UDC 004.75:004.85:519.86
DOI: 10.15587/2706-5448.2026.352435

**Dmytro Buhai,
Anatolii Zhuchenko,
Oleksii Zhuchenko,
Dmytro Kovaliuk,
Denys Skladannyy**

IMPROVING THE EFFECTIVENESS OF MEDICAL DECISION SUPPORT SYSTEMS BASED ON MACHINE LEARNING AND CLOUD SERVICES

The object of research is the process of developing and deploying decision support systems using ML models and cloud infrastructure. A significant problem solved in this research is the software implementation of such DSSs with ML models, as well as their further deployment for end-user access. As a result, a multipurpose scheme that combines the stages of local development and publication in a cloud infrastructure is proposed. Such approach is relevant for small companies and government agencies as it allows them to save financial resources on maintaining permanent IT specialists, maintenance and support. Its distinctive feature is that model training and its integration into a web application are performed at the local stage, while the publication stage uses cloud services to automatically update the project.

The research implements a comprehensive data preprocessing pipeline for stroke risk prediction, including KNN-based imputation for missing values and SMOTE + NCL for class balancing. Following a correlation analysis and data augmentation four classification algorithms: logistic regression, SVM, Random Forest, and eXtreme Gradient Boosting were evaluated. Logistic regression is identified as the top-performing model regarding recall after data augmentation. The final model is integrated into a Flask application via serialization and a dedicated inference module.

The application is published automatically from GitHub to Amazon's cloud environment using such services as EC2, S3, ECR, and Secrets Manager. The cost of maintaining such a project is significantly lower than using dedicated servers or third-party software with a subscription fee per user. The results can be used in various industries to create DSSs that require high availability and minimal maintenance costs.

Keywords: *stroke prediction, data analysis, machine learning, DSS, web-programming, cloud infrastructure.*

Received: 04.12.2025
Received in revised form: 26.01.2026
Accepted: 17.02.2026
Published: 28.02.2026

How to cite
Buhai, D., Zhuchenko, A., Zhuchenko, O., Kovaliuk, D., Skladannyy, D. (2026). Improving the effectiveness of medical decision support systems based on machine learning and cloud services. Technology Audit and Production Reserves, 1 (2 (87)), 75–84. https://doi.org/10.15587/2706-5448.2026.352435

1. Introduction

Today, machine learning models (ML models) are widely used as separate components of control system, monitoring, access control, and decision support systems (DSS). It makes possible through information technology development, which, firstly, allowed for the accumulation of a large amount of data from various subject areas and, secondly, ensured the growth of computing power for data processing, including in real time. The main areas of ML models' application are industry, agriculture, medicine, banking, the military, and the aerospace industry [1].

It should be noted that methods for obtaining machine learning models based on numerical data, which are mainly used to solve classification, regression, and classification tasks, are relatively well developed and are already implemented in various mathematical packages, cloud platforms, and machine learning software frameworks. From a technical point of view, an ML model is an object stored in a local environment as a file or files set and learned to search for relevant patterns. Using such models in the environment where they are obtained is not difficult and comes down to model initializing and prediction function calling for the relevant input values [1, 2].

In the case of fully automated systems, the obtained ML models can be transferred to the appropriate software and hardware platform and integrated into existing automated process control systems (APCS). However, this approach does not work for decision support systems, as these involve manual or semi-automatic data entry by end users: operators, doctors, bank employees, etc. A typical example is a DSS for medical diagnostics, where a doctor or patient enters clinical data, physiological indicators, and medical history into the system to make a diagnosis and predict disease risks [1, 2].

Nowadays, the number of papers exploring the implementation and deployment of ML models is significantly lower than the number of papers devoted to the development of ML models themselves. On the one hand, it is explained by the fact that not all developed ML models require industrial deployment, and on the other hand, the complexity of the model deployment process significantly increases. Such a process demands a comprehensive approach and the involvement of data science, software development, and DevOps specialists.

Analyzing existing work in this area, it can be concluded that implementing ML models is a complex task. It requires continuous collection of new data, retraining of models, the use of numerous data analysis libraries, and updating models in the production environment. For instance, the [3] demonstrates that employing the Machine Learning Operations methodology can significantly accelerate and optimize the deployment process of ML models. In general, the machine learning deployment workflow consists of the following stages: data management, model learning, model verification, and model deployment. As shown in [4], compared to classical software engineering, the machine learning systems implementation contains unique artifacts: datasets and models. This corrected to the emergence of an entire MLOps direction – best practices for creating, validating, and publishing machine learning models. As shown in [5], on addition to classic DevOps, data analysts are added to the team, who are responsible for data processing and model development.

To correctly implement the above steps, it is necessary to answer the following questions: how will the model be created and updated, and how will it be accessible to end users.

Let's note that it is possible to completely place all processes in the cloud service based on corporate solutions such as AWS SageMaker, Microsoft ML, TensorFlow TFX, and MLflow. However, this approach has certain drawbacks, such as the cost of using cloud resources, dependence on the provider, possible legal issues, and dependence on an internet connection.

In several papers dedicated to DSS implementation based on ML models, it is common practice for the data processing and model training stages to occur locally in the developer environment (Dev), with the subsequently tested models being updated in the production environment. For example, the authors of [6] provide a structured guideline for deploying ML models in production environments, focusing on specific use cases related to predictive quality. They demonstrate relevant solutions and the necessary steps for successful implementation. A similar approach has been successfully applied to DSS in medicine [7] and in 3D printing [8].

Interaction with models is carried out by calling the REST-API, and the models themselves are implemented as web applications to which https requests are made. Such an approach has better development flexibility, since the developer has complete control over the process. Local development also reduces the cost of using cloud computing in the early stages. Cloud hosting is a key element for the effective deployment and maintenance of ML solutions. The authors of [9] note that cloud technologies provide scalability, cost efficiency, innovation, and security, which are important factors for the deployment of ML solutions.

Paper [10] provides a various cloud services overview and their providers, highlighting the capabilities of remote file storage, document co-authoring, and the ability to use software solutions regardless of computer hardware specifications. This provides wide opportunities for the cloud technologies implementation in various fields, including ML-based control systems.

Thus, after analyzing existing approaches, let's propose the following DSS implementation based on ML models.

The above scheme has the following features:
- model training – locally;
- DSS implementation as a web application;
- DSS deployment – cloud infrastructure;
- DSS updates – automatically based on cloud services.

Thus, a review of the literature allows to conclude that there are different approaches to creating and implementing systems based on ML models. Each of them has its advantages and disadvantages. However, there is currently no universal approach to the implementation and deployment of ML models in cloud services that would be cost-effective for small institutions and could be scaled to an entire class of DSS tasks.

The object of research is the process of developing and deploying decision support systems using ML models and cloud infrastructure.

The aim of research is to improve the efficiency of DSS based on ML models through a universal approach to local development and deployment in cloud infrastructure.

To achieve this aim, the following *objectives* must be completed:

1. Build ML models based on statistical data.
2. Create a software application for model integration, user data entry, and forecasting.
3. Select a cloud platform, configure cloud services, and develop scripts for automatic deployment.

A set of machine learning models has been selected to build a DSS: logistic regression, support vector machine, Random Forest, and XGBoost. These algorithms are chosen for their high interpretability, which is critical for medical applications, and low computational resource requirements when deployed in containers.

The effectiveness of DSS can be assessed by the accuracy of the ML model and the economic indicators of the system's development and operation.

