ISSN 2224-087X. Електроніка та інформаційні технології. 2022. Випуск 17. С. 74–83
Electronics and information technologies. 2022. Issue 17. P. 74–83

УДК 519.85.004.42 DOI: https://doi.org/10.30970/eli.17.7

MLOPS BASED PROTOTYPE OF AI SYSTEM FOR EDGE COMPUTING

O. Sinkevych, Ya. Boyko, L. Monastyrskyy

Ivan Franko National University of Lviv,
50 Drahomanova str., Lviv, UA-79005, Ukraine
oleh.sinkevych@lnu.edu.ua

The paper represents an approach for developing the edge AI system which is based on the modern MLOps concept. For the edge hardware part we utilize Nvidia Jetson Nano microcomputer which provides server side for network requests processing, data storage and machine learning models of self-deployment. We propose the working MLOps pipeline fully designed by the industrial software solutions like TensorFlow 2, Mlflow, Apache Airflow, which is integrated into the developed application. The considered pipeline scheme consists of three operational stages: a) data storage and processing, which stands for fetching the data from database, cleansing and transformation; b) machine learning modeling with synchronous hyper-parameters optimization and model registration; c) model deployment and serving. The whole pipeline is wrapped by the REST API created via FastAPI micro-framework and orchestrated using Apache Airflow service. To implement the described pipeline we chose the time dependant temperature data to be learned and short-term predicted by the GRU-based recurrent neural network. The latter one is tuned in terms of hyper-parameters configuration by genetic algorithm which is embedded into the second stage of the pipeline. Also, a design which combines Nvidia Jetson Nano server with the inference edge device like STM32 H745 microcontroller via sockets is discussed.

Key words: edge computing, MLOps, machine learning, Mlflow, genetic algorithm.

Introduction

Despite the active development of AI cloud based solutions, the edge/fog computations are gaining popularity in many of industrial and academic topics. The reasons for this phenomenon are comprehensively discussed in publications [1-5], among which we highlight the autonomy and isolation. Such features of the boundary calculations allow to design and investigate the productivity and efficiency of hardware and software systems in the local environment without binding of the paid cloud servers. Quite an important step in the development of software, in particular, for the edge computing which uses machine learning, is to build the right automatic pipeline. That pipeline typically includes data ingestion and storage with the sequential processing, utilization of the machine learning models as well as model deployment, saving and monitoring stage. Unfortunately, by now there are not many papers devoted to the edge MLOps technologies. Nevertheless, an interesting example has been presented in [6], where the authors developed an Edge MLOps framework for automating machine learning at the edge, enabling continuous model training, deployment, delivery and monitoring. Also, in [7] the authors proposed the detailed Digital Twin MLOps architecture for

© Sinkevych O., Boyko Ya., Monastyrskyy L., 2022

personalized demand response suggestions based on online short-term energy consumption prediction.

Since this topic is quite new, in this paper we propose our own MLOps pipeline scheme. As the main edge device, we use Nvidia Jetson Nano microcomputer with a built-in 128-core GPU. To implement the MLOps pipeline, we have applied technologies such as SQLite for the data storage; TensorFlow 2 for the machine learning modelling; Mlflow for the model registration and serving; Apache Airflow for the data fetching – model training – model deployment orchestration. The complete process, consisting of interaction of these technologies and steps, is controlled through the REST API, created by the FastAPI micro-framework. As data for testing our pipeline, we chose the temperature measured in the Laboratory of Intelligent Autonomous Systems of the Faculty of Electronics and Computer Technologies, Ivan Franko National University of Lviv, Ukraine. The resulting model (GRU recurrent neural network) which is able to predict temperature data based on the studied patterns in the previous measurements and can be easily deployed on STM32 microcontroller via the client-server sockets [8]. The last step defines more complex twin edge computing system, where Nvidia Jetson Nano serves as the main computational hub while STM32 microcontroller runs the inference.

Hardware architecture

The Nvidia Jetson Nano microcomputer (fig. 1), on the basis of which the hub is implemented, has the following characteristics: 128 cores NVIDIA Maxwell™ GPU, quad-core ARM® A57 1479 MHz and 4 GB 64-bit LPDDR4 operational memory with a reading speed of 25.6 gigabytes / second.

Fig. 1. Hardware components: Nvidia Jetson Nano and STM32 H745 microcontroller

This provides sufficient computing resources both to run the web server and to train simple (single-layer) recurrent neural networks due to the presence of 128 core GPU.

