DOI: https://doi.org/10.36910/6775-2524-0560-2026-64-29
УДК 004.05
**Stefanyshyn Ivan**, PhD student
https://orcid.org/0009-0008-6930-528X
**Yatsyshyn Vasyl**, Candidate of Technical Sciences, Associate Professor
https://orcid.org/0000-0002-5517-6359
Ternopil Ivan Puluj National Technical University, Ternopil, Ukraine

# IMPLEMENTATION OF A TECHNOLOGY FOR AUTOMATED CONSTRUCTION AND OPTIMIZATION OF SOFTWARE COMPONENT PIPELINES FOR EEG SIGNAL CLASSIFICATION

**Stefanyshyn I., Yatsyshyn V. Implementation of a Technology for Automated Construction and Optimization of Software Component Pipelines for EEG Signal Classification.** The article proposes the implementation of a technology for automated construction and optimization of software component pipelines for the classification of electroencephalographic (EEG) signals. The relevance of the practical application of such a technology is determined by the fact that, in existing approaches, the construction of software component pipelines is mainly performed empirically or focuses on individual algorithms, whereas the formalized specification of admissible configurations, control of their correctness, computation, comparison, and optimization remain insufficiently implemented at the level of an integrated software system. The aim of the study is to implement a technology that provides automated construction, computation, evaluation, and optimization of software component pipelines for EEG signal classification. In the article, a software component pipeline is represented as a tree-like directed graph structure. To implement the proposed technology, functional and non-functional requirements are formulated, architectural decisions are substantiated, the main modules of the software system are defined, and its infrastructure is described. The software system implements a complete technological cycle: creating an experiment, uploading EEG data, constructing a tree-like directed graph structure, configuring computation and optimization parameters, performing local and cloud computations, launching optimization, and presenting results. Optimization involves multi-criteria evaluation of software component pipelines by accuracy, F1-score, area under the ROC curve, and normalized processing time per sample, taking into account weighting coefficients. The quality of the software implementation is confirmed by automated testing of the desktop application, server, computation script, and optimization script. Experimental verification was performed on real EEG data of motor activity of the fingers of the upper limbs. The obtained results confirmed the practical applicability of the proposed technology and the implemented software system.
**Keywords:** electroencephalographic signals, classification, architecture, technology, software system, infrastructure, algorithm, quality, domain-driven design, multi-tier architecture.

**Стефанишин І. М., Яцишин В. В. Імплементація технології автоматизованої побудови та оптимізації конвеєрів програмних компонентів для класифікації ЕЕГ-сигналів.** У статті запропоновано імплементацію технології автоматизованої побудови та оптимізації конвеєрів програмних компонентів класифікації електроенцефалографічних (ЕЕГ) сигналів. Актуальність практичного застосування такої технології зумовлена тим, що в існуючих підходах побудова конвеєрів програмних компонентів переважно виконується емпірично або зосереджується на окремих алгоритмах, тоді як формалізоване задання допустимих конфігурацій, контроль їхньої коректності, обчислення, порівняння та оптимізація залишаються недостатньо реалізованими на рівні цілісної програмної системи. Метою роботи є програмна реалізація технології, що забезпечує автоматизовану побудову, обчислення, оцінювання та оптимізацію конвеєрів програмних компонентів класифікації ЕЕГ-сигналів. У статті конвеєр програмних компонентів подано у вигляді деревовидної орієнтованої графової структури. Для реалізації запропонованої технології сформовано функціональні та нефункціональні вимоги, обґрунтовано архітектурні рішення, визначено основні модулі програмної системи та описано її інфраструктуру. Програмна система реалізує повний технологічний цикл: створення експерименту, завантаження ЕЕГ-даних, побудову деревовидної орієнтованої графової структури, налаштування параметрів обчислення й оптимізації, виконання локальних і хмарних обчислень, запуск оптимізації та подання результатів. Оптимізація передбачає багатокритеріальне оцінювання конвеєрів програмних компонентів за точністю, F1-мірою, площею під ROC-кривою та нормалізованим часом обробки одного зразка з урахуванням вагових коефіцієнтів. Якість програмної реалізації підтверджено автоматизованим тестуванням десктопного застосунку, сервера, скрипта обчислень і скрипта оптимізації. Експериментальну верифікацію виконано на реальних ЕЕГ-даних моторної активності пальців верхніх кінцівок. Одержані результати підтвердили практичну придатність запропонованої технології та реалізованої програмної системи.
**Ключові слова:** електроенцефалографічні сигнали, класифікація, архітектура, технологія, програмна система, інфраструктура, алгоритм, якість, предметно-орієнтоване проєктування, багаторівнева архітектура.

## Formulation of the problem.

In the context of ensuring the quality of software systems for EEG signal classification, the main scientific problem lies in the absence of a formalized approach to the construction, evaluation, and optimization of software component pipeline configurations. This is especially important for fine motor activity recognition, where the non-stationarity of EEG signals, individual variability, and the similarity of neurophysiological manifestations of finger movements complicate stable class differentiation [1]. The relevance of this problem is also confirmed in related engineering fields, where, in Boston Dynamics research, the recognition and reproduction of hand movements are identified as a complex stage of

controlling mechanical systems [2]. At the same time, existing studies mostly consider individual algorithms rather than the formalized construction of complete configurations, which complicates traceability, comparison of alternatives, and selection of the most effective solution [3, 4].

Solving this problem requires the development of a technology for automated construction and optimization of software component pipelines for EEG signal classification. The basis of this technology should be a tree-like directed graph structure that defines the space of admissible configurations, records the relationships between software components, and prevents the formation of incorrect computational routes. Optimization in this context should be performed as a formalized process of selecting the most optimal configuration based on a set of classification quality metrics, computational characteristics, weighting coefficients, and constraints of the applied task.

Solving this scientific problem directly determines the need to develop a software system capable of implementing the proposed technology as a complete controlled cycle. Such a system should not only launch individual algorithms but should support the formation of software component pipelines, execution of computations, recording of results, comparison of configurations, and presentation of a justified selection of the most effective configuration. It is the implementation of the software system that transfers the technology from the level of a formal approach to the level of a reproducible engineering tool for EEG signal classification.

**An analysis of the latest research and publications.**

A previous systematic analysis of the subject area of EEG signal classification demonstrates that the main limitation of current studies lies in the fragmented consideration of the software component pipeline [1]. The vast majority of works focus on the modification of isolated algorithms or individual stages of data processing, whereas the pipeline as an integrated software structure remains insufficiently formalized. In such approaches, admissible sequences of components, rules of their compatibility, configuration parameters, and the relationship between the pipeline structure and the final quality metrics of the software system are not always explicitly defined. The formation of computational routes in scientific experiments is mainly performed without specialized software tools for the automated construction and selection of configurations, which leads to a manual or partially formalized choice of methods for solving this task. As a result, evaluation is often based on a limited number of metrics, while insufficient traceability of computations complicates the objective comparison of alternative configurations and the reproduction of the obtained results [3, 4].

At the same time, the analysis of modern AutoML systems, in particular H2O AutoML [5], AutoGluon [6], and Google Cloud AutoML Tables [7], shows that their limitations are directly related to the stated scientific problem. First, these systems do not have mechanisms for the explicit representation of domain-specific data processing stages as controlled software components of a pipeline. Second, optimization in such software systems is mainly oriented toward a single primary metric, which does not allow the formal consideration of a combination of classification quality, result stability, and computational characteristics. Third, experiment traceability remains incomplete, since mostly the final model or results table is recorded rather than the complete structure of the software component pipeline configuration. As a result, existing AutoML systems do not provide a sufficient basis for the reproducible comparison of alternative configurations and the selection of the most optimal variant for a specific applied task.

**Presenting the main material.**

**Formalization of the Construction and Optimization of a Software Component Pipeline within the Technology.**

The technology proposed in this study involves the formalization of two interrelated processes: the construction of admissible configurations of a software component pipeline and the selection of its most optimal configuration. This ensures a transition from the informal combination of individual algorithms to a controlled software engineering process in which the composition of the pipeline, relationships between software components, execution parameters, and evaluation metrics are explicitly defined.