2. Materials and Methods

Let's consider the proposed concept for implementing the DSS (Fig. 1) using stroke prediction as an example.

According to WHO data [11], Ukraine is among the countries with the highest stroke mortality rates in the world. It is estimated that every year in Ukraine, nearly 130,000 people suffer a stroke, one in five patients dies before being discharged from the hospital, and two-fifths of patients die within one month of the onset of the disease. Worldwide, stroke is the leading cause of neurological disability. Demographic data and physiological indicators are used to predict stroke: gender, age, presence of cardiovascular disease, marital status, and lifestyle [12]. It should be noted that the dataset [12] is selected according to the subject area, however, the approach to deploying DSS with ML models proposed in the paper is universal and can include any other machine learning model.

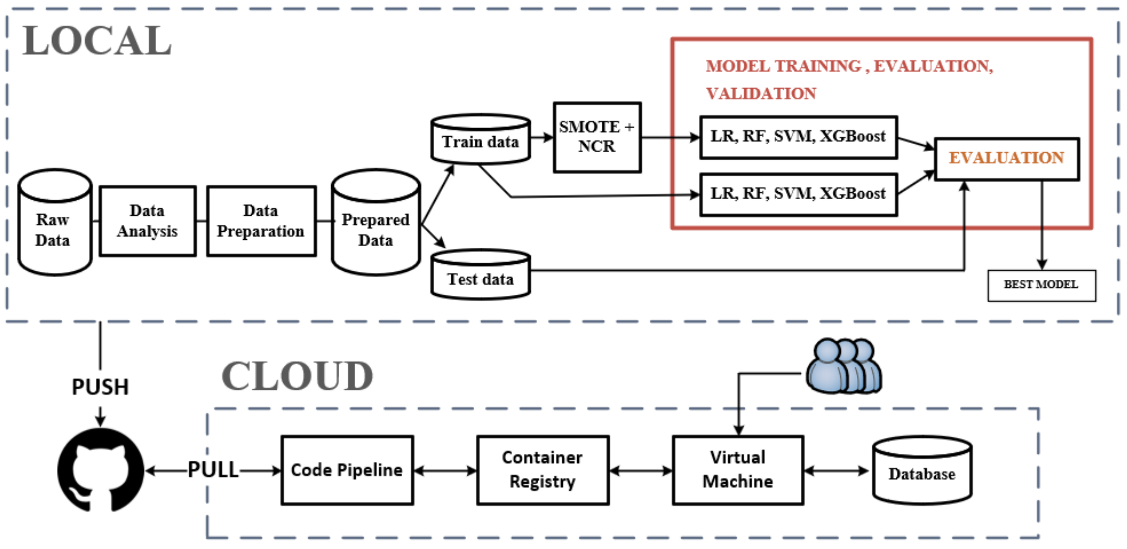


Fig. 1. Decision support system implementation based on machine learning models

The dataset factors [12] influence the choice of modelling methods and contain the following information: gender: "male", "female" or "other"; age of the patient, hypertension: 0 if the patient does not have hypertension, 1 if the patient has hypertension; heart disease: 0 if the patient does not have any heart diseases, 1 if the patient has a heart disease; married: "no" or "yes", work type: "children", "government job", "never worked", "private" or "self-employed"; residence type: "rural" or "urban"; average glucose level in blood; body mass index; smoking status: "formerly smoked", "never smoked", "smokes" or "unknown"; stroke: 1 if the patient had a stroke or 0 if not.

3. Results and Discussion

3.1. Stroke prediction ML models

Considering that the ML model is to be developed locally and later integrated into a web application, it is most appropriate to use the Python environment (Guido van Rossum, Netherlands), which contains all the necessary tools for both machine learning and web development. The first step in creating an ML model is data research: determining its type, identifying missing observations, and analyzing the relationships between factors. Next, the data is examined using visualization (Fig. 2), in particular, histograms and correlation matrices, to understand the relationship between factors.

After analyzing the dataset [12], it is possible to draw the following conclusions: there are missing values for BMI (Body Mass Index), and the dataset is imbalanced (95.13% of cases are non-stroke, while only 4.87% are stroke patients). To address these issues, data augmentation is applied. Missing values for the BMI factors are generated using a regression model based on the *k*-nearest neighbors method (KNeighborsRegressor, scikit-learn (David Cournapeau, France)). Data imbalance is a critical issue in medical tasks, as models tend to learn predicting the majority class, resulting in high accuracy but low recall. It means that the model rarely correctly identifies actual stroke cases. To solve this problem, the minority class ("stroke present") is expanded using the SMOTE (synthetic minority oversampling technique) [13]. Such technique generates a certain number of nearest neighbors of a data point with the same "stroke" label, selects one random neighbor, and places a new synthetic point at a random position along the line segment connecting the original point and its neighbor. As a result, 66.67% (3,889) of the samples belonged to the "no stroke" class, and 33.33% (1,944) are patients with stroke. Subsequently, the neighborhood cleaning rule (NCL) is applied, which removes instances of the majority class ("no stroke") that are located near points of the minority class ("stroke"). This process helps improve the classification algorithms performance. After applying SMOTE combined with NCL, the dataset comprised 63.9% (3,441) "no stroke" instances and 36.1% (1,944) "stroke" instances. It is important to note that the dataset is split into training (4,088 samples) and testing (1,022 samples) sets, and these techniques are applied only to the training data, since subsequent model evaluation must be conducted on real, not synthetic, data. A visual representation of the classes using PCA (principal component analysis) before and after augmentation is shown in Fig. 3.

To create the correlation matrix, the Pearson correlation coefficient is used. The matrix reveals the absence of any significant linear relationship both among the factors themselves and between the factors and the output variable. It allows to conclude that applying linear models for classification is not appropriate. The classifiers selected are logistic regression, support vector machines (SVM), Random Forest, and eXtreme Gradient Boosting (University of Washington group, USA). The creation of these models is illustrated in Fig. 4.

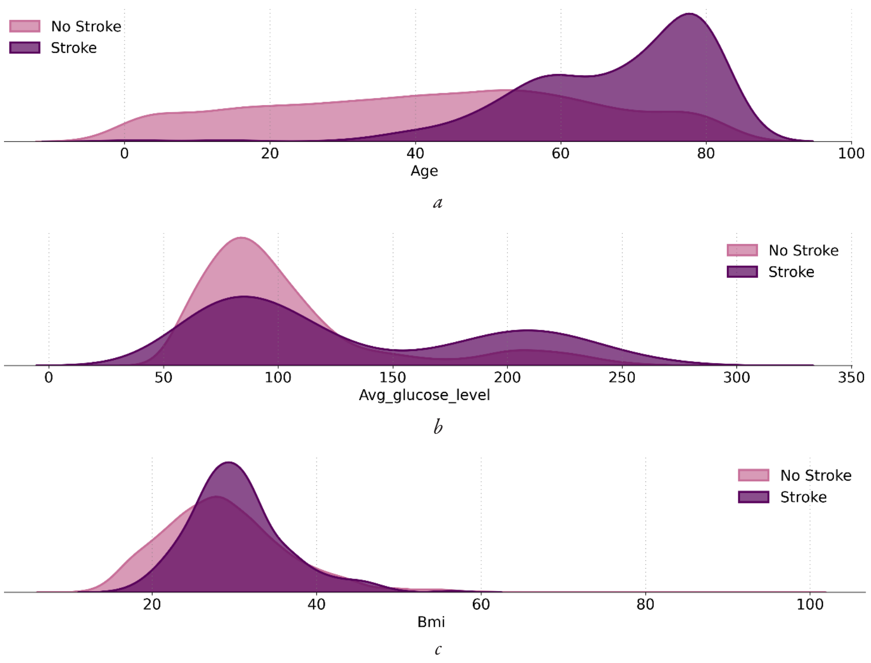


Fig. 2. Impact of individual factors on stroke incidence:
a – age distribution; *b* – glucose level distribution; *c* – BMI distribution