STM32H745 is built on a dual-core circuit (32-bit 480 MHz Cortex-M7 and 240 MHz Cortex-M4) and has 1 MB of SRAM and 2 MB of flash memory. It is able to work in different power modes.

The Nvidia Jetson Nano connects to the STM32 H745 microcontroller through a LAN switch. Also, if needed, the current system can be accessed via global Internet. If there is a separate Wi-Fi module on the hub, the connection between the microcontroller and the hub is made directly via an Ethernet network cable without an intermediate switch. To test the inference made by microcontroller with the deployed model, the temperature sensors, e.g.,

DS18B20 can be connected. The general hardware scheme of the proposed system is shown in fig. 2.

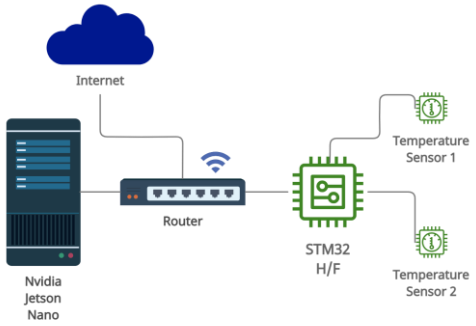

Fig. 2. Scheme of the hardware connection

Due to the modularity, the proposed hardware system can be easily expanded with the additional microcontroller equipment and supplemented with more sensors. In addition, access to the Internet allows one to develop the mobile services to manage the operation of this system, namely through the creation of mobile applications and integration with other remote IoT components.

Software architecture

The Nvidia Jetson Nano microcomputer runs on Linux4Tegra operating system based on Ubuntu 18.04 and uses the Nvidia JetPack SDK, which provides access to the CUDA-X graphics acceleration library. This library provides support for TensorRT and cudaDNN technologies used in machine learning to develop and train deep neural networks using the frameworks such as TensorFlow, PyTorch, MxNet and others.

The software implementation of the presented system is based on a two-component architecture: a) server side for query processing and monitoring the state of the deployed machine learning model; b) client-microcontroller STM32 H745 which interacts with a running server using TCP sockets to transfer data and update model files after monitoring its current state (fig. 3).

Server side of the system is designed using FastAPI [9] micro-framework which is based on the asynchronous web server Uvicorn. A FastAPI-based API's binds client requests, which income to special route functions (MVT pattern), with the event handlers defined by the functional logic of the developed API. With the use of TCP sockets, the requests are sent and processed by both server and client (microcontroller STM32 H745).

If data drift is detected, or the specified model accuracy is lost (TF model), the server can update the neural network model (*update_model*) deployed on the microcontroller. In turn, after processing the temperature data coming from the connected sensors (Temperature Sensor1 / 2), the client microcontroller sends these data to the server (*send_data*) for further processing and storage in the database. On the server side, such real-time communication is provided by a bidirectional transport protocol implemented by the sockets supporting library.

In addition to the microcontroller client, this architecture allows one to extend server users in the context of adding other devices (sending system or data notifications to the smart

phones, adding other sensors and handlers, etc.) and to scale the proposed system in order to improve it.

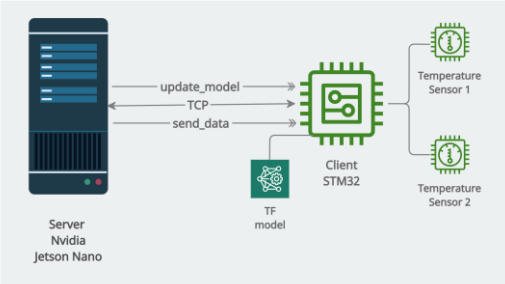

Fig. 3. Client-server connection

Server side details

For the developing the server the side preference was given to the FastAPI framework. The choice of the optimal framework among such popular web server design solutions as Django, Flask, FastAPI, Pyramid was made for the following reasons:

- deployment of the server on the edge computing device with limited computing resources;
- availability of the comprehensive documentation;
- availability of libraries to expand the functionality and interaction with the STM32 microcontroller;
- support for asynchronous programming, ASGI and the RESTful concept.

At present, testing the performance of these software frameworks demonstrates the advantage of FastAPI over competitors [10], what is a strong argument in terms of limit calculations. Also, since machine learning tasks in the context of edge computing may require significant computational time, it is necessary to ensure the possibility of asynchronous processing of other requests coming to the server. This is fully implemented by the basic FastAPI tools and thanks to the simplicity and convenience speeds up the prototyping of application software interfaces. FastAPI can handle both synchronous and asynchronous requests and has built-in support for data validation, JSON serialization, authentication and authorization.

