DAIsy – Developing AI ecosystems improving diagnosis and care of mental diseases

ITEA 4 – 21016

Work package 5 (WP5)

Platform Development

Deliverable 5.3.
Development of AI models

Document type	: Deliverable
Document version	: No. 1
Document Preparation Date	: Feb 2024
Classification	: Confidential
Due Date	: Dec 2025

25 years ITEA 4 Community 1998–2023

HISTORY

Document version #	Date	Remarks
1	Feb 2024	Initial version
2	Feb 2026	Final version

Table of Contents

1 Introduction

The deliverable outlines the development of various AI models as specified in Work Package 4 (WP4), which are integral to the project's success. It includes a decision support component to assist users in making informed decisions tailored to each use case. Furthermore, it details the necessary steps for integrating the AI system within the DIAsy framework, ensuring a seamless and functional structure for users and developers alike. This integration is crucial for leveraging the full potential of AI within the DAIsy ecosystem.

The various security mechanisms of the individual components described are discussed in detail in Deliverable 5.4, where the protective measures and security architecture are comprehensively documented.

2 Description of the method of integration into an existing system

2.1 Deployment as a microservice in a Docker container on a Kubernetes cluster

To deploy a service within a Docker container to a Kubernetes cluster, an image repository is essential. This repository will store the Docker images that Kubernetes will pull and run on its pods. Once the image repository is set up and linked to the Kubernetes cluster, the Kubernetes manifests is used to define the deployments, including the image to be used. The service is equipped with a Swagger interface. Swagger, also known as OpenAPI, provides a developer-friendly framework for describing the API of the service. This facilitates easier integration and control by providing a clear contract for the API and interactive documentation.

3 Real-time machine learning component for multimodal neurofeedback toolbox MultiPy

3.1 Objective

The main goal of the real-time machine learning component is to enable adaptive classification and feedback generation in the DAIsy use case Multimodal Neurofeedback. It is designed to identify meaningful patterns in multimodal neurophysiological data, specifically, EEG, fNIRS, or combined EEG-fNIRS signals, so that neural states or cognitive conditions can be inferred in real time. The machine learning algorithms implemented in this module, such as Support Vector Machines (SVM) and Linear Discriminant Analysis (LDA), are selected for their efficiency and interpretability, making them suitable for brain-computer interface (BCI) and neurofeedback applications. By processing incoming features extracted from the EEG and fNIRS preprocessing pipeline, this module translates physiological fluctuations into actionable outputs that can drive adaptive interfaces, adaptations of experimental tasks, or feedback visualization. The overall goal is to support closed experimental paradigms where the system continuously learns from the user's responses and dynamically adapts to promote engagement, learning, or therapeutic outcomes. The model design explicitly prioritizes interpretability and temporal stability to ensure that real-time predictions can be meaningfully assessed by experts during experimental or therapeutic sessions, rather than functioning as opaque decision mechanisms.

Furthermore, a classic machine learning analysis (correlation analysis) of the established depression parameters was performed. The data for this was first generated and then a corresponding analysis and visualisation was carried out.

3.2 Testing and Validation

Testing and validating the machine learning module ensures that the algorithms function reliably under real-time conditions and deliver meaningful results consistent with offline benchmarks. The component allows users to load pre-recorded EEG, fNIRS, or combined datasets for training, enabling systematic evaluation before deployment in live experiments. The final training process will include data normalization, feature selection, and cross-validation. Performance is assessed using accuracy, confusion matrices, and receiver operating characteristic (ROC) curves. These tools help determine how well the model distinguishes between mental states or task conditions and can uncover potential class imbalances or overfitting.

In real-time operation, validation will continue through online accuracy tracking and latency monitoring. During pilot sessions, the classifier output is compared to known markers or behavioral responses to confirm temporal agreement and interpretability. The machine learning module also visualizes classification boundaries, decision confidence, and feature relevance to facilitate expert evaluation. Validation is an ongoing process, as model performance can vary across participants and sessions. Therefore, retraining and recalibration mechanisms are integrated to ensure robustness throughout the experimental lifecycle.