The formalization of the process of constructing admissible pipeline configurations is based on representing its software components as a tree-like directed graph structure. Within this structure, nodes correspond to software components, while the relationships between them determine the admissible sequence of data transfer and transformation. This makes it possible to control component compatibility at the stage of configuration construction, prevent the formation of incorrect computational routes, and ensure the traceability of the transition from input EEG signals to classification results.

The formalization of the process of optimizing the software component pipeline assumes that, for each admissible configuration, the values of classification quality metrics are computed in the form of accuracy, F1-score, and area under the ROC curve, as well as computational characteristics, in particular the normalized processing time per data sample with corresponding weighting coefficients. To bring heterogeneous metrics to a common scale, their values are normalized, after which the metrics subject to maximization are converted into minimization form. The integral value for each configuration is determined by formula (1):

$$F(Accuracy, F1, ROC\_AUC, NTPS) = a_{Accuracy} Min\_Norm_{Accuracy} + \\ + a_{F1} Min\_Norm_{F1} + a_{ROC\_AUC} Min\_Norm_{ROC\_AUC} + a_{NTPS} Norm_{NTPS} \quad (1)$$

under the conditions $a_{Accuracy} + a_{F1} + a_{ROC\_AUC} + a_{NTPS} = 1$,

where $a$ – weighting coefficients that define the priorities of the applied task, $Min\_Norm$ – normalized metric values converted into minimization form, $Norm$ – the normalized value of the metric, $NTPS$ – the normalized processing time per data sample.

To select the most effective configuration, the mixed-integer linear programming method is used. Its application makes it possible to formalize the procedure for selecting one software component pipeline from the space of admissible configurations for which the value of the objective function (1) is minimal (2):

$$min \sum_{i=1}^{m} F(Accuracy, F1, ROC\_AUC, NTPS)_i x_i, \quad (2)$$

under the conditions $\sum_{i=1}^{m} x_i = 1, x_i \in \{0,1\}$,

where m – the number of admissible configurations of the software component pipeline, $x$ – a binary variable for configuration selection. The constraint guarantees that only one configuration will be selected from all admissible configurations. The configuration with the minimum value of the objective function is defined as the most appropriate one for the metrics, their specified weighting coefficients, and the conditions of the applied task.

**Formation of Requirements for the Technology.**

After formalizing the principles of construction and optimization of the software component pipeline, it is necessary to define the requirements for the technology, since they determine the scope of its functionality, the conditions of its practical application, and the basis for further architectural design.

**Functional Requirements of the Technology.**

The functional requirements (Fig. 1) define a set of actions that must be implemented in the software system to realize the complete cycle of automated construction and optimization of software component pipelines for EEG signal classification. Their formation is necessary in order to specify the boundaries of user interaction with the system, the distribution of responsibilities between roles, and the sequence of transition from experiment preparation to obtaining optimization results.

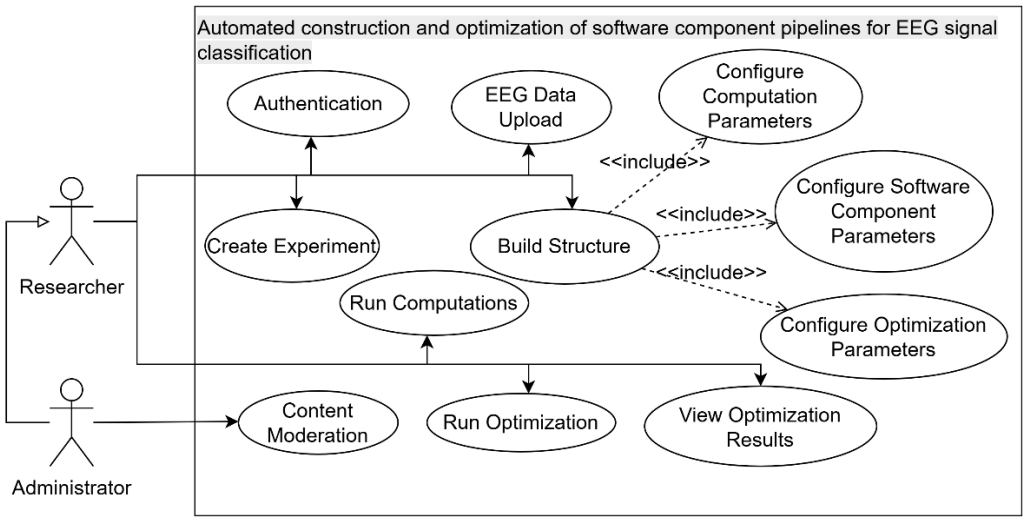



Fig. 1. Use case diagram for the implementation of the technology for automated construction and optimization of software component pipelines for EEG signal classification

In Fig. 1, the functional requirements are grouped around two roles: researcher and administrator. The researcher is the main actor, since this scenario covers the technological cycle of working with the software system: experiment preparation, working with EEG data, formation of a tree-like directed graph structure, parameter configuration, computation launch, optimization execution, and results review. The administrator supports the same main scenario of working with the technology and additionally performs service moderation functions. The diagram records not only the list of functions but also the boundaries of user responsibilities within the software implementation of the technology.

**Non-Functional Requirements of the Technology.**

The non-functional requirements for the implementation of the technology define the conditions of its practical use during the automated construction and optimization of software component pipelines for EEG signal classification. The performance of the technology must ensure prompt user interaction with the software system and support high-performance computations for a set of pipeline configurations. Scalability involves the correct operation of the software system as the number of software components, admissible configurations, and computational tasks increases. Reliability consists in the correct processing of input data, configuration parameters, and computation results without disrupting the processes of construction, computation, and optimization. Usability involves clear interaction between the researcher and the software system during software component pipeline configuration, computation and optimization launch, and results analysis. Portability must ensure the possibility of using the software system in different operating environments without changing the basic logic of the technology.

**Selection of Methods and Tools for the Stages of the Software Component Pipeline.**

The selection of methods and tools for the stages of the software component pipeline was carried out based on the results of the analysis of scientific studies presented in previous works [1], taking into account the prevalence of the corresponding approaches in EEG signal classification tasks and their suitability for implementation within the proposed technology. The following set of software components was used as the basic set: CSP [3] for the preprocessing stage, ASR [8] for the data enhancement stage, ICA [9] for the feature extraction stage, PCA [10] for the dimensionality reduction stage, and SVM [11] and CNN [12] for the classification stage.

The selected set covers the key stages of the EEG signal classification pipeline and is sufficient for the initial implementation and verification of the mechanisms for constructing, computing, and optimizing configurations within the software system. It is not exhaustive, since the architecture of the software system provides for the further expansion of the set of software components for each pipeline stage without changing the general logic of the technology.

**Architectural Principles in the Implementation of the Technology.**

**Architectural Decisions for Supporting the Technological Cycle.**

The implementation of the technology involves the use of a set of software engineering approaches and design patterns, the selection of which is determined by the need to ensure the structured organization of domain logic, coordinated interaction between functional modules, flexible data storage, and support for different modes of computation execution. This set of decisions forms the architectural basis for implementing the automated construction and optimization of software component pipelines for EEG signal classification. In designing the architecture of the software system for automated construction and optimization of software component pipelines for EEG signal classification, the following approaches were used:

- Domain-driven design was used to decompose the architecture of the software system into separate domain parts, which made it possible to separate the logic of construction, computation, optimization, reporting, and data management without mixing responsibilities.
- Client-server architecture was applied to separate the researcher's interactive work from server-side management of experiments, data, computations, and optimization.
- Multi-tier architecture was used to coordinate the three main layers of the technology: the desktop part, the server part, and external services.
- A modular monolith was applied to the server part, since domain-driven design made it possible to organize it as a set of isolated modules within a single application.
- The MVC design pattern was applied to separate state, interface, and user action processing.
- A document-oriented database was selected due to the nested and variable structure of the data, which made it possible to store data in a form natural to them and avoid excessive schema complexity.