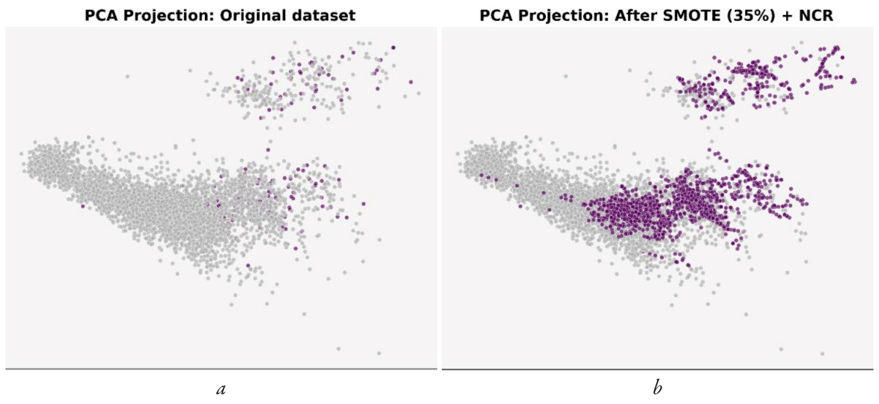


Fig. 3. Augmented dataset visualization: *a* – original dataset; *b* – dataset after resampling

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier

models = {
 "LogReg": LogisticRegression(max_iter=500),
 "RandomForest":     RandomForestClassifier(n_estimators=200,
random_state=42),
 "SVM": SVC(probability=True, random_state=42),
 "XGBoost": XGBClassifier(
                                   random_state=42,
 learning_rate=0.05,
 n_estimators=300,
 max_depth=4,
 subsample=0.9,
 colsample_bytree=0.9,
 eval_metric="logloss",
 )
}
```

Fig 4. Initialization of classification models

The accuracy metrics for the models are calculated on the test dataset. The quantitative values of these metrics are presented in Fig. 5, 6.

Fig. 7 shows the confusion matrices for the logistic regression model (other matrices are not shown due to their similarity) built on the original and augmented datasets.

As noted above, the accuracy metric is not suitable for this task, and it is more appropriate to focus on *recall*, which ensures fewer missed positive cases. This is critical for tasks where missing a positive case is worse than making an extra negative error: missing a sick patient (false negative, FN) is more dangerous than unnecessarily testing a healthy individual (false positive, FP). For further implementation, let's select the logistic regression model. Additionally, GridSearchCV and Stratified K-Fold Cross Validation are performed on each algorithm using both the original and augmented datasets, searching for the best parameters for each model as well as the optimal settings for the SMOTE technique. As a result, a significant improvement can be observed in some models; however, logistic regression remains the best in terms of *recall* (Fig. 8).

Since the primary goal of this paper is to demonstrate and explore the approach to implementing a DSS based on ML models, a model accuracy of 79% and *recall* of 0.8 is acceptable in this context.

Thus, by using data augmentation techniques, it was possible to ensure acceptable model accuracy. At the same time, the rate of truly ill patients' detection (*recall*) is significantly increased.

```
                   model  accuracy  precision  recall        f1   roc_auc    pr_auc
           LogReg (orig)  0.952055   1.000000    0.02  0.039216  0.842243  0.270203
      LogReg (resampled)  0.797456   0.168776    0.80  0.278746  0.843848  0.268299
     RandomForest (orig)  0.950098   0.333333    0.02  0.037736  0.787490  0.149276
RandomForest (resampled)  0.904110   0.157143    0.22  0.183333  0.771914  0.163258
              SVM (orig)  0.951076   0.000000    0.00  0.000000  0.608992  0.118244
         SVM (resampled)  0.819961   0.139785    0.52  0.220339  0.793313  0.188130
          XGBoost (orig)  0.948141   0.285714    0.04  0.070175  0.825144  0.218707
     XGBoost (resampled)  0.902153   0.179487    0.28  0.218750  0.807078  0.200338
```

Fig. 5. Model accuracy metrics after data augmentation

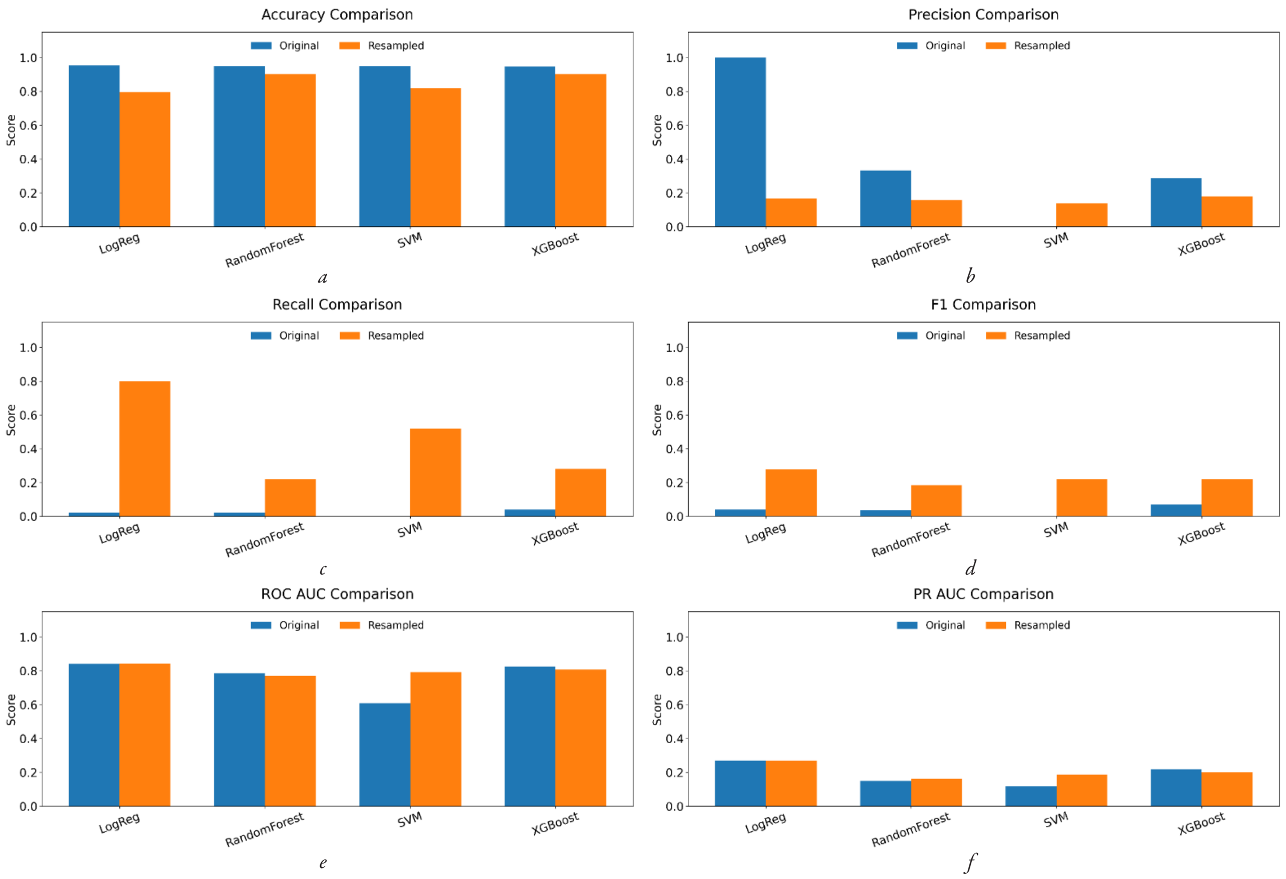


Fig. 6. Model accuracy metrics after CV: *a* – accuracy; *b* – precision; *c* – recall; *d* – F1-score; *e* – ROC-AUC; *f* – PR-AUC

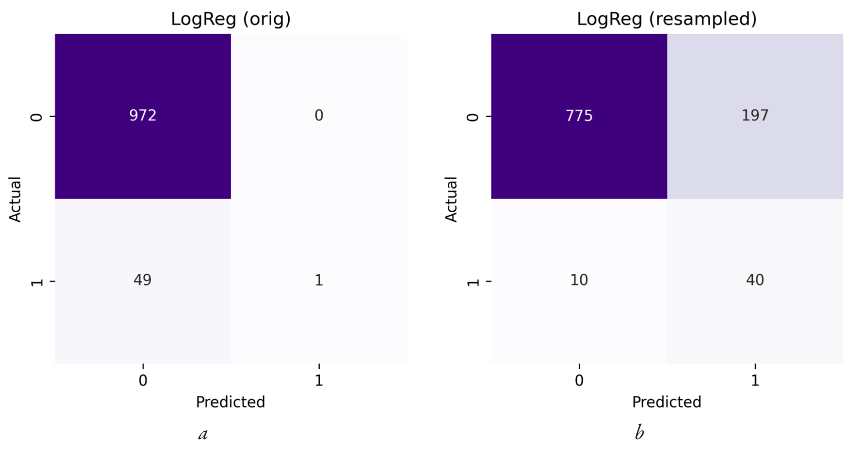


Fig. 7. Confusion matrices for the logistic regression model: *a* – original dataset; *b* – augmented dataset