Special attention is given to inter-session variability, as physiological patterns may differ across recordings. Validation therefore focuses not only on average performance, but also on consistency and robustness across sessions and subjects.

3.3 Performance Metrics

Performance expectations for the real-time machine learning component will focus on accuracy, latency, throughput, and computational efficiency. In typical deployments, the classifier should deliver up-to-date predictions with a response time of less than 200 ms to ensure a smooth feedback loop suitable for neurofeedback and BCI paradigms. Throughput depends on the sampling rate of the input data but is designed to support parallel multi-channel EEG and fNIRS streams without frame loss or buffer delays. Accuracy targets vary depending on the task. Resource utilization is optimized through lightweight algorithms and just-in-time compilation of feature extraction routines using Numba to ensure stable operation on standard laboratory computers. Memory and CPU utilization are monitored to prevent real-time performance degradation. Continuous performance logging allows users to identify processing or communication bottlenecks and supports iterative optimization of model parameters and system configuration.

Latency constraints are treated as a core requirement, as delayed feedback can disrupt closed-loop paradigms and negatively affect user engagement and learning effects.

3.4 Interfaces

The component communicates with the rest of the DAIsy system primarily through LSL interfaces. It receives preprocessed features from the signal processing pipeline as LSL inputs and outputs classification results or confidence values via dedicated LSL outputs. These outputs can be used by feedback visualization modules, task controllers, or other DAIsy components that require real-time status information.

The graphical user interface, implemented with PySide6, includes modules for model configuration, training data selection, and visualization of performance metrics such as accuracy curves, confusion matrices, and ROC plots. Users can switch between offline training and real-time streaming. Further interaction options include the selection of features or channels for classification, the choice of machine learning model type, and the optimization of hyperparameters such as regularization strength or kernel type. The design supports flexibility for both experimental investigations and reproducible research and allows adaptation to a wide variety of BCI and neurofeedback protocols.

3.5 Dependencies

The component is based on several Python libraries that enable real-time data processing, numerical computations, and machine learning. Core dependencies include Numpy, Scipy, and Pandas for numerical and statistical operations; Scikit-Learn for implementing SVM, LDA, and related models; and Numba for accelerating feature computation. Integration with MNE, MNE-LSL, and PyXDF enables multimodal EEG and fNIRS data processing, while PYLSL manages streaming data and model outputs. Visualization components are built with PySide6, PyQTgraph, and Matplotlib to ensure responsive and interactive renderings. These dependencies are consistent with the broader DAIsy environment, enabling seamless integration and reducing compatibility issues. The full list of dependencies is included in the project's requirements file, which contains fixed versions of all required libraries to ensure cross-system reproducibility. As development progresses, optimization and modularization efforts may lead to the separation of real-time and offline packages to reduce the runtime requirements of the deployed instances.

3.6 Architecture

The internal structure of the real-time machine learning component consists of several key layers that work together to enable continuous learning and feedback generation. The input layer handles the acquisition of feature vectors from the preprocessing pipeline via LSL and ensures temporal synchronization and buffering. The processing layer manages data normalization, feature selection, and model inference. Depending on the configuration, it can operate in single-modality mode (EEG or fNIRS) or multimodal fusion mode. The learning layer contains implementations of SVM and LDA classifiers with interfaces for training, validation, and incremental updates. This layer also manages model persistence, allowing trained models to be saved, reloaded, or optimized between sessions. The visualization layer enables real-time monitoring of classifier output, accuracy tracking, and diagnostic plots. Finally, the communication layer outputs predictions via LSL outputs and handles interaction with other DAIsy modules such as the feedback controller or the data logger.This modular architecture allows the component to be extended with new algorithms or additional preprocessing stages without changing the existing pipeline. Ongoing development aims to improve scalability and reduce processing latency by further optimizing the feature extraction and inference stages.

