Журнал Белорусского государственного университета. Математика. Информатика. 2026;1:140–155
Journal of the Belarusian State University. Mathematics and Informatics. 2026;1:140–155

УДК 621.391

МУЛЬТИМОДЕЛЬНЫЙ ПОДХОД С ВЕКТОРНЫМ ПОИСКОМ В ОПЕРАТИВНОМ УПРАВЛЕНИИ КРИТИЧНЫМ ИТ-СЕРВИСОМ

В. В. КРАСНОПРОШИН[1)]***, А. А. СТАРОВОЙТОВ***[1)]

[1)]*Белорусский государственный университет, пр. Независимости, 4, 220030, г. Минск, Беларусь*

Аннотация. Исследуются актуальные проблемы, связанные с задачей оперативного управления критичными ИТ-сервисами. Для повышения качества управления предлагается новый метод, расширяющий подход динамической локальной аппроксимации. Он реализует связывание векторных представлений паттернов нагрузки с обучаемыми нейросетевыми моделями, что позволяет оперативно строить предиктор для нового паттерна на основе векторного поиска семантически близких моделей в библиотеке. Излагаются оригинальные принципы проектирования системы управления, которая для адаптации к новым, связанным с неопределенностью паттернам нагрузки использует аппроксимацию данных, получаемых в режиме реального времени. Приводятся результаты экспериментов, подтверждающие эффективность предложенного подхода и показывающие, что при обработке нерегулярных и сложных нагрузок гибридные проактивные алгоритмы по ряду показателей являются более эффективными, чем реактивные алгоритмы.

Ключевые слова: принятие решений; проактивное управление; неопределенность внешней нагрузки; нейронная сеть; мультимодельная система; библиотека моделей; векторный поиск.

MULTI-MODEL APPROACH WITH VECTOR SEARCH IN OPERATIONAL MANAGEMENT CRITICAL IT SERVICE

V. V. KRASNOPROSHIN[a]***, A. A. STAROVOITOV***[a]

[a]*Belarusian State University, 4 Niezaliezhnasci Avenue, Minsk 220030, Belarus*

Corresponding author: V. V. Krasnoproshin (krasnoproshin@bsu.by)

Abstract. In the work are investigated current problems related to the task of operational management of critical IT services. To improve the quality of management, a new method is proposed, extending the dynamic local approximation approach. It implements linking vector representations of load patterns with trainable neural network models, which allows promptly constructing a predictor for a new pattern based on vector search for semantically close models in the library. Original principles of designing a control system are outlined, which for adaptation to new, uncertainty-related load patterns uses approximation of data obtained in real-time mode. The results of experiments confirmed the effectiveness

Образец цитирования:
Краснопрошин ВВ, Старовойтов АА. Мультимодельный подход с векторным поиском в оперативном управлении критичным ИТ-сервисом. *Журнал Белорусского государственного университета. Математика. Информатика.* 2026;1:140–155 (на англ.).
EDN: OYKWAO

For citation:
Krasnoproshin VV, Starovoitov AA. Multi-model approach with vector search in operational management critical IT service. *Journal of the Belarusian State University. Mathematics and Informatics.* 2026;1:140–155.
EDN: OYKWAO

Авторы:
Виктор Владимирович Краснопрошин – доктор технических наук, профессор; профессор кафедры информационных систем управления факультета прикладной математики и информатики.
Александр Александрович Старовойтов – аспирант кафедры информационных систем управления факультета прикладной математики и информатики. Научный руководитель – В. В. Краснопрошин.

Authors:
Viktor V. Krasnoproshin, doctor of science (engineering), full professor; professor at the department of information management systems, faculty of applied mathematics and computer science.
krasnoproshin@bsu.by
https://orcid.org/0000-0002-9463-4869
Aleksandr A. Starovoitov, postgraduate student at the department of information management systems, faculty of applied mathematics and computer science.
starovoytovaa@bsu.by
https://orcid.org/0009-0001-6531-4225

of the proposed approach and showed that when processing irregular and complex loads, hybrid proactive algorithms in a number of indicators are more effective than reactive ones.

Keywords: decision making; proactive control; external load uncertainty; neural network; multi-model system; model library; vector search.

Introduction

Maintaining critical IT services in operational condition with a guaranteed level of service (*service level objective*, SLO) is one of the key tasks of the operation of modern digital infrastructure. The dynamic and often unpredictable nature of external load requires the use of complex automatic scaling (autoscaling) mechanisms, capable not only of reacting to sharp load changes but also of predicting them [1].