The main architectural template for the design of the server part is the Model-View-Template (MVT), which covers the developed classes and their structuring in a three-step pipeline (fig.4-5).

In fig. 4 the conceptual MVT diagram of the application is shown. The MODEL is an abstraction that defines the object representation of data stored in a SQLite database. Such data are temperatures obtained as a result of measurements. In order to create a fast and working prototype of the edge system, the data model was limited to one temperature table, but can be easily supplemented with other objects, such as user classes and climatic indicators, if necessary. View (VIEW) is the implementation of the interaction of user or internal queries with the database, which involves performing CRUD (create / read / update / delete) actions with temperature data and contains calls to basic computational operations from the flowchart of fig. 5.

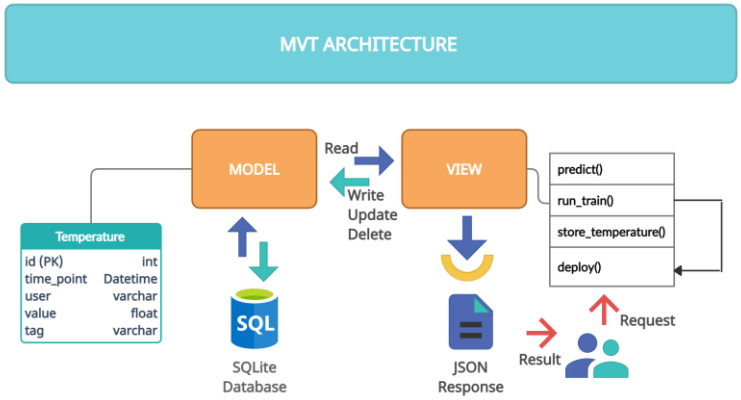

Fig. 4. MVT architecture scheme

The route functions defined in this block are responsible for the following operations:

- *predict()* predicts the temperature by the model deployed on the server for a given number of steps;
- *run_train()* starts the full process of training the neural network model with automatic selection of hyper-parameters by applying a genetic algorithm and;
- *store_temperature()* using network socket technology processes a TCP request from the microcontroller, which contains the processed temperature data, and stores it in the database;
- *deploy()* implements the embedding of the prepared neural network model in the microcontroller through a function call.

The results of routing functions are returned in JSON format, which can be embedded in the HTML page.

MLOps pipeline

To create a synchronized three-step process of data processing, model training and its deployment on the server we use the approaches of the modern concept of MLOps, which aims to flexibly implement experimental machine learning models in the production system. The need to implement such a process for the proposed system is caused by the constant accumulation and change of temperature data, on which the trained deployed model depends. Therefore, the neural network model needs to be retrained and re-adapted to new data while the system is running without stopping.

The project configuration and local environment settings should be specified in the corresponding .yaml, .json or .env file.

In fig. 5, it is shown the lifecycle of the deployment workflow. The first component of data processing and preparation contains the following modules:

- *collect_data* – implements interaction with the function of the microcontroller to save the obtained temperature measurements in the database;

- *get_data* – contains functions for obtaining data from SQLite, saving intermediate processed arrays with temperature data in the local directory of the project, which are used to train neural network models;
- *transform* – determines the class-handler for normalization, reduction to three-dimensional tensors (input format for the GRU recurrent neural network) and splitting of the array of temperature data into training and test samples;
- *store_local* – saves intermediate three-dimensional temperature arrays (training and test samples) in the local directory defined in the project configuration file.

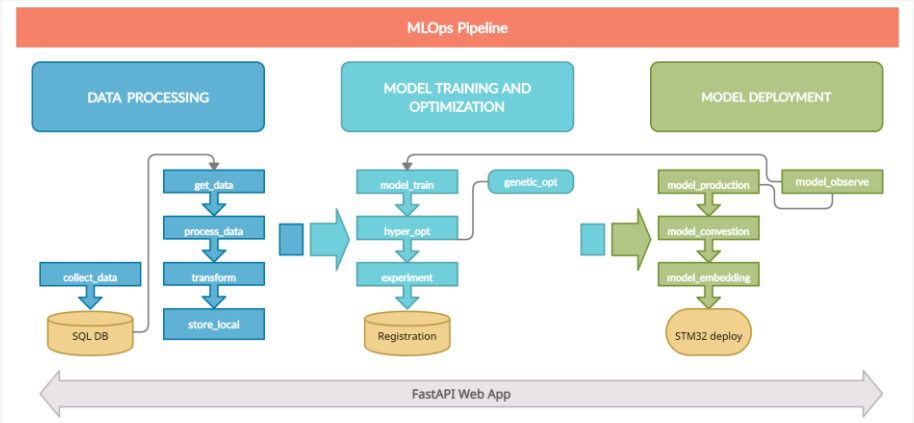

Fig. 5. MLOps pipeline

The second component of the fig. 5 consists of the following modules:

- *model_train* – contains functions for starting training of neural network models and estimating accuracy, creating models using the TensorFlow 2 and launching the Mlflow service for registration, saving in SQLite (default), tracking versions and parameters of models;
- *hyper_opt* – optimizes the hyper-parameters of neural network models, contains a class for calculating the hyper-parameters of single-layer recurrent and hybrid neural networks based on the genetic algorithm;
- *genetic_opt* – file with the implementation of the genetic optimization algorithm based on the DEAP [11] library;
- *experiment* – an auxiliary unit consisting of Jupyter Notebook files for conducting numerical experiments and model research in an interactive mode.

The last component – model deployment and monitoring is responsible for the direct launch of trained and configured models at the server level, which are ready for use on user requests. Also, the status of models and new data is monitored, based on which retraining can take place with subsequent re-deployment. It contains the following modules:

- *model_production* – deploys the model on the server locally using Mlflow Model technology, a standard format for packaging machine learning models, which can be used in various tools, such as real-time query service via REST API; saves the model to local directory in TensoFlow.keras h5 format;

- *model_conversion* – converts the model from HDF5 / h5 format to TensorFlow Lite format, which allows one to transform the model to a C-object and save it to a file with extension using the Linux command line utility *xxd*; also at this stage it is possible to quantize the model using TensorFlow Lite tools;
- *model_embedding* – embeds the model saved in C-object format in the STM32 H745 microcontroller using the sockets;
- *model_observe* – monitors the model; Evidently AI tool [] can be used to monitor new data for data drift and to evaluate the accuracy of model prediction.

The Apache Airflow platform for managing data workflows [13], which are defined by oriented acyclic graphs (DAG), is used to run each of the system components sequentially. This allows one to track the performance of each task: data processing and preparation, training neural network models with subsequent registration by Mlflow tools and obtaining the best model for its deployment both on the local server and on the microcontroller.

To achieve the sequence of the training process of neural network models, a corresponding DAG file was created, which starts the components and the corresponding modules from the pipeline in fig. 5. This DAG is triggered by *run_train*() route function defined via FastAPI.

In fig. 6, it is shown an acyclic graph scheme for training the models and their deployment. Each of the tasks defines the nodes of the Airflow graph which are responsible for running the component items. The next task cannot be started until the previous one is completed. Thus, Airflow takes care of the integrity and synchronization of the workflow.

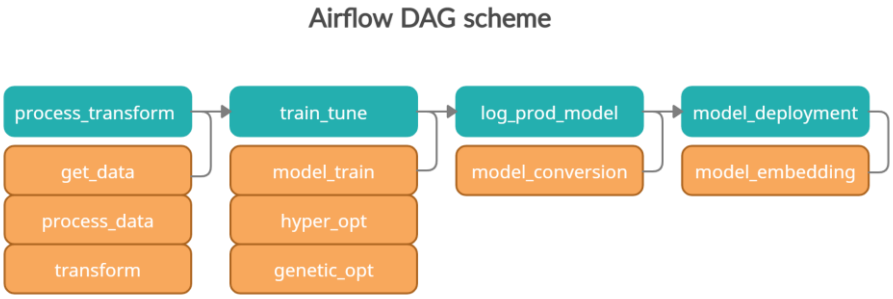

Fig. 6. DAG for running pipeline

Results and discussion

Temperature data (fig. 7) measured in the Laboratory of Intelligent Autonomous Systems of the Faculty of Electronics and Computer Technologies of the Ivan Franko National University of Lviv for the period from February 1, 2021 to September 31, 2021 were used to test the developed prototype system.

Applying the components of pipeline (fig. 5) to the obtained temperatures via running the DAG (fig. 6) by Apache Airflow server resulted in locally deployed models (fig. 8), which is demonstrated by Mlflow UI screenshot. The system is able to select the best performed model among the multiple experiments. Because of the short terms of development, the full microcontroller deployment process has not been completed yet, but the all preparatory works

have been done. Therefore,the *model_observe* module will be complemented as soon as client-microcontroller connects to the server.