4 Interpretable AI models to detect type of Eating disorder (ED) and predict treatment response in hospitalized Anorexia nervosa (AN) patients

4.1 Objective

To find the (i) predictors and prototypical representatives of different types of ED; (ii) clinical predictors of anomalies in treatment response in hospitalized AN patients.

The light-weight (data-frugal) and interpretable ML models applied by TU Eindhoven (TU/e) on the Routine outcome monitoring (ROM) data from GGz, provided the clinicians with the prototypical representatives of different types of ED, different inherent sub-demographics of ED patients who had had an intake at GGz OB. For every ED patient the applied models present their confidence in assigning an ED type to them. Such information will enabled clinicians to extract further knowledge about each type and subtype of ED, especially in differentiating among the diagnostically seemingly overlapping ED types, and prepare care plans more judiciously. Next, TU/e also applied a suite of data-frugal anomaly detection models on the temporally higher definition, clinical variables of hospitalized Anorexia patients to detect anomalies in their recovery patterns, including conditions known as re-feeding syndrome. and plan more cautiously. This helped them find possible early signs of non-response, and of refeeding syndrome. For clinicians this information was helpful for their resource management. Model interpretability is a central requirement, ensuring that predictions and anomaly

detections can be inspected and contextualized by clinicians rather than acting as automated diagnostic outputs.

4.2 Testing and Validation

To ensure that the developed model meets its intended purpose the trained models submitted to GGz OB by TU/e will be validated on a larger, ED dataset in the NL. Even though this is beyond the scope of this project, and of TU/e's, the tools and detailed pictorial guidebook and video tutorial provided by TU/e to GGz OB will support them in their validation effort when they proceed with it. The validation approach emphasizes transferability and clinical plausibility, supporting future evaluation on extended datasets while maintaining transparency of model behavior.

4.3 Dependencies

External dependencies are managed through Conda virtual environment and include numpy, pandas, matplotlib, scipy, scikit-learn, pip, sklearn-lvq, shap.

5 Insight Generation from Activity and Nutrition Data

5.1 Objective

The objective of the insight generator is to provide personalized, data-driven feedback to patients with eating disorders by analyzing nutrition and activity data. The system identifies patterns, generates comparisons, and highlights insights that are most relevant to the individual at a given moment. Importantly, it adapts to changes in patient preferences and priorities throughout the treatment journey, ensuring that feedback remains meaningful and supportive over time.

This adaptive mechanism supports clinicians and patients by facilitating more timely recognition of shifting needs, fostering patient engagement, and providing complementary guidance for monitoring recovery. Ultimately, the goal is to strengthen the personalization of care for eating disorder patients by offering insights that evolve with the patient's journey, while reducing the cognitive and emotional burden associated with interpreting complex data independently. The system is designed to support clinical monitoring by complementing, rather than replacing, clinician judgment, providing adaptive insights that evolve alongside patient behavior.

5.2 Testing and Validation

To verify that the insight generator fulfils its intended purpose, we designed a series of simulation studies based on real-world nutrition and activity data. The simulations were constructed to reflect realistic patient behaviors and variations in daily routines, including changes in calorie intake, exercise type, duration, and intensity.

The primary goal of validation was to assess whether the system could not only generate relevant comparisons but also adapt its output as user preferences shifted, which is a critical requirement in the context of eating disorder treatment where patient needs evolve over time. By introducing controlled preference changes in the simulations, we tested the model's ability to detect and respond to these shifts, ensuring that the generated insights remain aligned with the patient's current situation.

Performance was evaluated by comparing the model's predictions of "interesting" insights with ground truth labels derived from simulated user feedback. This setup allowed us to systematically measure sensitivity to preference changes and robustness across different simulated patient journeys. The validation confirmed that the insight generator can adapt dynamically, thereby meeting the requirements for providing personalized, evolving feedback

in clinical use. Validation focuses on longitudinal stability, ensuring that adaptation to preference changes does not introduce erratic or contradictory feedback over time.

