different music playing in the background, and their EEG activity and pulse rate were monitored [14].

The EEG activity was recorded using a BrainVision Recorder (see section 2.2.3). The data were later processed and analyzed using EEGLAB and ERPLAB (see sections 2.2.1 and 2.2.2). We can see the approach to EEG data processing in the list below:

1. Data filtration – Low-pass and high-pass filter applied.



2. Automated artifact detection and removal – Sliding window method to detect voltage outside of a limit was used.
3. Manual artifact detection and removal applied.
4. Computation and visualization of a frequency spectrum of an EEG signal – Fast Fourier transform used.

### **2.3.2 Attention of driver during simulated drive**

This experiment deals with methods of electroencephalography (EEG) and event-related potentials (ERP) under various conditions to investigate the driver's attention. Eleven students were stimulated with audio signals during a 20 minutes drive in four experimental sessions on a car simulator. For each participant, there were two drives in a day, one in the morning (between 9 and 12 AM) and one in the afternoon (between 1 and 4 PM). On the first day, they completed the drives after a usual night's sleep. The next day they completed the rides after a sleep restricted to a maximum of four hours [15].

The BrainVision Recorder (described in section 2.2.3) was used to record and store an EEG/ERP data. For data processing, the BrainVision Analyzer was used (see section 2.2.3). The recorded EEG/ERP data were further processed using the following workflow:

1. Data filtering – IIR filter was applied.
2. Data segmentation – The epochs were extracted from datasets in the area of each target stimulus.
3. Manual rejection of corrupted data was performed.
4. Baseline correction – The baseline correction was applied to mitigate uneven amplitude shifts.
5. Data averaging – The selected epochs were averaged.
6. Peak latency – Maximum amplitude in the time interval of possible occurrence of the P3 component was found.
7. Fractional 50% Peak latency – This technique marks the time point, when 50% of the maximum amplitude is reached.
8. Fractional 50% Area latency – This technique was used to compute the area under the component, and then to find the time point that divided the area into halves.
9. Visualization – Several plots with grand averages were created.

## 2.4 Future vision

The researchers from the neuroinformatic lab would like to switch to the software ecosystem that will be utterly open-source. Even though the EEGLAB and ERPLAB are open-source solutions, they depend on the Matlab, a proprietary software.

Besides, that BrainVision software is not open-source, the researchers would also like to work with standardized data and metadata, but the BrainVision metadata format does not comply with standardization.

# 3 Existing workflow management systems

In this chapter, I will introduce standards for electrophysiological data and their association with workflow management systems. Next, I will focus on the existing approaches and software tools used to design data workflows. I will describe them and their capabilities, the possibility of electrophysiological data processing, and describe the graphical user interface, if possible. As the last step, I will analyze the simplicity of extendability (i.e., writing additional addons or usage of existing libraries for data processing) of described solutions based on official documentation and my experience from experimenting with these systems.

As a part of this thesis, the assignment was to analyze workflow management systems for electrophysiological data processing, especially those integrated with community-respected data repositories. However, no system that would be integrated with such repositories was found.

## 3.1 Electrophysiological data standards

In the past years, several electrophysiological data standards have emerged. As the complexity of scientific data increases, the lack of standards for the neurophysiological data and metadata is the barrier to return-on-investment from neurophysiology experiments. The leading cause is a complicated interchange and reuse of data, and the reproduction of the experiments [16]. Some of the standards are, for example, Neurodata Without Borders: Neurophysiology (NWB:N), Brain Imaging Data Structure (BIDS), or Neuroscience information exchange format (NIX).

Even though the standards are present, the researchers are still utilizing various scripts and non-standard data formats for data processing. The workflow management systems could play a role in a transition from the non-standard to standard data formats. For example, if the system is straightforward to use, offers necessary analytical methods, and the possibility to convert data to standard formats, the researchers might start using it. Therefore the neurophysiological experiments will become more reproducible, and the data will comply with FAIR principles<sup>1</sup>.

---

<sup>1</sup>FAIR principles introduction – <https://www.go-fair.org/fair-principles/>

## 3.2 Orange

Orange is an open-source machine learning and data visualization tool. It is a desktop application written in Python and is available as a Python library as well [17, 18].

Users can design data workflows by visual programming or writing scripts in Python. In Figure 3.2, we can see an example of a workflow using visual programming.

Every visual block in the workflow is called a widget. In Orange, it is possible to connect compatible widgets only, making it more straightforward for the user to design functioning workflows without the necessity to verify the compatibility among widgets manually. The compatibility is secured on a syntactic level only, where each input and output slot of a widget has a defined data type, and users can connect only the slots with the same data type.

For some workflows, it is crucial to have the steps in the right order; thus, semantic checks would be beneficial; however, the semantic level of compatibility is not implemented in Orange.

Orange works in real-time, which means every adjustment made in an individual widget is directly reflected in the linked widgets. It is possible to pause the real-time propagation, but it is not possible to start the workflow only once and manually from the beginning.

### Electrophysiological data compatibility

Orange, as a standalone solution, does not provide functionality for electrophysiological data processing. Orange provides an Orange Bioinformatics addon, which can be accessed as a Python library or through a visual programming interface. This addon is focused on the analysis of genes as it provides several widgets, for example, *Genes* (information about genes from NCBI Gene database), *GEO Data Sets* (database stores curated gene expression DataSets), or *Marker Genes* (access to a public database of marker genes) [19]. At the moment, this addon does not support the analysis of electrophysiological data.

At the time of writing this thesis, there is currently no addon that would provide support for the electrophysiological data workflows. However, it is possible to create new addons, which is described in section Extendability.

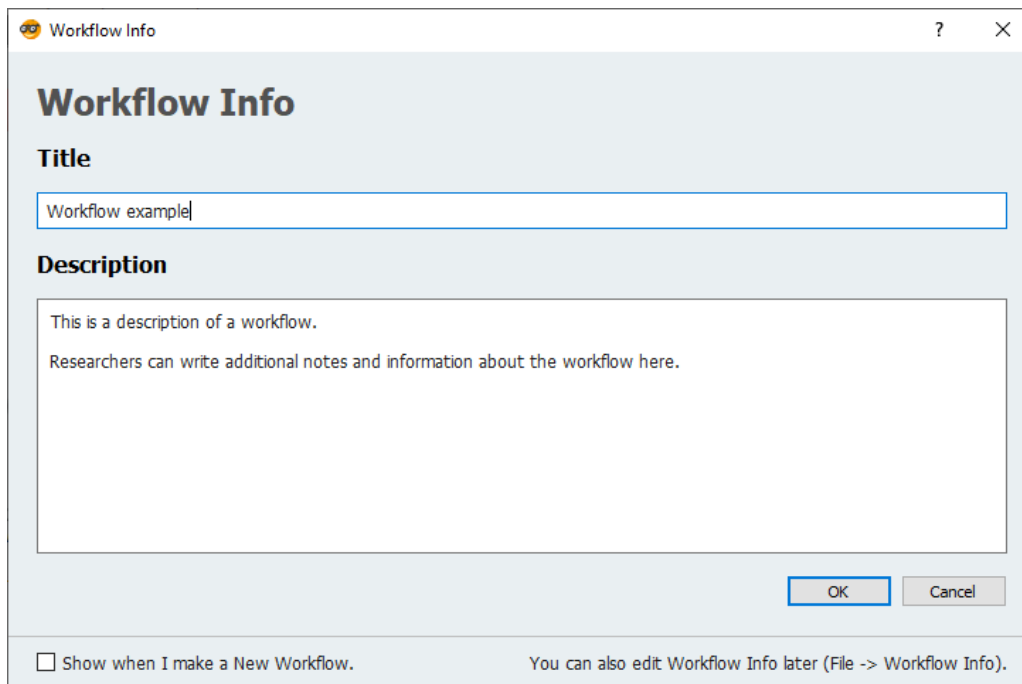


Figure 3.1: An example of a textual description of a workflow in Orange

## Graphical User Interface

The graphical user interface of Orange is relatively straightforward to use, even for an inexperienced user, as there are only a few functional buttons present. The respective buttons have an icon and a tooltip, which describes their purpose.

The researchers may provide a textual description of the workflows, which can help others quickly get an idea of how the particular workflow works. The textual description of a workflow can be seen in Figure 3.1. On top of that, it is also possible to append the workflow with annotations. Annotations can be textual or in the form of an annotation arrow, as can be seen in Figure 3.2. Such annotations can help to describe intricate parts of a workflow.

New widgets can be added to a workflow using two approaches. The first approach is to drag and drop the specific widget from the gallery into a workspace. The second approach is to right-click in a workspace and either write the name of a widget or find it in a contextual menu. In the workflow designer part of the application, it is possible to automatically align widgets to a grid, making the workflow more eye-pleasing and clear.

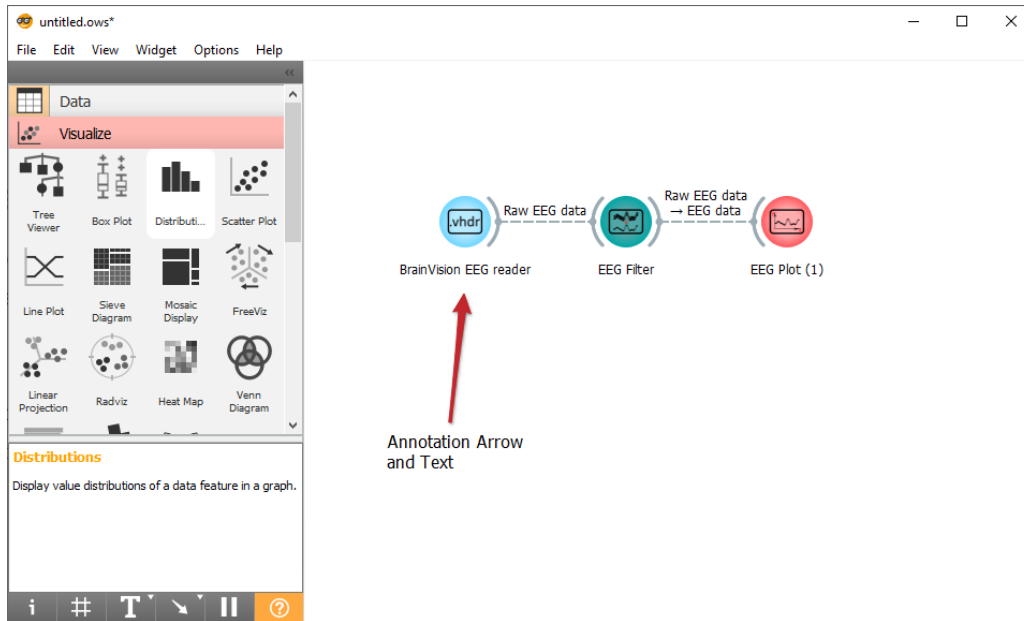


Figure 3.2: Graphical user interface of Orange with annotation text and arrow in a workspace

## Extendability

In Orange, it is possible to extend the functionality with custom widgets as Orange provides official documentation for widget development. The documentation is extensive, and apart from a technical description, it contains code examples, making it more obvious to understand how the widget development works [20].

Each widget contains information such as name, description in plain text, or icon. This information is used to distinguish among different widgets. Widgets include information about its inputs and outputs; therefore, the user immediately knows which other widgets they may combine with a particular widget.

In Figure 3.3, we can see a sample widget that demonstrates this functionality. In the picture there is our created library called „Sample EEG Widgets“ (1), and the library may include multiple widgets. The library includes a sample widget (2), and below that is its plain text description (3). On the right side in the picture, we can see a box that shows the widget’s inputs and outputs (4). We can notice additional information about a link among widgets that displays to which input slot is widget connected (5). We can see the source code for the sample widget in section B.1.

The current workflows at the neuroinformatics lab take advantage of

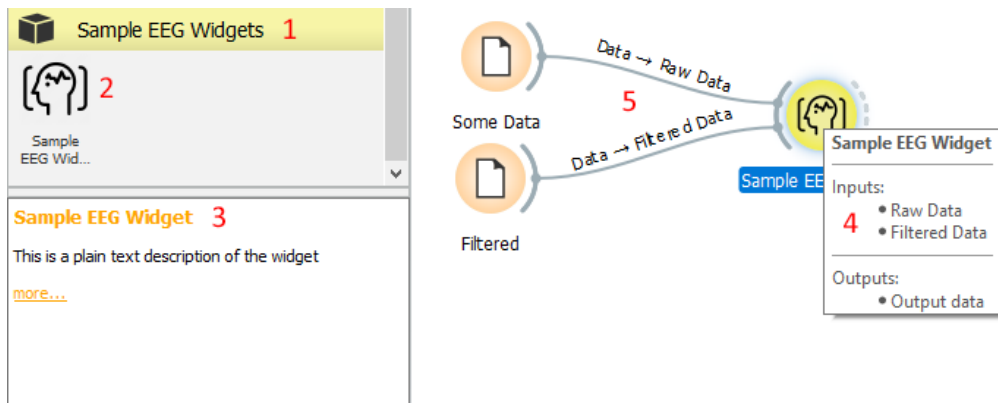


Figure 3.3: Example of a sample widget and its properties in Orange

some methods from the MNE-Python library<sup>2</sup> (more in Section 5.1.1). As widgets in Orange are Python scripts as well, it is possible to write widgets that are utilizing the MNE library. The possibility of using the MNE library was presented in a term project that adds some EEG workflow widgets to Orange [21].

### 3.3 Snakemake

The Snakemake workflow management system is a tool to create reproducible and scalable data analyses. Workflows are described via human-readable, Python-based language.

Workflows are Python scripts extended by declarative code to define rules. Rules are used to describe how to create output files from input files. They can produce outputs in various ways, e.g., by running the inline Python code defined in a rule, executing a shell command, or executing an external script. Snakemake currently allows integration of external scripts utilizing R, R Markdown, and Julia language. Snakemake does not have any syntactic compatibility check, i.e., users have to manually verify that each input for each rule is in a correct format.

Snakemake workflows can be executed on workstations, clusters, the grid, or in a cloud, without a need for modification. Usage of resources can be constrained, e.g., by available CPU cores, memory, or number of GPUs [22, 23].

Since Snakemake does not provide any graphical user interface, the user must have at least some programming experience. The Snakemake workflows

<sup>2</sup>Magnetoencephalography (MEG) and Electroencephalography (EEG) in Python

are defined in a file called Snakefile. We can see an example of a simple workflow in Listing 3.1.

## Electrophysiological data compatibility

As Snakemake works with command-line tools or Python scripts, it is possible to use Snakemake to process electrophysiological data. Snakemake supports the utilization of reusable modules, or wrappers (see section Extendability). However, at the time of writing this thesis, there are currently no modules with a focus on the electrophysiological data. The major part of the modules focuses on the processing of genome sequencing data [24].

## Graphical User Interface

As the Snakemake does not provide any graphical user interface, in this section, the Snakefile will be described instead.

```
1 | rule targets:
2 |     input:
3 |         "plots/dataset1.pdf",
4 |         "plots/dataset2.pdf"
5 |
6 | rule plot:
7 |     input:
8 |         "raw/{dataset}.csv"
9 |     output:
10 |        "plots/{dataset}.pdf"
11 |     shell:
12 |        "somecommand {input} {output}"
```

Listing 3.1: Snakemake file example

In Listing 3.1, there are two rules, rule `targets`, and rule `plot`. In the rules, there are defined some directives, `input`, `output`, and `shell`. The `input` and `output` directives are followed by a list of files that are expected to be used or created by the rule. The `shell` directive contains the shell command to execute, which is used to create output files from input files.

If the user wants to write Snakefile rules, they have to be sure that each shell command receives the input data in a correct format, because Snakemake does not have any semantic checks.

Snakemake does not provide any graphical user interface for workflows creation.



## Extendability

As Snakemake does not have any graphical user interface and works with tools available in the command-line shell at the run time, it is possible to use various software tools to create a workflow. This means that it is not necessary to extend the functionality of the tool as such.

Instead of addons, the Snakmake has reusable modules that make it easier for the user to create complex workflows. Modularization in the Snake-make comes at different levels listed below.

1. Wrappers
2. Split Snakefiles into smaller reusable parts
3. Subworkflows

The most fine-grained level are so-called wrappers. Wrappers are scripts designed around command-line tools that make it more straightforward for the user to use such tools. For example, the user does not have to remember the order of arguments for the specific tool, as the wrapper solves this problem for them. To make Wrappers available to use, they must be published at the Snakemake Wrapper Repository, as Snakemake automatically downloads them from this repository only. Currently, there is no other way to use the Wrappers locally without publishing into the repository.

The second level is to split reusable parts of a larger workflow into smaller Snakefiles and include them into a master Snakefile via the include statement.

The third and last level is sub-workflows. A sub-workflow is executed independently before the primary workflow is executed. Thereby, Snake-make ensures that all files needed for the main workflow are created or updated [22].

Same as for the previous tools, I will show an example code on how to concatenate two strings using the second level of modularity, i.e., split the larger Snakefile into smaller reusable parts.

```
1 | rule string_concatenation:
2 |     output:
3 |         "{concat_output}"
4 |     shell:
5 |         "echo {concat_strings} > {concat_output}"
```

Listing 3.2: String concatenation in Snakemake

In Listing 3.2, we can see a rule that concatenates strings together and print the result in an output file. The rule is saved in a reusable standalone

file. It uses two variables, `concat_string`, and `concat_output`, the variables are defined in the main Snakefile.

In Listing 3.3, we can see the main Snakefile and how the rule for string concatenation is included using the `include` directive on line 4. On line 1 in Listing 3.3, we can see the definition of strings that we want to concatenate.

```
1 | concat_strings=["hello", "world"]
2 | concat_output="concat_output.txt"
3 |
4 | include: "string_concat/Snakefile"
5 |
6 | rule all:
7 |     input: expand("{file}", file=concat_output)
```

Listing 3.3: Demonstration of the include statement in the Snakemake

## 3.4 GIN

GIN (G-Node Infrastructure) services are a free data management system designed for comprehensive and reproducible scientific data management [25].

GIN is a web-accessible repository store for scientific data based on git and git-annex accessible securely from anywhere while keeping data in sync, backed up, and easily accessible [26].

Git is a free and open-source distributed version control system [27]. Git-annex allows managing files with git, without checking the file contents into git, which is useful when working with larger files than git can currently easily handle [28].

The user could use a combination of the git and git-annex directly; however, GIN provides a tool called Gin client [29] that takes care of a lot of details in the background and thus is easier to use.

For the creation of workflows, GIN offers a tool/micro-service called gin-proc.

### gin-proc

The micro-service named gin-proc lets users design efficient workflows by automating Snakemake (described in section 3.3) and building the workflows with the open-source version of Continuous Integration (CI) service Drone [30].

Gin-proc is a middle-man connecting GIN repository and Drone server. It is divided into two components. The first component is a backend applic-

ation running on Python, and the second component is a web frontend for the application running on Nuxt.js.

In Figure 3.4, we can see a web graphical user interface for gin-proc where users can design workflows. It is possible to choose from two workflow styles – custom and Snakemake. The custom workflow style is used to manually define commands which will be executed to produce output files. The Snakemake workflow style will use the Snakemake tool to create output files.

When a configured workflow is submitted, gin-proc will clone the user’s repository from GIN, generate a `drone.yml` file in the repository and commit it. The `drone.yml` is the configuration file for the Drone service, where workflows are built and executed. It will also automatically configure webhooks in the given repository, which are utilized to notify the Drone service to run workflows.

## Electrophysiological data compatibility

As gin-proc uses the Snakemake to run workflows, the same data compatibility, as described in section 3.3, applies.

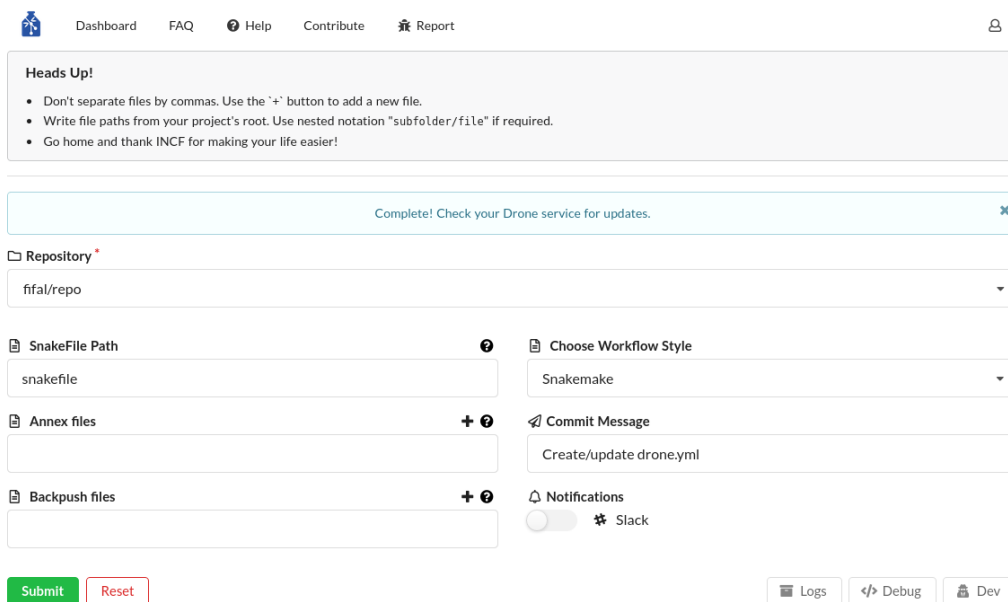


Figure 3.4: Web GUI for gin-proc

## 3.5 VisTrails

VisTrails is an open-source scientific workflow and provenance management system that supports data exploration and visualization. It is written in Python and is available for Mac, Unix, and Windows.

VisTrails enables reproducibility and makes the process of creation, and maintainability of visualizations simpler. Additionally, it allows researchers to explore data effectively. Researchers can examine visualizations from previous versions of dataflow, use different data for dataflow, or compare different results. VisTrails offers a syntactical compatibility check amongst blocks only [31].

Since 2016, the VisTrails is not maintained, and thus no support is available.

### Electrophysiological data compatibility

Although VisTrails provides plugin infrastructure, there are no plugins explicitly designed to process the electrophysiological data. However, VisTrails can interact with various libraries and programming languages through Python [32].

Many Python libraries include methods for signal processing (for example, the NumPy, SciPy, or MNE for Python), which can be applied to electrophysiological data. From such libraries, standalone plugins might be created, or the library might be called directly from a Python source module in a workflow.

### Graphical User Interface

The graphical user interface of the VisTrails is quite extensive; however, the basic use is straightforward, even for an inexperienced user.

The VisTrails stores history of a particular workflow. It is possible to tag every version of a workflow; therefore, it is possible to go back in history. The history feature is available even without the tagging, and every change is stored. The researchers may also include notes to every version of a workspace. We can see the history feature in Figure 3.5.

Modules are ordered in categories and can be added to the workspace using the drag and drop approach. Individual modules do not have any icons, which would quickly distinguish them from others; however, a textual description of the widget is available. The workspace allows for an automatic module alignment.

The graphical user interface of the VisTrails can be seen in Figure 3.6.

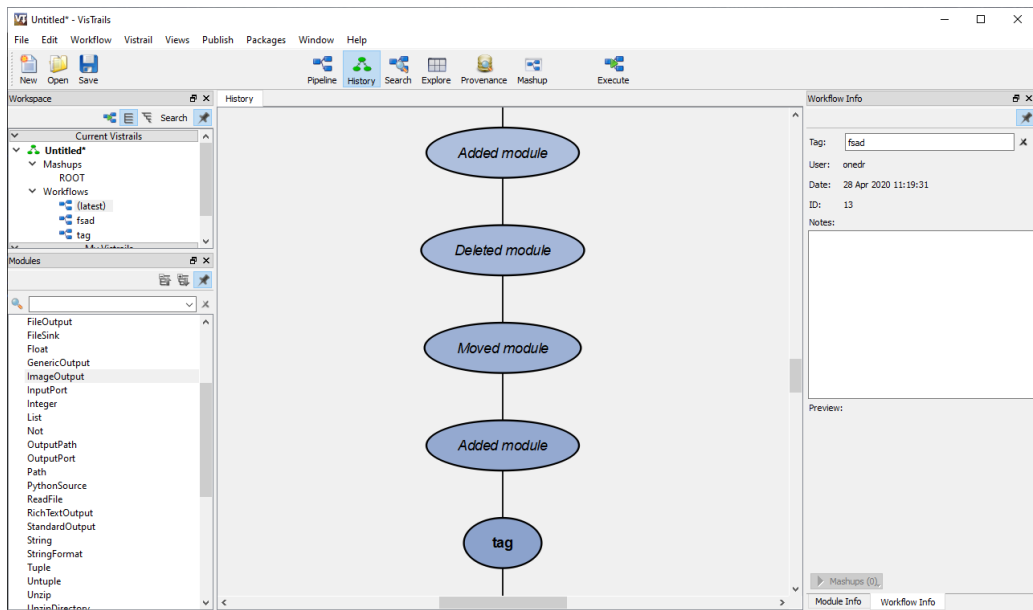


Figure 3.5: The history feature of the VisTrails

## Extendability

Similarly, like the Orange tool, VisTrails provides a plugin infrastructure to integrate user-defined functions and libraries, as well as a developer’s documentation on how to write such packages for VisTrails. A VisTrails package is a collection of Python classes, where each represents a new module [33].

In Figure 3.6, we can see a sample workflow that demonstrates the possibility of utilizing self-created modules. On the left side, there is the created package with one module (1). The module has the same functionality as a widget that was previously created for the Orange tool. We can see two string input modules (2), our module called „SampleEEGModule“ (3), and the output module. In the console (4), we can see the text „Hello World“, which results from our workflow. On the right side (5), there is information about our module. We can see the source code in Section B.2.

## 3.6 Workflow Designer

Workflow Designer is a prototype web-based application written in Java and developed at the University of West Bohemia. It allows drag-and-drop creation, edit, and processing workflows from a predefined library (packages) of methods (blocks). Workflows can be exported or imported in the JSON format to ensure reusability. The Workflow Designer can be used to process

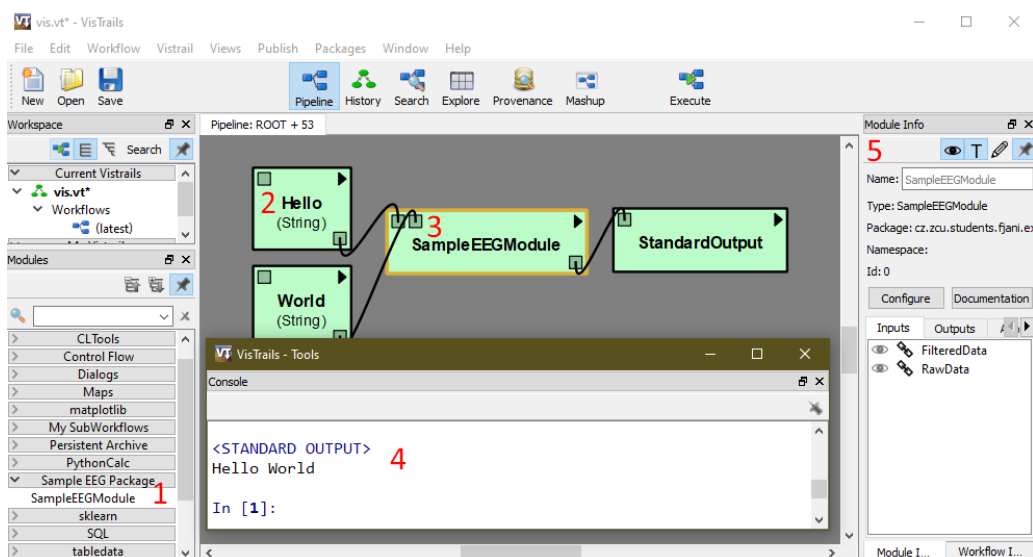


Figure 3.6: Example of a sample workflow in VisTrails

any general computation if the custom method library is available.