Existing industrial solutions (Kubernetes HPA, AWS Auto Scaling, Azure Autoscale) are often based on threshold rules, not always effective under complex, non-stationary load profiles. Research approaches, using machine learning and time series forecasting, show high accuracy, but may require long training and are not sufficiently prompt in adapting to new, previously unseen load patterns. As an example illustrating the complexity of the load profile in fig. 1 are presented data on user requests to the web server of the NASA Kennedy Space Centre in Florida on 6 July 1995[1].

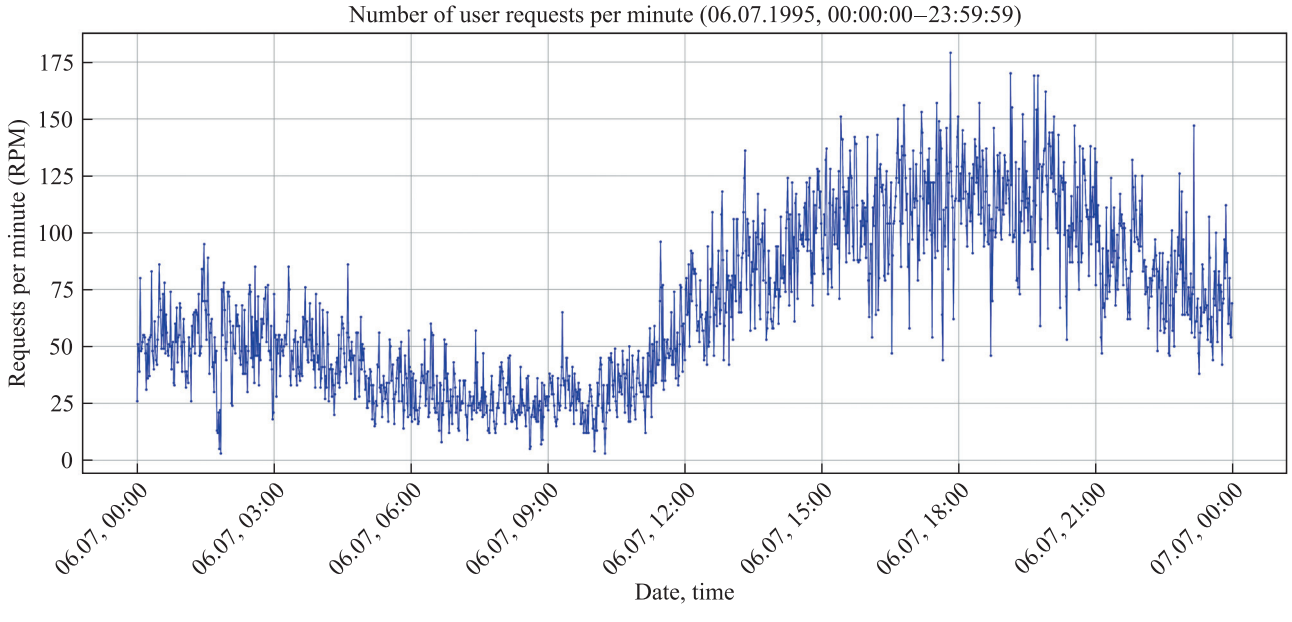

Fig. 1. Example of external load

In previous work [2], the authors proposed a multi-model method DLANNLIB (*dynamic local approximation by neural network models with library*). For each system state (determined by the volume of allocated resources) in real time is built and saved into a library a local neural network model, trained on data of the current load pattern. Such a mechanism allows accumulating «experience» and using it for predicting the transition of the model system to other states.

This article develops the indicated approach, proposing to use a vector database for semantic search of models in the library based on the similarity of load patterns. This idea is applied to solve the key task: to ensure the proactivity of control at the beginning of a new state, when data for training a new model is still insufficient. But it is already possible to assess the similarity of the current situation with previously observed ones, select corresponding models from the library, prepare a predictor, and based on the forecast prepare a control decision.

Method of dynamic local approximation by neural network models with library and vector search of *k*-nearest models

The proposed development of the DLANNLIB method consists in the addition of a new component – a vector database, which helps implement the mechanism of search and selection of models from the library. The name DLANNLIB_kNM (*dynamic local approximation by neural network models with library with k-nearest models vector search*) will be used in the further exposition.

[1]NASA-HTTP // Internet Traffic Archive [Electronic resource]. URL: https://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html (date of access: 09.09.2025).

Formation of vector representations. For each neural network model, saved in the library, additionally is formed its vector representation (embedding). This feature vector is linked to a specific system state S_i, the load pattern on which the model was trained.