```
             model  accuracy  precision  recall        f1   roc_auc    pr_auc
0  LogReg_SMOTE_NCR  0.797456   0.168776    0.80  0.278746  0.843848  0.268299
1     SVM_SMOTE_NCR  0.790607   0.155462    0.74  0.256944  0.831317  0.248830
2      RF_SMOTE_NCR  0.867906   0.170543    0.44  0.245810  0.810473  0.183081
3   LogReg_baseline  0.952055   1.000000    0.02  0.039216  0.841605  0.263564
4       RF_baseline  0.950098   0.333333    0.02  0.037736  0.787490  0.149276

5      SVM_baseline  0.951076   0.000000    0.00  0.000000  0.607942  0.118313
```

Fig. 8. Model accuracy metrics after cross validation

3.2. DSS software application

According to the pipeline proposed in Fig. 2, the next step after obtaining the ML model is its integration it into a web application, which essentially constitutes the DSS implementation, the target users interact with. Since the ML model for detecting the stroke presence is developed using machine learning libraries (*scikit-learn*, *scipy*) in the Python programming language, it is natural (as confirmed by a review of previous research) to develop the web application using Python tools.

The first step in integrating the ML model into a web application is packaging, which means converting the model into a form that can be easily used on other systems, typically a binary format. This process is called serialization, and the most commonly accepted formats are Pickle (Python Software Foundation, USA), ONNX (Facebook, Microsoft, USA), and Joblib (Gael Varoquaux, France). In this research, the use of Pickle is proposed. In the training script (Fig. 9), the model is saved as a file, which is then copied into the web application.

```
import pickle
with open('Models/stroke_model.pkl', 'wb') as model_file:
pickle.dump(clf, model_file)
```

Fig. 9. Model serialization using Pickle

To use the model in the web application, it needs to be loaded and called a prediction method (Fig. 10).

For implementing the web interface and API for machine learning models, the most popular Python frameworks are Flask (Pallets Projects, USA) and Django (Django Software Foundation, USA). Both have their advantages and are used depending on the specifics of the project. Flask is noted for its ease of use and minimalism. It provides basic functionality that can be extended according to the project's needs. Django, on the other hand, is a powerful framework for developing complex web applications. It offers many features, including database support, user authentication, and an administrative panel, making it suitable for large projects that require scalability and extensive functionality. For this task, the Flask framework is more appropriate.

```
import pickle
with open(BASE_DIR / "models" / "stroke_model.pkl", "rb") as f:
 STROKE_MODEL = pickle.load(f)
with open(BASE_DIR / "models" / "preprocess.pkl", "rb") as f:
 PREPROCESS = pickle.load(f)