The workflows are put together in the web application, and the individual steps of a workflow are called blocks. Each block can have input and output ports, and properties for the block configuration. It is necessary to define a data type for each port and property, as the compatibility amongst blocks is secured on the syntactic level only. Workflow Designer currently offers no inspection on the semantic level [34].

## Electrophysiological data compatibility

The Workflow Designer has been successfully tested on workflows from the ERP domain that utilized signal processing and machine learning. The Designer was integrated with the cloud environment where the datasets for the EEG/ERP data were stored; thus, making it easier to run workflows on the data in the cloud [34].

The custom Java library used for electroencephalographic signal processing is called EEGWorkflow, and it is available as open-source software.

## Graphical User Interface

Workflow Designer has a graphical user interface that is relatively simple featurewise, and from the inexperienced user's point of view, it is understandable at first glance.

New blocks are added to a workspace using a drag and drop method.

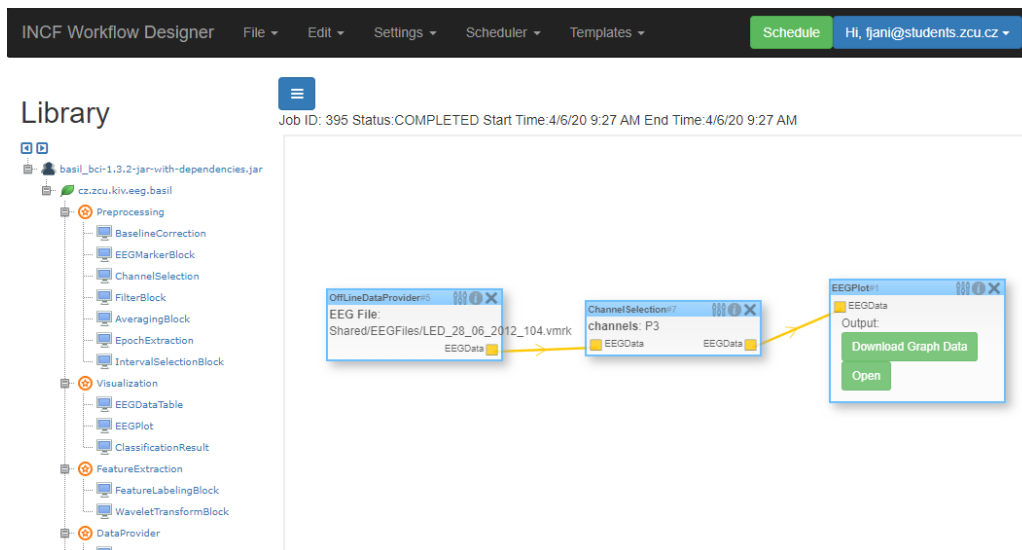


Figure 3.7: Graphical user interface of the Workflow Designer

The blocks on the workspace contain information about the connection port datatype; therefore, it is easy to distinguish compatible blocks. There is no way how to annotate created workflows. This feature could be useful for more extensive, complex workflows.

The graphical user interface of the web application can be seen in Figure 3.7.

## Extendability

The Workflow Designer does not provide any developer’s guide; however, it provides a Git repository, which contains examples and short documentation on how to use the Workflow Designer’s annotation library. These examples consist of different design patterns and project architectures, in which the library can be used.

The annotation library is used to create packages of blocks that are utilized to extend the functionality of the Workflow Designer. Documentation for the annotation library is quite short, but on the other hand, it is easy to follow [35]. Moreover, if something is ambiguous, it is possible to look at the provided examples and learn how to use various annotation types.

In Figure 3.8, we can see the sample workflow that utilizes a custom package. The workflow is used for the concatenation of an array of strings. A result of the workflow is saved in a file that can be downloaded. On the left side in the picture (1), we can see our package with created blocks. In the middle, there is our workflow with two string input blocks (2), block

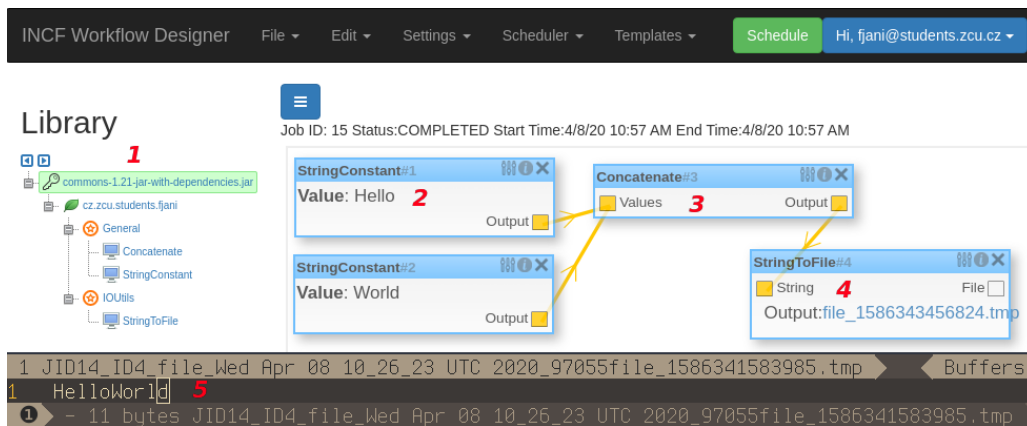


Figure 3.8: Sample workflow for string concatenation in the Workflow Designer

for concatenation (3), and a block that converts a string into a downloadable file (4). At the bottom of the picture, we can see the contents of the result file (5). The source code for the concatenation block (3) is listed in a Section B.3.

### 3.7 Apache Taverna

Taverna is an open-source domain-independent Workflow Management System – a suite of tools used to design and execute scientific workflows. Taverna is written in Java and includes Taverna Engine, Taverna Workbench (the desktop client application), and Taverna Server (used to execute remote workflows). As a part of the Taverna suite, there is also a command-line tool that is available for faster execution of workflows from a terminal [36].

Similarly, like the other described workflow management systems, Taverna provides a syntactical level of compatibility check amongst modules based on data types. Additionally, Taverna deals with the semantical level of the check as well; however, it is a formal semantics; thus, restrictions for input/output parameters only are defined. The semantics of connections between modules (if the link makes sense) is not solved [37].

During the writing of this thesis, the community voted to retire Taverna as a project from Apache Software Foundation on 2020/02/19, which means that the codebase is no longer maintained [38].

### Electrophysiological data compatibility

Taverna Workbench appears in various editions, such as astronomy, enterprise, digital preservation, or bioinformatics. The difference among editions



is at module bundles, which differs with each edition. The bioinformatics release could be interesting for the processing of the electrophysiological data, but on the closer inspection, it is an edition focused mainly on the genomic data processing. For example, one of the modules the bioinformatics edition includes is the BioMart plugin, which is a data warehouse aimed at complex genomic data sets [39].

Taverna Workbench was used to process a high-resolution Stereotactic Electroencephalogram (SEEG) and to compute various measurements for the study of epilepsy seizure networks. However, the processing, as such, took place utilizing a Java-based RESTful API. The Workbench was used only to support the large scale signal processing task and to track the processing pipeline visually [40].

Currently, there are no specialized modules focused on the processing of electrophysiological data.

## Graphical User Interface

The graphical user interface of Taverna Workbench may be overwhelming for inexperienced users, as it is quite extensive and complex.

Similarly, like in the other workflow management systems, new blocks can be added to a workspace by dragging and dropping from the blocks library. The particular input/output ports of a block can not be distinguished at first glance, as they are not displayed in a workspace. However, it is possible to click on a respective block and show its details or add annotations.

The graphical user interface of Taverna Workbench can be seen in Figure 3.9.

## Extendability

The Taverna Workbench is available in version 2.5 and has not been updated since the transition under Apache Software Foundation. It is composed of modules. It is possible to extend the Taverna functionality by creating new modules; however, the developer's guide is incomplete and hard to follow in comparison with VisTrails or Orange guides. The developer's guide is available for the Taverna in version 2.5 only. For the more current 3.0 release, there is no guide.

To make plugin creation for Taverna more straightforward, it provides a Maven archetype; however, this archetype was not updated since May 2014. Even though I was able to generate this archetype using Eclipse IDE, I could not make the plugin work. Apache Taverna is the only system in

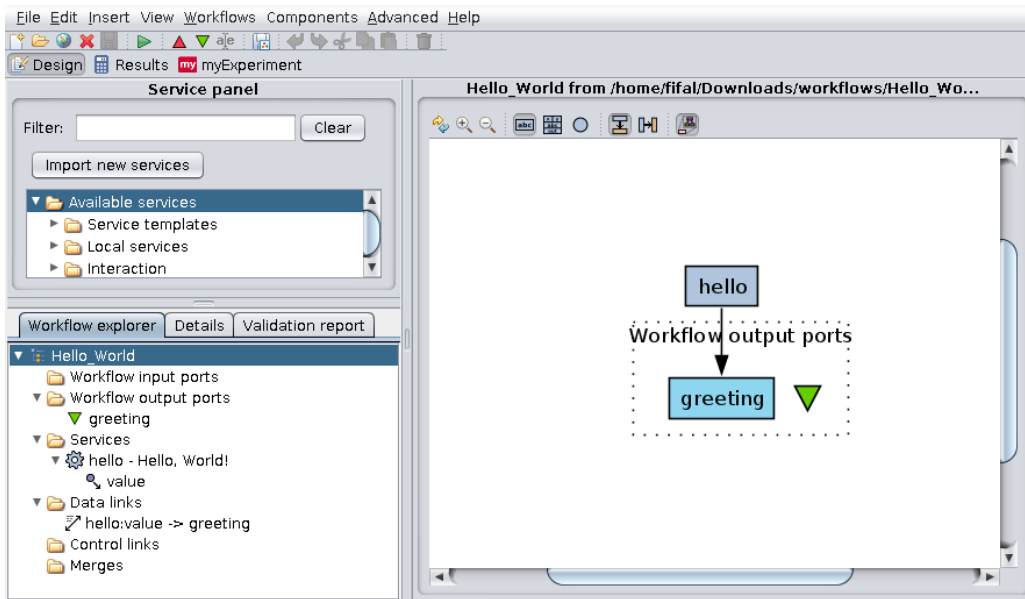


Figure 3.9: Graphical user interface of the Taverna Workbench

which I was not able to extend its functionality [41].

### 3.8 NeuroPype Suite

NeuroPype suite is an application suite that includes a computational engine for biosignal processing (NeuroPype), an open-source visual workflow designer (Pipeline Designer), and tools for interfacing hardware.

NeuroPype is a python-based signal processing and dataflow programming application that is available locally or on the cloud over a REST API. It is released under various proprietary licenses, such as Enterprise, Academic, or Developer.

Pipeline Designer is a Python application that provides a graphical user interface for workflow design. It is based on Orange 3, and the interface is identical as in the Orange. Therefore there are many similarities amongst those applications as far as functionality goes. For example, the Pipeline Designer provides a syntactical level of compatibility check only, same as the Orange. Currently, there is no compatibility check amongst widgets on the semantical level [42, 43].

The graphical user interface can be seen in Figure 3.10. As we can see, the interface is identical to the Orange interface.

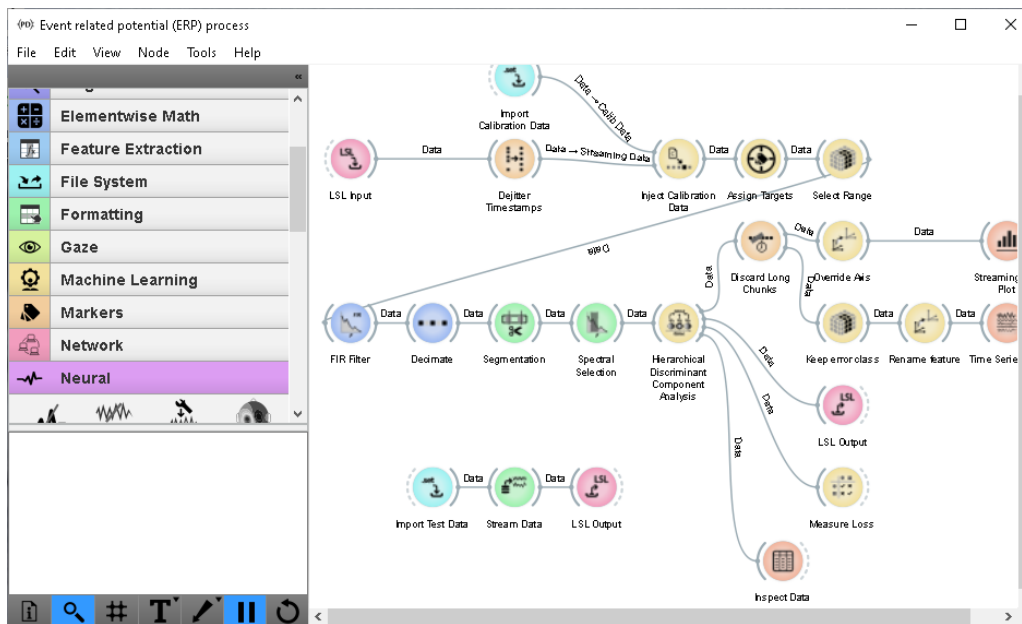


Figure 3.10: Graphical user interface of the Pipeline Designer from the NeuroPype suite

## Electrophysiological data compatibility

NeuroPype suite is designed with a focus on the processing of biological data. Pipeline Designer contains various categories of widgets for biosignal processing. Some of those categories are, for example, cardiac signals, eye tracking analysis, feature extraction, machine learning, or neural signals. NeuroPype suite is thus compatible with electrophysiological data processing, as it provides most of the widgets and analytical methods needed for workflows at the Neuroinformatics lab.

## Graphical User Interface

The NeuroPype’s Pipeline Designer is based on the Orange; therefore, the functionality is the same as described in Section 3.2.

## Extendability

Similarly, like the Orange, NeuroPype’s functionality can be extended with custom nodes written in Python. NeuroPype offers quite a broad developer’s guide that describes the development of custom nodes, and the NeuroPype REST API. Apart from the NeuroPype engine class references, the developer’s guide contains examples of source code. This approach makes

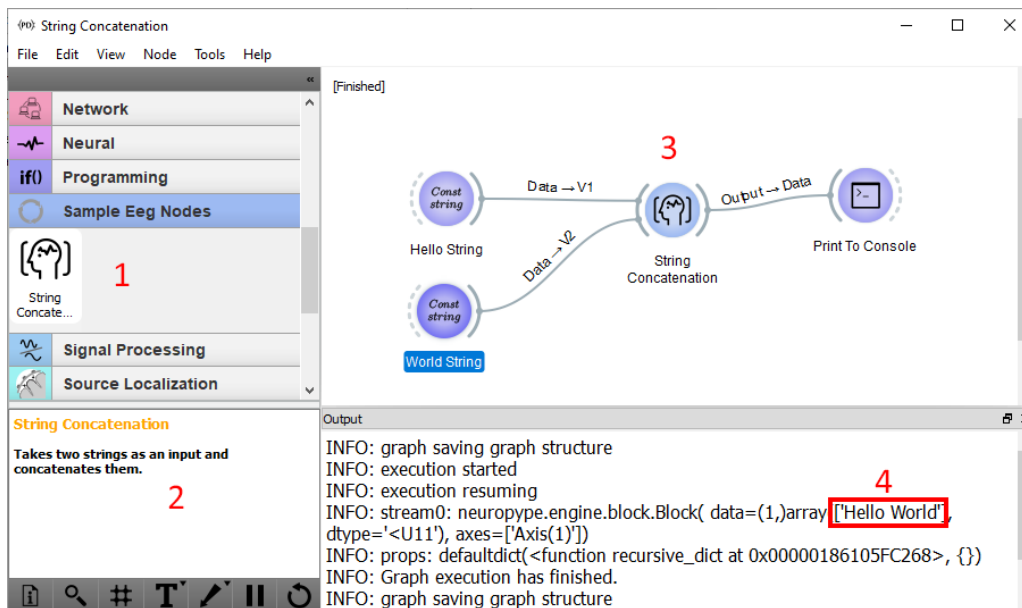


Figure 3.11: Example of a sample node in the Pipeline Designer

it easier for potential developers. However, some parts are undocumented, e.g., how to assign an icon to the custom node. Even though the Pipeline Designer is based on the Orange, the development process of building custom nodes is different [44].

In Figure 3.11, we can see a sample custom node that concatenates two strings. On the left side in the picture (1), we can see a custom node that was automatically imported at the start of the Pipeline Designer. Below the node, there is a textual description (2) of the node. In the middle of the screen (3), we can observe the workflow used to concatenate two strings. Below the workflow, there is a console output. In the console output, we can inspect the result of the workflow marked in a red rectangle (4).

### 3.9 Other large infrastructures

This section will briefly describe infrastructures related to electrophysiological data processing and workflow management that do not fit the needs of the Neuroinformatics lab. Nevertheless, the following infrastructures deserve mention as they are quite large.

## DataJoint

DataJoint is a free, open-source framework for programming scientific databases and computational data pipelines. It is a simplified and conceptually refined relational data model that allows data structure definition, data querying, and visualization of entities and relationships among them. It has been adopted in neuroscience labs for fluent interaction with scientific data pipelines [45].

Every pipeline is composed of one or more tables. Each table represents a specific set of data, and the data can be entered, either manually or automatically. DataJoint allows for computation in scientific data pipelines. The computation, as well as a pipeline definition, can be done using a Python or MATLAB language, and there is no graphical user interface available.

## Human Brain Project

The Human Brain Project (HBP) is a research infrastructure to advance neuroscience, medicine, and computing. There are six ICT research Platforms in the HBP infrastructure – Neuroinformatics, Brain Simulation, High-Performance Analytics and Computing, Medical Informatics, Neuromorphic Computing, Neurorobotics. An HBP Collaboratory is a part of the HBP, and it collects tools from the HBP Platforms in one place [46].

The HBP Collaboratory can be used to collect and organize tools, share data, or use Jupyter<sup>3</sup> notebooks to share ideas, code, and workflows. The workflows shared utilizing the Jupyter notebooks are created by the scripting approach, i.e., users have to write the source code manually. There is no graphical user interface present for workflow creation other than the Jupyter itself.

---

<sup>3</sup>Jupyter – <https://jupyter.org/>

# 4 Workflow management systems requirements

There are several requirements for scientific workflow systems to fit the needs of the neurophysiological lab. In this chapter, I will introduce them and briefly describe them. These requirements will be used in the following chapter to assess the analyzed workflow management systems.

## 4.1 Availability of analytical methods

The primary and essential requirement is the availability of analytical methods for electrophysiological data processing. The workflow management system should include analytical methods for treating such data or provide a way to utilize third-party libraries (see Section 5.1) containing analytical methods either by extending the functionality by custom addons or by the possibility of calling external libraries.

The systems will be evaluated based on the possibility of utilizing suitable libraries and extendability.

## 4.2 Graphical user interface

Workflow systems should offer a graphical user interface that is reasonably straightforward and understandable even for large workflows. It should be clear, which output port is connected to which input port or what are their data types, at first glance to make the analysis and processing more effective.

The graphical user interfaces will be assessed based on the subjective opinion of an inexperienced user who used the workflow management systems for the first time.

## 4.3 Syntactical and semantical compatibility

As workflows' creation process can be quite extensive, the system needs to have some form of a syntactical compatibility check amongst blocks. Syntactical check guarantees that the user can immediately distinguish between compatible blocks (or input/output ports on a block) and cannot use mutually incompatible blocks in a workflow.

It would be helpful if the system dealt with a semantical compatibility check as well. Even though the blocks can be compatible on the syntactic level, they do not necessarily indicate that the order in which they are connected makes sense or is correct.

## 4.4 Community size

Another requirement is the size of the community that is actively using and maintaining a particular system.

The scientific workflow systems solve problems specific to scientific domains that are not present in general workflow management systems. The development of such systems (or additional libraries) is time-consuming and larger communities have a better chance of getting the problem fixed, or adding new functionality in a shorter amount of time.

As it is almost impossible to find out how many researchers are utilizing a particular system, the evaluation of this factor is primarily based on the developer community's activity on respective repositories. The activity anticipates the number of commits, the number of stars, the number of contributors in the repository, and the ratio of open/closed issues in the issue tracking system.

## 4.5 User's manual

Scientific workflows can be particular for each domain, and the same applies to the workflow management systems. Such complex systems should have a comprehensive user's documentation or manual. Besides documentation, the tutorials or workflow examples may provide an additional advantage to the user's learning curve.

The systems will be assessed based on the extent and availability of the before-mentioned materials, as well as ease of finding such materials.

## 4.6 Ease of development

Sometimes, it is necessary to extend the functionality of the workflow management system to fit the specific scientific domain requirements. Besides the possibility of creating custom modules, creating custom modules should be quick and easily understandable. That is guaranteed by a quality developer's guide, type of a development language, the extent of a user's doc-

umentation, code quality, and the scope of source code needed for such modules to work.

In the following sections, other than on before-mentioned factors, the workflow management systems will be assessed based on the experience with the development of an uncomplicated concatenation module for every system. Similarly, like with the graphical user interface, the evaluation is based on an inexperienced user's opinion.



# 5 Comparison of existing workflow management systems

This chapter will introduce suitable third-party libraries that could extend the functionality of analyzed systems. This will allow for better assessment based on the availability of analytical methods. Then I will compare the systems from various points of view based on the requirements presented in Chapter 4. In the last section, I will choose the best workflow management system for the neurophysiological lab's needs.

## 5.1 Suitable third-party libraries

As it is achievable to extend functionality in the analyzed systems to allow for electrophysiological data processing utilizing various third-party libraries, in this section, I will describe available libraries that are suitable for such data processing.

### 5.1.1 MNE-Python

MNE software suite is an open-source Python software for exploring, visualizing, and analyzing human neurophysiological data (MEG, EEG, sEEG, and more).

The MNE software suite includes an open-source software package called MNE-Python. This package provides algorithms in Python that cover multiple data preprocessing methods, source localization, statistical analysis, and estimation of functional connectivity between distributed brain regions. It is tightly integrated with NumPy, SciPy, matplotlib, and Mayavi, core Python libraries for scientific computation and visualization. Moreover, the MNE-Python package has extensive documentation that contains tutorials and examples for new users, user manual, and API reference [47].

The community around the MNE software suite and the MNE-Python package is highly active. It has one or more commits per day in a GitHub repository (total over 15 000 commits), 1 300 stars, and 204 contributors [48].

### 5.1.2 MNE-BIDS

The MNE-Python ecosystem is a significant software package for electrophysiological data analysis and processing. As the user base grew, the support for the BIDS standard was implemented as a dedicated Python software package called MNE-BIDS, to provide an interface for BIDS datasets with MNE-Python.

MNE-BIDS allows researchers to re-organize data into BIDS format, store associated metadata, extract information for preprocessing and read the data into MNE-Python objects for further analysis [49].

The community is relatively active, and the project is under continuous development. In GitHub repository, it has over 850 commits, 28 stars, and 19 contributors [50].

### 5.1.3 NIX-MNE conversion tool

The NIX-MNE conversion tool is a package of Python scripts, that allows conversion of an EDF and BrainVision data formats into the NIX standard, utilizing the MNE-Python library. It also allows reading NIX data files into MNE-Python objects for further analysis and processing.

The package's codebase is quite small; its GitHub repository has 72 commits, zero stars, and one contributor [51].

### 5.1.4 Elephant

Electrophysiology Analysis Toolkit (Elephant) is an open-source library for the analysis of electrophysiological data in the Python programming language.

Elephant focuses on generic analysis functions for spike train data and time-series recordings from electrodes (e.g., local field potentials, or intracellular voltages). Besides, the Elephant project aims to provide a consistent and homogeneous modular analysis framework.

The Elephant's community is active, and the project is updated continuously. The GitHub repository has over 370 commits, 72 stars, and 32 contributors [52, 53].

### 5.1.5 Conclusion

As we can see from the before-mentioned third-party libraries, it is relatively easy to find libraries for the electrophysiological processing that are focused mainly on the Python programming language, as the Python is widespread

amongst the scientific community. In the next sections – summary and comparison, the workflow management system’s programming language will be taken into account, as well as other factors.

## 5.2 Summary of analyzed systems

In this section, I will summarize the analyzed workflow management systems from Chapter 3, each will be assessed based on the requirements described above.

### 5.2.1 Orange

Orange, as is, does not provide any analytical methods for electrophysiological data processing. However, Orange is written in Python and allows for functionality extendability utilizing packages with custom widgets.

The graphical user interface of the Orange is straightforward to use and is understandable even for larger workflows. In the Orange, it is clear, what is port’s data type or where the port is connected, at first glance.

Orange provides a syntactical compatibility check amongst widgets, which does not allow the user to use mutably incompatible blocks. However, the semantical compatibility check is not solved in the Orange.

The community around the Orange is quite active. On GitHub, it has more than 12 000 commits, 2 200 stars, 80 contributors, and it has at least one commit nearly every day. The ratio of open/closed issues is 28/1391.

Orange offers quite an extent user’s manual that includes YouTube video tutorials, workflow examples, a description of individual widgets, and textual tutorials. The hypertext links to respective materials are gathered on a single web page; therefore, it is easy to find relevant pieces of information.

The development of packages with custom widgets is well documented and is easy to follow. Most of the source code files are commented in great detail. However, there are some exceptions, where the file does not contain any comment at all. The size of a source code needed for a simple concatenation module is appropriate, as can be seen in Listing B.1.

### 5.2.2 Snakemake

Snakemake does not provide any modules for electrophysiological data processing. However, Snakemake works with command-line tools or Python scripts; thus, it is possible to utilize third-party libraries (e.g., MNE-Python) and use Snakemake to process such data.

Snakemake does not come with any graphical user interface; therefore, it is not as easy to use as other systems are. Workflows are written in the Snakefiles that are Python-like script files.

The Snakefiles scripts do not have any compatibility check whatsoever, whether it is a syntactic or semantical check.

Snakemake is actively in development, and on its GitHub repository, it has over 3 500 commits, 500 stars, and over 120 contributors. The ratio of open/closed issues is 138/78, which may suggest that developers are focused more on their vision than what the users want, or the development process of fixing issues is sluggish. It is a popular tool as its article was cited 70 times in the first third of the year 2020 already, which suggests it is receiving interest in the Bioinformatics field<sup>1</sup>.

The user's documentation for Snakemake is adequate for the system functionality. It is available online, and it contains tutorials with examples and descriptions of various rules usages. Additionally, it offers links to external resources and talks.

Snakemake has quite broad and up to date documentation. The documentation includes tutorials and API reference. The tutorials are easy to follow, even though they are focused on the genomic domain. The development itself is not as easy as in the systems that provide a graphical user interface and requires some programming knowledge.