5.3 Performance Metrics

The insight generator achieves an accuracy of over 90% in predicting which insights users find relevant, and it maintains this performance consistently over time, even as preferences change. In addition to accuracy, consistency of insight relevance over extended periods is considered a key indicator of system reliability in clinical contexts.

5.4 Interfaces

The insight generator accesses nutrition and activity data from the central database and processes this information to produce personalized insights. These insights are transmitted to the patient-facing app, where they are displayed in an accessible format. The app also provides an interface for patients to give feedback on the presented insights, which is collected and returned to the system to refine future predictions. All model outputs are exposed through standardized interfaces that allow downstream components to display predictions together with confidence indicators, supporting cautious interpretation in applied settings.

5.5 Dependencies

Python, TensorFlow

6 Major Depressive Disorder (Distinguishing Bipolar and Unipolar Depressive Disorder)

6.1 Objective

The objective of this work is to develop AI models that support the analysis and differentiation of MDD and BD subtypes by combining information derived from neuroimaging data with available clinical findings.

Model training is based on structured feature representations generated by the upstream processing workflow. These representations include descriptors derived from MRI processing outputs as well as region-based gray matter information obtained from structural MRI. Clinical variables are included as complementary inputs to reflect diagnostic context typically considered in clinical practice.

The main goals of the model development are:

- to learn from imaging-derived feature representations that are produced consistently by the platform,
- to integrate clinical findings alongside imaging features in a controlled manner,
- to limit unnecessary complexity in the feature space through feature selection and dimensionality reduction,
- to provide model outputs that can be interpreted in terms of contributing input features,
- to evaluate different types of machine learning approaches, ranging from classical models to neural network and deep learning based methods.

The models are designed to operate on prepared feature representations provided by the platform and do not access raw imaging data directly.

6.2 Testing and Validation

Testing and validation are carried out to assess how reliably the models perform across different data subsets and feature combinations.

The evaluation procedure includes:

- separating data used for training and evaluation to avoid bias,
- applying cross-validation to assess consistency across different data splits,
- comparing model performance when trained on different feature sets, such as imaging features alone or imaging features combined with clinical variables.

Feature selection methods are applied during model development to identify which inputs contribute most to the model outcome and to reduce redundancy in the feature space. Dimensionality reduction techniques are used where appropriate to simplify the feature representation while preserving relevant variation.

Several model types are evaluated, including classical machine learning approaches operating on structured feature representations, as well as neural network–based methods applied to imaging-derived representations. The outcomes of these evaluations are used to guide model selection for further integration into the platform.

6.3 Performance Metrics

Model performance is evaluated using standard measures commonly applied in classification tasks.

These include:

- overall classification accuracy,
- class-sensitive measures such as the F1-score,
- discrimination measures such as the area under the ROC curve.

In addition to prediction quality, the models provide confidence values that indicate how strongly a prediction is supported by the input data. Explainability outputs are used to identify which features have the greatest influence on the model decision.

All reported performance measures describe the behavior of a trained model under evaluation conditions and are not interpreted as guarantees for individual clinical cases.

6.4 Interfaces

The AI model component is invoked by the orchestration layer after upstream processing has prepared the required representations. The model consumes prepared feature representations and optional clinical findings as input, and produces a classification result together with confidence values and a brief interpretation summary. Model outputs are passed to downstream components for structured reporting and clinical visual review. The model does not access raw imaging data and does not write intermediate outputs to the repository.

For visual interaction, the **Viewer** is implemented using **VTK-based visualisation components**, enabling volumetric brain rendering and diffusion-derived tract overlays suitable for clinical inspection. The **Doctor Dashboard** provides a **React-based user interface** for initiating analysis requests, monitoring completion status, and accessing results through the Viewer.

6.5 Dependencies

Neuroimaging-related preparation steps use **FSL** where applicable. Model development and evaluation are conducted in **Python** using **NumPy**, **Pandas**, **SciPy**, **scikit-learn**, **PyTorch**, and **Matplotlib**. Dependencies are managed in isolated **Conda** environments to support reproducible execution and avoid interference with other DAIsy platform components.