     X = PREPROCESS.transform(df)
     proba = STROKE_MODEL.predict_proba(X)[0, 1]
```

Fig. 10. Importing the serialized model and preprocessor for inference

To facilitate interaction between the web server and the web application implementing the DSS, it is proposed to use the Python standard for web servers – WSGI (Web Server Gateway Interface). In the proposed architecture, the role of the WSGI server is fulfilled by Gunicorn (Benoit Chesneau, France), which runs inside a Docker (Docker Inc., USA) container. When an HTTP request is received from a client, Gunicorn accepts the request, passes it to the WSGI application (a Flask object), and returns the generated response. The user request processing scheme of the web application is shown in Fig. 11. The client browser sends an HTTPS request to the public address of the cloud server. This request reaches the Docker container, where the WSGI server Gunicorn invokes the functions of the Flask application. Those functions call modules for ML inference and modules for integration with cloud storage.

In developing the web application process for the DSS, it is also important to consider the possibility of performance optimization, as discussed by the authors in [14], which allow to use cloud service resources in the future efficiently.

The ML model is integrated into a web application based on the Flask framework. The DSS operation results are shown in Fig. 12.

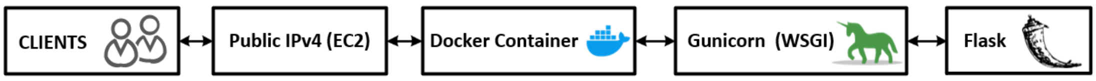
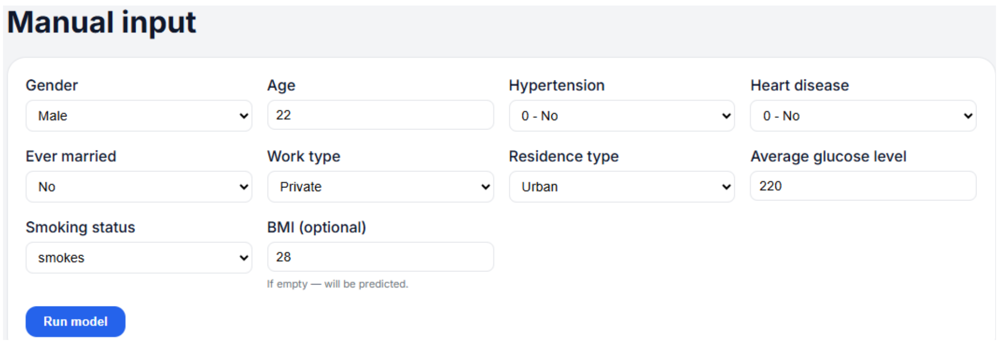


Fig. 11. User request handling scheme

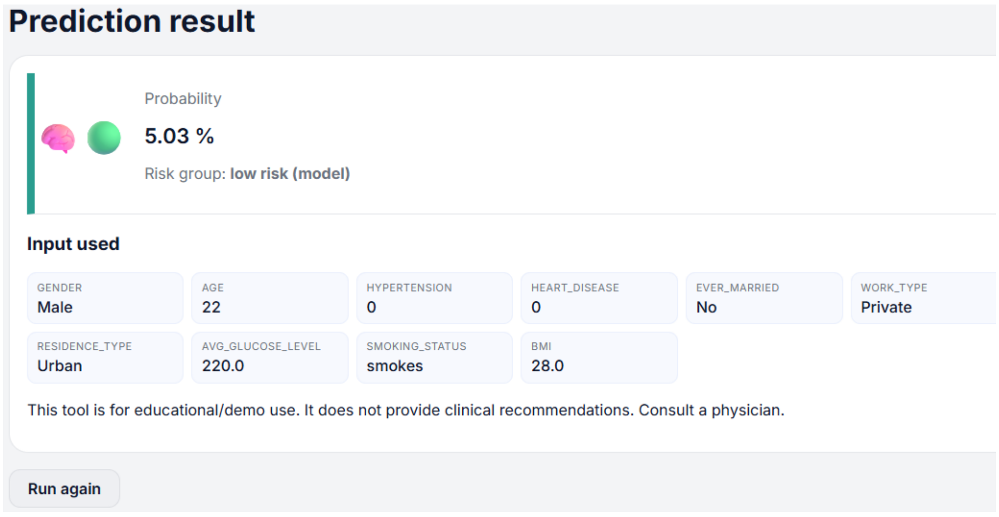


Manual input

Gender	Age	Hypertension	Heart disease
Male	22	0 - No	0 - No
Ever married	Work type	Residence type	Average glucose level
No	Private	Urban	220
Smoking status	BMI (optional)		
smokes	28		

If empty — will be predicted.

a



Prediction result

Probability

5.03 %

Risk group: low risk (model)

Input used

GENDER	AGE	HYPERTENSION	HEART_DISEASE	EVER_MARRIED	WORK_TYPE
Male	22	0	0	No	Private
RESIDENCE_TYPE	AVG_GLUCOSE_LEVEL	SMOKING_STATUS	BMI		
Urban	220.0	smokes	28.0		

This tool is for educational/demo use. It does not provide clinical recommendations. Consult a physician.

b

Fig. 12. Decision support system demonstration in operation: *a* – manual input; *b* – prediction result

The example confirms the operability of the implemented DSS: the system correctly processes user-defined input parameters, performs inference using the serialized ML model, and provides an interpretable prediction result.

3.3. Automatic deployment to cloud infrastructure

A literature review showed that the most practical approach is to deploy ML models in a cloud infrastructure using ready-made services [15]. The choice of cloud platform depends on the specifics of the task, budget, model complexity, and the resources required for its deployment. AWS (Amazon Web Services, USA) and GCP offer the broadest functionality for large projects, while Heroku is better suited for smaller or experimental systems. Microsoft Azure is a good choice for integration with enterprise solutions, while IBM Cloud is well-suited for analytics and business applications [10].

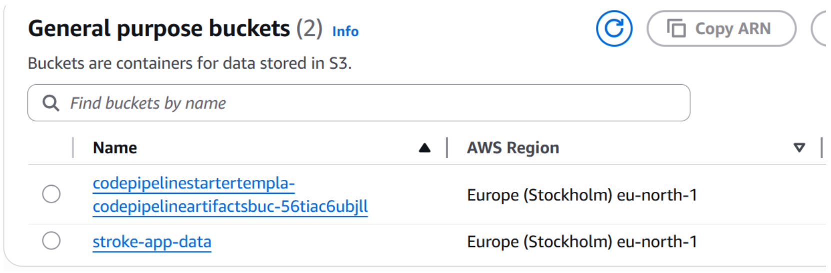
The practical implementation of the DSS is carried out on the AWS platform. However, the proposed architecture is cloud-independent thanks to the use of Docker technology. This allows the system to be easily transferred to Google Cloud or Azure platforms without modifying the software code, if necessary. The choice of AWS for demonstration is due to the availability of advanced automation services that allow optimizing infrastructure costs within the Free Tier.

ML models are often used by government institutions, small private companies, or startup developer teams that face significant financial constraints. Note that it is possible to use a cheaper cloud solution during development, testing, and demonstration, and a more expensive tariff plan during mass launch. In such cases, it is preferable to deploy the model on a cloud service with minimal or no cost. At the same time, performance metrics, software availability, and reliable data and personal information security cannot be compromised. Given these requirements, the authors consider AWS EC2 the most appropriate solution. Its main advantage lies in the ability to create a virtual server with customizable parameters (processor types, memory size, operating system), allowing precise alignment with project needs. On EC2, a Docker daemon (a background process managing Docker objects) is deployed, which runs a container hosting the web application. This container includes the entire necessary stack: a trained ML inference module, a Flask application, the WSGI server Gunicorn, and supporting libraries.

Amazon S3 object storage service is used to store user requests and forecasting results. The web application writes JSON files containing anonymized session data both to the local container directory and to an S3 bucket, ensuring reliable long-term storage of results and enabling further analysis. Access to S3 is configured through an IAM role assigned to the EC2 instance, adhering to the principle of least privilege (Fig. 13).

Access to external services or hidden variables, when needed, is managed via AWS Secrets Manager. All secrets are stored centrally in encrypted form and are not included in the source code or repository configuration files. During deployment, a special script on the EC2 instance reads the secret values through the AWS SDK and then passes them into the Docker container as environment variables. Inside the container, the Flask application simply reads these variables without having direct access to Secrets Manager.