The vector can include various statistical metrics of the training dataset (mean, variance, entropy, correlation dimension, etc.). However, for determining some of them, such as the correlation dimension by the Grassberger – Procaccia method [3] is required a large number of samples and the calculation process itself can take additional time, reducing the operational efficiency.

Most naturally to use as a vector the dataset on which the model was trained or its part. Here various options are possible. The simplest is the use of a vector, built from the first or last samples of the training set, with dimensionality equal to the number of features in the model (window size).

Vector search for a new pattern. Upon the system's transition to a new state S_{i+1} and the arrival of the first utilisation data, before the readiness of a dataset for training a new model, the following actions are performed:

1. Based on these data, the feature vector of the current pattern is determined.
2. A search in the vector database is conducted by the metric of scalar product, cosine similarity, Euclidean, Manhattan distances and other metrics for finding a certain number of the closest vectors by the nearest neighbours method (k-NN).
3. Previously trained neural network models, corresponding to the found vectors, are extracted from the library.

Construction and use of an ensemble predictor. The found models are used to construct an ensemble predictor for the current state. The weights of models in the ensemble can be determined:

- by distance (inversely proportional to the distance between the vector of the current pattern and the vector of the found model);
- quality of the found models (inversely proportional to the errors of the models at the validation stage);
- a hybrid option using both distances and validation errors simultaneously.

The resulting forecast is formed as a weighted sum of the forecasts of individual models of the ensemble.

Training a new model and updating the database. In parallel with the operation of the ensemble predictor, the system continues to accumulate data for the new state. As soon as a sufficient amount of data (samples) has been accumulated, the standard process of training a new local model by the DLANN method is launched. After successful training:

1. The new model together with metadata is saved into the library.
2. For it, a vector representation is determined based on the training sample (sample elements, metadata, statistical parameters).
3. This vector is added to the vector database and linked to the new model.

Forgetting mechanism. In the process of the system's operation, new models and data accumulate in the vector database. A forgetting mechanism can be implemented, which allows limiting the number of semantically close models, leaving only the best models, with minimal validation error.

Model system

Model system consists of a set of computational modules, on which operate instances of application software, between which is carried out balancing of external requests. For request generation is used a load system, allowing to set load and receive statistics on processed requests. The system supports a scaling mechanism (state change). In fig. 2 is presented a diagram of the model system. For managing service configurations is used the Infrastructure as a Code (IaC) approach in which service configurations are described in YAML files.

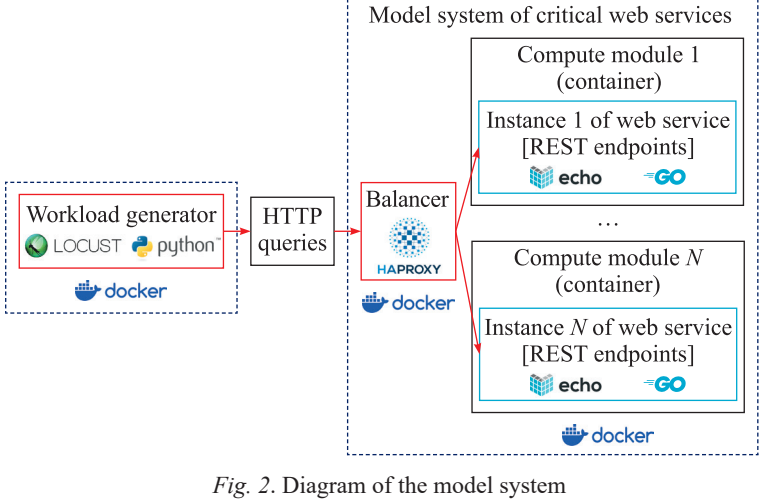

Fig. 2. Diagram of the model system

We will describe the main blocks of the model system.

Block of computational modules. As computational modules is used Docker[2]. This solution allows creating computational modules (Docker containers), managing them, collecting various utilisation metrics. There are ready libraries (e. g., Docker SDK for Python), for working with Docker API Engine. Possible use of other solutions, e. g., Kubernetes clusters, etc.

Block of load balancing. As a load balancer across containers with instances of the application web service is used HAProxy[3], which allows automatically adding new instances to the load balancing. Previously in experiments was used load balancing by service name using the mechanism built into Docker, from which had to be abandoned due to uneven balancing when creating new containers.