- Hybrid local-cloud execution was incorporated into the technology to support experiments of different computational complexity and to combine local work with the possibility of scaling in a cloud environment.

**Architectural Structure.**

The architectural structure of the software system defines the way in which the construction of the tree-like directed graph structure, computation, and optimization of software component pipeline configurations are combined within a single system, taking into account the needs of the applied task (Fig. 2). This ensures a transition from the empirical formation of pipelines to a controlled process of selecting the most effective configuration.

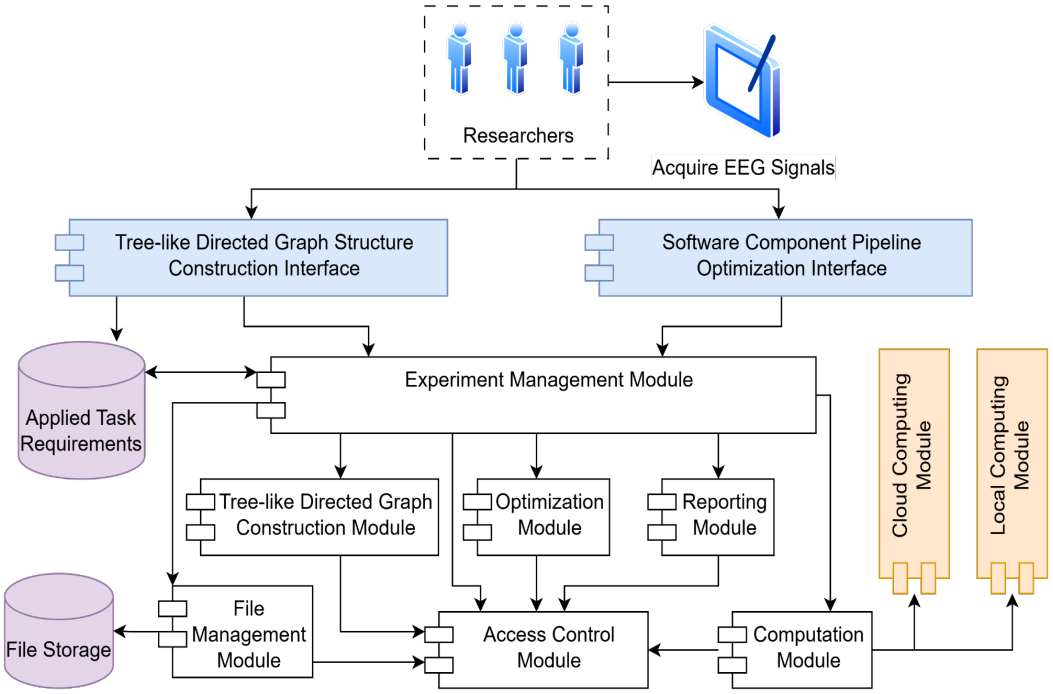



Fig. 2. Architecture of the software system for the automated construction and optimization of software component pipelines for EEG signal classification

The architecture shown in Fig. 2 combines input elements, interface control tools, domain modules, and computational infrastructure within a single cycle of construction, computation, and optimization of configurations. It specifies how the selected architectural decisions support the technological cycle of automated construction and optimization of the software component pipeline. Domain-driven design ensured the decomposition of the system into domain modules. Client-server and multi-tier organization separated interface interaction, experiment coordination, and computation execution. The modular monolith preserved the integrity of the server logic, while local-cloud execution transferred the computational layer to a scalable environment.

Based on domain-driven design, the software system includes modules for experiment management, access control, file management, construction of the tree-like directed graph structure, computation, optimization, and reporting. The division into these modules separates the logic of configuration formation, computation execution, selection of the most appropriate configuration, and presentation of results.

In addition to domain modules, the architecture includes interface and infrastructure elements that support the execution of the technology. The construction interface provides the formation of the tree-like directed graph structure and simultaneous specification of the needs of the applied task. The formed needs are then used by the experiment management module to coordinate construction, computation, and optimization parameters within the experiment. The optimization interface provides the launch of the procedure for selecting the most effective configuration and work with the obtained results. The file storage stores EEG signals, intermediate results, and final artifacts of the experiment. Local and cloud computations execute the formed configurations depending on the computational complexity of the task.

Within the proposed architecture of the software system, EEG signals are supplied to the construction interface, where, together with the needs of the applied task, a tree-like directed graph structure is formed and the space of admissible configurations is defined. Then, the experiment management module coordinates the transition to computing these configurations in a local or cloud environment. The obtained results are transferred to the optimization module, where the most effective configuration is determined. After that, the reporting module forms the final representation of the results for further analysis and use.

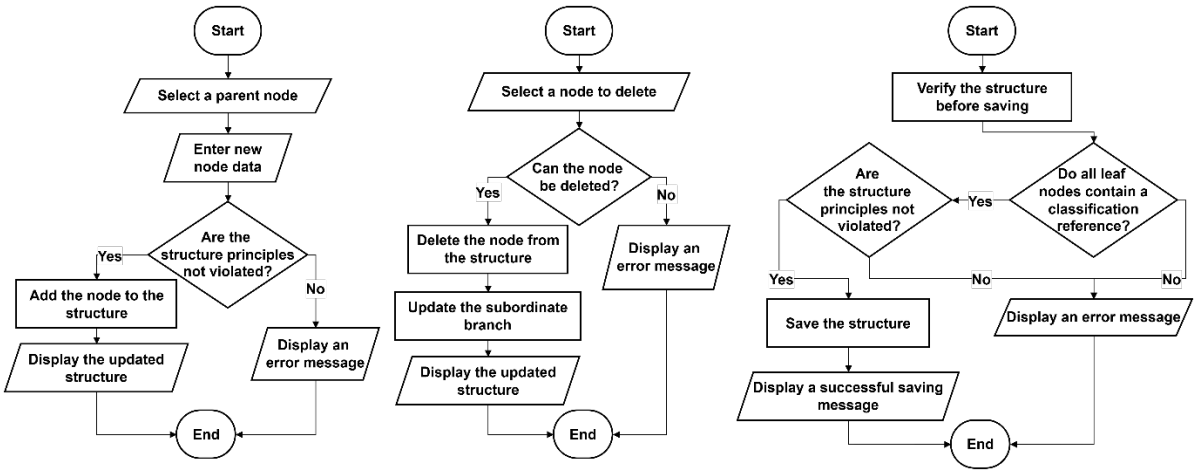
**Scalability and Extensibility of the Architecture.**

The extensibility of the architecture of the software system for automated construction and optimization of software component pipelines for EEG signal classification is embedded at the level of representing the software component pipeline as a tree-like directed graph structure. Such a representation allows new algorithms, node types, and variants of their composition to be added without revising the basic architecture of the technology. Due to this, the software system can be adapted to new applied tasks, other datasets, and new variants of EEG signal processing while preserving the unified logic for constructing admissible configurations.

The scalability of the architecture is implemented at the level of the computational layer. The increase in the number of configurations and computational tasks is supported by the separation of local and cloud execution, with both modes relying on a common computational logic. This ensures the consistency of results in different environments, reduces maintenance complexity, and creates a basis for expanding computational resources without changing the logic of configuration construction and optimization.

**Flowcharts for Constructing the Software Component Pipeline.**

To formalize the construction of the tree-like directed graph structure of the software component pipeline, Fig. 3 presents flowcharts that specify the admissible operations for modifying the structure and the conditions for verifying its correctness. The flowcharts describe the construction process through states, transitions, and checkpoints, which prevent the formation of inadmissible configurations.




a b c

Fig. 3. Flowcharts for constructing the tree-like directed graph structure of the software component pipeline: a) node addition; b) node deletion; c) structure saving

Fig. 3 distinguishes three operations for working with the tree-like directed graph structure: node addition, node deletion, and structure saving. The division into separate flowcharts makes it possible to describe step by step the rules for modifying the structure and verifying its correctness.