### 5.2.3 GIN and gin-proc

Gin-proc is not a workflow management system, as it might seem from the project description. Gin-proc is a middle-man between a GIN repository and a Drone Continuous Integration service. It automates the process of running the Snakemake workflows after the changes are committed into a repository. Therefore, the compatibility with electrophysiological data processing is the same as for the Snakemake (see Section 5.2.2).

Gin-proc has a web graphical user interface used to choose a repository, edit some parameters for Snakemake workflow, and generate a `drone.yml` file for the selected repository.

The syntactical or semantical compatibility check is not present, as it is based on Snakemake.

Gin-proc's community is not active. The project has 70 commits, 0 stars, and four contributors. The last commit is from September 2019, since then, the only change was an update of dependencies to a newer version. The ratio of open/closed issues is 11/29.

---

<sup>1</sup><https://badge.dimensions.ai/details/id/pub.1018944052>

The user’s documentation or manual is non-existent, as the only information about this project is on its GitHub repository that contains only the developer’s guide.

The documentation for the project is insufficient. Even getting the project to work is a nuisance, as many required steps are not documented, e.g., various environment variables and required tools.

### 5.2.4 VisTrails

VisTrails, similarly, like the other analyzed systems does not include any plugins specifically for the processing of electrophysiological data. It is, however, possible to create custom plugins and utilize some of the existing libraries for signal processing or call third-party libraries directly from the Python source module.

VisTrails comes with a graphical user interface that is reasonably easy to use and to understand even for new users. In VisTrails, nevertheless, it is not clear at first glance what is the port’s data type, which can be a slight inconvenience when managing more extensive workflows.

The compatibility check amongst blocks is solved in VisTrails in a similar way as in the other systems, i.e., VisTrails offers the syntactical check only.

The VisTrail’s community is not active anymore, as the project was discontinued in 2016. On its GitHub repository, it has over 6 500 commits, 92 stars, and 24 contributors. However, the repository was not updated in three years, since the last commit date is three years old. The ratio of open/closed issues is 151/934 and the last opened issue is from March of 2018.

The user’s guide is available online, and it contains the description of the installation process and multiple examples on how to use VisTrails. The examples are well documented and include a sufficient amount of images. The documentation is missing the description of individual modules.

The development process of custom plugins is straightforward, and the developer’s guide is easy to comprehend. The documentation is quite broad, and it comprises examples and API reference. The size of the source code for the simple concatenation block is appropriate, as can be seen in Listing B.2.

### 5.2.5 Workflow Designer

Workflow Designer was tested on workflows from the ERP domain that utilized signal processing and machine learning. There is a custom open-source Java library for electroencephalographic signal processing available. Workflow Designer allows for functionality extendability by custom Java

libraries.

Workflow Designer comes with a simple web graphical user interface that is understandable and obvious to use. The users can see the data type of every port at first glance, and together with the fit to screen feature, it allows for straightforward work even for comprehensive workflows.

The compatibility within individual blocks is guaranteed on the syntactical level, and the Workflow Designer does not provide any additional check, for example, on the semantical level.

Workflow Designer is a local project that is developed at the University of West Bohemia. The community is, therefore, not large, as we can see on the GitHub repository statistics. The project has 200 commits, two stars, four contributors, and the last commit is from August 2019.

Workflow designer has a user's manual on its GitHub repository, although it is more of a developer's guide. It contains a textual description accompanied by images on basic usage followed by pieces of information for developers. Nevertheless, more extensive documentation is not present.

The Workflow Designer does not provide the developer's guide; however, it provides a Git repository, which contains examples and short documentation to custom plugin development. The documentation is clear to follow, and the Workflow Designer's annotation library is simple to use. The size of the source code for the simple concatenation plugin is appropriate (see Listing B.3).

### 5.2.6 Apache Taverna

Taverna Workbench, which is a part of a Taverna suite, appears in various editions, and one of them is the bioinformatics edition. Nevertheless, like in other systems, the bioinformatics add-on mainly focuses on the processing of genomic data sets. The Taverna was used to process SEEG data; however, processing took place utilizing Java-based RESTful API; thus, no specialized module is available. There are no modules for the electrophysiological data processing available neither.

Taverna suite comes with a graphical user interface, which is quite challenging to use for inexperienced users. The data types of individual ports are not apparent at first glance.

Taverna offers a syntactical compatibility check and partial semantical compatibility check as well. Semantical check is provided for the input/output parameters only. The semantics of connection amongst modules is not solved.

The Apache Taverna's community decided to retire the Taverna project

from the Apache Software Foundation; therefore, the codebase is no longer maintained. Apache Taverna has 2267 commits, 11 stars, and six contributors. The issue tracker is not available in the GitHub repository; however, it is available on Apache servers. The open/closed issue ratio is 874/189.

The user manual for the Apache Taverna is available online, although, it is fragmented over several web pages. It offers examples, video tutorials, and textual descriptions. However, most of the documentation is not up to date as it was written for the older version of Taverna.

The development process of custom modules is quite challenging. The developer's guide is incomplete and does not provide a sufficient amount of examples. I could not create a custom module for the Taverna Workbench by following the developer's guide, as it was not updated since 2014.

### 5.2.7 NeuroPype Suite

NeuroPype suite is designed with a focus specifically on the processing of biological data. It offers widgets for various biosignal processing categories, e.g., cardiac signals, neural signals, feature extraction, or machine learning. NeuroPype suite provides most of the analytical methods needed for workflows at the Neuroinformatics lab; however, it has a proprietary license.

The Pipeline Designer, which is the graphical user interface of the NeuroPype Suite, is based on Orange's GUI, the functionality is identical and was described above in Section 5.2.1.

Similarly, like the Orange system, the Pipeline Designer offers the syntactical compatibility check only.

The license for the NeuroPype suite is proprietary; however, NeuroPype offers a free Academic license. Although it is uncertain whether the conditions for a free Academic license will change in the future, and the license might become paid. The project is actively developed and maintained. As the project is closed-source, it is impossible to find out repository statistics.

The documentation for the users is quite extensive and is available online. It contains example workflows, installation process, and description of every widget in respective categories. Pictures accompany the description of examples in a vast amount; thus, it is more straightforward for the user to understand it properly. All the information is gathered in one place. Therefore, it is easy to find the required information quickly.

Custom nodes can extend NeuroPype's functionality, and an extensive developer's guide is available; however, some unimportant features are undocumented, e.g., how to assign an icon to the custom node. The development process is relatively straightforward, as the developer's guide contains

Workflow System	Language	Availability of methods		GUI	Syn. / Sem. check	Community size	User Experience	
		Existing methods	Addons				Ease of development	User manual
Orange	Python	no	yes	yes	yes / no	large	easy	excellent
Snakemake	Python-like	no	partial	no	no / no	large	medium	adequate
gin-proc	Python / JS	no	no	partial	no / no	small	medium	insufficient
VisTrails	Python	no	yes	yes	yes / no	medium	easy	adequate
Workflow Designer	Java	partial	yes	yes	yes / no	small	easy	insufficient
Apache Taverna	Java	no	yes	yes	yes / partial	large	hard	insufficient
NeuroPype Suite	Python	yes	yes	yes	yes / no	data unavailable	easy	excellent

Table 5.1: Comparison of analyzed scientific workflow systems

examples of source code. The size of the source code for the custom node is appropriate, similarly to other systems (see Listing B.4).

## 5.3 Conclusion

The best workflow management system will be chosen by the exclusion method, i.e., I will gradually exclude systems that fit the requirements the least, until only one is left.

In Table 5.1, we can see a comparison of analyzed systems assessed by requirements.

At first, we can exclude software that is not a workflow management system, but a utility that helps to connect repository and continuous integration service – the *gin-proc*.

The next software we can exclude does not offer a graphical user interface, which might be a problem for scientists that do not have scripting or programming skills. Moreover, it does not provide any compatibility check whatsoever – *Snakemake*.

During the analysis of existing systems and scientific workflows, in general, I get to the conclusion that most of the scientists like to use scripting languages, such as Python, R, or Matlab. Only two of the analyzed systems do not utilize Python – *Workflow Designer* and *Apache Taverna*. Furthermore, the Workflow Designer’s community is small, as it is a local project. In terms of the Taverna, it is not maintained anymore, and on top of that, the development process is considerably complicated.

Even though the NeuroPype suite offers most of the methods needed for the Neuroinformatics lab and it has an understandable graphical user interface, it is proprietary software. The researchers from the Neuroinformatics lab would like to switch to an utterly open-source system. Therefore we can



exclude the *NeuroPype Suite*.

This leaves us with a pair of systems – VisTrails and Orange 3. Neither of those systems has a module for electrophysiological data processing.

If we compare the graphical user interface of the workspace between these two tools (see Figures 1 and 2), we can see one significant difference. On the VisTrails workspace, there is no indicator above the connections among modules. So at first glance, it is not immediately clear which output connects into which input port. If the user wants to find this information out, he has to mouse over each port of the module, which may be inconvenient when working with larger workflows. On top of that, the VisTrail was discontinued in 2016, meaning that the community is not as active as Orange’s community.

Therefore, the tool that fits the requirements of the Neuroinformatics lab the most is the **Orange 3**.

# 6 Implementation

In this chapter, I will describe the required changes and their implementation in a selected scientific workflow management system to meet the workflows' needs at the neuroinformatics lab.

## 6.1 Changes to Orange 3

In this section, I will describe the changes to Orange 3 needed to respect the workflows at the neuroinformatics lab.

There will be no changes required to the Orange 3 itself. Nevertheless, it is necessary to implement additional widgets to extend the tool's functionality to assure the possibility of processing electrophysiological data.

Some of the analytical methods required were described in Chapter 2, the Python libraries for electrophysiological data processing were described in Section 5.1. The most commonly used library (based on the repository's activity) is the MNE for Python (see Section 5.1.1). On top of that, the library includes most of the analytical methods required. Therefore, this library will be used to implement new widgets.

Additionally, it is necessary to use some libraries that include machine learning methods. One of the experiments utilized for the verification of the implemented library's functionality uses convolutional neural networks and linear discriminant analysis (see Section 7.3). As the experiment uses Keras, Tensorflow, Scikit, and NumPy libraries in its workflow, the same libraries will be used in new widgets to make the implementation more straightforward.

## 6.2 Implemented Widgets

In this section, I will introduce the created widgets for respective categories of data processing described in Chapter 2. The library's repository can be found on a GitLab<sup>1</sup>, together with an installation guide.

---

<sup>1</sup>MNE Widgets for Orange 3 – <https://gitlab.com/fifal/orange-mne-library>

### 6.2.1 Data IO

The recorded electrophysiological data may come in various data formats. Widgets in this category are used to load or save such data files.

#### BrainVision EEG reader

This widget allows researchers to load data files produced by the proprietary BrainVision Recorder software.

We can see the graphical user interface of the widget in Figure 6.1. The widget has one clickable button, read-only text input, and a combo box. After the researcher selects the data `.vhdr` file using the button, the file path is displayed in a text input, so it is evident at first glance, which file is being processed. The combo box is utilized to select the montage type. Montages are useful for various visualizations that require location parameters for each electrode.

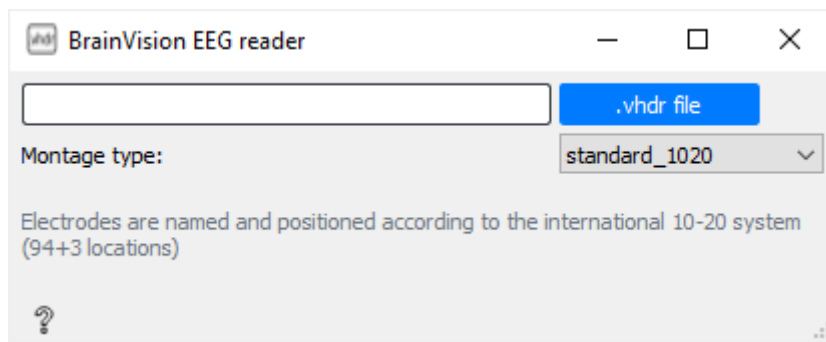


Figure 6.1: The graphical user interface of the BrainVision EEG reader widget

#### EEGLAB reader

EEGLAB reader is a widget that allows loading of `.set` data files.

We can see the graphical user interface of the widget in Figure 6.2. The interface is identical to the widget for BrainVision data file loading. It contains one button for the file selection, read-only text input to show the loaded file's path, and a combo box for the montage type selection.

#### Matlab File Reader

This widget is utilized to load data files in a Matlab format (`.mat`).

The graphical user interface is pictured in Figure 6.3. We can see that the GUI is almost identical to previous file readers, except it does not have

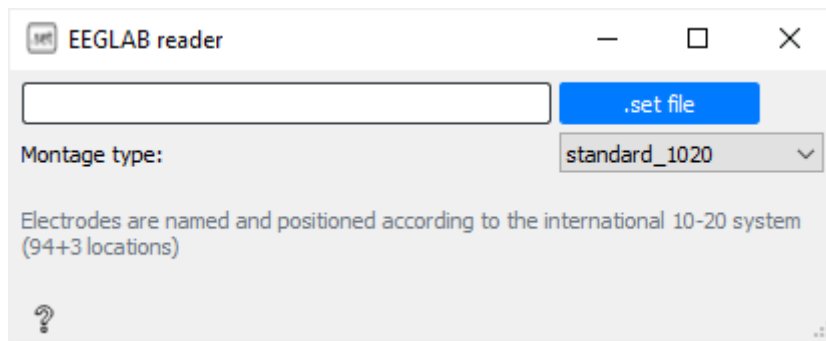


Figure 6.2: The graphical user interface of the EEGLAB reader widget

a combo box for the montage selection. The functionality is otherwise the same as in the previous readers.

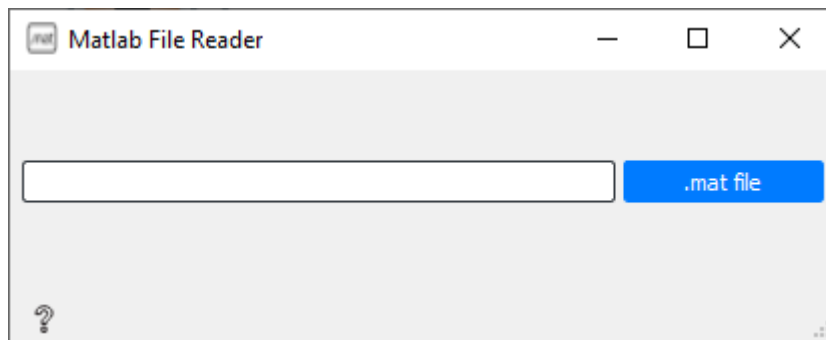


Figure 6.3: The graphical user interface of the Matlab Reader widget

### Fif File Save

If the processing takes a long time, or when we need to load the data in another workflow, it may be useful to save the processed data into a file. The Fif File Save widget has this functionality. It is possible to save three different types of processed data – raw, epochs, or evokeds. Each type has its suffix after the file is saved. The suffix is essential because when loading the data using the MNE library, each type requires a different method.

The graphical user interface can be seen in Figure 6.4. It contains one button which opens a file save dialog to choose the output file path and name.

### Fif Reader

The Fif Reader widget is used to load data files saved by the Fif File Save widget.

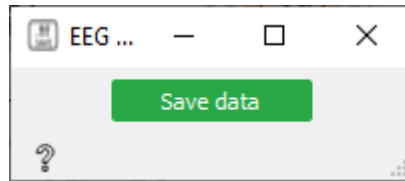


Figure 6.4: The graphical user interface of the Fif File Save widget

The graphical user interface is in Figure 6.5. We can see it is identical to previously described readers (BrainVision reader, or EEGLAB reader). The functionality is the same as well.

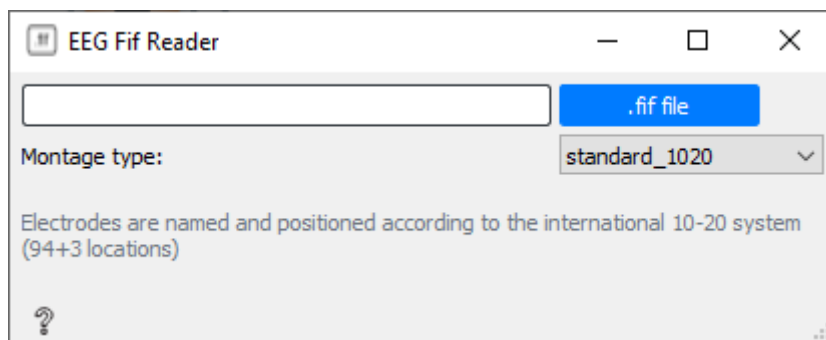


Figure 6.5: The graphical user interface of the Fif Reader widget

## 6.2.2 Preprocessing

Widgets in this category are utilized to prepare the raw data for processing. Such widgets include epoch extraction, channel selection, filtering, and more.

### Channel Select

The raw recorded electrophysiological data may contain several channels, but researchers may need only a few of them for their processing. For this purpose, a Channel Select widget is present in the library. The Channel Select allows researchers to select specific channels from the raw data.

The graphical user interface of the Channel Select widget is pictured in Figure 6.6. If data are loaded and sent to this widget, the users can see what channels are present in the data, and select required channels, by writing channel names, comma-separated, into the text input.

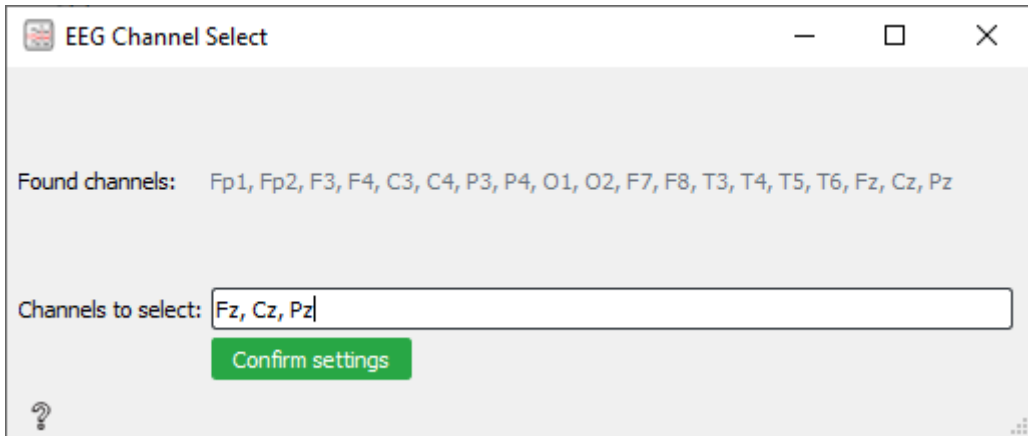


Figure 6.6: The graphical user interface of the Channel Select widget

## Epoch Extraction

The electrophysiological signal recordings can be very long. However, in some cases, the researchers are interested only in part of the data, where particular stimuli have occurred. For such cases, epoch extraction methods are available.

In Figure 6.7, we can see a graphical user interface of the Epoch Extraction widget. In the widget, there are two text inputs and a section with checkboxes. Text inputs are utilized to select the interval of epochs to extract. The checkboxes show stimuli that are available for extraction.

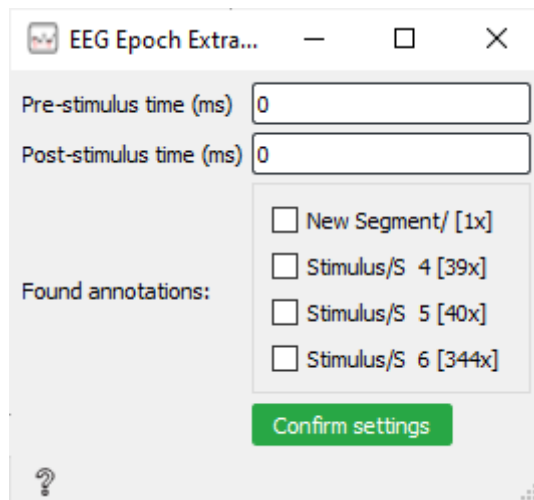


Figure 6.7: The graphical user interface of the Epoch Extraction widget

## Filter

Filtering is an essential step of preprocessing, as recorded signals can contain a lot of signal noise. For example, electrophysiological recordings can be noisy because of power line interference.

The filter widget supports two methods of filtering – IIR or FIR. The graphical user interface is dynamically changing based on the selected method.

In Figure 6.8, we can see the graphical user interface of the Filter widget for the IIR method. The essential parameters of the IIR method are lower and upper cutoff frequency. Additionally, there are several advanced parameters, such as type and order of the filter and their specifying parameters. The types of the filter can be Chebyshev, Butterworth, Elliptic, or Bessel/Thompson.

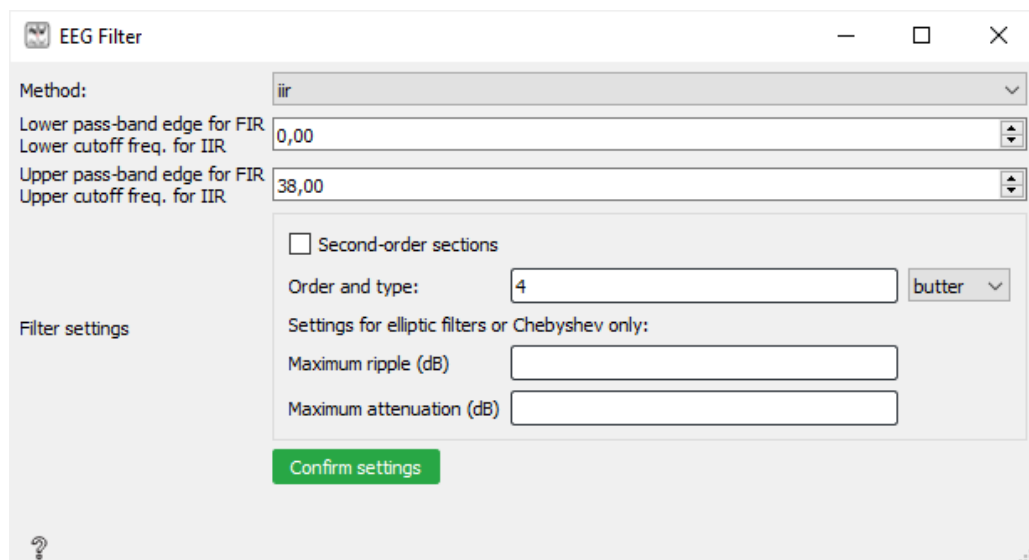


Figure 6.8: The graphical user interface of the Filter widget for the IIR settings

The graphical user interface for the FIR method can be seen in Figure 6.9. Similarly, like the IIR method, the FIR method offers the setting of essential parameters – lower and upper pass-band edge. The specific parameter settings for the FIR method are filter length, FIR window type, and phase.

## Baseline Correction

Baseline correction is a significant part of preprocessing. An uneven amplitude shifts may occur in the recorded signal. For further processing and analysis, it is necessary to compensate for such amplitude shifts.

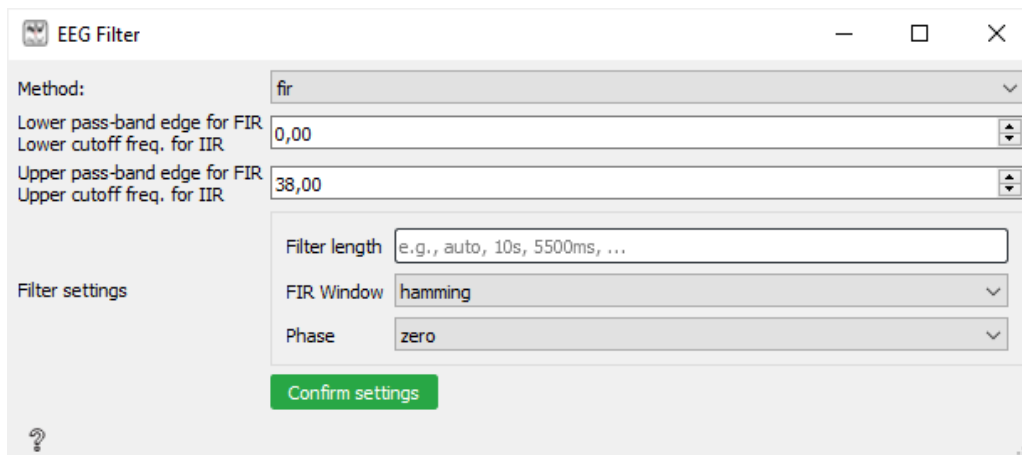


Figure 6.9: The graphical user interface of the Filter widget for the FIR settings

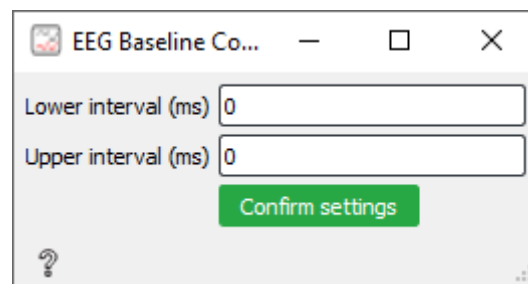


Figure 6.10: The graphical user interface of the Baseline Correction widget

In Figure 6.10, we can see a graphical user interface of the Baseline Correction widget. The widget has two text inputs that are utilized to define an interval from the whole epoch length, which will be used for the baseline correction.

## Artifact Rejection

Individual epochs extracted from the recorded signal may contain artifacts, which may bias the results. Such artifacts may be caused by, for example, eye movement (eyewink), or head movement. The simplest method to detect such artifacts is by the amplitude of the signal.

The implemented library offers a widget called EEG Artifact Rejection, which is utilized to remove corrupted epochs. The widget is pictured in Figure 6.11. The widget has one input, which is used to set the amplitude threshold in microvolts. After the limit is set, the widget iterates over epochs and finds an amplitude peak for each. If the absolute value of the peak is higher than the threshold, the epoch is rejected.



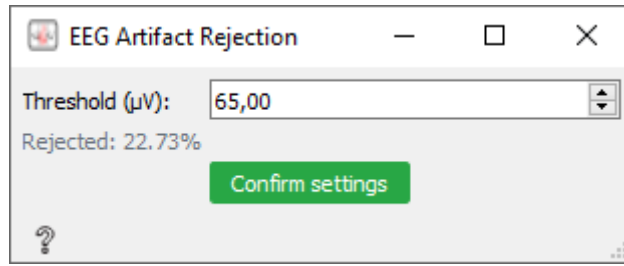


Figure 6.11: The graphical user interface of the Artifact Rejection widget

## Averaging

Evoked potentials in electrophysiological signals are composed of several components. The signal averaging methods can be used to highlight those components, to get a better picture of the signal's final form. For this purpose, there is an EEG Averaging widget available in the created library.

In Figure 6.12, we can see a graphical user interface of the EEG Averaging widget. When the widget receives the data, it shows a group of checkboxes, where each checkbox represents a stimulus category. Apart from the name of the stimulus, a count of stimuli is displayed. Users can select specific stimulus over which will be the signal from epochs averaged. Although it is usually useful to average signals that belong to one particular stimulus, users can average signals over more than one stimulus.