The production-ready model, deployed on the server can be accessed directly via *mlflow models serve* command from the terminal or via FastAPI API route function. Then, the process of model embedding into the microcontroller can be invoked.

stm_neural > data > raw > train.csv

```
1   ,time_point,user,value,tag,id
2   0,2021-02-02T14:51:01.026563,home,11.937,indoor,20
3   1,2021-02-02T15:01:02.932900,home,12.0,indoor,21
4   2,2021-02-02T15:11:04.852834,home,12.125,indoor,22
5   3,2021-02-02T15:21:06.772924,home,12.187,indoor,23
6   4,2021-02-02T15:31:08.532852,home,12.125,indoor,24
7   5,2021-02-02T15:41:10.372329,home,12.125,indoor,25
8   6,2021-02-02T15:51:12.212860,home,12.125,indoor,26
9   7,2021-02-02T16:01:13.972915,home,12.125,indoor,27
10  8,2021-02-02T16:11:15.892863,home,12.062,indoor,28
```

Fig. 7. Temperature data sample

During the experiments on Nvidia Jetson Nano we explored the following nuances: 1) the database SQLite can be easily replaced by more powerful PostgreSQL, because the computational resources of microcomputer allow that; 2) increasing the number of layers of recurrent neural network model which has been used for the temperature prediction causes too significant increase in model size (leads to issue with memory limits on microcontroller); 3) genetic algorithm for hyper-parameters optimization performs accurately, but lacks of speed and consumes a lot of computational resources.

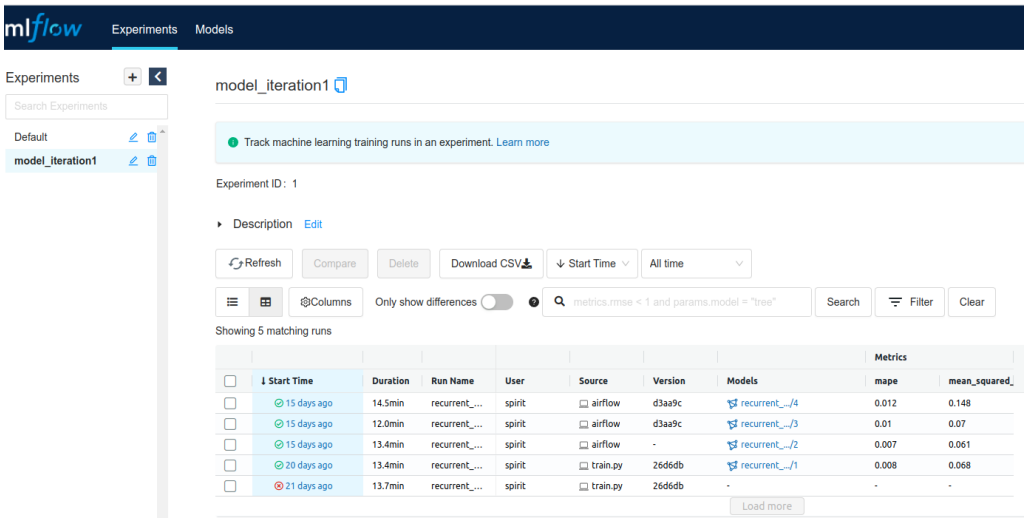

Fig. 8. Mlflow registered models

Conclusions and future work

The approach for developing the MLOps pipeline for edge computing devices based on Nvidia Jetson Nano and STM32 H745 microcontroller is presented. To complete the full exploration of the presented system we have to link the server side with microcontroller by integrating trained and optimized models into STM32 H745. Such an end-to-end scheme will be investigated and discussed in future research. Also, the exploration on asynchronous calls for CPU bound problem (model training) and I/O operations performed on the microcomputer will be of interest in a future work.

REFERENCES

[1] Edge computing use case examples [Електронний ресурс]. – 2020. – Mode of access: https://stlpartners.com/edge_computing/10-edge-computing-use-case-examples/.

[2] *Sriram G*. Edge Computing vs. Cloud Computing: An overview of Big Data challenges and opportunities for large enterprises / G. Sriram. // International Research Journal of Modernization in Engineering Technology and Science. – 2022. – №4. – P. 1186 – 1190.

[3] Bringing computation closer towards user network: Is edge computing the solution? / [A. Ejaz, A. Ahmed, I. Yaqoob та ін.]. // IEEE Communications Magazine. – 2017. – №55. – С. 138–144. DOI: 10.1109/MCOM.2017.1700120.