During node addition, as shown in Fig. 3a, the parent node is first selected and the data of the new node are entered. After that, the system verifies whether the new element can be added to the structure. If the operation does not violate the principles of constructing the tree-like directed graph structure, the node is added, and the user receives an updated representation of the structure. If the addition is inadmissible, the system displays an error message.

The flowchart in Fig. 3b describes the process of node deletion. After a node is selected, the system verifies whether it can be removed without violating the correctness of the structure. If the verification is successful, the node is deleted, the subordinate branch is updated, and the user receives an updated

representation of the structure. If the deletion is inadmissible, the system does not modify the structure and reports an error.

Fig. 3c presents the process of saving the constructed structure. Before saving, the system verifies whether the principles of structure construction have been violated and whether all leaf nodes contain a classification reference. Satisfying these conditions confirms that the constructed branches can be considered admissible configurations of the software component pipeline. If the verification is successful, the structure is saved, and the user receives a message confirming successful saving. If at least one of the conditions is violated, the system displays an error message.

**Class Diagram of the Subsystem for Constructing the Tree-like Directed Graph Structure of the Software Component Pipeline.**

To specify the static organization of the subsystem for constructing the tree-like directed graph structure of the software component pipeline, a class diagram is presented in Fig. 4. It reflects the software implementation of the subsystem at the level of the main software entities, their relationships, and distribution of responsibilities, and also shows how the formation, modification, and correctness verification of the tree-like directed graph structure are organized within the software system.

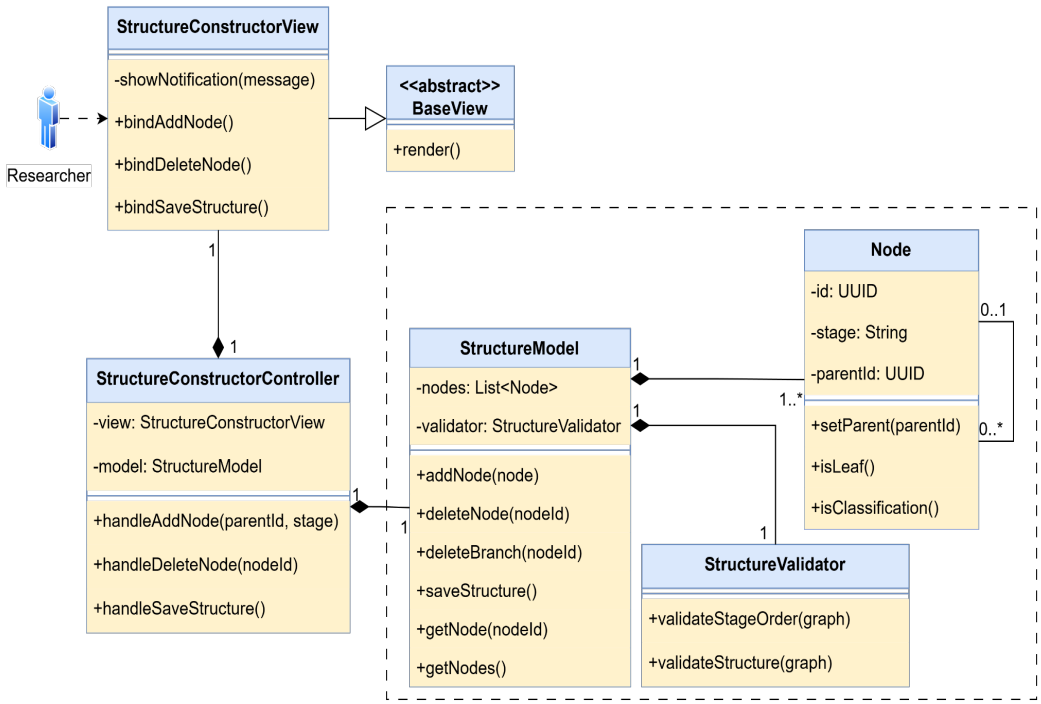



Fig. 4. Class diagram of the subsystem for constructing the tree-like directed graph structure of the software component pipeline

The construction subsystem is organized according to the MVC design pattern. Within it, interface interaction, management of operations for modifying the tree-like directed graph structure, preservation of its current state, and verification of the correctness of the formed software component pipeline configuration are separated. The central element of this organization is the *StructureConstructorController* class, through which operations for adding nodes, deleting nodes, and saving the structure are coordinated. The interface layer, or View, is represented by the *StructureConstructorView* class, which interacts with the system user and inherits the base abstraction *BaseView*. This separates the logic of displaying and processing interface events from the logic of modifying the tree-like directed graph structure. The Model layer is implemented by the *StructureModel* class, which contains the current state of the tree-like directed graph structure and the basic operations for changing it. Individual nodes of this structure are represented by the *Node* class, which reflects the node's belonging to the corresponding processing stage and its relationship with the parent node. Verification of the correctness of the formed software component pipeline configuration is assigned to the *StructureValidator* class, which encapsulates the logic for controlling the order of processing stages and compliance with the requirements for the tree-like directed graph structure.

**Sequence Diagram for the Optimization of Software Component Pipeline Configurations.**

The sequence diagram for the optimization of software component pipeline configurations specifies the order of message exchange between the researcher, the experiment management module, and the optimization module during the selection of the most appropriate configuration (Fig. 5).

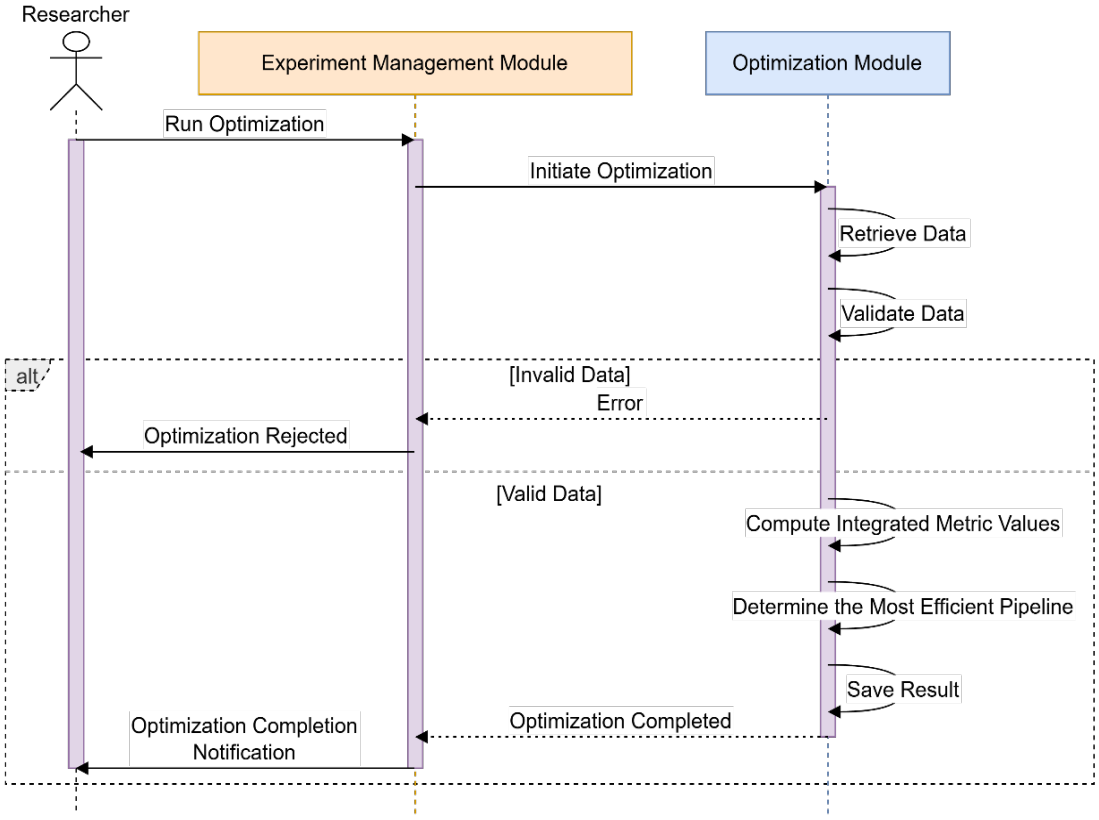



Fig. 5. Sequence diagram for the optimization of software component pipeline configurations

The diagram shows that the researcher initiates optimization through the experiment management module, which sends a request to the optimization module for further processing. In the optimization module, data retrieval, data validation, computation of integral indicators, determination of the most effective configuration, and saving of the result are performed sequentially. The alternative combined fragment defines two execution branches depending on the validity of the input data. If the data do not meet the validation criteria, the procedure is terminated with an error message and optimization rejection. If the data are successfully validated, the full cycle of optimization processing is performed, after which the result is transferred to the experiment management module, and the researcher receives a notification that optimization has been completed.