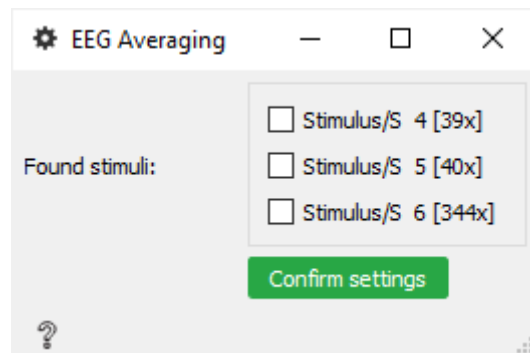


Figure 6.12: The graphical user interface of the Averaging widget

## Concatenation

Orange allows for multiple data inputs through one signal line into a widget; however, it does not allow the output of several data utilizing one signal line out of a widget. This functionality may be a limitation for researchers that have multiple input files. The Concatenation widget is included in

the library to get around this limitation. All inputs must have the same number of channels to concatenate multiple data inputs correctly.

We can see the graphical user interface in Figure 6.13. The GUI is simple as it contains one button. After the button is clicked, all input files are concatenated and sent to the widget's output.

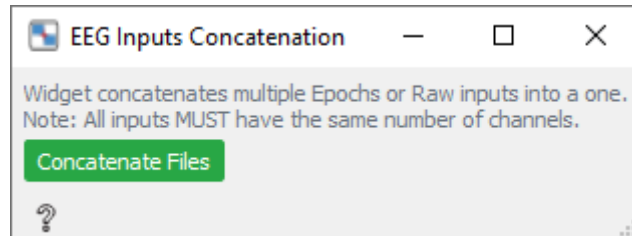


Figure 6.13: The graphical user interface of the Concatenation widget

## Resample

The sampling frequency rate of electrophysiological data can be quite high (up to 1000 Hz). For some types of experiments and processing methods, such high resolution of the signal is not required and may slow down the processing of the data without any advantages. The Resample widget may resample the sampling frequency of the extracted epochs to a range of 1–1000 Hz.

The graphical user interface is pictured in Figure 6.14. The widget contains input for the sampling frequency in Hz and a confirmation button.

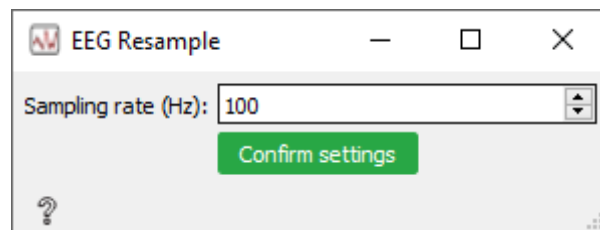


Figure 6.14: The graphical user interface of the Resample widget

## Grand Average

The grand average is an average calculated from multiple averages. This can be beneficial, for example, when examining the data from several testing subjects, as this approach can mitigate some abnormalities or artifacts.

We can see the graphical user interface in Figure 6.15. The widget contains only a label with information about its functionality.

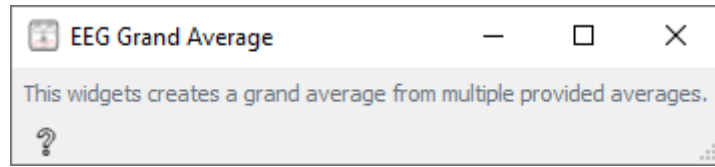


Figure 6.15: The graphical user interface of the Grand Average widget

### 6.2.3 Feature Extraction

Large data sets contain a large number of variables that require a lot of computation resources. The goal of feature extraction widgets is to reduce the number of variables, thus reducing computation resources requirements while accurately describing the original data set.

#### Peak Latency

For some experiments, it is beneficial to find signal peak latency after a particular stimulus has occurred (time delay between stimulus and maximal signal amplitude). To extract this information simply, the library includes a Peak latency widget.

In Figure 6.16, we can see the graphical user interface of the Peak Latency widget. The widget has three inputs for the parameter settings. There are two text inputs for the time interval and a combo box for the mode selection. The time interval is useful if we want to find a peak in a specific range of an epoch. There are three modes of peak finding available – Positive, Negative, and Absolute. The Positive mode will find the peak in the positive values, the Negative mode is the exact opposite, and it will find the peak in the negative values. The Absolute mode will find the most significant peak across positive and negative values. Apart from the peak latency, the widget also shows the peak amplitude.

#### Epochs to Vector

Classification methods require only essential data for them to work correctly. For example, the convolutional neural networks usually work with feature vectors. An Epochs to Vector widget was implemented, to convert processed data to a vector.

In Figure 6.17, we can see that the widget’s graphical user interface is quite minimalistic as it does not have any interactive inputs. The widget contains one label with information about how many epochs were converted to a vector in total.

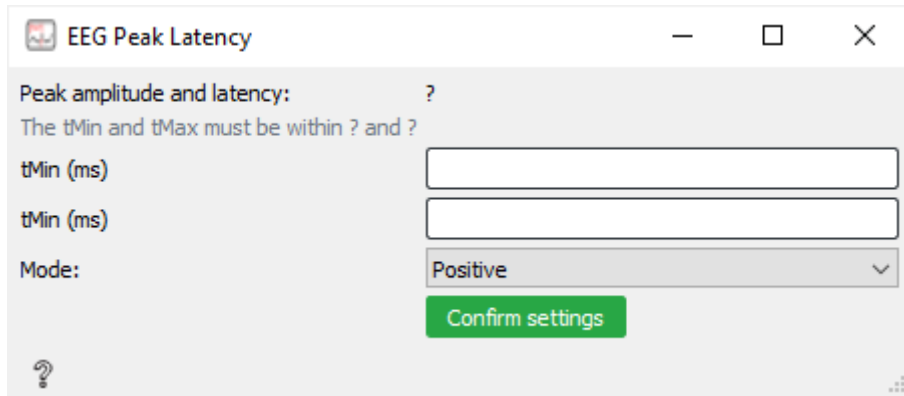


Figure 6.16: The graphical user interface of the Peak Latency widget

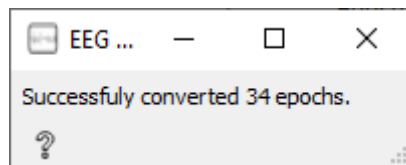


Figure 6.17: The graphical user interface of the Epochs to Vector widget

## 6.2.4 Classification

The purpose of the classification widgets is to find a structure based on mutual relations in data.

Note that there are widgets for preprocessing and feature extraction in this section, not only classification widgets. For example, there is a Reject Amplitude widget with the same functionality as the Artifact Rejection widget, but with one difference. The Reject Amplitude is used in classification workflows to reject amplitude in vectors instead of MNE Python's objects. To prevent user's confusion, the widgets associated with classification are in one category together.

### Prepare Vectors

This widget is specific to the Evaluation of the convolutional neural networks experiment (see Section 7.3). Its source code was taken over from the original source code [54] and modified to fit the widget requirements. Modified parts are commented on in a source code of the widget. Widget expects two input vectors – target and non-target and creates the Classification struct from them, which is required for the classification process. Even though it is specific to the one experiment, it is possible to use this widget in other workflows.

We can see the graphical user interface in Figure 6.18. Its GUI is quite simplistic as it has a label with a description of the widget functionality and one button.

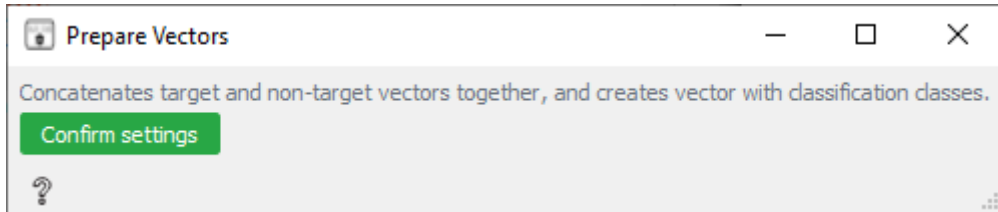


Figure 6.18: The graphical user interface of the Prepare Vectors widget

### Create Classification Struct

The Create Classification Struct widget, similarly, like the Prepare Vectors widget creates a Classification struct from the input vector; however, the classification classes of the vector are based on the stimulus selected in the epoch extraction step. Additionally, it requires only one input vector instead of two input vectors as opposed to the Prepare Vectors widget.

The graphical user interface of the widget is in Figure 6.19.

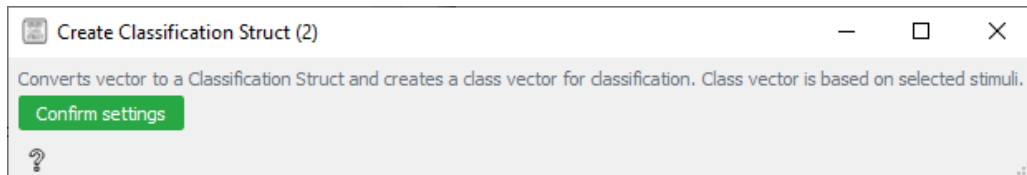


Figure 6.19: The graphical user interface of the Create Classification Struct widget

### Concatenate Classification Structs

The Concatenate Classification Structs widget, as the name suggests, is useful to concatenate multiple Classification structs together. On top of that, it crops all vectors size to match vector with the minimum size to prevent overfitting. For example, when one stimulus has 400 events, and the other has only 20, the classifier would be overfitted on the first stimulus.

The graphical user interface of the widget is in Figure 6.20. As we can see, it is identical to the previous two widgets.

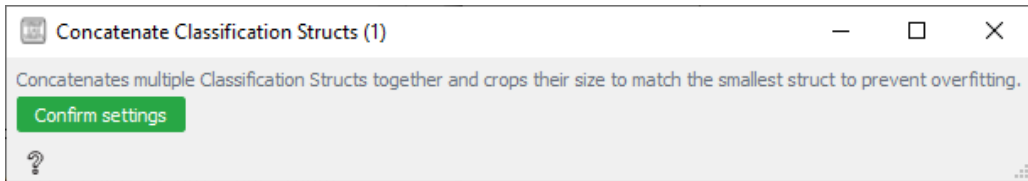


Figure 6.20: The graphical user interface of the Concatenate Classification Structs widget

## Windowed Means

The Windowed Means widget extracts features in the following way. It splits the extracted epochs into intervals based on the minimal latency, maximum latency, and the number of steps. It then calculates the average value for each EEG channel for each range. Those averages form a feature vector. The functionality of this widget was taken from the original source code [54] and modified to the widget's needs. Modified parts are commented on in the source code of the widget.

The graphical user interface of the widget is in Figure 6.21. The widget has five inputs in total and one confirmation button. The first two inputs (Min. latency, and Max. latency) are used to set the interval borders after the stimuli occurrence. The next input (Number of steps) allows us to set how many intervals will be averaged. The pre-epoch input is used to set the time of the pre-stimulus. The last input is used to set the sampling frequency rate of the signal in the extracted epochs.

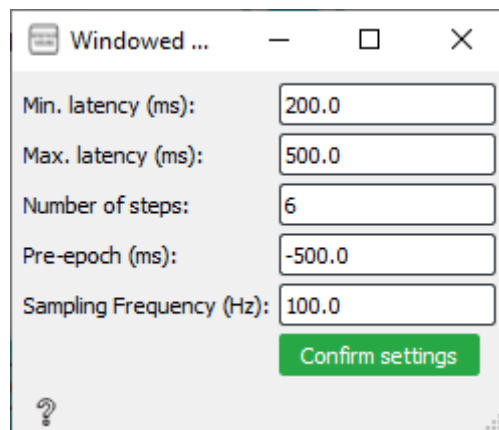


Figure 6.21: The graphical user interface of the Windowed Means widget

## Reject Amplitude

Identically as the Artifact Rejection, the Reject Amplitude widget rejects epochs whose amplitude exceeds the threshold limit. The difference is that this widget works with vectors opposed to the Artifact Rejection, which works with MNE's library objects. Additionally, this widget has a switch between units ( $V$  and  $\mu V$ ) because data from the original experiment are in microvolts, and data processed in Orange are in volts.

We can see the graphical user interface of the widget in Figure 6.22.

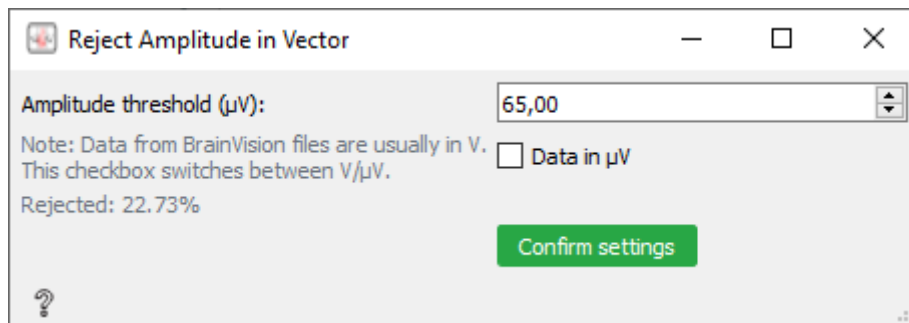


Figure 6.22: The graphical user interface of the Reject Amplitude vector

## CNN Reshape

The CNN Reshape is another widget that is explicitly implemented for the Evaluation of the convolutional neural networks experiment (see Section 7.3). This widget adds a singleton dimension to enable CNN classification using the Convolutional Neural Network widget. The code was taken from the original source code, and the modified parts are commented on in a source code of the widget.

The graphical user interface is in Figure 6.23.

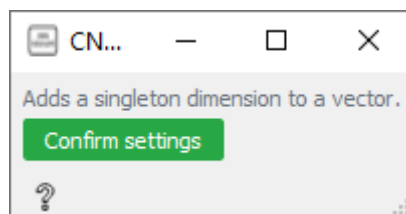


Figure 6.23: The graphical user interface of the CNN Reshape widget.

## Train Test Split

It is necessary to provide training and testing dataset to use classification methods that utilize supervised learning. It is possible to split one extensive dataset into two parts – training and testing, to get such datasets. The Train Test Split widget offers this functionality.

In Figure 6.24, we can see the graphical user interface of the Train Test Split widget. It contains one input for numbers and one button. Using the input for numbers, researchers can easily set the percentage representing the proportion of the dataset to include in the test split.

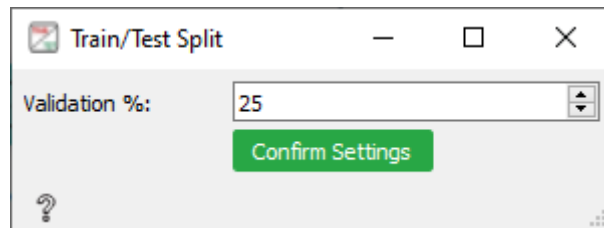


Figure 6.24: The graphical user interface of the Train Test Split widget

## Train Test Select

Similarly, like the Train Test Split, this widget is utilized to prepare the classification data. The only difference is that this widget requires two input datasets instead of one – training and testing, widget then converts datasets into a structure that necessary in classification widgets.

We can see the graphical user interface in Figure 6.25. It contains a label with information about its functionality and a confirmation button.

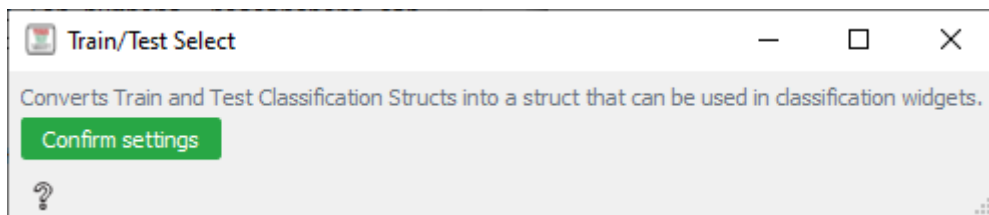


Figure 6.25: The graphical user interface of the Train Test Select widget

## Neighbor Average

The Neighbor Average widget is similar to the Windowed Means widget, except instead of intervals, it averages N number of trials in epochs together.



The source code was taken from the original experiment's source code, and modified parts are commented on in the widget's source code.

We can see the graphical user interface in Figure 6.26. The widget contains two checkboxes to select which datasets will be averaged, and two inputs to set how many trials average together.

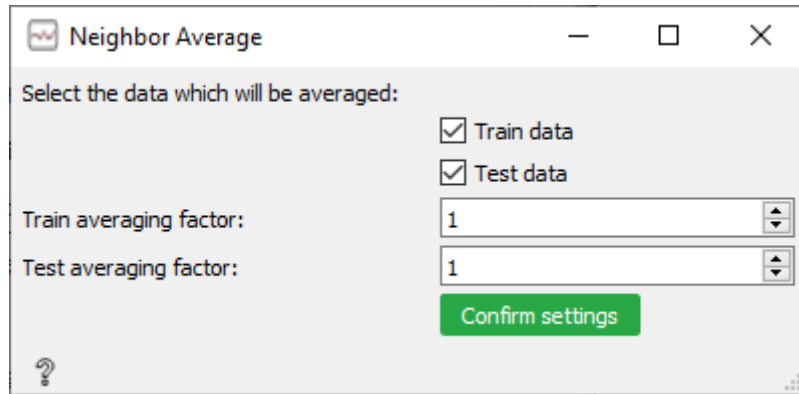


Figure 6.26: The graphical user interface of the Neighbor Average widget

## Linear Discriminant Analysis

One of the classification methods utilized in electrophysiological experiments is a Linear Discriminant Analysis (LDA). It is a classifier with a linear decision boundary. The LDA classification from the scikit library<sup>2</sup> is available through the Linear Discriminant Analysis widget.

The graphical user interface is pictured in Figure 6.27. It has one input for the number of iterations, one confirmation button, and a label with the results. The results include four metrics for both validation and test data – Accuracy, AUC (Area under the ROC Curve), Precision, and Recall. The results include standard deviations for each metric as well.

## Convolutional Neural Network

Convolutional neural networks can be used for the classification of electrophysiological signals. A simple widget that uses Keras<sup>3</sup> and Tensorflow<sup>4</sup> libraries is called Convolutional Neural Network.

We can see the graphical user interface in Figure 6.28. The widget contains two inputs – the number of epochs and the number of iterations, and

<sup>2</sup>scikit-learn – <https://scikit-learn.org>

<sup>3</sup>Keras – <https://keras.io/>

<sup>4</sup>Tensorflow – <https://www.tensorflow.org/>

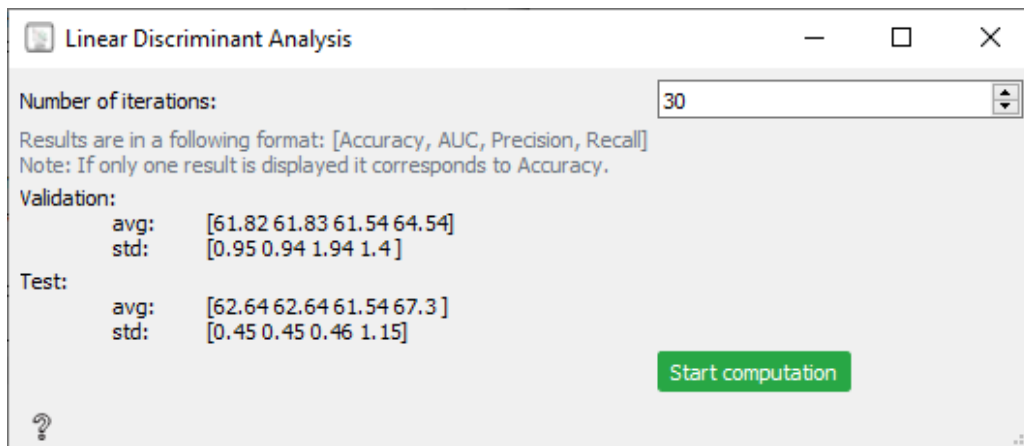


Figure 6.27: The graphical user interface of the Linear Discriminant Analysis widget

one button to start the computation. The implemented model of CNN is specific for the experiment described in Section 7.3. It would be possible to let the researchers specify the model's details; however, the graphical user interface would be highly complex as the models have numerous different settings. For this reason, only the model from the original experiment was implemented as a proof of concept.

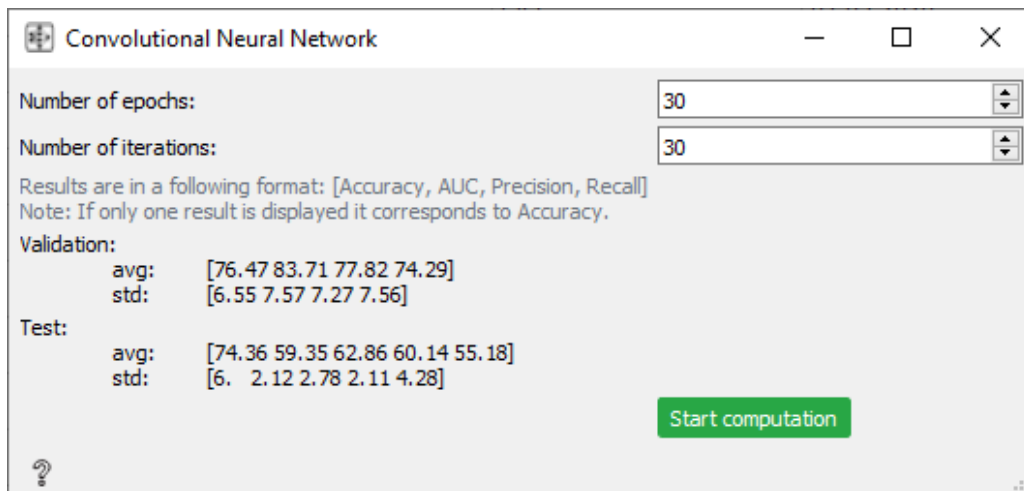


Figure 6.28: The graphical user interface of the Convolutional Neural Network widget

## 6.2.5 Visualization

These widgets are useful to understand the results of the analysis better, as pictures or graphs are more easily understandable for humans than a table of numbers, for example

### EEG Plot

To better understand the raw data, it is necessary to have a way to visualize them. This widget takes advantage of the plot function available on all instances of `mne.io.Raw`, `mne.Epochs`, or `mne.Evoked` from the MNE-Python library.

For each instance, the plot window has different features. For example, the epochs' plot allows setting how many epochs to show at once. The plot of the averaged signal enables to create scalp map (see example map in Figure 7.9) when a part of the averaged signal is selected using the mouse cursor.

In Figure 6.29, we can see an example plot of the raw electrophysiological data, as a result of this widget.

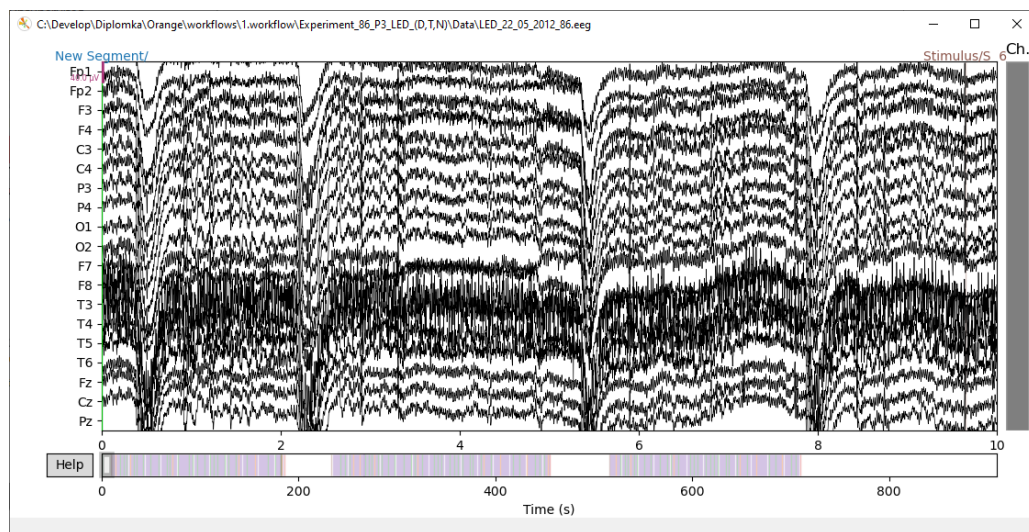


Figure 6.29: The visualization of the raw data using the EEG Plot widget

### Compare Evoked Plot

It can be useful to visualize multiple averages on a single plot to compare respective signals against each other. The Compared Evoked Plot widget ensures this functionality.

We can see an example of the visualization in Figure 6.30.

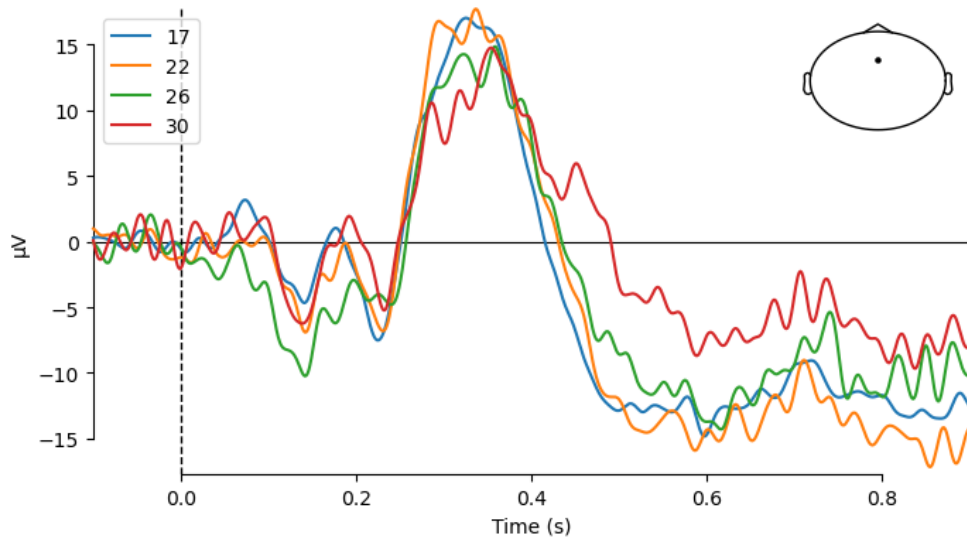


Figure 6.30: The visualization of the multiple averages in a single plot using the Compare Evoked Plot

# 7 Verification of proposed solution

In this chapter, I will verify the proposed solution on selected electrophysiological data. Verification of the created library will be assessed on three existing electrophysiological experiments from the Neuroinformatics lab. In the last part of the chapter, I will discuss the results and describe the limitations of the proposed solution.

## 7.1 Attention of driver during simulated drive

The first experiment on which the proposed solution will be assessed is an experiment that was described in more detail earlier in Section 2.3.2.

The original research deals with EEG and ERP methods under various conditions to investigate the driver's attention. The subjects were stimulated with audio signals during a drive on a car simulator after a usual night's sleep and after a sleep restricted to a maximum of four hours. The P300 components of the EEG signal for ordinary sleep and restricted sleep were analyzed and discussed.

### Experiment steps

The steps of the processing are the following:

1. *Data Filtering* – IIR Filter was applied to data from the Fz, Cz, and Pz electrodes.
2. *Epoch Extraction* – The epochs were extracted in the time interval (-100 ms, 900 ms) in the area of occurrence of the target stimulus.
3. *Rejection of Corrupted Data* – The epochs containing artifacts were rejected.
4. *Baseline Correction* – The baseline was corrected using the interval (-100 ms, 0 ms) before the occurrence of each target stimulus.
5. *Data Averaging* – The epochs of each participant were averaged and stored. Then the grand averages for each experimental session were computed.