[4] *Verma A*. Comparative Study of Cloud Computing and Edge Computing: Three Level Architecture Models and Security Challenges / A. Verma, V. Verma. // International Journal of Distributed and Cloud Computing. – 2021. – №9. – С. 13–17.

[5] Recent Advances in Evolving Computing Paradigms: Cloud, Edge, and Fog Technologies / [N. Angel, D. Ravindran, P. Vincent та ін.]. // Sensors. – 2022. – №22. – С. 196–234. DOI: 10.3390/s22010196.

[6] Edge MLOps: An Automation Framework for AIoT Applications / E.Raj, D. Buffoni, M. Westerlund, K. Ahola. // IEEE International Conference on Cloud Engineering (IC2E). – 2021. – С. 191–200. DOI: 10.1109/IC2E52221.2021.00034.

[7] A Digital Twin Architecture Model Applied with MLOps Techniques to Improve Short-Term Energy Consumption Prediction / [T. Yukio, V. Hayashi, R. Arakaki та ін.]. // Machines. – 2022. – №10. – С. 23–49. DOI: 10.3390/machines10010023.

[8] *Fowler M*. Python Concurrency with asyncio / Matthew Fowler. – Shelter Island : Manning Publications, 2022. – 378 p.

[9] FastAPI [Електронний ресурс] // FastAPI. – Режим доступу: https://fastapi.tiangolo.com(дата звернення: 22.05.2022). – Title from screen.

[10] Benchmarks - FastAPI [Електронний ресурс] // FastAPI. – Режим доступу: https://fastapi.tiangolo.com/benchmarks/ (дата звернення: 22.05.2022).

[11] DEAP documentation – DEAP 1.3.1 documentation [Електронний ресурс] // DEAP documentation – DEAP 1.3.1 documentation. – Режим доступу: https://deap.readthedocs.io/en/master/ (дата звернення: 22.05.2022).

[12] *Harenslak B*. Data Pipelines with Apache Airflow / Bas Harenslak, Julian de Ruiter. – [Б. м.]: Manning Publications, 2021. – 480 с.

MLOPS ПРОТОТИП СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ГРАНИЧНИХ ОБЧИСЛЕНЬ

О. Сінькевич, Я. Бойко, Л. Монастирський

Львівський національний університет імені Івана Франка,
вул. Драгоманова, 50, 79005 Львів, Україна
oleh.sinkevych@lnu.edu.ua

У статті представлено підхід до розробки системи штучного інтелекту, що базується на сучасній концепції MLOps. Така система розроблена в контексті граничних обчислень та представляє собою апаратно-програмний комплекс, який не залежить від доступу до хмарних сервісів та може розгортатися локально. Для периферійної апаратної частини ми використовуємо мікрокомп'ютер Nvidia Jetson Nano, який виступає в якості серверної частини для обробки мережевих запитів, зберігання даних і саморозгортання моделей машинного навчання. Запропоновано робочий конвеєр MLOps, повністю розроблений з використанням таких промислових програмних рішень як TensorFlow 2, Mlflow, Apache Airflow. Такий конвеєр повністю інтегрується у розроблений прикладний програмний інтерфейс. Архітектурна схема конвеєра складається з трьох операційних етапів: а) зберігання та обробки даних, що означає отримання даних з бази даних, очищення та їх перетворення для використання у процесі тренування моделі машинного навчання – рекурентної нейтронної мережі б) створення моделі з синхронною оптимізацією гіперпараметрів та реєстрацією моделі; в) розгортання та обслуговування моделі. Увесь конвеєр обгорнутий REST API, створеним за допомогою мікро-фреймворку FastAPI і організований за допомогою служби Apache Airflow. Для реалізації описаного конвеєра ми вибрали температурні дані, які оброблялися рекурентною нейронною мережею на основі GRU, що здійснює короткотермінове прогнозування температури. Гіперпараметри цієї мережі можна оптимізувати за допомогою генетичного алгоритму, який реалізовується під час другого етапу конвеєра та викликається паралельно у процесі її тренування. Також на основі технології сокетів наведена схема приєднання сервера Nvidia Jetson Nano до іншого граничного пристрою – мікроконтролера STM32 H745, що слугує об'єктом розгортання натренованої на сервері нейромережі.

Ключові слова: граничні обчислення, MLOps, машинне навчання, Mlflow, генетичний алгоритм.

Стаття: надійшла до редакції 12.05.2022,
прийнята до друку 16.05.2022