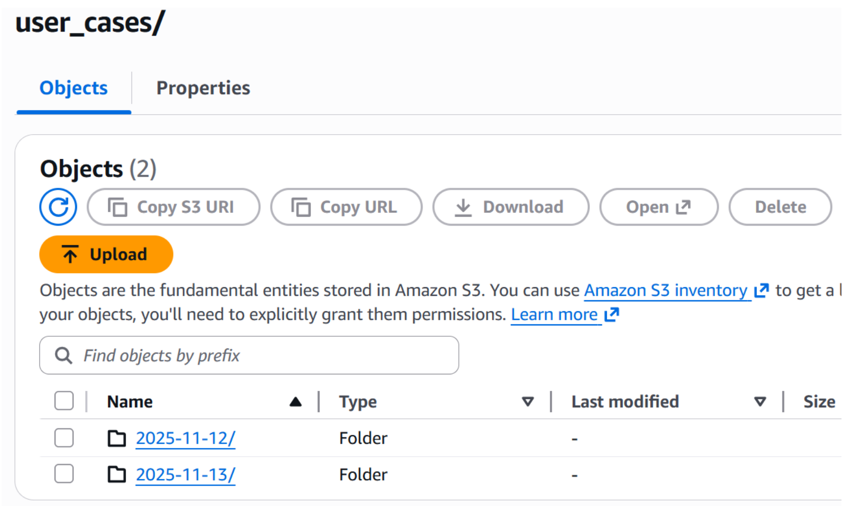
To ensure a reproducible and automated container build process, the Amazon Elastic Container Registry (ECR) and AWS CodeBuild services are used. The web application's source code is stored in a GitHub repository. With each code update (push to a specified branch), an AWS CodePipeline pipeline is triggered. In the first stage, CodePipeline retrieves the latest version of the code from GitHub (GitHub Inc., USA). In the second stage, it passes the code to CodeBuild, which, based on the Dockerfile, builds the Docker image and uploads it to ECR (Fig. 14). This ensures that the most recent version of the container with the updated DSS implementation is always available in ECR (Fig. 15).



Buckets are containers for data stored in S3.

Name	AWS Region
codepipelinestartertempla-codepipelineartifactsbuc-56tiac6ubjll	Europe (Stockholm) eu-north-1
stroke-app-data	Europe (Stockholm) eu-north-1

a

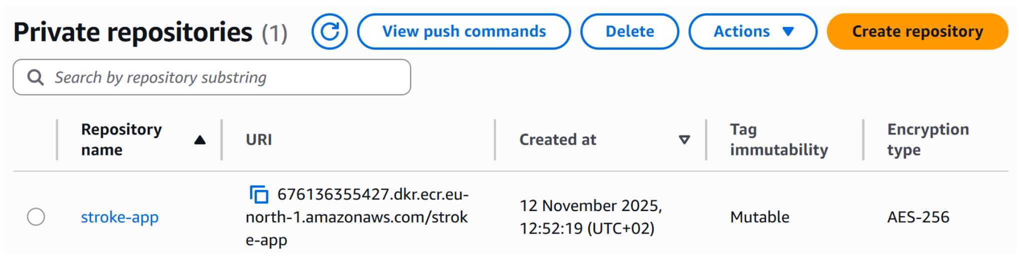


Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a l
your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size
2025-11-12/	Folder	-	
2025-11-13/	Folder	-	

b

Fig. 13. S3 buckets: *a* – buckets; *b* – objects



Repository name	URI	Created at	Tag immutability	Encryption type
stroke-app	676136355427.dkr.ecr.eu-north-1.amazonaws.com/stroke-app	12 November 2025, 12:52:19 (UTC+02)	Mutable	AES-256

a

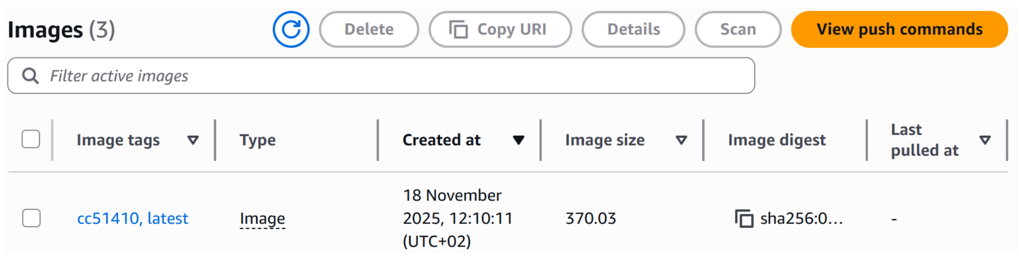
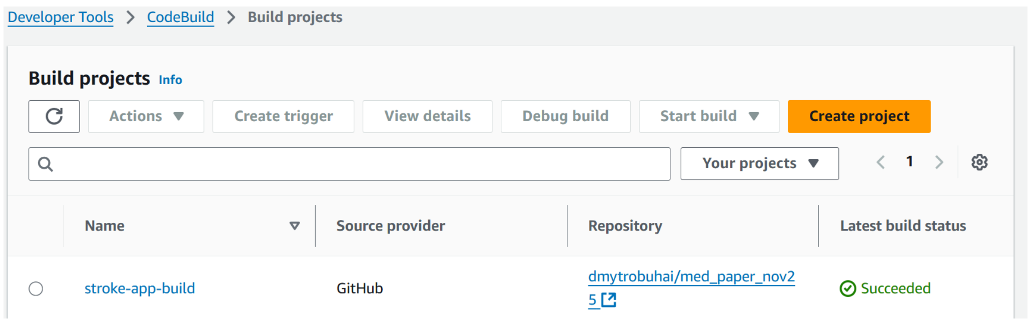


Image tags	Type	Created at	Image size	Image digest	Last pulled at
cc51410, latest	Image	18 November 2025, 12:10:11 (UTC+02)	370.03	sha256:0...	-

b

Fig. 14. Elastic container registry: *a* – private repositories; *b* – images



Name	Source provider	Repository	Latest build status
stroke-app-build	GitHub	dmytrobuhai/med_paper_nov25	Succeeded

Fig. 15. CodeBuild

After successfully completing the Docker image build stage in CodeBuild, CodePipeline moves on to the deployment stage (Fig. 16). At this stage, an AWS Lambda function is called, that acting as an intermediary between the CI/CD pipeline and the EC2 instance. The Lambda function has access to the EC2 instance identifier and the command to run the deploy script. It requests to the AWS Systems Manager (SSM) service and sends an AWS-RunShellScript command, which executes the deployment script on the EC2 instance. The deploy script sequentially performs the following actions:

– determines the AWS region and account;
– checks the availability of the S3 bucket;
– reads the secret value from AWS Secrets Manager;
– authenticates with ECR;
– loads the latest Docker image of the web application;
– stops the previous container;
– starts a new container instance with the updated code.

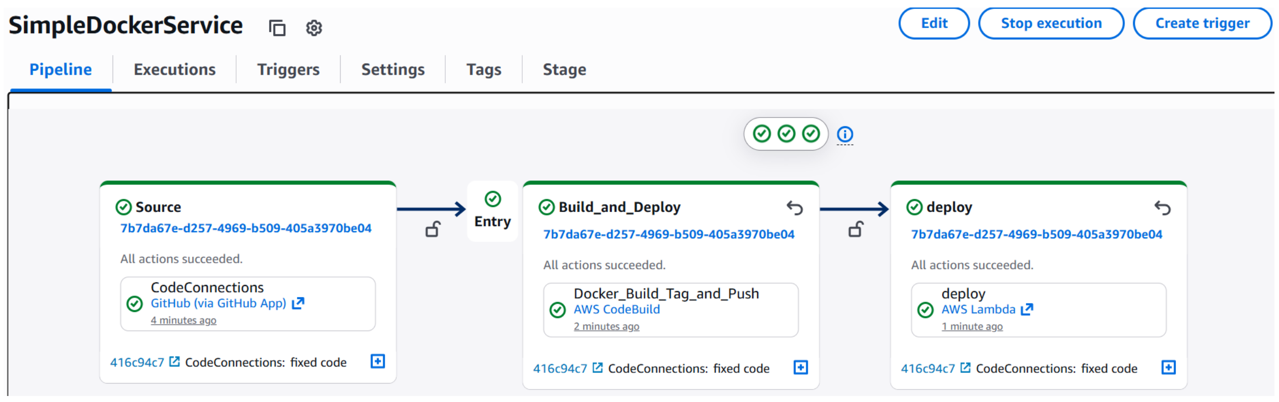
All necessary parameters such as the bucket name, secret identifier, server port, etc. are passed to the script as constants or environment variables.

This approach ensures a fully automated DSS update: after a code change in the GitHub repository, the system independently builds a new container, publishes it to ECR, and deploys it on an EC2 instance without any manual developer intervention. Throughout this process, all activities are logged in Amazon CloudWatch and AWS CodePipeline services, allowing for monitoring the status of builds, deployments and respond in a timely manner to possible errors. The overall implementation scheme is shown in Fig. 17.

The described architecture combines the flexibility of local ML model development with the reliability and scalability of cloud infrastructure, providing automated updates and continuous availability of the web-oriented DSS for stroke prediction.

3.4. Discussion

This paper demonstrates the development process of a DSS featuring a human-machine interface in the form of a web application that includes an ML model. The cloud service stack used for its deployment and automatic updating is presented.


SimpleDockerService

Edit | Stop execution | Create trigger

Pipeline | Executions | Triggers | Settings | Tags | Stage

Source
7b7da67e-d257-4969-b509-405a3970be04
All actions succeeded.
CodeConnections
GitHub (via GitHub App)
4 minutes ago
416c94c7 CodeConnections: fixed code

Entry

Build_and_Deploy
7b7da67e-d257-4969-b509-405a3970be04
All actions succeeded.
Docker_Build_Tag_and_Push
AWS CodeBuild
2 minutes ago
416c94c7 CodeConnections: fixed code

deploy
7b7da67e-d257-4969-b509-405a3970be04
All actions succeeded.
deploy
AWS Lambda
1 minute ago
416c94c7 CodeConnections: fixed code

Fig. 16. CodePipeline

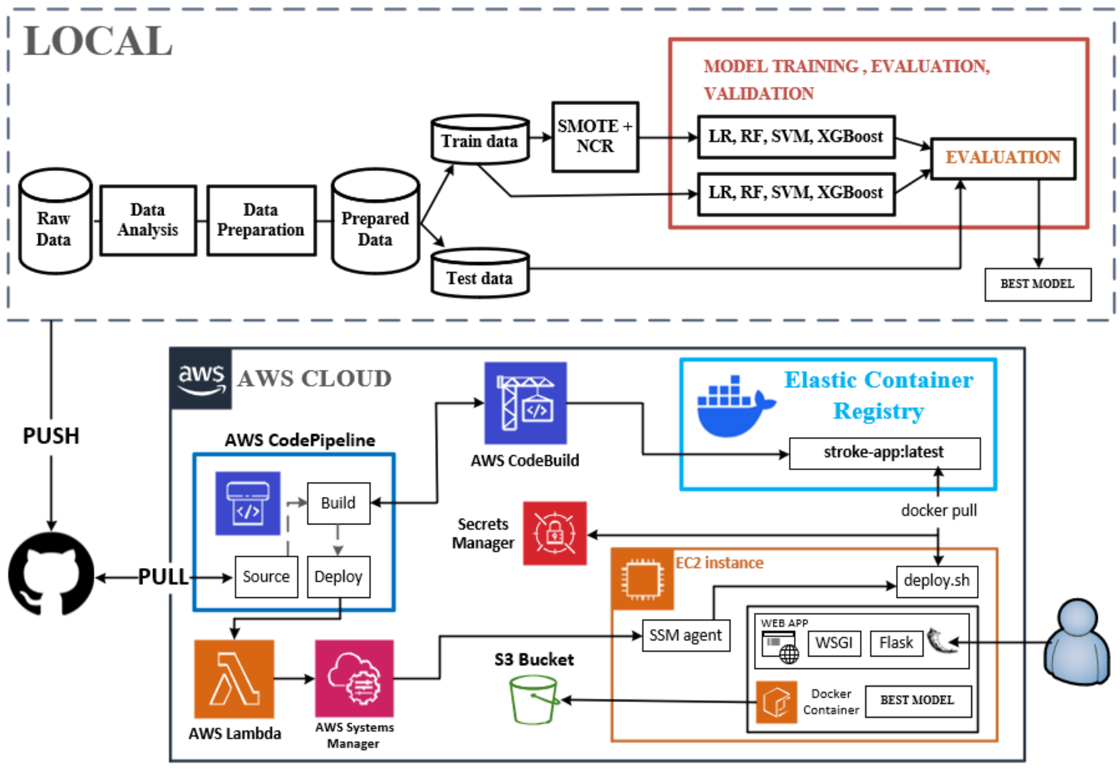


Fig. 17. Overall implementation scheme of the DSS

Using a stroke prediction dataset, a complete model-building cycle was conducted: data analysis, sample preparation, missing value augmentation, class balancing (using SMOTE and NCR), classifier training, and performance comparison. Based on the recall metric (the most critical in medical applications), the logistic regression model was selected. The model development and its integration into the web application were performed locally using the Python programming language.