6. *P3 Peak Latency* – Maximum amplitude was found in the time interval of a possible occurrence of the P3 component. Additionally, Fractional 50% Peak Latency and Fractional 50% Area Latency were found.

## Workflow

We can see the whole workflow in Orange 3, pictured in Figure C.1. To replicate the experiment in the Orange 3, the set of custom widgets listed below, was utilized (parameters of individual widgets are in Table 7.1):

1. *Data Filtering* – For this part, an EEG Channel Select widget was used to select the Fz, Cz, and Pz electrodes. Then an EEG Filter widget was used to apply IIR Filter.
2. *Epoch Extraction* – An EEG Epoch Extraction widget was utilized to select epochs for target stimulus.
3. *Rejection of Corrupted Data* – For the rejection of the corrupted data, an automatic method was utilized instead of manual rejection. A widget called EEG Artifact Rejection was used, which rejects such data, that has a peak higher than the configured threshold.
4. *Baseline Correction* – The baseline was corrected using an EEG Baseline Correction widget.
5. *Data Averaging* – For the data averaging, an EEG Averaging widget was utilized.
6. *P3 Peak Latency* – An EEG Peak Latency widget was utilized to find a P3 Peak latency. Additionally, an EEG Compare Evoked widget was used to visualize individual grand averages.

## Results

A subset of the raw data from the experiment was selected to verify the custom widgets' functionality. Data from three subjects – 165, 167, and 174, were chosen to reproduce the experiment. The P3 Peak latency was found in the grand average using the maximum amplitude technique only.

If we compare the results (see Table 7.2) with the results from the original experiment article (Table 2 in [15]), we can see that our findings' latency is slightly higher. Higher latency can be caused by the IIR filter parameters that were applied, or by the rejection of the corrupted data technique.

Widget	Parameters				
<i>EEG Channel Select</i>	<i>selected channels</i>				
	FZ				
<i>EEG Filter</i>	<i>method</i>	<i>lower cutoff</i>	<i>upper cutoff</i>	<i>order</i>	<i>type</i>
	IIR	0.1	38	4	butter
<i>EEG Epoch Extraction</i>	<i>pre-stimulus time</i>	<i>post-stimulus time</i>	<i>annotations</i>		
	-100	900	Stimulus/S 5		
<i>EEG Baseline Correction</i>	<i>lower interval</i>	<i>upper interval</i>			
	-100	0			
<i>EEG Artifact Rejection</i>	<i>threshold</i>				
	38				

Table 7.1: Values of parameters for individual widgets used in the workflow

Note that it was impossible to assign an exact data file to each result in the original experiment. Furthermore, it was not possible to identify which data files were recorded under sleep deprivation and vice versa.

The exact parameters for the IIR filter were not discussed in the before-mentioned article; therefore, suitable parameters for electrophysiological data were chosen in our workflow. Additionally, the rejection of corrupted data in the original experiment was done manually. In our workflow, a simple threshold method was utilized for the rejection, which could bias the results even more.

Subject No.	1. recording [ms]	2. recording [ms]	3. recording [ms]	4. recording [ms]
165.	325	337	359	354
167.	343	346	283	357
174.	344	313	327	360
<i>Grand Average</i>	<i>344</i>	<i>339</i>	<i>321</i>	<i>357</i>
<b>Grand Average Combined</b>	<b>GA12</b> [ms]	<b>GA34</b> [ms]	<b>GA13</b> [ms]	<b>GA24</b> [ms]
	<i>342</i>	<i>357</i>	<i>346</i>	<i>353</i>

Table 7.2: Table of the results from the reproduced experiment showing the latency of the P3 component on the Fz electrode, using the maximum amplitude method

If we look at the plot in Figure 7.1 and compare it with the original experiment article (Figure 7.2), we can see that the voltage range is broader in our plot. This may be caused by averaging only a subset of the whole

recorded data. Nevertheless, we can see that the curves of the data are similar.

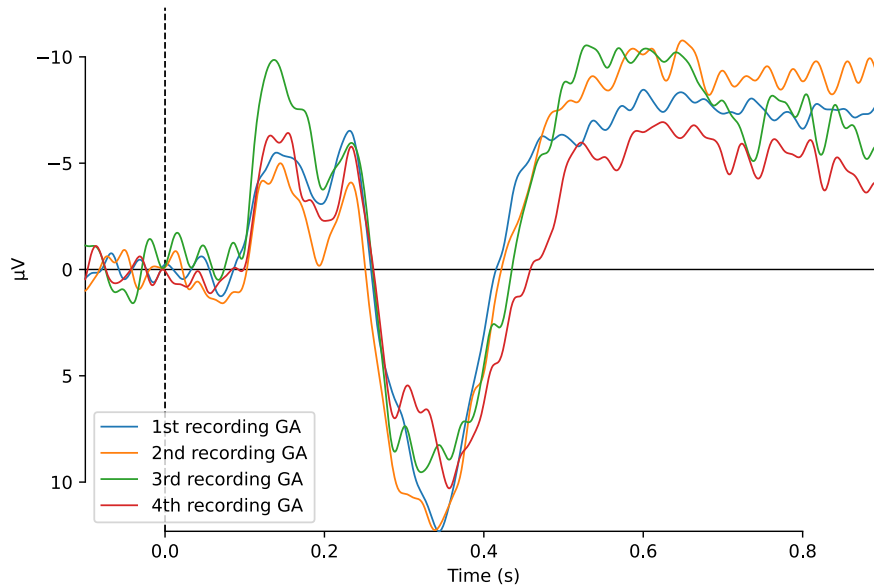


Figure 7.1: Grand Averages computed from three subjects for the 1st, 2nd, 3rd and 4th recording

In Figure 7.3, we can see a grand average for a combination of recordings from the three subjects' averages. We can compare this plot with the original experiment (see Figure 7.4) as well. In this case, similarly, like in the original research, the latency is delayed for the sleep-deprived data (2nd and 4th recording) compared to 1st and 3rd measurements (usual sleep).

## 7.2 Event-related potential datasets based on a three-stimulus paradigm

The event-related potentials technique is widely used in cognitive neuroscience research. The P300 waveform has been explored in many research articles, such as lie detection or brain-computer interfaces (BCI). However, very few datasets are publicly available [55].

The original article [55] presents EEG/ERP data acquired using an odd-ball hardware stimulator based on the three-stimulus paradigm. The data



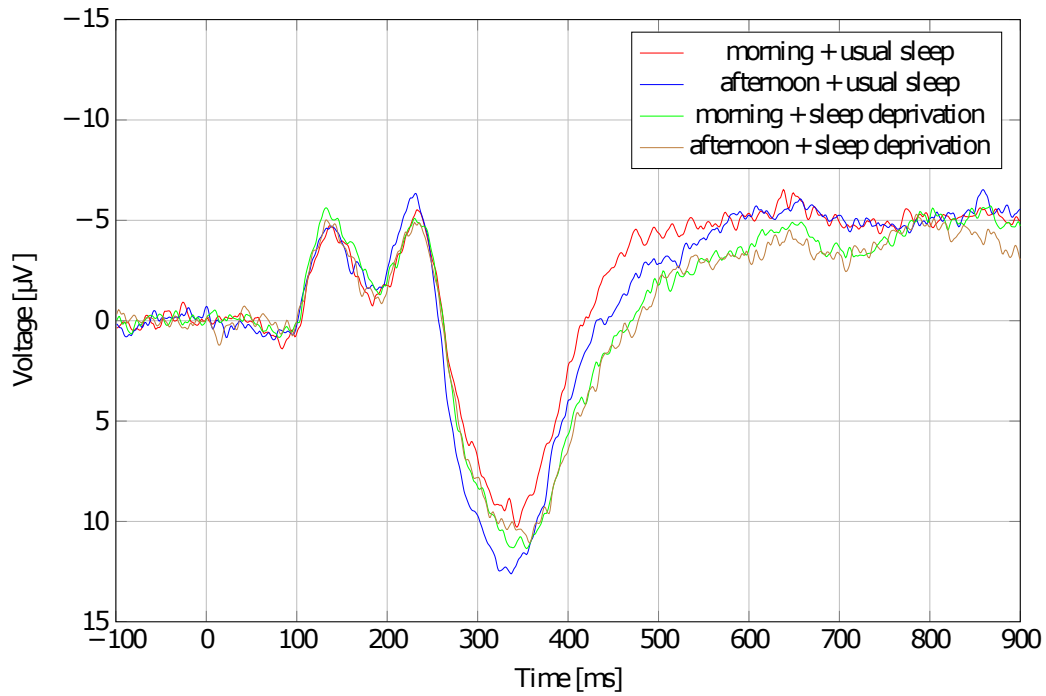


Figure 7.2: Grand Average on the electrode Fz - experimental sessions differ in the daytime and duration of sleep [15]

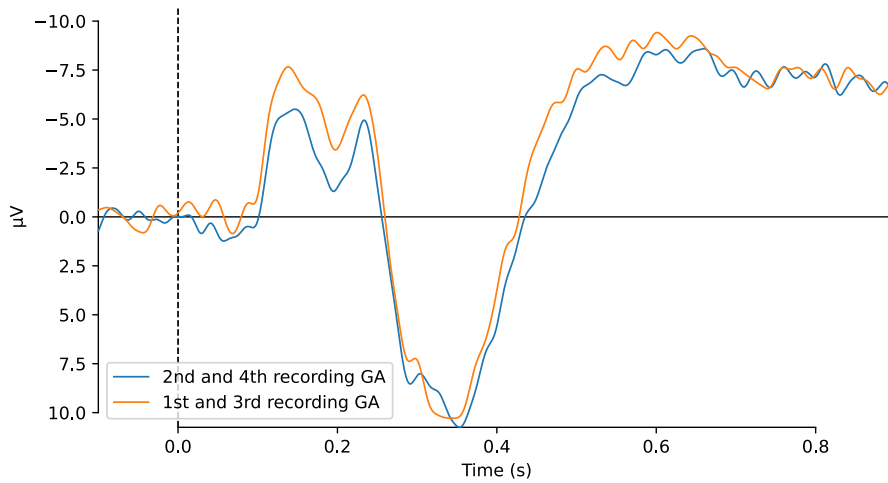


Figure 7.3: Comparison of Grand Averages for the combination of 1st + 3rd recording and 2nd + 4th recording

and metadata are shared in the EEG/ERP Portal<sup>1</sup>. Additionally, the article describes the process and validation results of the presented data. The ori-

<sup>1</sup>EEG/ERP Portal – <https://eegdatabase.kiv.zcu.cz>

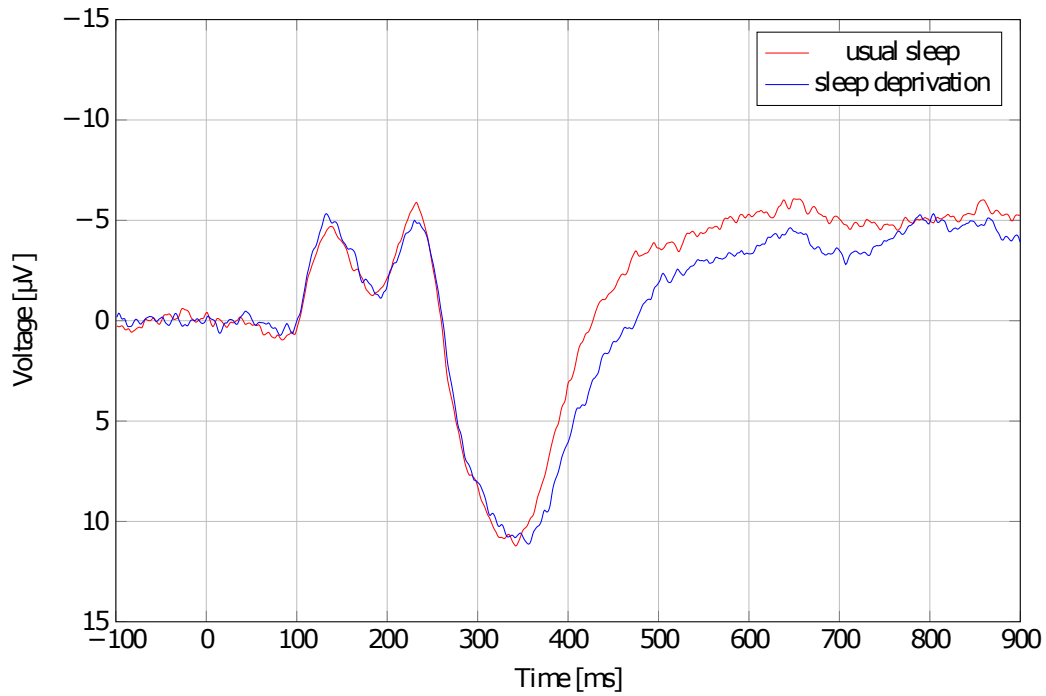


Figure 7.4: Grand Average on the electrode Fz after usual sleep and sleep deprivation [15]

ginal workflow from EEGLAB was recreated in Orange 3 utilizing the created library and will be described in this section.

## Experiment steps

The steps of the experiment's data processing are the following:

1. *Epoch extraction* – The epochs were extracted in the time interval (-500ms, 1000ms) in the area of occurrence of the target stimuli.
2. *Baseline correction* – The baseline was corrected using the (-500ms, 0ms) time interval before the occurrence of the target stimuli.
3. *Filtering* – The signal was band-pass-filtered with the cutoff frequencies of 0.1 Hz and 8 Hz.
4. *Resampling* – Each epoch was down-sampled to 100 samples.
5. *Feature Extraction* – For each epoch, an average in six intervals following the occurrence of stimuli were calculated for each channel and extracted into the final feature vector.

6. *Classification* – A Linear Discriminant Analysis was trained on the training dataset and then evaluated using the test datasets.
7. *Grand Averages* – Additionally, grand averages and related scalp maps were generated to show the ERP response.

## Workflow

The whole workflow in Orange 3 to reproduce this experiment is pictured in Figure C.2. For each step of the processing, the following widgets were utilized (parameters of each widget are in Table 7.3):

1. *Epoch extraction* – An EEG Epoch Extraction widget was utilized to extract epochs for target and non-target stimuli.
2. *Baseline correction* – An EEG Baseline Correction widget was used.
3. *Filtering* – The filtering was performed using an EEG Filter widget. The method of filtering was set to IIR.
4. *Resampling* – An EEG Resample widget was utilized to down-sample the signal to 100 Hz.
5. *Feature Extraction* – Extracted epochs were converted to a vector using the EEG Epochs to Vector widget. Then classification struct was created from the vector by Create Classification Struct widget. An EEG Windowed Means widget was utilized to extract the features.
6. *Classification* – Structs with testing and training data were sent to a Linear Discriminant Analysis widget.
7. *Grand Averages* – Multiple testing datasets were concatenated together using an EEG Concatenation widget. Then required stimuli were extracted, and the average was calculated using an EEG Average widget.

## Results

The results are split into two sections. In the first section, the results of the classification part of the workflow will be discussed. In the second section, the results of grand averages will be compared to the original article.

<b>Widget</b>	<b>Parameters</b>				
<i>Channel Select</i>	<i>selected channels</i>				
	Cz, Fz, Pz				
<i>Epoch Extraction</i>	<i>pre-stimulus</i>		<i>post-stimulus</i>		<i>selected stimuli</i>
	-500		1000		2, 4
<i>Baseline Correction</i>	<i>lower interval</i>			<i>upper interval</i>	
	-500			0	
<i>Resample</i>	<i>sampling rate</i>				
	100				
<i>Filter</i>	<i>method</i>		<i>lower cutoff</i>		<i>upper cutoff</i>
	IIR		0.1		8
<i>Artifact Rejection</i>	<i>amplitude threshold</i>				
	100				
<i>Windowed Means</i>	<i>min. latency</i>	<i>max. latency</i>	<i>number of steps</i>	<i>pre-epoch</i>	<i>sampling frequency</i>
	200	500	6	-500	100
<i>Linear Discriminant Analysis</i>	<i>number of iterations</i>				
	30				

Table 7.3: Values of parameters for individual widgets used in the workflow

## Classification

The results of the LDA classification are visualized in a plot in Figure 7.5. The plot has the same colors and visualization style as the original article’s plot (see Figure 7.6) to allow for a more obvious comparison amongst them.

If we compare the plots, we can see that our workflow results have a higher error rate (red bars) than in the original experiment. The higher error rate may be caused by the classifier’s different training process, as the original research uses EEGLAB, and our classifier uses Python’s library. Additionally, the epochs with artifacts were rejected in the testing dataset. The amplitude threshold was different for each experiment dataset so that the percentage of rejected epochs would be as close as possible to the artifacts percentage in the original article. If we compare the artifact percentage (blue bars) in both plots, we can see that they are almost identical. However, even without the artifact rejection, the error rate was higher in comparison to original results.

In Table C.1 in appendices, we can see the exact results of our workflow. Even though the results are slightly worse than in the original experiment, the accuracy’s median is close to 70 percent, which means that the classifier is significantly better than random guessing.

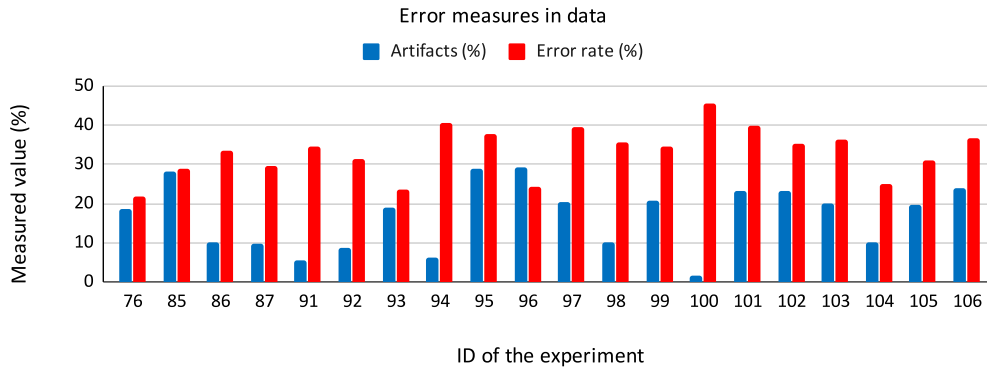


Figure 7.5: The results from the reproduced workflow of the LDA classification visualized in a plot

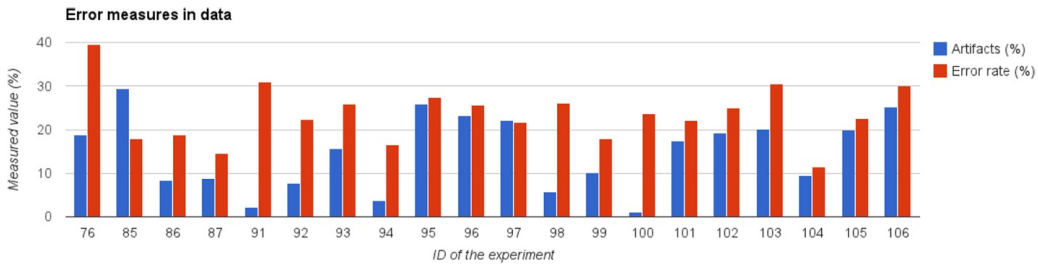


Figure 7.6: The results from the original article of the LDA classification visualized in a plot [55]

## Grand Averages

The last part of this experiment deals with grand averages and related scalp maps that were generated to show how each stimulus type creates a different ERP response.

If we compare our results (Figure 7.7) with the results from the original experiment (Figure 7.8) we can see that the grand averages are identical; thus, this part of the workflow is working correctly.

Even though there is no widget to create scalp maps, it is possible using the EEG Plot widget. If the EEG Plot widget receives averaged epochs, researchers can select a part of the signal in the plot, and the scalp map will be generated. The resulting scalp maps produced utilizing this technique can be seen in Figure 7.9.

If we compare the two scalp maps, we can see that they are similar, but not identical. When particular stimuli were averaged to create scalp maps, I noticed that signal from some channels was corrupted as it contained artifacts. The problematic channels were excluded to create more accurate

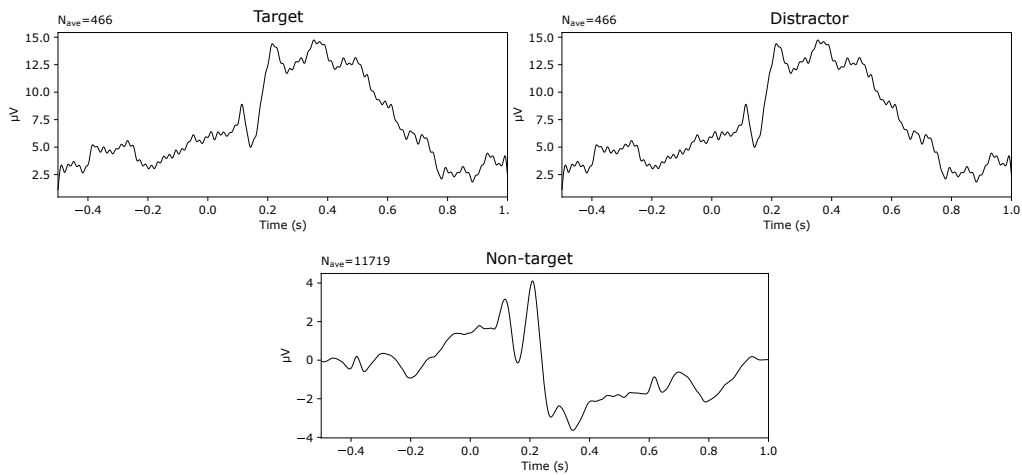


Figure 7.7: Grand averages (Pz electrode) for each stimulus type

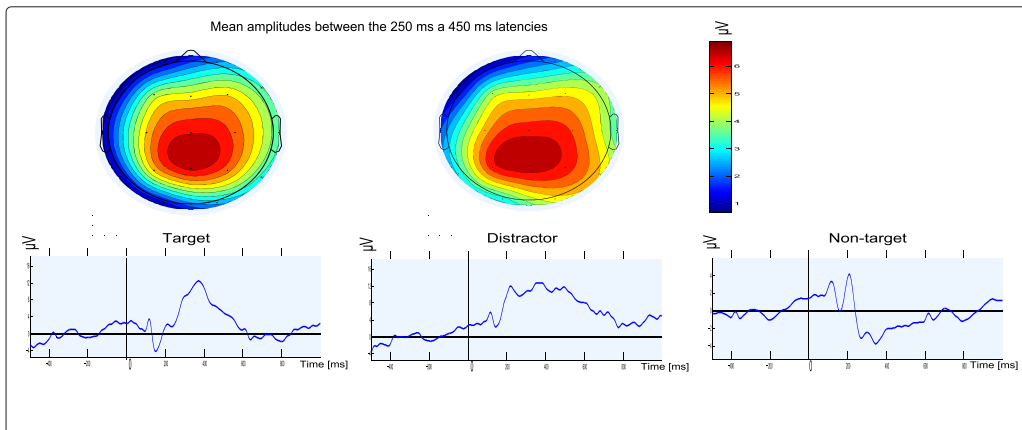


Figure 7.8: Results of grand averages (Pz electrode) for each stimulus type from the original article [55]

maps. However, the resulting scalp maps are slightly different opposed to maps from the original article.

### 7.3 Evaluation of convolutional neural networks using a large multi-subject P300 dataset

Deep neural networks have been studied in various machine learning areas. The signal classification of an event-related potential (ERP) is a highly complex task potentially suitable for deep neural networks. In this experiment,

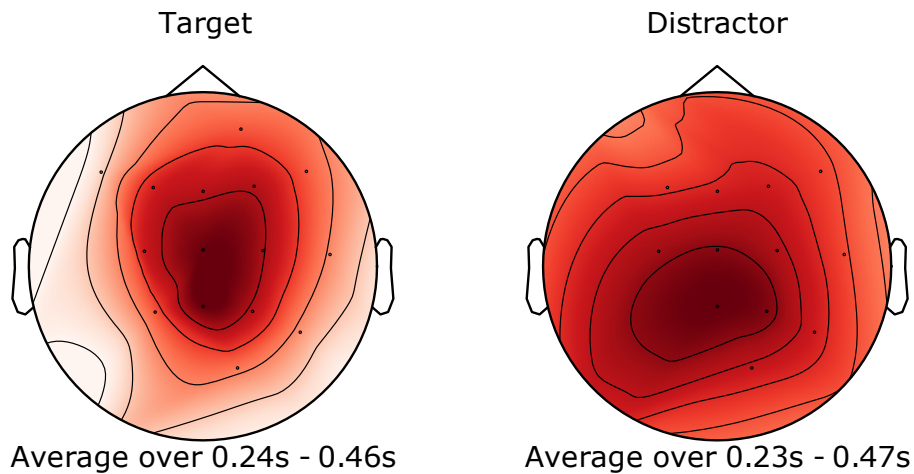


Figure 7.9: The scalp maps for the target and distractor stimuli

convolutional neural networks (CNN) have been compared to linear discriminant analysis (LDA) and support vector machines (SVM) using a large multi-subject publicly available P300 dataset [56].

## Experiment steps

The steps of the experiment's data processing are the following:

1. *Epoch extraction* – Intervals between 200 ms prestimulus and 1000 ms poststimulus were extracted.
2. *Baseline correction* – The prestimulus interval between -200 ms and 0 ms was used for the baseline correction.
3. *Artifact rejection* – Amplitude threshold was set to  $100 \mu V$  to reject severely damaged epochs caused by eye blinks.
4. *Feature extraction* – Feature extraction based on averaging time intervals (Windowed Means) was used. The interval between 300 and 500 ms after stimuli was chosen.
5. *Classification* – Linear Discriminant Analysis, Support Vector Machine, and Convolutional Neural Networks were used for the classification.

## Workflow

The preprocessing of multiple file inputs using the Orange 3 is not as straightforward as when utilizing scripting languages. This is due to the graphical user interface that needs to be quite complex to provide the required functionality.

For this experiment, two workflows in Orange are prepared. The reason for the two workflows is to verify the functionality of implemented widgets using the same input dataset as in the original experiment. The second workflow contains the simplified creation of the input dataset (due to limitations mentioned above), in addition to the classification part. Both workflows will be described in the respective sections.

### Workflow with the original dataset

We can see the whole workflow in Orange 3, pictured in Figure C.3. For the replication of the experiment with original dataset, the following widgets were utilized (the parameters of individual widgets are listed in Table 7.4):

1. *Split Matlab into vectors* – This is a specialized widget implemented only for the requirements of this workflow. It is required to transform the Matlab file into vectors that are further processed in the following widgets.
2. *Prepare Vectors* – It is used to convert vectors into a structure required for the next steps in the workflow.
3. *Reject Amplitude in Vector* – This widget was utilized to reject the corrupted parts of the data.
4. *CNN Reshape* – Widget adds a singleton dimension to the data to enable for CNN Keras classification.
5. *Windowed Means* – Averages selected time intervals that have been calculated using the set parameters. It is used in the linear discriminant analysis branch of the workflow.
6. *Train/Test Split* – Splits the dataset to training and testing parts.
7. *Neighbour Average* – Averages every N number of trials together in the extracted epochs.
8. *Convolutional Neural Network* – Widget that runs CNN classification.
9. *Linear Discriminant Analysis* – Widget that runs LDA classification.