**Construction of the Software System as an Implementation of the Technology for Automated Construction and Optimization of Software Component Pipelines for EEG Signal Classification.**

The construction of the software system provides its practical software implementation. From the perspective of software engineering, the construction of the software system makes it possible to verify not only the correctness of individual algorithms but also the ability of the proposed technology to operate as an integrated engineering tool.

**Software System Infrastructure.**

The software system infrastructure defines the way in which the software system for automated construction and optimization of software component pipelines is practically deployed. It ensures the distribution of responsibilities between system elements, the isolation of resource-intensive computations from interface interaction, centralized storage of experimental data and results, as well as support for local and cloud execution of software component pipeline configurations. To implement these requirements, the software system is represented as a set of interconnected infrastructure elements, each of which supports a separate part of the technological cycle (Fig. 6).

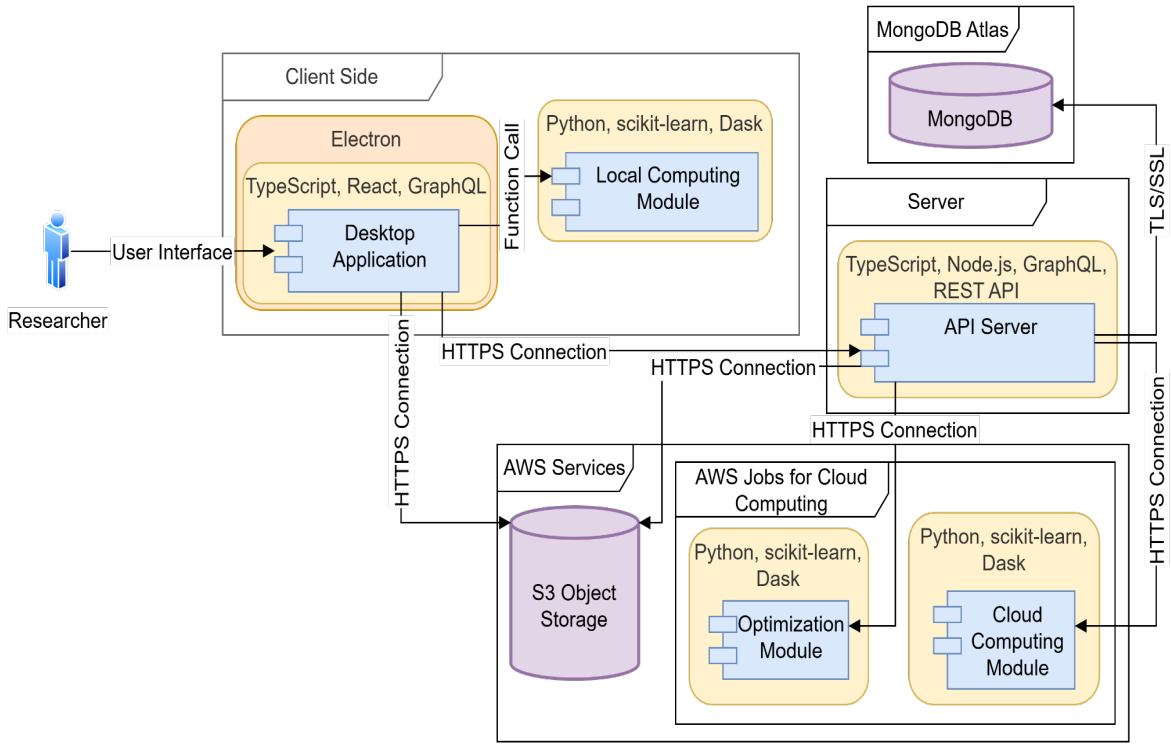



Fig. 6. Infrastructure of the software system for automated construction and optimization of software component pipelines for EEG signal classification

The software system infrastructure consists of four main elements: the client part, the server, the MongoDB Atlas cloud database management service, and AWS services for storing file artifacts, performing cloud computations, and launching optimization (Fig. 6). Such an organization ensures the deployment of the software system as a set of interconnected infrastructure layers that support the operation of the technology for automated construction and optimization of software component pipelines for EEG signal classification.

The desktop application was implemented using Electron, React, TypeScript, and GraphQL [13–16]. The choice of Electron was determined by the need to deploy the client part in the researcher's local environment. React was used to build the user interface through a component-based model, while TypeScript was used to type the client logic, which improves the controllability of working with experiment data, parameters of the tree-like directed graph structure, and optimization results. GraphQL was applied for data exchange between the client and server layers, since it makes it possible to explicitly define the composition of data required for individual scenarios of the software system. The client part includes a local computation module implemented using Python, scikit-learn, and Dask [17–19]. Python and scikit-learn were used to implement data processing algorithms, machine learning, and metric computation, while Dask was used to organize parallel execution of computations and optimization.

The server was implemented using Node.js, TypeScript, and GraphQL [15, 16, 20]. Node.js was used to build the server layer aligned with the ecosystem of the desktop application. TypeScript provides typing of server logic, domain models, workflow states, software component pipeline configurations, and optimization results. GraphQL was applied for working with the domain data of the software system, while REST API was used for file operations, since uploading and retrieving EEG files have a clearly defined operational nature.

The MongoDB Atlas cloud database management service was used to store the structured data of the software system [20]. Its selection was determined by the need to store variable document-oriented structures: experiments, parameters of the tree-like directed graph structure, software component pipeline configurations, metric values, and optimization results. The server connection to MongoDB Atlas is established through a TLS/SSL connection.

AWS services were used to store file artifacts and execute the cloud computational layer [21]. Amazon S3 was applied as object storage for EEG files and generated reports [22]. The choice of AWS tasks for cloud computations and optimization was determined by the need to run resource-intensive

procedures outside the researcher's local environment and to scale experiment execution as the number of software component pipeline configurations increases.

Interaction between the client part, server, AWS services, and computational modules is carried out through HTTPS connections. As a result, the software system infrastructure provides the technological basis for local and cloud execution of computations, storage of EEG data, recording of experiment results, and determination of the most appropriate configuration of the software component pipeline for EEG signal classification.

**Implementation of the Algorithm for Optimizing Software Component Pipeline Configurations.**

The implementation of the optimization algorithm provides the software realization of the formalized selection of the most effective software component pipeline configuration. Due to this, the software system implements a reproducible procedure for selecting a configuration according to the needs of the applied task (Listing 1).