For deploying the DSS in a production environment, AWS cloud infrastructure was utilized. The application is containerized using Docker, with its build and deployment organized according to CI/CD principles. AWS CodePipeline monitors changes in the GitHub repository, CodeBuild automatically creates the Docker image and publishes it to Amazon Elastic Container Registry (ECR), after which AWS Lambda initiates the container update on an EC2 instance via AWS Systems Manager. For secure access to external services, Secrets Manager is used, while Amazon S3 stores the results of user interactions.

Unlike typical approaches oriented at large corporations, a pipeline specifically adapted to the limited financial and personnel resources of small companies and public medical institutions is proposed. This allows to provide industrial-level reliability without creating a separate DevOps team.

The originality of this work lies in combining a "lightweight" single-node cloud architecture – one EC2 instance with a containerized DSS – with a full-fledged access and secrets management scheme, specially tailored for medical decision support systems. A template is proposed and implemented in which:

– anonymized user session data is stored in S3 through an EC2 IAM role following the least privilege principle;
– all external keys and confidential parameters are passed into the container exclusively via Secrets Manager and environment variables, without being hardcoded in the code or Docker image.

This approach demonstrates how basic security requirements for a medical decision DSS can be met using publicly available AWS services without deploying heavy enterprise solutions.

The obtained results have practical significance for creating real-time DSS and do not require complex infrastructure. The proposed approach allows to use locally created ML models in a cloud environment with automatic updates, interaction logging, and secure management of confidential parameters.

The proposed implementation scheme can be extended to other medical tasks and used in areas with similar requirements: banking systems, equipment status monitoring systems, automated quality control, etc. Thanks to low operating costs and the CI/CD approach, the solutions are suitable for institutions with limited budgets.

The implementation of the proposed CI/CD approach allows for improved economic, infrastructural, and operational metrics. Deployment Time: thanks to the complete automation of container assembly, the deployment time for new model versions has been reduced from several hours to 10–15 minutes. This saves approximately 15 to 20 man-hours per month in the active phase of DSS development. The effectiveness is confirmed by the stability of the Uptime system (99.9%), which is minimized in the AWS cloud environment thanks to automatic instance restart mechanisms.

Through a web interface, the user inputs clinical parameters, after which the system normalizes the features, queries the ML model, and returns a stroke risk assessment. The main advantages of this demonstrated approach are economic efficiency – since some costs are saved during the local development phase – and reliability, thanks to the use of cloud services. Additionally, the proposed solution for automatic DSS updates ensures independence from external specialists while maintaining full control over the project's code.

At the same time, the proposed approach has several limitations related to the DSS architecture and the integration principles with the cloud infrastructure. The system is deployed on a single-node architecture (one EC2 instance), which provides sufficient functionality for small medical institutions. In case of a need for scaling due to an increase in the number of requests, it is necessary to implement balancing mechanisms, container replication, and automatic recovery in case of failures.

The DSS further development can proceed along several directions. Firstly, a promising avenue is the creation of modules for automatic import of clinical data through integration with electronic medical information systems, which would eliminate the need for manual parameter entry. In this scenario, the DSS could operate not only via a human-machine interface but also through direct use of REST-API by external medical systems. Secondly, the model's functionality could be expanded – for example, by adding an assessment of the factors that most influence the prognosis or by applying additional algorithms to enhance the model's sensitivity for rare events.

4. Conclusions

1. During the research, a complete cycle of data preparation for stroke risk prediction is carried out: elimination of gaps using the nearest neighbor method, sample balancing using SMOTE + NCL, and analysis of correlation relationships. Tests of four classification algorithms showed that after data augmentation, the logistic regression model was the best in terms of recall (0.80). The quantitative results obtained, with an accuracy of 79% and ROC AUC of 0.84; confirm that the model is suitable for use in DSS as a primary screening tool.

2. Based on the trained model, a web application is implemented that provides input of indicators, their preliminary processing and real-time forecasting. The model is integrated through serialization (Pickle) and an inference module built into the Flask application. As a result, a functional human-machine interface suitable for practical use in medical institutions is obtained.