Widget	Parameters				
<i>Reject Amplitude</i>	<i>amplitude threshold</i>		<i>data in <math>\mu V</math></i>		
	100		selected		
<i>Train/Test Split</i>	<i>validation %</i>				
	25				
<i>Windowed Means</i>	<i>min. latency</i>	<i>max. latency</i>	<i>number of steps</i>	<i>pre-epoch</i>	<i>sampling frequency</i>
	300	1000	21	-200	1000
<i>Neighbor Average</i>	<i>train data</i>	<i>test data</i>	<i>train averaging factor</i>	<i>test averaging factor</i>	
	selected	selected	1	1	
<i>Linear Discriminant Analysis</i>	<i>number of iterations</i>				
	30				
<i>Convolutional NeuralNetwork</i>	<i>number of iterations</i>		<i>number of epochs</i>		
	30		30		

Table 7.4: Values of parameters for individual widgets used in the workflow

### Workflow with the input dataset creation

Three recordings for each guessed number were selected (3x9) to create a target input dataset. Twelve recordings were chosen to create a non-target dataset. Those two datasets were converted to vectors and utilizing the Prepare Vectors widget then connected to the classification part of the workflow.

We can see the whole workflow in Orange 3, pictured in Figure C.4. The widgets numbered 2-9 described in section before were utilized with the same parameters. Additionally, the following widgets were used to reproduce the experiment with input dataset creation:

1. *Inputs Concatenation* – Widget was used to concatenate multiple input files with the same target stimulus (guessed number) before epoch extraction. After the epoch extraction, multiple epochs were concatenated together.
2. *Channel Select* – To assure only channels Fz, Cz, and Pz was present.
3. *Epoch Extraction* – Used to extract target stimuli.

## Results

Two workflows were created to verify the correct functionality of the custom widgets. In this section, the results will be discussed for each respective workflow.

## Workflow with the original dataset

We can see the results from the original and reproduced experiment for LDA and CNN classification in Table 7.5.

As we can see, the LDA classification results are almost identical for both the original and the reproduced versions, which means that the implemented widgets work correctly.

However, the results from the reproduced experiment for the CNN classification are different from the original results described in the article [56]. As the source code for CNN is taken from the original source code, I used the debugger and found out that the `fit` method of the CNN model receives the same input values and has the same parameters; however, the results are different. I was not able to identify the cause. The only part that was changed opposed to the original experiment is that the Tensorflow and Keras libraries were updated to work with a newer version of Python and thus work with Orange 3.

Experiment	AUC	Accuracy	Precision	Recall
LDA – original	61.77% (0.9)	61.76% (0.91)	61.45% (1.9)	64.64% (1.48)
LDA – reproduced	62.26% (1.04)	62.28% (1.05)	62.2% (1.62)	65.34% (1.78)
CNN – original	66.12% (0.68)	62.18% (0.88)	62.76% (1.96)	61.29% (2.49)
CNN – reproduced	84.29% (7.68)	77.26% (6.97)	78.17% (7.67)	75.79% (7.39)

Table 7.5: Comparison of the results from the original and reproduced experiment for the LDA and CNN classification. The value inside parentheses is standard deviation.

## Workflow with the input dataset creation

The results from the original and reproduced experiment can be seen in Table 7.6.

As we can see, similarly, like in the workflow with the original dataset, the LDA classification results are comparable to the results of the original experiment, even though there are small variations in standard deviation.

The CNN classification results are, however, very different. If we compare the accuracy, we can see that the results differ in only four percent. Nevertheless, if we compare the standard deviation (value inside parentheses), we can see that the deviation is several times bigger in the reproduced version. I tried to debug the classification process but could not identify a glaring error in the source code. Because of this, I think such results are caused by the smaller training dataset, as only a subset of all recordings from the original experiment was used. Another thing that could bias the results can be

the updated version of Keras and Tensorflow libraries, as was discussed in the section before.

Experiment	AUC	Accuracy	Precision	Recall
LDA – original	61.77% (0.9)	61.76% (0.91)	61.45% (1.9)	64.64% (1.48)
LDA – reproduced	69.24% (2.91)	62.11% (3.53)	73.88% (3.43)	83.66% (3.65)
CNN – original	66.12% (0.68)	62.18% (0.88)	62.76% (1.96)	61.29% (2.49)
CNN – reproduced	58.34% (14.82)	58.55% (14.99)	48.7% (29.56)	73.33% (44.22)

Table 7.6: Comparison of the results from the original and reproduced experiment for the LDA and CNN classification with custom created input dataset. The value inside parentheses is standard deviation.

## 7.4 Conclusion

In this section, I will evaluate the results of the reproduced experiments and describe the limitations of the proposed solution.

Three existing experiments were chosen for verification. It was shown that it is possible to reproduce the experiments in Orange 3, utilizing the custom library. Most of the results (with one exception) are comparable to the original experiments' results, indicating the correct functionality of the implemented widgets. The one exception is the Convolutional Neural Networks widget, as the results were different in comparison to the original research results. The possible reasons for different results based on my observation might be smaller training dataset or the version of the used libraries, as the source code was taken from the original research source code. Further debugging and research would be necessary to ensure the widget's functionality.

In terms of limitations of using the graphical user interface for the electrophysiological data processing, there are some nuisances in comparison to the scripting approach. For example, Orange 3 allows for multiple inputs to a widget using one signal line but does not provide this functionality for the widget's output. This may cause lengthy workflow creation in the case of multiple input files. It would be possible to solve this by wrapping the data in custom classes; however, this approach would introduce unnecessary complexity to the syntactical check amongst widgets' connections, as it would be necessary to have multiple different wrapper classes. This solution was verified and is available in the original git repository in a branch called „multiple-files“.

Another limitation is that the implemented widgets do not utilize multiple computation threads, which means when the computation takes a long time, the GUI freezes. It is possible to take advantage of multiple threads in Orange 3; however, this was not implemented because of the time reasons.

One more limitation is that when each widget receives input data, it copies them. This may be a problem when working with multiple or large datasets as it increases RAM requirements with each connected widget. Widgets were implemented in this way because Orange does not provide a way to restart a whole workflow from the start. If users were to use a widget that modifies the data, such as channel select, which drops unwanted channels and later decided to have dropped channels present, it would be impossible as the widget would not have a copy of original data. They would have to manually set the widgets from the beginning of the workflow to recover the dropped channels, which would be tedious.

The last limitation is the widgets' complexity needed to ensure correct functionality, which applies mainly to more complicated analytical methods implemented in widgets. For example, the convolutional neural networks have numerous different parameters and settings, making it challenging to create a proper graphical user interface that allows only the compatible settings to be selected. Moreover, the input data may be diverse, which is another challenging factor. On top of that, syntactic compatibility has to be solved, which adds to the complexity of widgets.

## 8 Conclusion

The analysis revealed that there are many existing workflow management systems; however, just a few of them are usable for electrophysiological data processing. On top of that, no system integrated with community-respected repositories for electrophysiological data was found. Suitable systems were further analyzed and assessed on several requirements to fit the needs of the Neurophysiological laboratory KIV/NTIS of the University of West Bohemia. The requirements were the following: availability of analytical methods, graphical user interface, syntactical and semantical compatibility check, community' size, user's manual, and ease of development. Additionally, suitable libraries with electrophysiological data processing methods were described. As a result of the evaluation, an Orange tool was chosen as the most appropriate tool to fit the laboratory's needs.

For the selected workflow management system – Orange 3, a custom library was implemented. The custom library provides 30 widgets for electrophysiological data processing.

The implemented library's functionality was verified on three existing experiments from the Neurophysiological laboratory KIV/NTIS. The workflows of experiments were reproduced in Orange 3, utilizing the custom library, and results were discussed. The experiments were successfully reproduced with comparable results to original experiments with one exception. The one exception is the Convolutional Neural Networks widget, where results differentiated significantly from the original experiment's findings, and further debugging and research is advised.

There are some limitations to the implemented library that were discussed. The most significant is the complexity of the widgets needed to ensure the required functionality. As the analytical methods have various parameters and settings, it is challenging to create a usable graphical user interface. On top of that, it is quite challenging to design the widget to work in general experiments and not only in specific ones.

Overall, it was shown that it is possible to use Orange 3 as an open-source alternative to currently used software at the laboratory. However, the library requires further development to become an alternative to the scripting approach of workflows creation, as such an approach is more flexible. In the current state, the library can be easily used in workflows with a smaller amount of input dataset files, and thus may be an alternative to a scripting approach.

# List of abbreviations

- API** Application programming interface
- BIDS** Brain Imaging Data Structure
- CNN** Convolutional Neural Network
- EEG** Electroencephalography / Electroencephalogram
- ERP** Event-related potential
- GIN** G-Node Infrastructure
- GUI** Graphical user interface
- LDA** Linear Discriminant Analysis
- MEG** Magnetoencephalography
- NIX** Neuroscience information exchange format
- NWB:N** Neurodata Without Borders: Neurophysiology
- REST** Representational state transfer
- SEEG** Stereotactic Electroencephalogram

# Bibliography

- [1] JAYAPANDIAN, C. P. et al. Electrophysiological signal analysis and visualization using cloudwave for epilepsy clinical research. *Studies in health technology and informatics*. 2013, vol. 192. Available at: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4451213/>.
- [2] MCPHILLIPS, T. et al. Scientific workflow design for mere mortals. *Future Generation Computer Systems*. 2009, vol. 25, no. 5. Available at: <https://www.sciencedirect.com/science/article/abs/pii/S0167739X08000873>.
- [3] WANG, Z. – LEE, P. – MCKEOWN, M. A Novel Segmentation, Mutual Information Network Framework for EEG Analysis of Motor Tasks. *Biomedical engineering online*. 06 2009, vol. 8. doi: 10.1186/1475-925X-8-9.
- [4] COHEN-BOULAKIA, S. et al. Scientific workflows for computational reproducibility in the life sciences: Status, challenges and opportunities. *Future Generation Computer Systems*. 2017, vol. 75.
- [5] GIL, Y. et al. Examining the challenges of scientific workflows. *Computer*. 2007, vol. 40, no. 12.
- [6] LUDÄSCHER, B. et al. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*. 2006, vol. 18, no. 10. doi: 10.1002/cpe.994. Available at: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.994>.
- [7] ATKINSON, M. et al. Scientific workflows: Past, present and future. *Future Generation Computer Systems*. 2017. Available at: <https://doi.org/10.1016/j.future.2017.05.041>.
- [8] DIVIŠ, J. Datový model pro analytické metody a systém workflow v elektrofyzilogických experimentech. Master's thesis, University of West Bohemia, 2015. Available at: <http://hdl.handle.net/11025/17902>.
- [9] BRUNNER, C. – DELORME, A. – MAKEIG, S. Eeglab—an open source matlab toolbox for electrophysiological research. *Biomedical Engineering/Biomedizinische Technik*. 2013.
- [10] ARNO. *Graphical user interface of EEGLAB* [online]. 2010. [Accessed 2020/02/13]. Available at: [https://scn.ucsd.edu/wiki/File:Eeglab\\_small.jpg](https://scn.ucsd.edu/wiki/File:Eeglab_small.jpg).

- [11] *Graphical user interface of ERPLAB* [online]. ERP Info, 2014. [Accessed 2020/02/13]. Available at: <https://erpinfo.org/erplab>.
- [12] *BrainVision Analyzer 2 User Manual* [online]. Brain Products GmbH Munich, 2019. [Accessed 2020/03/18].
- [13] *Graphical user interface of BrainVision Analyzer 2* [online]. Brain Products GmbH., 2015. [Accessed 2020/02/18]. Available at: <http://brainvision.co.uk/daily-lfe/5-brainvision-analyzer-2-tips-and-tricks>.
- [14] BERÁNKOVÁ, K. Využití elektrofyziologických dat pro analýzu vlivu hudby na aktivitu lidského mozku během mentální zátěže. Master's thesis, University of West Bohemia, 2019.
- [15] MOUCEK, R. – KOSAR, V. Attention of Driver during Simulated Drive. In *HEALTHINF*, pages 543–550, 2014.
- [16] RUEBEL, O. et al. NWB: N 2.0: An Accessible Data Standard for Neurophysiology. *bioRxiv*. 2019.
- [17] *Orange Data Mining* [online]. University of Ljubljana, 2019. [Accessed 2019/11/24]. Orange Data Mining. Available at: <https://orange.biolab.si/>.
- [18] DEMŠAR, J. et al. Orange: Data Mining Toolbox in Python. *Journal of Machine Learning Research*. 2013, vol. 14. Available at: <http://jmlr.org/papers/v14/demsar13a.html>.
- [19] *Orange3 Bioinformatics* [online]. 2019. [Accessed 2020/03/05]. Orange Bioinformatics. Available at: <https://orange3-bioinformatics.readthedocs.io/en/latest/index.html>.
- [20] BIOLAB. *Orange Development* [online]. University of Ljubljana, 2015. [Accessed 2020/03/09]. Available at: <https://orange-development.readthedocs.io/>.
- [21] *MNE EEG Workflows for Orange* [online]. University of West Bohemia, 2019. [Accessed 2020/03/12]. Available at: <https://gitlab.com/Dumby7/zswi-bci>.
- [22] *Snakemake* [online]. Snakemake, 2019. [Accessed 2019/11/03]. Snakemake Documentation. Available at: <https://snakemake.readthedocs.io/en/stable/>.
- [23] KÖSTER, J. – RAHMANN, S. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*. 08 2012, vol. 28, no. 19. ISSN 1367-4803. doi: 10.1093/bioinformatics/bts480. Available at: <https://doi.org/10.1093/bioinformatics/bts480>.



- [24] *Snakemake Wrappers* [online]. Snakemake, 2019. [Accessed 2020/03/27]. Snakemake Wrappers. Available at: <https://snakemake-wrappers.readthedocs.io/en/stable/>.
- [25] *Overview* [online]. G-Node, 2019. [Accessed 2019/11/03]. G-Node GIN. Available at: <https://gin.g-node.org/G-Node/info/wiki>.
- [26] *What is GIN?* [online]. G-Node, 2019. [Accessed 2019/11/03]. G-Node GIN. Available at: <https://gin.g-node.org/G-Node/Info/wiki/GinAdvantagesStructurei>.
- [27] *Git* [online]. Git, 2019. [Accessed 2019/11/03]. Git. Available at: <https://git-scm.com/>.
- [28] *Git-annex* [online]. Git-annex, 2019. [Accessed 2019/11/03]. Git-annex. Available at: <https://git-annex.branchable.com/>.
- [29] *Gin client* [online]. G-Node, 2019. [Accessed 2019/11/03]. Gin client. Available at: <https://gin.g-node.org/G-Node/Info/wiki/GinUsageTutorial>.
- [30] *gin-proc Microservice for GIN* [online]. G-Node, 2019. [Accessed 2019/11/03]. G-Node GIN. Available at: <https://github.com/G-Node/gin-proc>.
- [31] CALLAHAN, S. P. et al. VisTrails: Visualization Meets Data Management. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 745–747, New York, NY, USA, 2006. Association for Computing Machinery. doi: 10.1145/1142473.1142574. Available at: <https://doi.org/10.1145/1142473.1142574>. ISBN 1595934340.
- [32] ANDERSON, E. W. *The analysis and visualization of electroencephalography data using a provenance-enabled environment and its applications to visualization*. PhD thesis, School of Computing, University of Utah, 2011.
- [33] *Writing VisTrails Packages* [online]. vistrails.org, 2014. [Accessed 2020/03/17]. Available at: <https://www.vistrails.org/usersguide/v2.2/html/packages.html>.
- [34] JEŽEK, P. – VAŘEKA, L. Workflow Designer-A web Application for Visually Designing EEG Signal Processing Pipelines. In *2019 IEEE 19th International Conference on Bioinformatics and Bioengineering (BIBE)*, pages 368–373. IEEE, 2019.

- [35] *INCF Workflow Designer Manual* [online]. University of West Bohemia, 2019. [Accessed 2020/04/06]. Available at: [https://github.com/NEUROINFORMATICS-GROUP-FAV-KIV-ZCU/workflow\\_designer/blob/master/UserManual.md](https://github.com/NEUROINFORMATICS-GROUP-FAV-KIV-ZCU/workflow_designer/blob/master/UserManual.md).
- [36] *Apache Taverna* [online]. The Apache Software Foundation. [Accessed 2020/03/16]. Available at: <https://taverna.incubator.apache.org/introduction/>.
- [37] ŠTĚBETÁK, J. Analytic methods and workflows for EEG/ERP domain: technical report number: DCSE/TR-2013-02. Technical report, University of West Bohemia, 2013.
- [38] *Vote to retire Taverna from Apache Software Foundation* [online]. 2020. [Accessed 2020/03/23]. Available at: <https://lists.apache.org/thread.html/r559e0dd047103414fbf48a6ce1bac2e17e67504c546300f2751c067c%40%3Cdev.taverna.apache.org%3E>.
- [39] *Taverna Workbench Bioinformatics 2.5* [online]. University of Manchester. [Accessed 2020/04/08]. Available at: <http://www.taverna.org.uk/download/workbench/2-5/bioinformatics/>.
- [40] SOCRATES, V. – GERSHON, A. – SAHOO, S. S. Computation of Brain Functional Connectivity Network Measures in Epilepsy: A Web-Based Platform for EEG Signal Data Processing and Analysis. *Studies in health technology and informatics*. 2019, vol. 264.
- [41] *Taverna – Developer documentation* [online]. 2010. [Accessed 2020/04/14]. Available at: <http://dev.mygrid.org.uk/wiki/display/developer/Home>.
- [42] *NeuroPype* [online]. 2020. [Accessed 2020/04/09]. Available at: <https://www.neuropype.io/>.
- [43] *Intheon NeuroPype* [online]. 2020. [Accessed 2020/04/09]. Available at: <https://intheon.io/>.
- [44] *NeuroPype Documentation* [online]. 2020. [Accessed 2020/04/09]. Available at: <https://www.neuropype.io/docs/common>.
- [45] YATSENKO, D. – WALKER, E. Y. – TOLIAS, A. S. DataJoint: A Simpler Relational Data Model. *CoRR*. 2018, vol. abs/1807.11104. Available at: <http://arxiv.org/abs/1807.11104>.
- [46] *Short Overview of the Human Brain Project* [online]. Human Brain Project, 2020. [Accessed 2020/08/05]. Available at: <https://www.humanbrainproject.eu/en/about/overview/>.

- [47] GRAMFORT, A. et al. MEG and EEG data analysis with MNE-Python. *Frontiers in neuroscience*. 2013, vol. 7.
- [48] *MNE : Magnetoencephalography (MEG) and Electroencephalography (EEG) in Python* [online]. MNE, 2020. [Accessed 2020/04/15]. Available at: <https://github.com/mne-tools/mne-python>.
- [49] APPELHOFF, S. et al. MNE-BIDS: Organizing electrophysiological data into the BIDS format and facilitating their analysis. *The Journal of Open Source Software*. 2019, vol. 4, no. 44.
- [50] *MNE-BIDS Github* [online]. MNE, 2020. [Accessed 2020/04/27]. Available at: <https://github.com/mne-tools/mne-bids>.
- [51] *NIX-MNE conversion tool* [online]. G-Node, 2020. [Accessed 2020/04/27]. Available at: <https://github.com/G-Node/nix-mne>.
- [52] *Elephant - Electrophysiology Analysis Toolkit* [online]. Elephant team, 2020. [Accessed 2020/04/22]. Available at: <https://elephant.readthedocs.io/>.
- [53] *GitHub - Elephant - Electrophysiology Analysis Toolkit* [online]. Elephant team, 2020. [Accessed 2020/04/22]. Available at: <https://github.com/NeuralEnsemble/elephant>.
- [54] VAŘEKA, L. *CNNforGTN* [online]. 2019. [Accessed 2020/05/18]. Available at: <https://bitbucket.org/lvareka/cnnforgtn/src/master/>.
- [55] VAREKA, L. – BRUHA, P. – MOUCEK, R. Event-related potential datasets based on a three-stimulus paradigm. *GigaScience*. 2014, vol. 3, no. 1.
- [56] VAŘEKA, L. Evaluation of convolutional neural networks using a large multi-subject P300 dataset. *Biomedical Signal Processing and Control*. 2020, vol. 58. ISSN 1746-8094. doi: <https://doi.org/10.1016/j.bspc.2019.101837>. Available at: <http://www.sciencedirect.com/science/article/pii/S1746809419304185>.

# Appendices

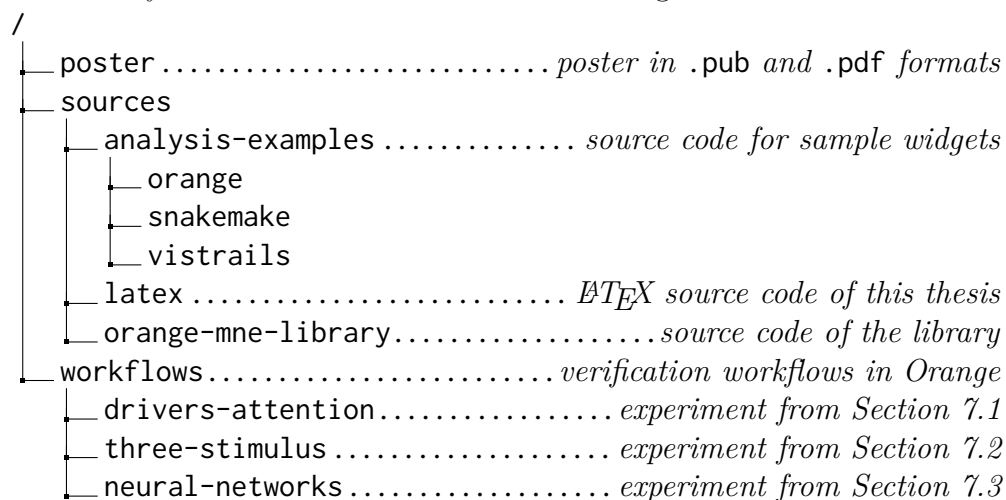
# A Contents of the attached DVD

## Repository

In addition to the enclosed DVD, it is possible to find the library's source codes in the repository on GitLab<sup>1</sup>. However, the repository does not include thesis' poster, experiment datasets, and latex source code.

## DVD

The hierarchy of the DVD's folders is the following:



## Installation and usage

First, it is essential to have a Python installed, at least in version 3.7, and then install the Orange 3 tool. Then it is possible to install the library using the `pip` tool. More information can be found in the `README.md` file in the library's source code directory.

The best way to run the created experiments is to copy the `workflows` directory's content into this path: `c:\dev\orange`, as the input dataset files are loaded automatically when the workflow is opened, and they have an absolute path set. And then open the workflow in Orange.

---

<sup>1</sup>MNE Widgets for Orange 3 – <https://gitlab.com/fifal/orange-mne-library>

# B Source code for sample modules

## B.1 Orange

In Listing B.1, we can see a Python source code for the VisTrails module, which concatenates two strings together.

```
1 from Orange.widgets import widget, gui
2 from Orange.widgets.utils.signals import Input, Output
3
4 class SampleWidget(widget.OWidget):
5     name = "Sample EEG Widget"
6     description = "This widgets concatenates two strings
7     together."
8     icon = "icons/test.png"
9     priority = 10
10
11     v_1 = ""
12     v_2 = ""
13
14     # Definition of Inputs
15     class Inputs:
16         v1 = Input("Raw Data", str)
17         v2 = Input("Filtered Data", str)
18
19     # Definition of Outputs
20     class Outputs:
21         output = Output("Output data", str)
22
23     want_main_area = False
24
25     # Widget initialization
26     def __init__(self):
27         super().__init__()
28
29         # GUI
30         self.label = gui.widgetLabel(self.controlArea, "The
31         result is: ??")
32
33     # Setter for input 1
34     @Inputs.v1
35     def set_constant(self, data):
36         self.v_1 = data
37         self.commit()
```

```

35
36     # Setter for input 2
37     @Inputs.v2
38     def set_data(self, data):
39         self.v_2 = data
40         self.commit()
41
42     # Computation
43     def commit(self):
44         result = self.v_1 + " " + self.v_2
45         self.label.setText("The result is: " + result)
46         self.Outputs.output.send(result)

```

Listing B.1: Source code of the sample module for Orange

## B.2 VisTrails

In Listing B.2, we can see a Python source code for the VisTrails module, which concatenates two strings together.

```

1  from vistrails.core.modules.config import IPort, OPort
2  from vistrails.core.modules.vistrails_module import Module,
   ModuleError
3
4  class SampleEEGModule(Module):
5     # Definition of input ports
6     _input_ports = [
7         IPort(name="RawData", signature="basic:String"),
8         IPort(name="FilteredData", signature="basic:String")
9     ]
10
11    # Definition of output ports
12    _output_ports = [
13        OPort(name="OutputData", signature="basic:String")
14    ]
15
16    # Module initialization
17    def __init__(self):
18        Module.__init__(self)
19
20    # Definition of module function
21    def compute(self):
22        v1 = self.get_input("RawData")
23        v2 = self.get_input("FilteredData")
24
25        self.set_output("OutputData", v1 + " " + v2)
26

```

```
27 | _modules = [SampleEEGModule,]
```

Listing B.2: Source code of the sample module for VisTrails

## B.3 Workflow Designer

In listing B.3, we can see a Java source code for the concatenation module written for the Workflow designer.

```
1 | package cz.zcu.students.fjani;
2 |
3 | import cz.zcu.kiv.WorkflowDesigner.Annotations.*;
4 | import cz.zcu.kiv.WorkflowDesigner.Type;
5 |
6 | import java.util.List;
7 |
8 | @BlockType(type = "Concatenate", family = "General")
9 | public class Concatenate
10 | {
11 |     // Definition of input port
12 |     @BlockInput(name = "Values", type = Type.STRING_ARRAY)
13 |     private List<String> values;
14 |
15 |     // Definiton of output port
16 |     @BlockOutput(name = "Output", type = Type.STRING)
17 |     private String output = "";
18 |
19 |     // Function which is executed when the~workflow is run
20 |     @BlockExecute
21 |     public void process(){
22 |         for (String value: values)
23 |         {
24 |             output += value;
25 |         }
26 |     }
27 | }
```

Listing B.3: Source code of the Concatenation module for the Workflow Designer

## B.4 NeuroPype Suite

In Listing B.4, we can see a Python source code for the node, which concatenates two strings.

```
1 | from ...engine import *
```



```

2
3 class SampleEEGNode(Node):
4     # Input ports
5     v1 = DataPort(str, "Raw data", IN)
6     v2 = DataPort(str, "Filtered data", IN)
7
8     # Output ports
9     output = DataPort(Packet, "Output data", OUT)
10
11     v1_str = ""
12     v2_str = ""
13
14     def __init__(self, **kwargs):
15         self._has_emitted = False
16         super().__init__(**kwargs)
17
18     @classmethod
19     def description(cls):
20         """Declare descriptive information about the~node"""
21         return Description(name='String Concatenation',
22                             description="""\
23 concatenates them.
24
25 url='https://example.cz',
26 version='1.0.0', status=DevStatus.
27 production)
28
29 @Node.update.setter
30 def update(self, v):
31     # Updates class properties if not null
32     if not self.v1 == None:
33         self.v1_str = self.v1
34     if not self.v2 == None:
35         self.v2_str = self.v2
36
37 @output.getter
38 def output(self):
39     # Returns output in a~Packet, that can be displayed by
40     the~Print to Console node
41     block = Block(data=[self.v1_str + " " + self.v2_str])
42     chunk = Chunk(block=block)
43     return Packet([chunk])