Listing 1. Implementation of the algorithm for optimizing software component pipeline configurations

```
def optimize_configurations(configurations, weights):
    if abs(sum(weights.values()) - 1.0) > 1e-9:
        raise ValueError("The sum of metric weights must be equal to 1.")
    prepared = []
    for configuration in configurations:
        if configuration["sampleCount"] == 0:
            raise ValueError("Sample count must be greater than zero.")
        metrics = {
            "accuracy": vector_length(configuration["accuracyScores"]),
            "f1": vector_length(configuration["f1Scores"]),
            "rocAuc": vector_length(configuration["rocAucScores"]),
            "ntps": configuration["duration"] / configuration["sampleCount"],
        }
        prepared.append({
            "configurationId": configuration["configurationId"],
            "metrics": metrics,
        })
    accuracy_norm = min_max_normalize([c["metrics"]["accuracy"] for c in prepared])
    f1_norm = min_max_normalize([c["metrics"]["f1"] for c in prepared])
    roc_auc_norm = min_max_normalize([c["metrics"]["rocAuc"] for c in prepared])
    ntps_norm = min_max_normalize([c["metrics"]["ntps"] for c in prepared])
    results = []
    best_configuration = None
    best_score = None
    for i, configuration in enumerate(prepared):
        score = (
            weights["accuracy"] * (1.0 - accuracy_norm[i]) +
            weights["f1"] * (1.0 - f1_norm[i]) +
            weights["rocAuc"] * (1.0 - roc_auc_norm[i]) +
            weights["ntps"] * ntps_norm[i]
        )
        results.append({
            "configurationId": configuration["configurationId"],
            "score": score,
        })
        if best_score is None or score < best_score:
            best_score = score
            best_configuration = configuration["configurationId"]
    return {
        "bestConfigurationId": best_configuration,
```

```
    "bestScore": best_score,
    "results": results,
}
```

Listing 1 presents the software implementation of optimization processing for software component pipeline configurations. The algorithm forms an integral value for each configuration and determines the most effective configuration with the minimum integral value, which ensures automated and reproducible optimization execution in the software system.

**Ensuring Code Quality and Maintainability of the Software System.**

The code quality of the software system was ensured through automated testing of its main software layers. For each layer, a set of tests was formed to verify key execution scenarios and reduce the risk of regressions during further development of the software system (Table 1).

Table 1. Test coverage of the software layers of the software system

| Part of the software system | Number of tests | Code test coverage |
|---|---|---|
| Desktop application | 58 | 62.67% |
| Server | 101 | 87.05% |
| Computation script | 15 | 100.00% |
| Optimization script | 11 | 100.00% |

Testing covers the desktop application, server, computation script, and optimization script, which ensures control over the correctness of the main software layers. The highest coverage was achieved for the computation and optimization scripts, since they implement the formalized logic of the critical stages of the technological cycle. The lower coverage of the desktop application and server is explained by the presence of interface, integration, and service logic. The obtained indicators confirm that the main computational and optimization layers of the software system have been tested at a level sufficient for the further extension of the implementation.

**Experimental Verification of the Software System for Automated Construction and Optimization of Software Component Pipelines for EEG Signal Classification.**

**Experimental Conditions for Verifying the Software System.**

The software system was verified on a private dataset of EEG signals of motor activity of human upper-limb fingers, formed in previous studies [6–8]. These data were used as a real applied scenario for verifying the complete cycle of the software system operation: uploading EEG data, constructing a tree-like directed graph structure, computing software component pipeline configurations, and optimizing configurations according to the defined metrics.

**Verification of the Technology Implementation in the Software System.**

Verification of the implemented software system for automated construction and optimization of software component pipelines is important from the perspective of software engineering, since it is at this stage that it is confirmed that the proposed methods have been brought to the level of an integrated software tool. This concerns not only the correctness of individual computations but also the verification of the complete applied scenario of the software system operation: from uploading EEG data to determining the most effective software component pipeline configuration.

Work with the software system begins with user authorization, after which an experiment is created and an EEG data file is uploaded to it. These actions are initial steps for the subsequent construction of the tree-like directed graph structure and the formation of software component pipeline configurations.

After the experiment is created, the tree-like directed graph structure is constructed, on the basis of which admissible software component pipeline configurations are formed. Within the verification, the mechanism of automatic structure generation was used for this purpose, making it possible to form a set of admissible configurations based on predefined software components and rules for their combination (Fig. 7).

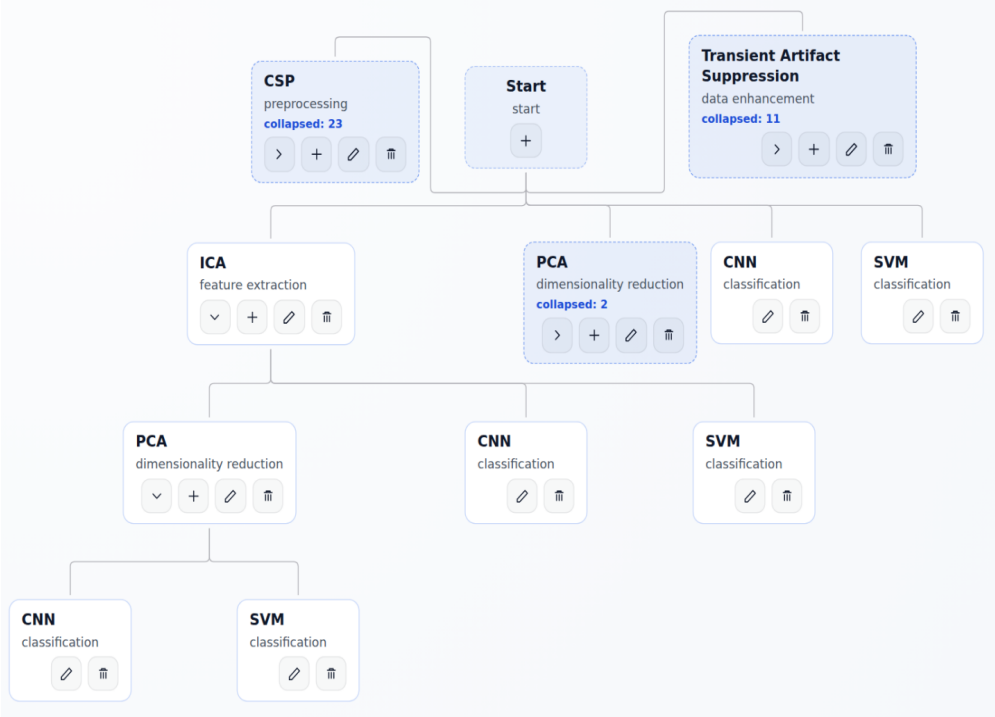



Fig. 7. Example of a constructed tree-like directed graph structure of the software component pipeline

Fig. 7 shows the construction of a tree-like directed graph structure from all software components currently supported by the software system. As a result of the construction, 32 admissible software component pipeline configurations were formed for further computation and optimization. Parameter configuration is provided for software components; however, within the verification, their manual configuration was not performed. Parameter values are selected by the hyper-optimization mechanism during configuration computation, which preserves the automated nature of the construction and optimization of the software component pipeline.

The next stage is the specification of computation and optimization parameters, which determine the way configurations are executed and the subsequent selection of the most appropriate configuration (Fig. 8).