3. The feasibility of using AWS as a cloud environment for deploying DSS is justified. Architecture based on EC2, S3, ECR, and Secrets Manager is selected, which ensures secure parameters storage, scalability, and the ability to use minimal resources. This approach allows industrial DevOps practices to be adapted to budgetary or government institutions. An automated CI/CD process is implemented, including CodePipeline, CodeBuild, ECR, and container updates on EC2 via AWS Systems Manager. After each code update in GitHub, a new Docker image is created on the server, automatically replacing the previous one. It minimizes manual intervention, reduces the error risk, and ensures the reproducibility of DSS operation in the production environment.

Conflict of interest

The authors declare that they have no conflict of interest in relation to this research, whether financial, personal, authorship or otherwise, that could affect the research and its results presented in this paper.

Financing

The research was performed without financial support.

Data availability

The manuscript contains data included as supplementary electronic material [12].

Use of artificial intelligence

The authors confirm that they did not use artificial intelligence technologies when creating the current work.

Authors' contributions

Dmytro Buhai: Software, Validation, Investigation; ***Anatolii Zhuchenko***: Conceptualization, Methodology; ***Oleksii Zhuchenko***: Formal analysis, Methodology, Writing – original draft; ***Dmytro Kovaliuk***: Conceptualization, Methodology, Software, Validation; ***Denys Skladannyy***: Methodology, Validation, Writing – review and editing.

References

1. Kubat, M. (2017). *An Introduction to Machine Learning.* Springer International Publishing, 348. https://doi.org/10.1007/978-3-319-63913-0
2. Shalev-Shwartz, S., Ben-David, S. (2014). *Understanding Machine Learning.* Cambridge University Press, 449. https://doi.org/10.1017/cbo9781107298019
3. Chowdary, M. N., Sankeerth, B., Chowdary, C. K., Gupta, M. (2022). Accelerating the Machine Learning Model Deployment using MLOps. *Journal of Physics: Conference Series, 2327 (1),* 012027. https://doi.org/10.1088/1742-6596/2327/1/012027
4. Paleyes, A., Urma, R.-G., Lawrence, N. D. (2022). Challenges in Deploying Machine Learning: A Survey of Case Studies. *ACM Computing Surveys, 55 (6),* 1–29. https://doi.org/10.1145/3533378
5. Treveil, M., Omont, N., Stenac, C., Lefevre, K., Phan, D., Zentici, J. et al. (2020). *Introducing MLOps How to Scale Machine Learning in the Enterprise.* O'Reilly Media, Inc. Available at: https://itsocial.fr/wp-content/uploads/2021/04/Comment-mettre-%C3%A0-l%E2%80%99%C3%A9chelle-le-Machine-Learning-en-entreprise.pdf
6. Heymann, H., Kies, A. D., Frye, M., Schmitt, R. H., Boza, A. (2022). Guideline for Deployment of Machine Learning Models for Predictive Quality in Production. *Procedia CIRP, 107,* 815–820. https://doi.org/10.1016/j.procir.2022.05.068
7. Corbin, C. K., Maclay, R., Acharya, A., Mony, S., Punnathanam, S., Thapa, R. et al. (2023). DEPLOYR: a technical framework for deploying custom real-time machine learning models into the electronic medical record. *Journal of the American Medical Informatics Association, 30 (9),* 1532–1542. https://doi.org/10.1093/jamia/ocad114
8. Heymann, H., Schmitt, R. H. (2023). Machine Learning Pipeline for Predictive Maintenance in Polymer 3D Printing. *Procedia CIRP, 117,* 341–346. https://doi.org/10.1016/j.procir.2023.03.058
9. Kozak, Ye. B. (2021). Data Analysis and Machine Learning in Cloud and Fog Platforms as a Basis for Efficient Data Transfer. *Scientific Notes of Taurida National V. I. Vernadsky University. Series: Technical Sciences, 5,* 100–107. https://doi.org/10.32838/2663-5941/2021.5/16
10. Panchenko, T., Tuzova, I., Tuzov, O., Chumak, O. (2024). Khmarni servisy ta ohliad yikh postachalnykiv. *InterConf, 43 (193),* 550–559. https://doi.org/10.51582/interconf.19-20.03.2024.053
11. Report on Situational Analysis Results of Acute Stroke Care in Ukraine (2024). *World Health Organization.* Available at: https://www.who.int/ukraine/publications/WHO-EURO-2024-9677-49449-73972
12. Stroke Prediction Dataset. *Kaggle Inc.* Available at: https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset
13. SMOTE. *Imbalanced-learn.* Available at: https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html
14. Kovaliuk, D. O., Kovaliuk, O. O., Pinaieva, O. Y., Kotyra, A., Kalizhanova, A. (2019). Optimization of web-application performance. *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2019*, 210. https://doi.org/10.1117/12.2537163
15. Singh, C. (2023). Automate deployment (CI/CD) of React JS application in AWS by using CodePipeline and EBS. *L&G Consultancy.* Available at: https://lng-consultancy.com/automate-deployment-ci-cd-of-react-js-application/

Dmytro Buhai, *Automation Hardware and Software Department, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine, ORCID: https://orcid.org/0009-0001-0112-1432*

Anatolii Zhuchenko, *Doctor of Technical Sciences, Professor, Automation Hardware and Software Department, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine, ORCID: https://orcid.org/0000-0002-1552-8372*

Oleksii Zhuchenko, *Doctor of Technical Sciences, Professor, Automation Hardware and Software Department, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine, ORCID: https://orcid.org/0000-0001-5611-6529*

Dmytro Kovaliuk, *PhD, Associate Professor, Automation Hardware and Software Department, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine, ORCID: https://orcid.org/0000-0002-9729-1443*

✉***Denys Skladannyy,*** *PhD, Associate Professor, Automation Hardware and Software Department, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine, e-mail: skl_den@ukr.net, ORCID: https://orcid.org/0000-0003-3624-5336*

✉*Corresponding author*