```

Listing B.4: Source code of the Concatenation module for the Pipeline Designer

# C Verification

Experiment ID	Accuracy [%]	Error rate [%]	Artifacts [%]
76	77.76	21.87	18.60
85	71.69	28.78	28.30
86	66.89	33.55	9.94
87	70.37	29.65	9.85
91	66.52	34.65	5.33
92	68.73	31.33	8.71
93	76.17	23.64	18.84
94	57.37	40.58	6.12
95	62.79	37.69	28.87
96	74.57	24.19	29.13
97	61.97	39.43	20.34
98	62.7	35.70	10.06
99	68.56	34.67	20.76
100	53.3	45.60	1.62
101	59.71	40.00	23.32
102	67.63	35.21	23.15
103	62.69	36.42	19.90
104	75.6	25.00	10.20
105	71.42	30.86	19.50
106	66.09	36.66	24.00
<i>Median</i>	67.26	34.66	-
<i>Standard Deviation</i>	6.5	6.32	-

Table C.1: The results of the LDA classification in reproduced experiment using Orange 3 and custom library

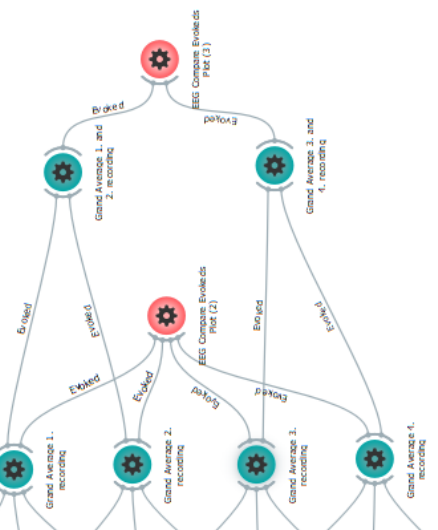
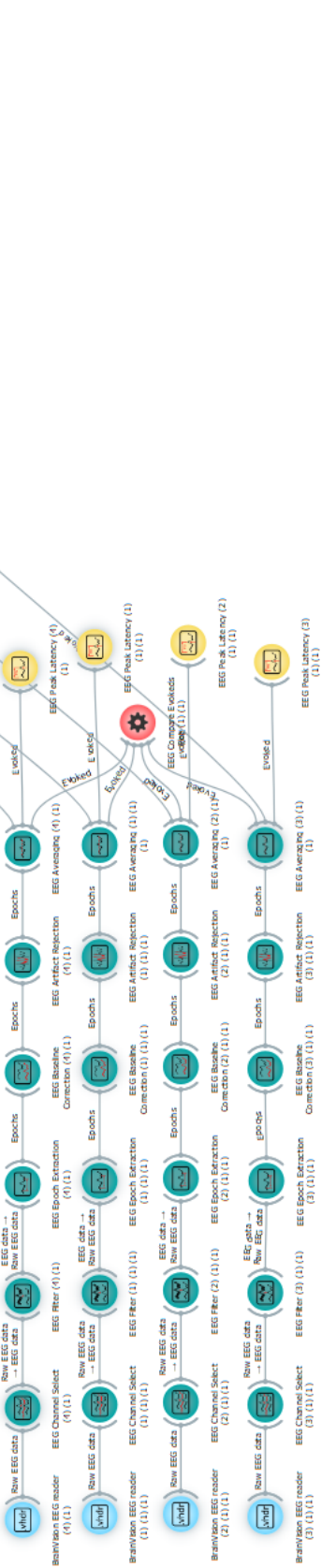
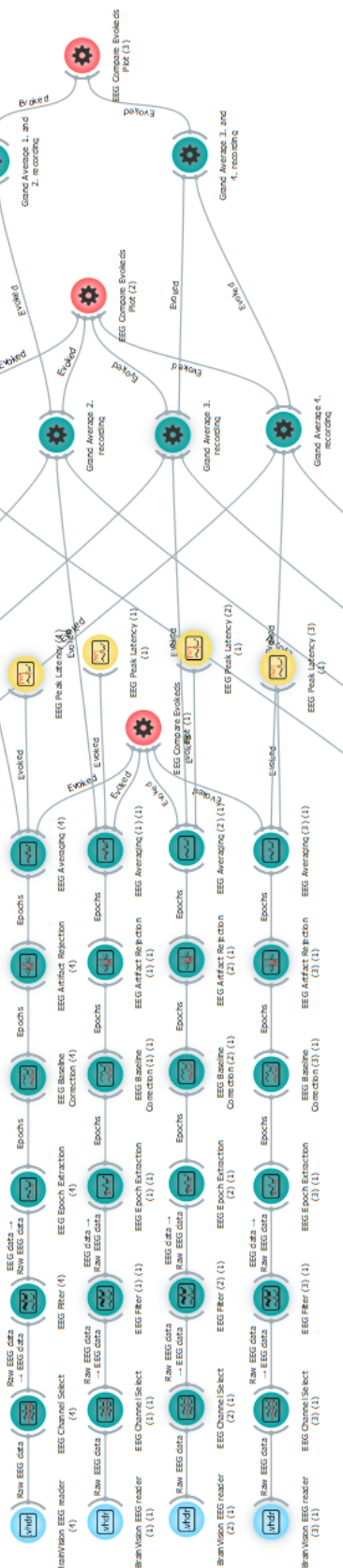
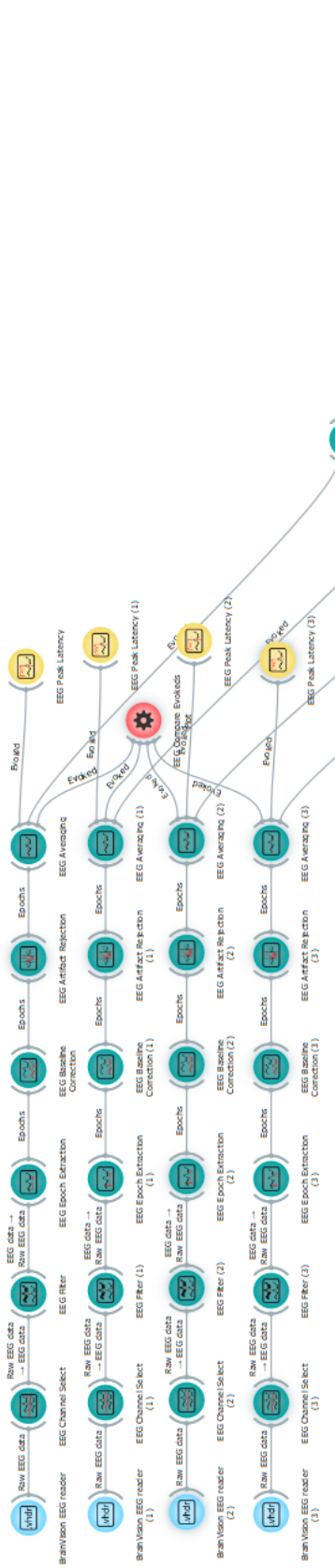


Figure C.1: Workflow used to replicate the Attention of driver experiment

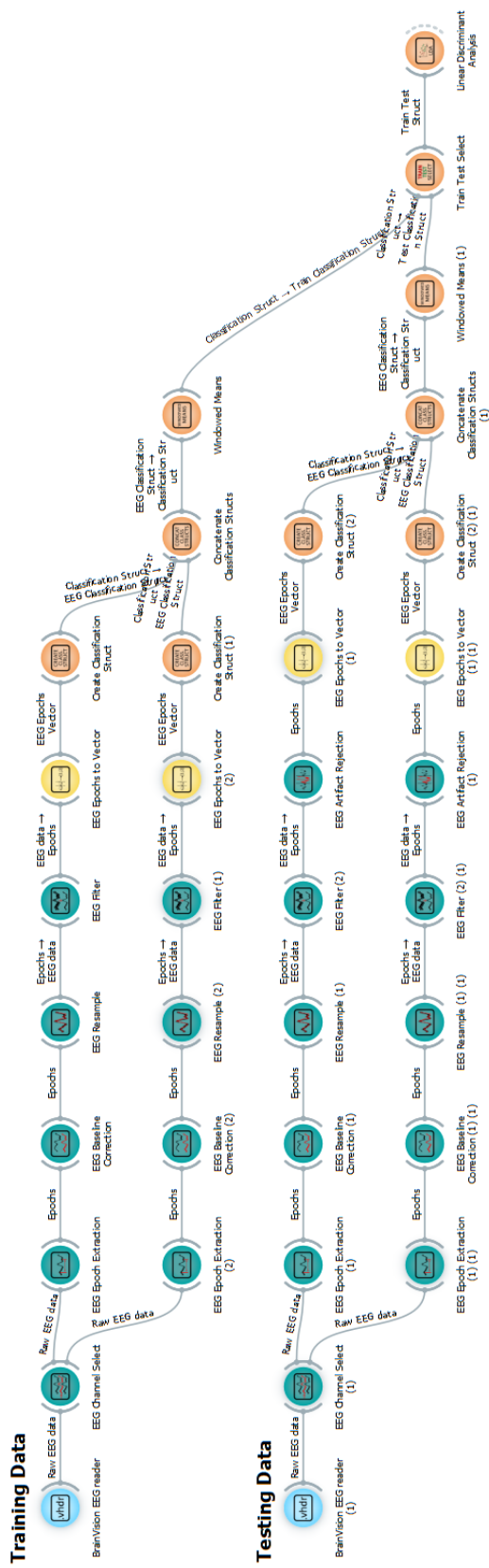


Figure C.2: Workflow used to replicate the Event-related potential datasets based on a three-stimulus paradigm experiment

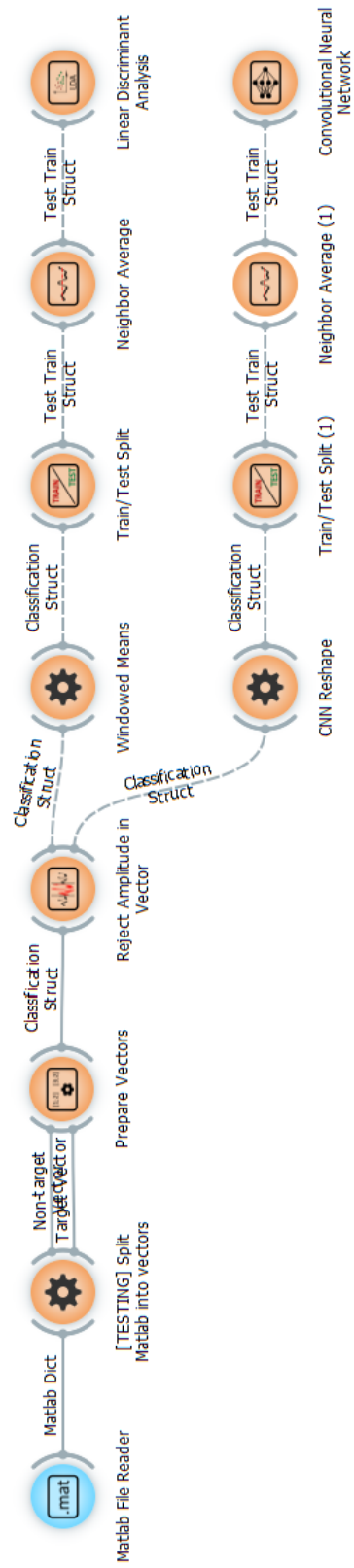


Figure C.3: Workflow used to replicate the Convolutional neural networks experiment

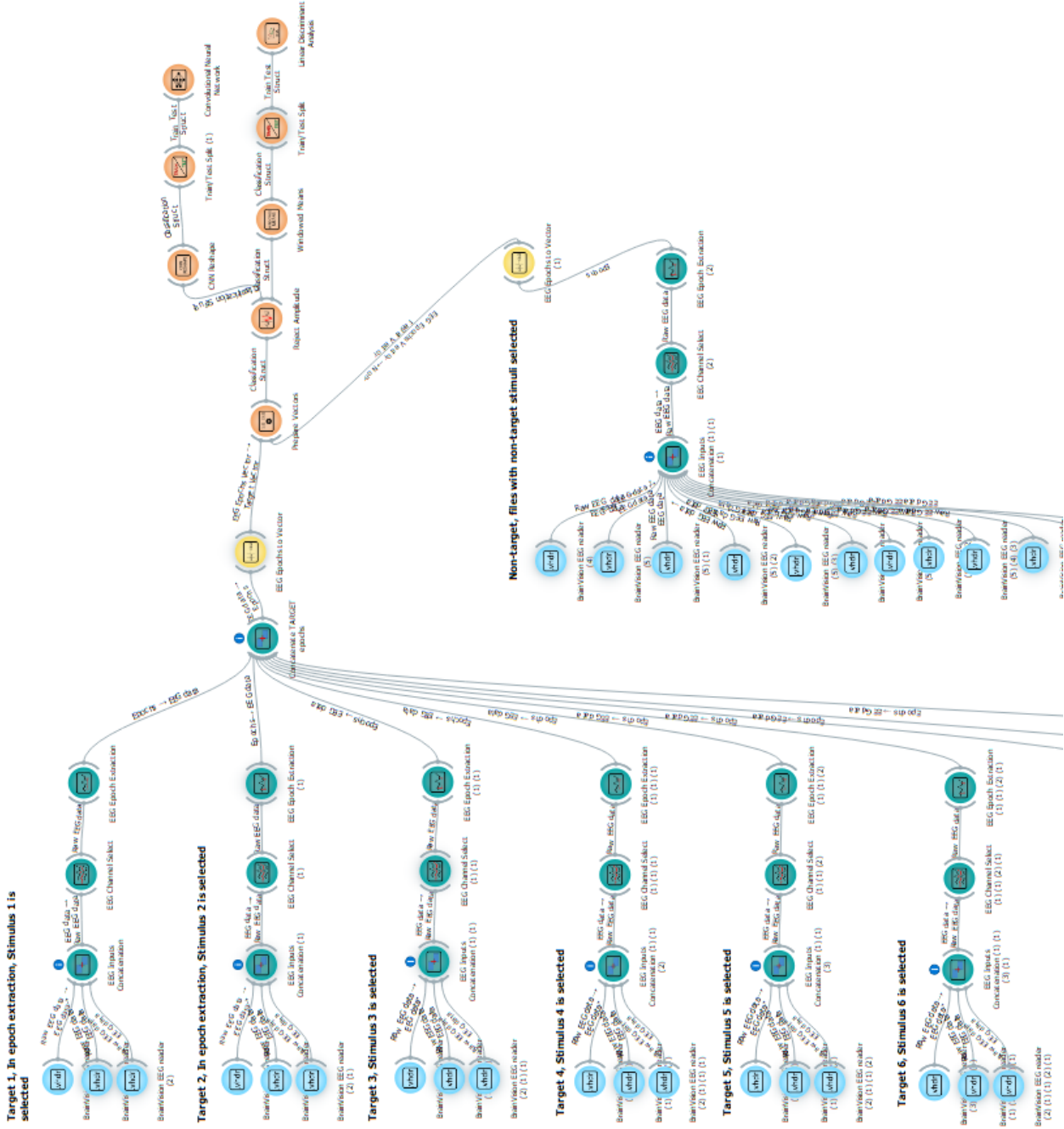


Figure C.4: Workflow used to replicate the Convolutional neural networks experiment with input dataset creation

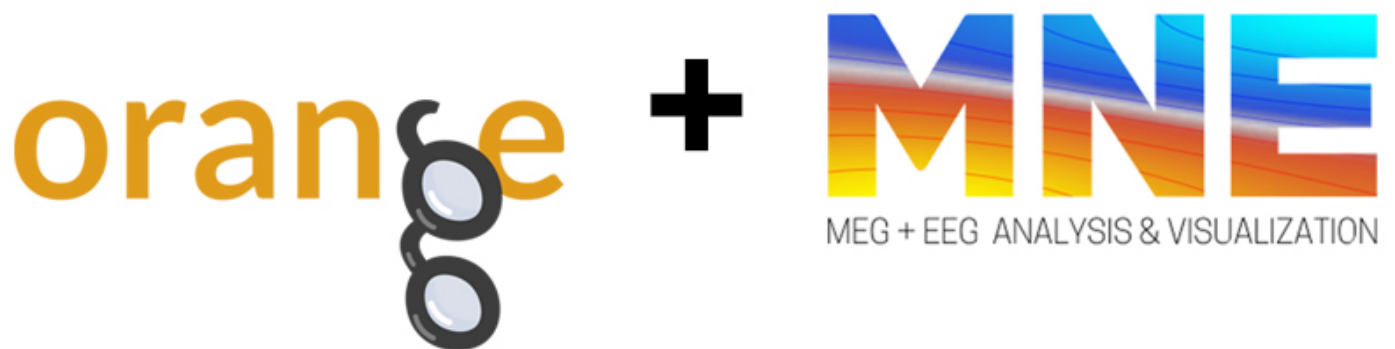
# D User manual

The User manual is on the following pages. the manual was generated using the MkPdfs for MkDocs<sup>1</sup> and uses their sample template.

---

<sup>1</sup>MkPdfs for MkDocs – <https://comwes.github.io/mkpdfs-mkdocs-plugin/index.html>

# MNE Widgets for Orange 3 (Orange3-MNE)





# Table of Contents

---

## Home

---

● User Documentation	4
● Installation	4
● Pip Method	4
● GUI Method	4

---

## Widgets

---

● Widgets	5
● Data IO	5
● BrainVision EEG Reader	5
● EEGLAB Reader	5
● Matlab File Reader	6
● Fif File Save	7
● Fif Reader	7
● Preprocessing	8
● Channel Select	8
● Epoch Extraction	9
● Filter	10
● Baseline Correction	11
● Artifact Rejection	12
● Averaging	12
● Concatenation	13
● Re-sample	14

● Grand Average	14
● Feature Extraction	15
● Peak Latency	15
● Epochs to Vector	15
● Classification	16
● Prepare Vectors	16
● Create Classification Struct	16
● Concatenate Classification Structs	17
● Windowed Means	17
● Reject Amplitude	18
● CNN Reshape	19
● Train Test Split	19
● Train Test Select	20
● Neighbor Average	20
● Linear Discriminant Analysis	21
● Convolutional Neural Network	22
● Visualization	22
● EEG Plot	23
● Compare Evoked Plot	23



# User Documentation

Orange3-MNE is a python package, that provides methods from MNE for Python for Orange 3 in a form of widgets, to allow for electrophysiological data processing.

*Note: This library was created as a part of the master's thesis to show that it is possible to use Orange 3 as a workflow management system for electrophysiological data processing. The widgets' functionality was verified on three existing experiments. Nevertheless, the library requires further development.*

## Installation

The installation process is quite straightforward, first we need to install the Orange 3 tool:

*Note: If you have Orange 3 already installed, you can skip this step.*

```
virtualenv orange          # Create a virtual environment
./orange/Scripts/activate # Activate the environment
pip install Orange3 PyQt5 # Install Orange 3 and PyQt library
```

Then it is possible to install the library using one of the following methods.

### Pip Method

```
pip install Orange3-MNE
```

### GUI Method

1. Run Orange: `python -m Orange.canvas`
2. In Orange navigate to Options -> Add-ons
3. Click on `Add more...` and enter the package name: `Orange3-MNE`
4. Confirm the settings and Orange will install the library
5. Restart Orange and the electrophysiological data processing library will be available

# Widgets

In this chapter, available widgets will be briefly described.

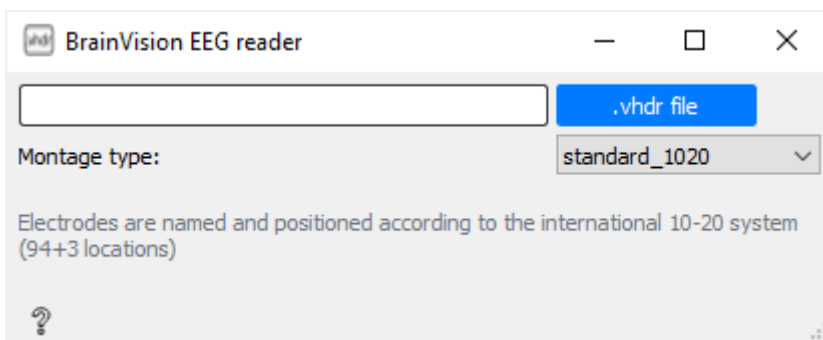
## Data IO

Widgets from this category are utilized to load input files, or save the results.

---

### BrainVision EEG Reader

This widget is used to load `.vhdr` files recorded from the BrainVision.



The widget window contains a clickable button, which opens the file selection dialog and a combo box, which allows to set the montage type to visualize various plots. After the file is selected, or the montage is changed, the loaded data are automatically sent to the output port of the widget.

**Input** - BrainVision's `.vhdr` file

**Output** - MNE-Python's object with type of `mne.io.Raw`

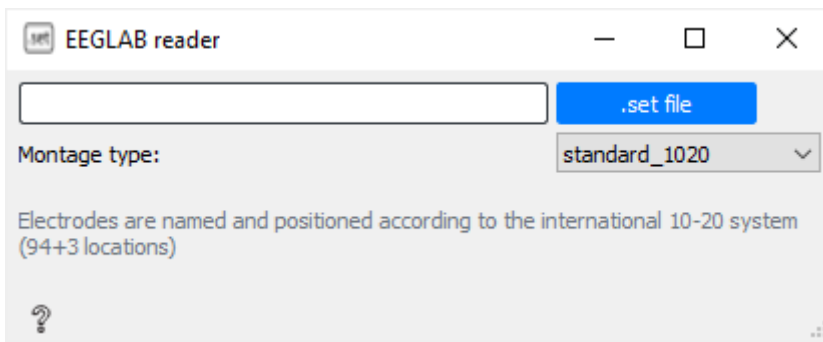
---

### EEGLAB Reader



EEGLAB set files  
reader

This widget is used to load `.set` files from EEGLAB.



The widget window contains a clickable button, which opens the file selection dialog and a combo box, which allows to set the montage type to visualize various plots. After the file is selected, or the montage is changed, the loaded data are automatically sent to the output port of the widget.

**Input** - EEGLAB's `.set file`

**Output** - MNE-Python's object with type of `mne.io.Raw`

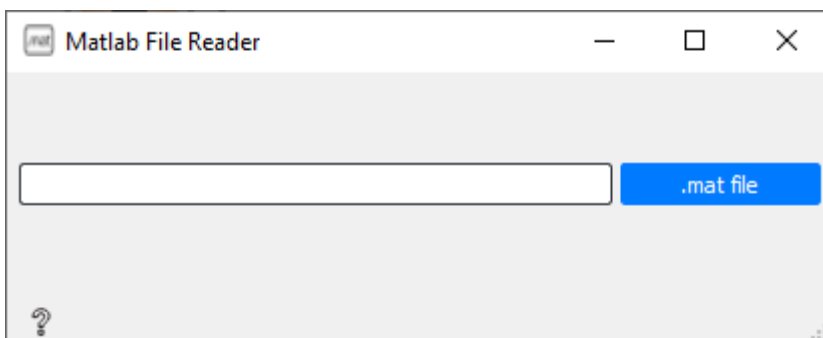
---

## Matlab File Reader



### Matlab File Reader

This widget is utilized to load data files in a Matlab format (`.mat`).



The widget window contains a clickable button, which opens the file selection dialog. After the file is selected the loaded data are automatically sent to the output port of the widget.

**Input** - Matlab's `.mat` file

**Output** - `dict`

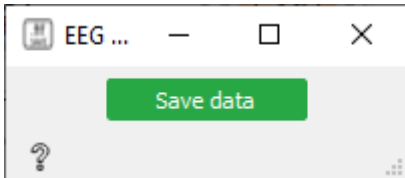
---

## Fif File Save



EEG File Save

It is possible to save three different types of processed data – raw, epochs or evoked.



If the processing takes a long time, or to load the data in another workflow, it may be useful to save the processed data into a file first and load them later. The Fif File Save widget has this functionality.

Each type of processed data has its suffix after the file is saved. The suffix is important because when loading the data using the MNE library, each type requires a different method:

Raw: `-raw.fif`

Epochs: `-epo.fif`

Evoked: `-ave.fif`

**Input** - MNE-Python's object with type of `mne.io.Raw`, `mne.Epochs`, or `mne.Evoked`

**Output** - Fif File

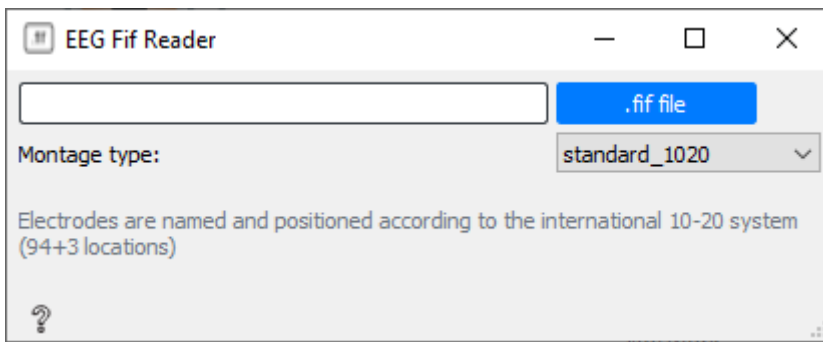
---

## Fif Reader



EEG Ff Reader

The Fif Reader widget is used to load data files that were saved by the [Fif File Save widget](#).



Input - FIF File

Output - MNE-Python's object with type of `mne.io.Raw`, `mne.Epochs`, or `mne.Evoked`

---

## Preprocessing

Widgets in this category are utilized to prepare the raw data for processing. Such widgets include epoch extraction, channel selection, filtering, and more.

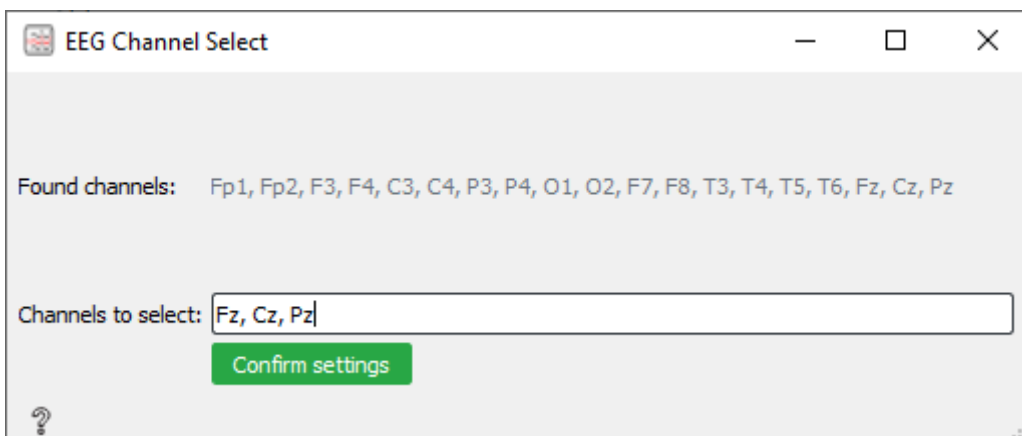
---

### Channel Select



#### EEG Channel Select

The raw recorded electrophysiological data may contain several channels, but for their processing, researchers may need only a few of them. For this purpose, a Channel Select widget is present in the library. The Channel Select allows researchers to select specific channels from the raw data.