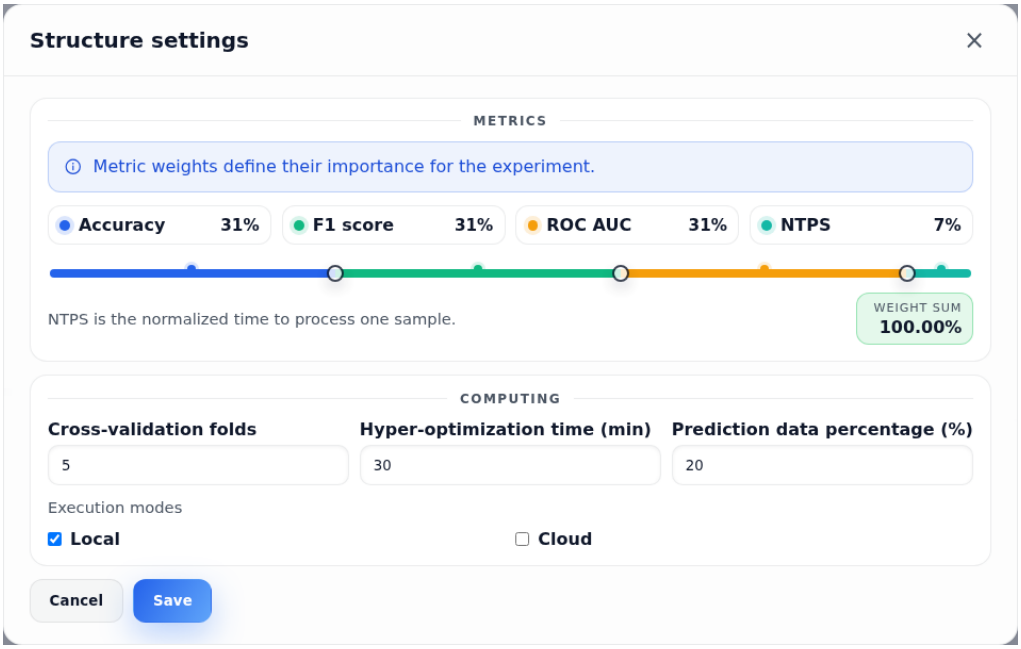



Fig. 8. Configuration of computation and optimization parameters

Fig. 8 shows the configuration of the weighting coefficients of the metrics used during the optimization of software component pipeline configurations. In this experiment, weights of 31% were assigned to Accuracy, F1-score, and area under the ROC curve, while 7% was assigned to NTPS. This ratio of weights defines the optimization priorities for the EEG signal classification task, where the main emphasis is placed on recognition quality, while computational costs are considered as an additional characteristic of the software component pipeline configuration. This also confirms that the software system implements a separate stage for configuring the experiment execution conditions before launching computations.

After configuration is completed, computations are launched for the formed configurations. During execution, the software system displays the current computation status, which makes it possible to view progress, stop execution, or run it again. This ensures the implementation of a complete computational stage within the software system. After the computations are completed, optimization of software component pipeline configurations is launched. As a result, the software system forms an ordered list of configurations according to their integral values. Since the total number of obtained configurations was 32, the article presents only the first 8 positions of the ranked list (Fig. 9).

**Leaderboard**

| PIPELINE | INTEGRAL VALUE | ACCURACY | F1 SCORE | AREA UNDER ROC CURVE | NORMALIZED PROCESSING TIME PER DATA SAMPLE | EXECUTION | STATUS |
|---|---|---|---|---|---|---|---|
| CSP → PCA → CNN | **0.000347** | 0.927764 | 0.927899 | 0.998277 | 0.000138 s/sample | Cloud queue | Completed |
| ICA → CNN | **0.013274** | 0.924256 | 0.924294 | 0.998068 | 0.000149 s/sample | Cloud queue | Completed |
| CSP → ICA → CNN | **0.013975** | 0.924136 | 0.924075 | 0.998051 | 0.000148 s/sample | Cloud queue | Completed |
| ICA → PCA → CNN | **0.016227** | 0.923384 | 0.923167 | 0.998053 | 0.000144 s/sample | Cloud queue | Completed |
| PCA → CNN | **0.017880** | 0.922929 | 0.922954 | 0.998002 | 0.000146 s/sample | Cloud queue | Completed |

Previous Next Page 1 of 7 Rows per page 5

Fig. 9. Results of optimizing software component pipeline configurations

Fig. 9 presents a fragment of the results of optimizing software component pipeline configurations. The results are ordered by integral value, where a lower value corresponds to a better configuration assessment according to the specified metric weights. The classification metrics for the first configurations are very close, so without optimization their direct comparison would be ambiguous. The use of the integral value made it possible to take Accuracy, F1-score, area under the ROC curve, and NTPS into account simultaneously and determine the most effective configuration. In this experiment, this configuration is «CSP → PCA → CNN». For additional evaluation of the optimization result, Fig. 10 presents the confusion matrix for the most appropriate software component pipeline configuration.

| | 1 | 12 | 12345 | 13 | 14 | 2 | 23 | 2345 | 24 | 3 | 34 | 4 | 5 | RETURN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 99.30% | 0.05% | 0.12% | 0.01% | 0.04% | 0.17% | 0.05% | 0.03% | 0.02% | 0.03% | 0.01% | 0.05% | 0.09% | 0.02% |
| 12 | 0.03% | 92.44% | 0.11% | 2.74% | 2.89% | 0.01% | 1.47% | 0.03% | 0.17% | 0.01% | 0.01% | 0.01% | 0.02% | 0.06% |
| 12345 | 0.09% | 0.10% | 91.12% | 0.05% | 0.11% | 0.01% | 0.16% | 2.84% | 0.03% | 0.00% | 0.42% | 0.00% | 0.01% | 5.04% |
| 13 | 0.01% | 3.09% | 0.13% | 88.89% | 5.84% | 0.01% | 1.61% | 0.04% | 0.30% | 0.01% | 0.03% | 0.00% | 0.00% | 0.03% |
| 14 | 0.03% | 2.21% | 0.16% | 2.71% | 87.13% | 0.01% | 6.74% | 0.04% | 0.83% | 0.02% | 0.06% | 0.01% | 0.01% | 0.06% |
| 2 | 0.34% | 0.03% | 0.01% | 0.01% | 0.03% | 97.08% | 0.02% | 0.01% | 0.01% | 1.54% | 0.01% | 0.40% | 0.50% | 0.01% |
| 23 | 0.06% | 1.11% | 0.18% | 0.84% | 6.74% | 0.01% | 86.73% | 0.06% | 4.16% | 0.01% | 0.04% | 0.00% | 0.00% | 0.07% |
| 2345 | 0.04% | 0.04% | 5.35% | 0.02% | 0.02% | 0.01% | 0.10% | 92.64% | 0.02% | 0.00% | 0.45% | 0.00% | 0.01% | 1.30% |
| 24 | 0.06% | 0.35% | 0.20% | 0.46% | 1.72% | 0.01% | 10.17% | 0.03% | 86.74% | 0.01% | 0.06% | 0.01% | 0.01% | 0.17% |
| 3 | 0.10% | 0.02% | 0.03% | 0.01% | 0.04% | 1.31% | 0.02% | 0.01% | 0.01% | 95.96% | 0.02% | 1.88% | 0.59% | 0.01% |
| 34 | 0.02% | 0.06% | 0.93% | 0.05% | 0.06% | 0.01% | 0.14% | 0.62% | 0.10% | 0.01% | 97.21% | 0.01% | 0.03% | 0.75% |
| 4 | 0.10% | 0.03% | 0.05% | 0.00% | 0.02% | 0.52% | 0.01% | 0.01% | 0.01% | 2.81% | 0.01% | 94.91% | 1.49% | 0.02% |
| 5 | 0.12% | 0.04% | 0.04% | 0.01% | 0.03% | 0.28% | 0.01% | 0.01% | 0.01% | 0.62% | 0.01% | 1.66% | 97.14% | 0.02% |
| RETURN | 0.05% | 0.12% | 7.42% | 0.04% | 0.06% | 0.01% | 0.26% | 1.07% | 0.14% | 0.01% | 0.40% | 0.01% | 0.01% | 90.40% |

Fig. 10. Confusion matrix for the most appropriate software component pipeline configuration

Fig. 10 presents the confusion matrix for the most effective software component pipeline configuration. The largest values are concentrated on the main diagonal, which confirms the correct

recognition of most classes of finger motor activity. The main errors occur between close finger combinations, in particular 23 → 24, 14 → 23, 14 → 13, as well as 12345 → RETURN.

The conducted verification demonstrated the compliance of the implemented software system with the main stages of the process of automated construction and optimization of software component pipelines for EEG signal classification. Within the applied scenario, the correctness of EEG data uploading, construction of the tree-like directed graph structure, computation of admissible configurations, optimization execution, and presentation of classification results was confirmed. The obtained results demonstrate the practical feasibility and suitability of the developed software system for automated selection of the most effective software component pipeline configuration according to the specified priorities of the applied task.