If data were sent to the widget, the widget will display all found channels. The user then can select desired channels only by writing their names comma separated in a line edit.

**Input** - MNE-Python's object with type of `mne.io.Raw`

**Output** - MNE-Python's object with type of `mne.io.Raw`

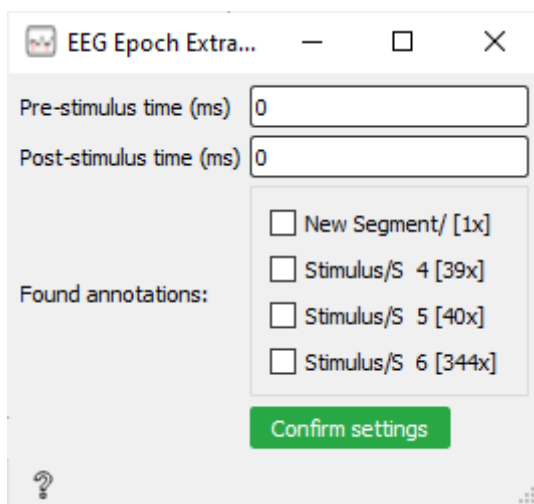
---

## Epoch Extraction



### EEG Epoch Extraction

The electrophysiological signal recordings can be very long. However, in some cases, the researchers are interested only in part of the data, where particular stimuli have occurred. For such cases, epoch extraction methods are available.



Parameter  $t_{Min}$  corresponds to Pre-stimulus time to extract. Parameter  $t_{Max}$  corresponds to Post-stimulus time to extract.

If data were already loaded and annotations were found a block with checkboxes will be displayed. In this block, users can select particular stimuli for the extraction.

**Input** - MNE-Python's object with type of `mne.io.Raw`

**Output** - MNE-Python's object with type of `mne.Epochs`

---

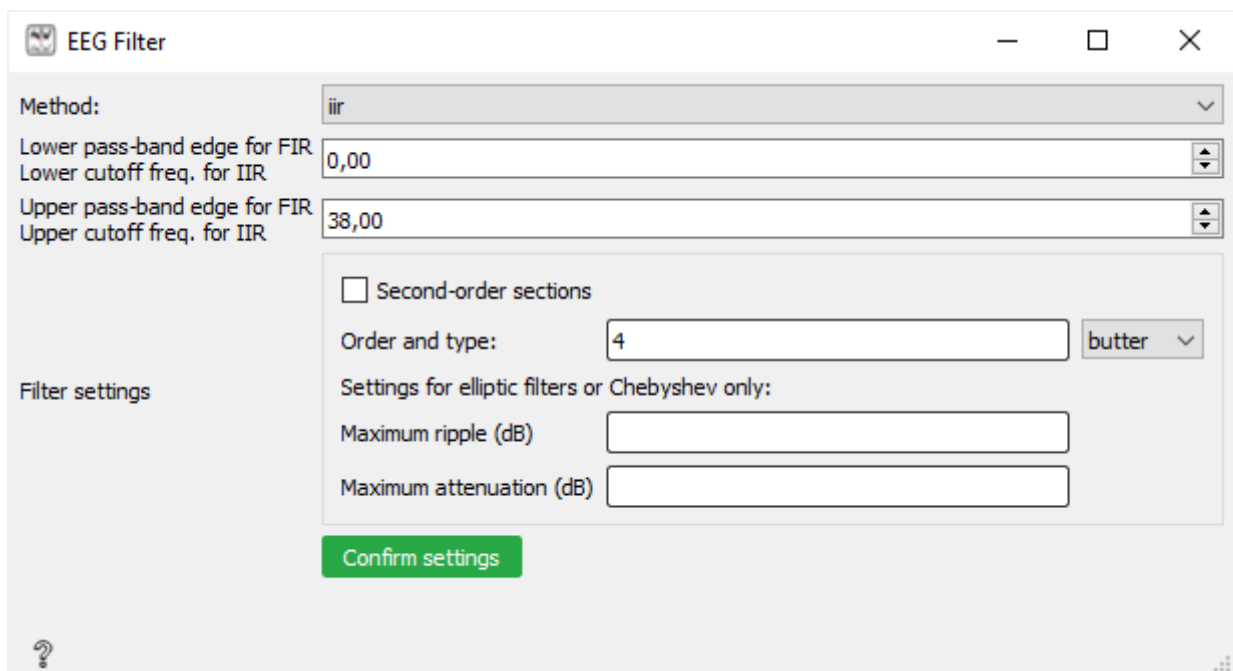
## Filter



EEG Filter

Filtering is an essential step of preprocessing, as recorded signals can contain a lot of signal noise. For example, electrophysiological recordings can be noisy because of power line interference.

The filter widget supports two methods of filtering – IIR or FIR. The graphical user interface is dynamically changing based on the selected method.



EEG Filter

Method: iir

Lower pass-band edge for FIR  
Lower cutoff freq. for IIR: 0,00

Upper pass-band edge for FIR  
Upper cutoff freq. for IIR: 38,00

Second-order sections

Order and type: 4 butter

Settings for elliptic filters or Chebyshev only:

Maximum ripple (dB)

Maximum attenuation (dB)

Confirm settings

The essential parameters of the IIR method are lower and upper cutoff frequency. Additionally, there are several advanced parameters, such as type and order of the filter and their specifying parameters. The types of the filter can be Chebyshev, Butterworth, Elliptic, or Bessel/Thompson.

Similarly, like the IIR method, the FIR method offers the setting of essential parameters – lower and upper pass-band edge. The specific parameter settings for the FIR method are filter length, FIR window type, and phase.

**Input** - MNE-Python's object with type of `mne.io.Raw`, `mne.Epochs`, or `mne.Evoked`

**Output** - MNE-Python's object with type of `mne.io.Raw`, `mne.Epochs`, or `mne.Evoked`

## Baseline Correction



EEG Baseline Correction

Baseline correction is a significant part of preprocessing. An uneven amplitude shifts may occur in the recorded signal. For further processing and analysis, it is necessary to compensate for such amplitude shifts.

The widget has two text inputs that are utilized to define an interval from the whole epoch length, which will be used for the baseline correction.

**Input** - MNE-Python's object with type of `mne.Epochs`

Output - MNE-Python's object with type of `mne.Epochs`

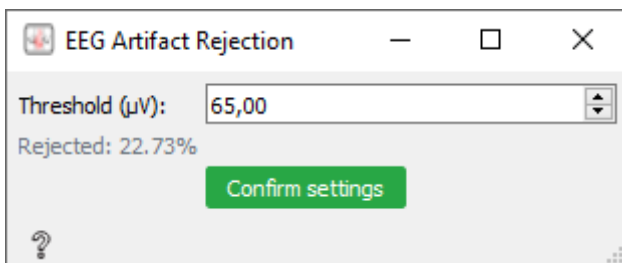
---

## Artifact Rejection



### EEG Artifact Rejection

Individual epochs extracted from the recorded signal may contain artifacts, which may bias the results. Such artifacts may be caused by, for example, eye movement (eyewink), or head movement. The simplest method to detect such artifacts is by the amplitude of the signal.



After the threshold is set, the widget iterates over epochs and finds an amplitude peak for each. If the absolute value of the peak is greater than the threshold, the epoch is rejected.

Input - MNE-Python's object with type of `mne.Epochs`

Output - MNE-Python's object with type of `mne.Epochs`

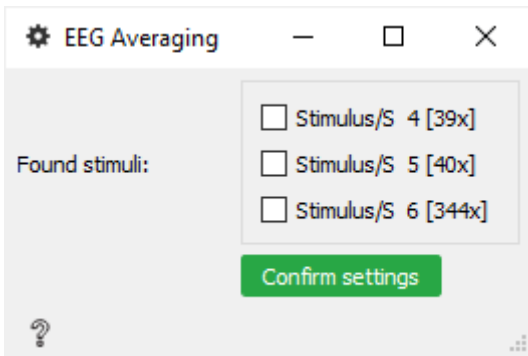
---

## Averaging



### EEG Averaging

Electrophysiological data may contain artifacts even after filtering, to get a better picture of the final signal's final form, it is useful to take advantage of the signal averaging methods. For this purpose, there is an EEG Averaging widget available in the created library.



When the widget receives the data, it shows a group of checkboxes, where each checkbox represents a stimulus category. Apart from the name of the stimulus, a count of stimuli is displayed. Users can select specific stimulus over which will be the signal from epochs averaged. Although it is usually useful to average signals that belong to one particular stimulus, users have the possibility to average signals over more than one stimulus.

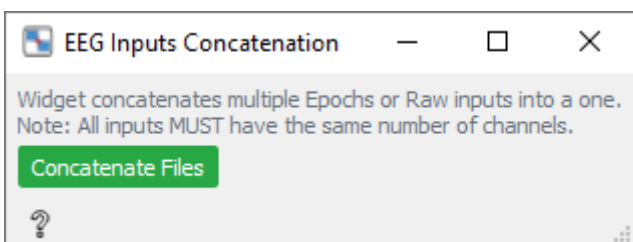
**Input** - MNE-Python's object with type of `mne.Epochs`

**Output** - MNE-Python's object with type of `mne.Evoked`

## Concatenation



EEG Inputs  
Concatenation



Orange allows for multiple data inputs through one signal line into a widget; however, it does not allow to output several data utilizing one signal line out of a widget. This functionality may be a limitation for researchers that have multiple input files. The Concatenation widget is included in the library to get around this limitation.

To concatenate multiple data inputs correctly all inputs must have the **same number of channels**.

**Input** - MNE-Python's object with type of `mne.io.Raw`, `mne.Epochs`

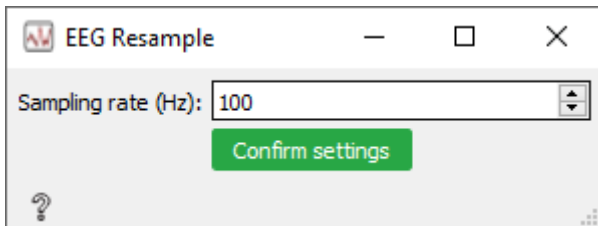
**Output** - MNE-Python's object with type of `mne.io.Raw`, `mne.Epochs`

## Re-sample



EEG Resample

Resample widget can be used to resample extracted epochs to required sampling frequency.



The Resample widget may resample the sampling frequency of the extracted epochs to a range of 1-1000 Hz. This may be useful for workflows where high resolution of the signal is not necessary and would slow down the processing of the data.

**Input** - MNE-Python's object with type of `mne.Epochs`

**Output** - MNE-Python's object with type of `mne.Epochs`

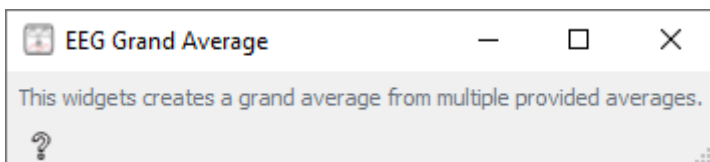
---

## Grand Average



EEG Grand Average

The grand average is an average calculated from multiple averages.



This can be beneficial, for example, when examining the data from several testing subjects, as this approach can mitigate some abnormalities or artifacts.

**Input** - Multiple MNE-Python's objects with type of `mne.Evoked`

**Output** - MNE-Python's object with type of `mne.Evoked`

---

## Feature Extraction

---

### Peak Latency



EEG Peak Latency

The screenshot shows a window titled "EEG Peak Latency" with standard window controls (minimize, maximize, close). The interface includes a title bar, a question mark icon, and a text input field for "Peak amplitude and latency:". Below this is a label "The tMin and tMax must be within ? and ?" followed by two text input fields for "tMin (ms)". A "Mode:" dropdown menu is set to "Positive". A green "Confirm settings" button is located at the bottom right. A help icon (?) is in the bottom left corner.

There are two text inputs for the time interval and a combo box for the mode selection. The time interval is useful if we want to find a peak in a specific range of an epoch. There are three modes of peak finding available – *Positive*, *Negative*, and *Absolute*. The *Positive* mode will find the peak in the positive values, the *Negative* mode is the exact opposite, and it will find the peak in the negative values. The *Absolute* mode will find the most significant peak across positive and negative values. Apart from the peak latency, the widget also shows the peak amplitude.

**Input** - MNE-Python's object with type of `mne.Evoked`

**Output** - None

---

### Epochs to Vector



EEG Epochs to Vector

The screenshot shows a window titled "EEG ..." with standard window controls (minimize, maximize, close). The interface displays a success message: "Successfully converted 34 epochs." A help icon (?) is in the bottom left corner.

Epochs to Vector is a widget that converts `mne.Epochs` into a vector which may be utilized in classification workflows. The graphical user interface of the widget is quite minimalistic as it does not have any interactive inputs. The widget contains one label with information about how many epochs were converted to a vector in total.

**Input** - MNE-Python's object with type of `mne.Epochs`

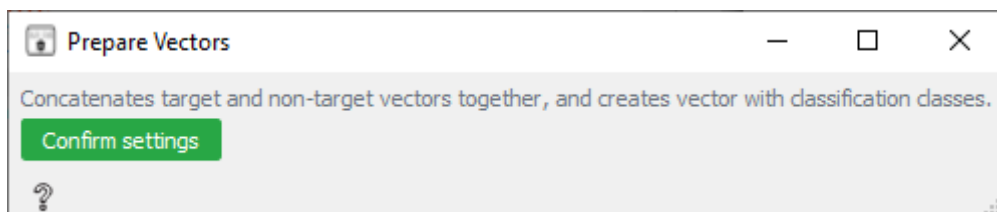
**Output** - dict with the following format: - { data: <vector\_with\_the\_data>, sfreq: <sampling\_frequency>, stimuli: <selected\_stimuli>, channels: <channel\_names> }

## Classification

### Prepare Vectors



Prepare Vectors



Widget expects two input vectors – target and non-target and creates the Classification struct from them, which is required for the classification process

**Input** - dict – Target vector - dict – Non-target vector

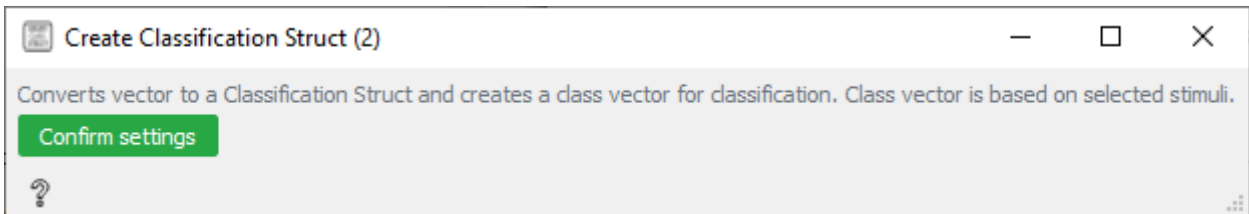
**Output** - `EegClassification.structs.ClassificationStruct`

### Create Classification Struct



Create Classification Struct





The Create Classification Struct widget, similarly, like the Prepare Vectors widget creates a Classification struct from the input vector; however, the classification classes of the vector are based on the stimulus selected in the epoch extraction step.

**Input** - dict - Input vector

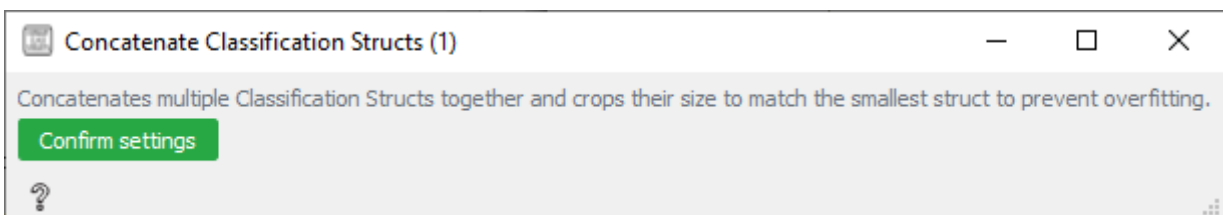
**Output** - `EegClassification.structs.ClassificationStruct`

---

## Concatenate Classification Structs



Concatenate  
Classification Structs



The Concatenate Classification Structs widget, as the name suggests, is useful to concatenate multiple Classification structs together. On top of that, it crops all vectors size to match vector with the minimum size to prevent overfitting, for example, when one stimulus has 400 events, and the other has only 20, the classifier would be overfitted on the first stimulus.

**Input** - multiple dict - Input vectors

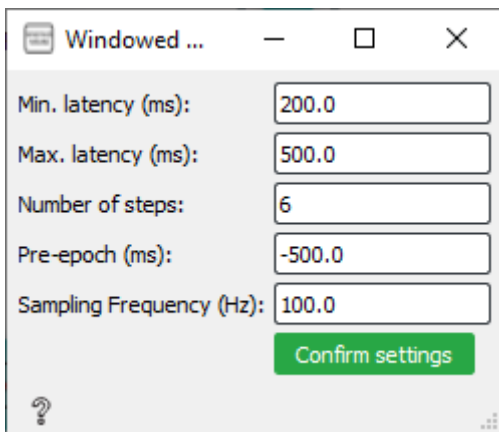
**Output** - `EegClassification.structs.ClassificationStruct`

---

## Windowed Means



Windowed Means



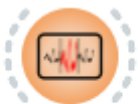
The Windowed Means widget extracts features in the following way. It splits the extracted epochs into intervals based on the minimal latency, maximum latency, and the number of steps. It then calculates the average value for each EEG channel for each range.

Parameters: - `Min. latency`: Lower border for the intervals (after the stimulus) - `Max. latency`: Upper border for the intervals (after the stimulus) - `Number of steps`: Number of intervals on selected range (lower, upper) - `Pre-epoch`: Sets how much time was extracted pre-stimulus in epochs - `Sampling frequency`: Sets the sampling frequency of the signal

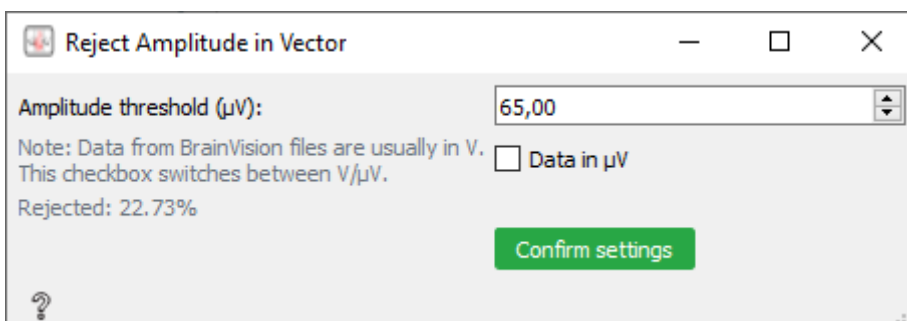
**Input** - `EegClassification.structs.ClassificationStruct`

**Output** - `EegClassification.structs.ClassificationStruct`

## Reject Amplitude



Reject Amplitude in  
Vector



Identically as the [Artifact Rejection](#), the Reject Amplitude widget rejects epochs whose amplitude exceeds the threshold limit. The difference is that this widget works with vectors opposed to the Artifact Rejection, which works with MNE's library objects.

Additionally, this widget has a switch between units (V and  $\mu\text{V}$ )

**Input** - `EegClassification.structs.ClassificationStruct`

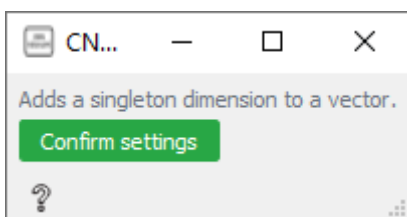
**Output** - `EegClassification.structs.ClassificationStruct`

---

## CNN Reshape



CNN Reshape



This widget adds a singleton dimension to enable CNN classification using the Convolutional Neural Network widget.

Note: This widget is specific to one experiment ([Evaluation of the convolutional neural networks experiment \(https://www.sciencedirect.com/science/article/abs/pii/S1746809419304185\)\)](https://www.sciencedirect.com/science/article/abs/pii/S1746809419304185))

**Input** - `EegClassification.structs.ClassificationStruct`

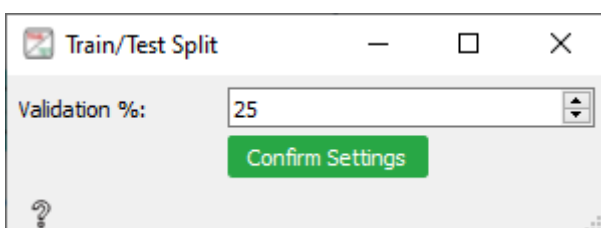
**Output** - `EegClassification.structs.ClassificationStruct`

---

## Train Test Split



Train/Test Split



It is necessary to provide training and testing dataset to use classification methods that utilize supervised learning. To get such datasets it is possible to split one extensive dataset into two parts – training and testing. The Train Test Split widget offers this functionality.

The Train Test Split widget contains one input for numbers and one button.

Using the input for numbers, researchers can easily set the percentage that represents the proportion of the dataset to include in the test split.

**Input** - `EegClassification.structs.ClassificationStruct`

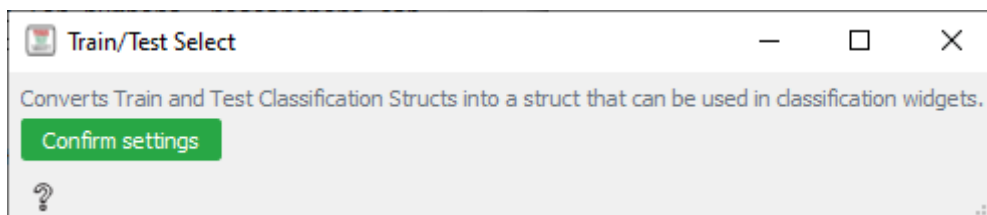
**Output** - `EegClassification.structs.TestTrainStruct`

---

## Train Test Select



Train/Test Select



Similarly, like the [Train Test Split](#), this widget is utilized to prepare the data for the classification. The only difference is that this widget requires two input datasets instead of one – training and testing, widget then converts datasets into a structure that necessary in classification widgets.

**Input** - `EegClassification.structs.ClassificationStruct`

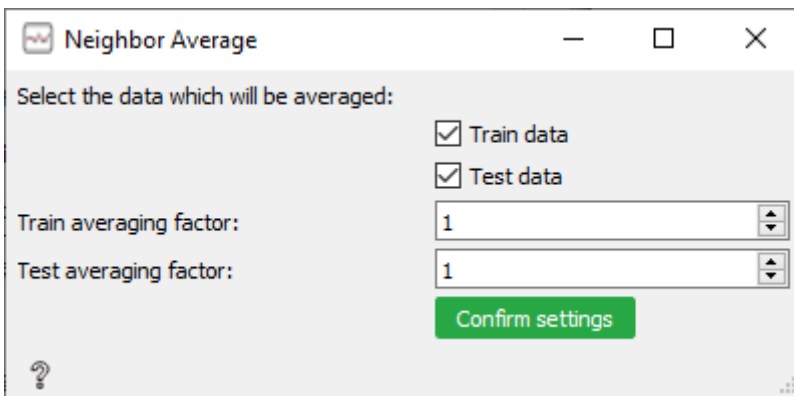
**Output** - `EegClassification.structs.TestTrainStruct`

---

## Neighbor Average



Neighbor Average



The Neighbor Average widget is similar to the [Windowed Means](#) widget, except instead of intervals, it averages N number of trials in epochs together.

The widget contains two checkboxes to select which datasets will be averaged.

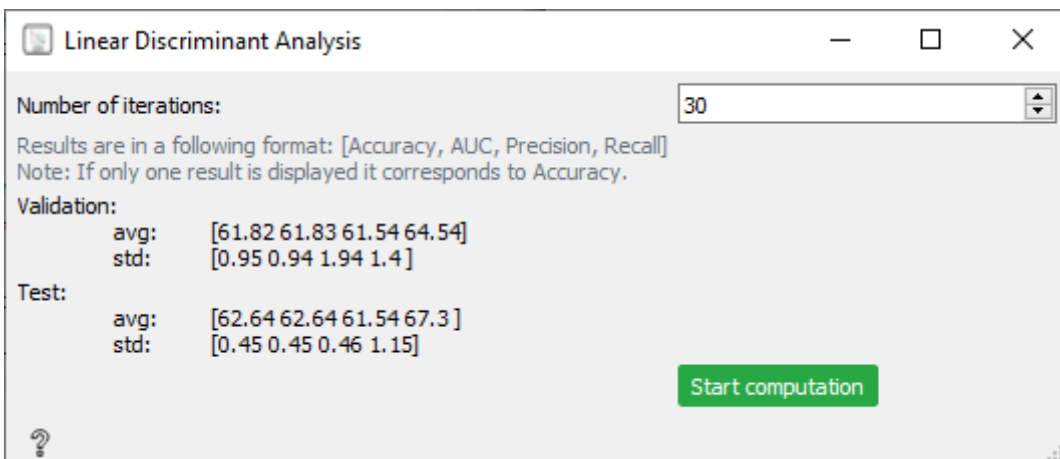
**Input** - `EegClassification.structs.TestTrainStruct`

**Output** - `EegClassification.structs.TestTrainStruct`

## Linear Discriminant Analysis



Linear Discriminant  
Analysis



The LDA classification from the scikit library is available through the Linear Discriminant Analysis widget.

The results include four metrics for both validation and test data – Accuracy, AUC (Area under the ROC Curve), Precision, and Recall. Additionally, the results include a standard deviation as well.

Note: If only one result is displayed it corresponds to **Accuracy**

**Input** - `EegClassification.structs.TestTrainStruct`

**Output** - None

---

## Convolutional Neural Network



### Convolutional Neural Network

A simple widget that uses Keras and Tensorflow libraries

Validation:	
avg:	[76.47 83.71 77.82 74.29]
std:	[6.55 7.57 7.27 7.56]
Test:	
avg:	[74.36 59.35 62.86 60.14 55.18]
std:	[6. 2.12 2.78 2.11 4.28]

The implemented model of CNN is specific for the experiment described in original [article \(https://www.sciencedirect.com/science/article/abs/pii/S1746809419304185\)](https://www.sciencedirect.com/science/article/abs/pii/S1746809419304185). It would be possible to let the researchers specify the model details; however, the graphical user interface would be highly complex as the models have numerous different settings. For this reason, only the model from the original experiment was implemented as a proof of concept.

**Input** - `EegClassification.structs.TestTrainStruct`

**Output** - None

---

## Visualization

Widgets in this category are usefull for the visualization of the raw or processed data.

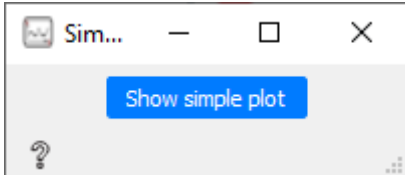
---

## EEG Plot



Simple Plot

EEG plot widget utilizes the `plot()` function from the MNE library.



The widget contains a clickable button, which opens the generated plot for the data.

**Input** - MNE-Python's object types: `mne.io.Raw`, `mne.Epochs`, or `mne.Evoked`

**Output** - None

---

## Compare Evoked Plot



EEG Compare Evoked  
Plot

It can be useful to visualize multiple averages on a single plot to compare respective signals against each other. The Compared Evoked Plot widget ensures this functionality.

The graphical user interface is the same as for the [EEG Plot](#) widget.