**Conclusions.**

The article considers the implementation of a technology for automated construction and optimization of software component pipelines in the form of a software system for EEG signal classification. Within the study, a scientific and applied problem was solved, namely the transition from empirical selection of individual algorithms for EEG signal processing and classification to a controlled software engineering process in which the software component pipeline acts as an object of construction, computation, evaluation, and optimization.

The construction of the software component pipeline is based on a tree-like directed graph structure, which defines admissible relationships between software components and forms a set of admissible configurations. Configuration optimization is considered as a minimization problem using classification quality metrics and computational resources, weighting coefficients, and mixed-integer linear programming to select the optimal software component pipeline configuration.

The article forms a software engineering basis for the software implementation of the technology for automated construction and optimization of software component pipelines for EEG signal classification. For this purpose, functional and non-functional requirements were defined, researcher work scenarios were described, architectural decisions were substantiated, the architectural structure of the technology was developed, and its scalability and extensibility were considered. As a result, the architecture of the software system is presented as an integrated set of engineering decisions that ensures the construction of a tree-like directed graph structure, computation of admissible configurations, optimization, and presentation of results.

The article implements a software system for automated construction and optimization of software component pipelines for EEG signal classification. This system supports experiment creation, EEG data uploading, construction of a tree-like directed graph structure, configuration of software component parameters, specification of metric weights, execution of local and cloud computations, optimization launch, and presentation of results. The software system infrastructure includes the client part, the server, the MongoDB Atlas cloud database management service, and AWS services. The quality of the software implementation was confirmed by automated testing of the desktop application, server, computation script, and optimization script.

The article presents experimental verification of the software system on a private dataset of EEG signals of motor activity of human upper-limb fingers. Within the verification, a tree-like directed graph structure was constructed from the supported software components, 32 admissible software component pipeline configurations were formed, their computation was performed, and optimization was carried out according to the specified metric weights. For the scenario with priority given to classification quality, the configuration "CSP → PCA → CNN" obtained the minimum integral value. The confusion matrix for this configuration showed a predominant concentration of correct classifications on the main diagonal and revealed the main errors between close combinations of finger motor activity.

The obtained results confirmed the practical suitability of the developed software system for automated construction and optimization of software component pipelines for EEG signal classification. For software engineering, the significance of the study lies in ensuring a reproducible process of configuration construction, traceability of the transition from EEG data to optimization results, controlled consideration of applied task priorities, and the possibility of further extending the software system with new software components and computation modes.

Prospects for further research are related to the development of the proposed technology toward its broader use on real EEG data and applied scenarios of EEG signal classification. A separate direction is the expansion of the set of software components for different pipeline stages, which will make it possible to

increase the space of admissible configurations, verify the technology on a broader set of signal processing and classification methods, and improve its suitability for different conditions of practical use.

**References**

1. Boyko I., Pastukh O., Stefanyshyn I., Lyashuk O. (2026) Review of EEG signal classification approaches in finger movement recognition. CEUR Workshop Proceedings. Vol. 4160. Available at: https://ceur-ws.org/Vol-4160/paper23.pdf (accessed: 10 March 2026).
2. Boston Dynamics. Atlas' Evolution From Research Robot to Industrial Humanoid. Available at: https://bostondynamics.com/blog/atlas-evolution-from-research-robot-to-industrial-humanoid/ (accessed: 10 March 2026).
3. Farhana U., Ferdous M. J. (2021) Improving Motor Imagery EEG Signals Classification Accuracy with CSP by Available Machine Learning Approach. Journal of Engineering Science. Vol. 12, no. 2. P. 67–77. DOI: https://doi.org/10.3329/jes.v12i2.54632.
4. Shelishiyah R., Thiyam D. B., Margaret M. J. (2023) Machine Learning-Based Classification of Hybrid BCI Signals using Mayfly-Optimized Multiclass Weighted Random Forest. International Journal on Recent and Innovation Trends in Computing and Communication. Vol. 12, no. 1. P. 29–35. DOI: https://doi.org/10.17762/ijritcc.v12i1.7907.
5. H2O.ai. H2O AutoML: Scalable and automatic machine learning. Available at: https://www.h2o.ai/platform/h2o-automl/ (accessed: 10 March 2026).
6. Erickson N., Mueller J., Shirkov A., Zhang H., Larroy P., Li M., Smola A. (2020) AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. arXiv preprint. arXiv:2003.06505. Available at: https://arxiv.org/abs/2003.06505 (accessed: 10 March 2026).
7. Google Cloud. AutoML Tables: Automatically build and deploy state-of-the-art machine learning models. Available at: https://cloud.google.com/automl-tables (accessed: 10 March 2026).
8. Ma R., Chen Y. F., Jiang Y. C., Zhang M. (2023) A New Compound-Limbs Paradigm: Integrating Upper-Limb Swing Improves Lower-Limb Stepping Intention Decoding From EEG. IEEE Transactions on Neural Systems and Rehabilitation Engineering. Vol. 31. P. 3823–3834. DOI: https://doi.org/10.1109/TNSRE.2023.3315717.
9. Chen R., Xu G., Jia Y., Zhou C., Wang Z., Pei J., Han C., Wang Y., Zhang S. (2023) Enhancement of Time-Frequency Energy for the Classification of Motor Imagery Electroencephalogram Based on an Improved FitzHugh–Nagumo Neuron System. IEEE Transactions on Neural Systems and Rehabilitation Engineering. Vol. 31. P. 282–293. DOI: https://doi.org/10.1109/TNSRE.2022.3219450.
10. Wei F., Xu X., Jia T., Zhang D., Wu X. (2023) A Multi-Source Transfer Joint Matching Method for Inter-Subject Motor Imagery Decoding. IEEE Transactions on Neural Systems and Rehabilitation Engineering. Vol. 31. P. 1258–1267. DOI: https://doi.org/10.1109/TNSRE.2023.3243257.
11. Lu Y., Wang W., Lian B., He C. (2024) Feature Extraction and Classification of Motor Imagery EEG Signals in Motor Imagery for Sustainable Brain–Computer Interfaces. Sustainability. Vol. 16, no. 15. Article 6627. DOI: https://doi.org/10.3390/su16156627.
12. Hwaidi J. F., Chen T. M. (2022) Classification of Motor Imagery EEG Signals Based on Deep Autoencoder and Convolutional Neural Network Approach. IEEE Access. Vol. 10. P. 48071–48081. DOI: https://doi.org/10.1109/ACCESS.2022.3171906.
13. Electron. Electron Documentation. Available at: https://electronjs.org/docs/latest (accessed: 10 March 2026).
14. Meta Open Source. React Documentation. Available at: https://react.dev (accessed: 10 March 2026).
15. Microsoft. TypeScript Documentation. Available at: https://www.typescriptlang.org/docs (accessed: 10 March 2026).
16. GraphQL Foundation. GraphQL Documentation. Available at: https://graphql.org/learn (accessed: 10 March 2026).
17. Python Software Foundation. Python Documentation. Available at: https://docs.python.org (accessed: 10 March 2026).
18. scikit-learn Developers. scikit-learn User Guide. Available at: https://scikit-learn.org/stable/user_guide.html (accessed: 10 March 2026).
19. Dask Developers. Dask Documentation. Available at: https://docs.dask.org (accessed: 10 March 2026).
20. OpenJS Foundation. Node.js API Documentation. Available at: https://nodejs.org/api/documentation.html (accessed: 10 March 2026).
21. MongoDB Inc. MongoDB Atlas Documentation. Available at: https://www.mongodb.com/docs/atlas (accessed: 10 March 2026).
22. Amazon Web Services. AWS Documentation. Available at: https://docs.aws.amazon.com (accessed: 10 March 2026).
23. Amazon Web Services. Amazon Simple Storage Service Documentation. Available at: https://docs.aws.amazon.com/s3 (accessed: 10 March 2026).

Історія статті:
Отримано: 20.05.2026 Доопрацьовано: 21.09.2026 Прийнято до друку: 23.09.2026 Опубліковано: 29.09.2026