Block of application web service. As a web application was decided to implement a microservice, based on the popular high-performance minimalist web framework Echo[4], written in the high-level language Go[5]. Service instances implement functionality in the form of REST API, consisting of several REST endpoints, which process incoming HTTP requests from client systems (external load). Each endpoint is associated with a specific microfunctionality, which has a varying impact on computational resource consumption.

Block of load formation. As a load generator is used Locust[6], written in Python[7]. Locust has less functionality compared to Apache JMeter[8], simple to configure, allows describing load profiles in the form of Python classes, less demanding on resources, supports distributed configuration of instances, allows setting the load profile in the form of a function or in the form of pre-prepared data.

Load profiles

For generating synthetic profiles for Locust is written a Python script, which at input receives a description of stages in the form of a list of dictionaries, in which are defined the durations of stages, target functions, width of variance, function for changing variance and others. For generating profiles based on real data is implemented a Python script, which allows forming the structure of stages for launching load for Locust. At the initial stages of system development mainly were used simple synthetic profiles «ramp», «triangle», «trapezoid» and the same profiles with noise. It is worth noting that even simple profiles like «ramp» generate a sufficiently complex dependence of the target metric on time `%CPU_Util`, for various system states. For example, in fig. 3 and 4 are presented user load profiles «triangle» and «triangle with noise».

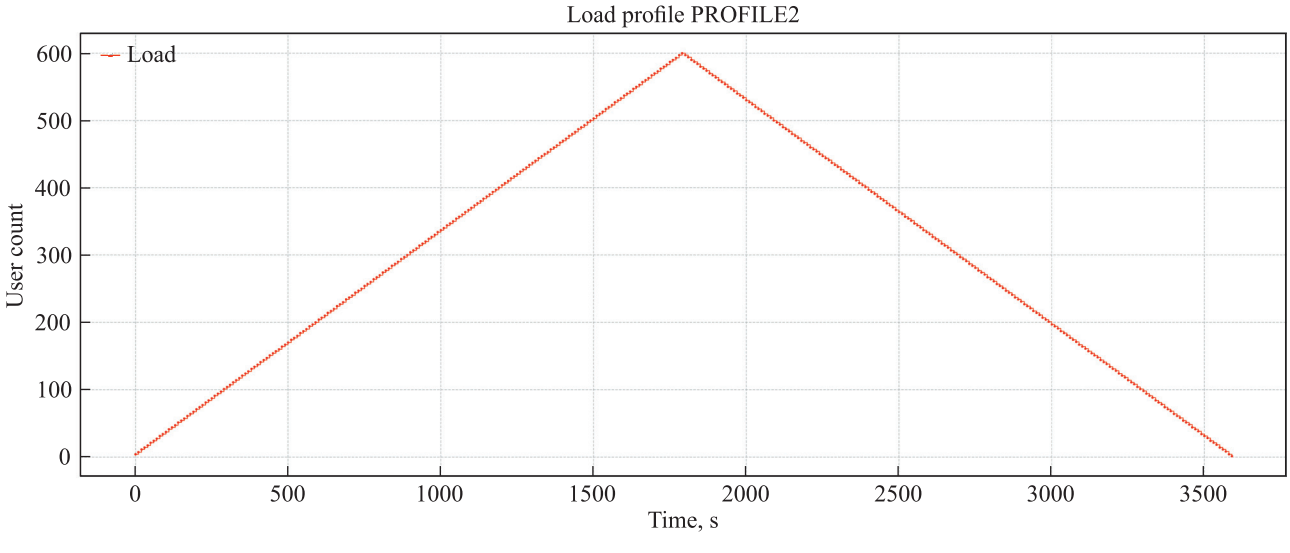

Fig. 3. Load profile «triangle», change the number of users over 1 h

[2]Docker Compose // Docker Docs : website. URL: https://docs.docker.com/compose (date of access: 09.09.2025).

[3]HAProxy: the reliable, high-performance TCP/HTTP load balancer [Electronic resource]. URL: https://www.haproxy.org (date of access: 09.09.2025).

[4]Echo: high-performance, extensible, minimalist Go web framework [Electronic resource]. URL: https://echo.labstack.com (date of access: 09.09.2025).

[5]Go: an open-source programming language supported by Google [Electronic resource]. URL: https://go.dev (date of access: 09.09.2025).

[6]Locust: an open-source load testing tool [Electronic resource]. URL: https://locust.io (date of access: 09.09.2025).

[7]Python: programming language [Electronic resource]. URL: https://www.python.org (date of access: 09.09.2025).

[8]Apache JMeter: open-source software [Electronic resource]. URL: https://jmeter.apache.org (date of access: 09.09.2025).

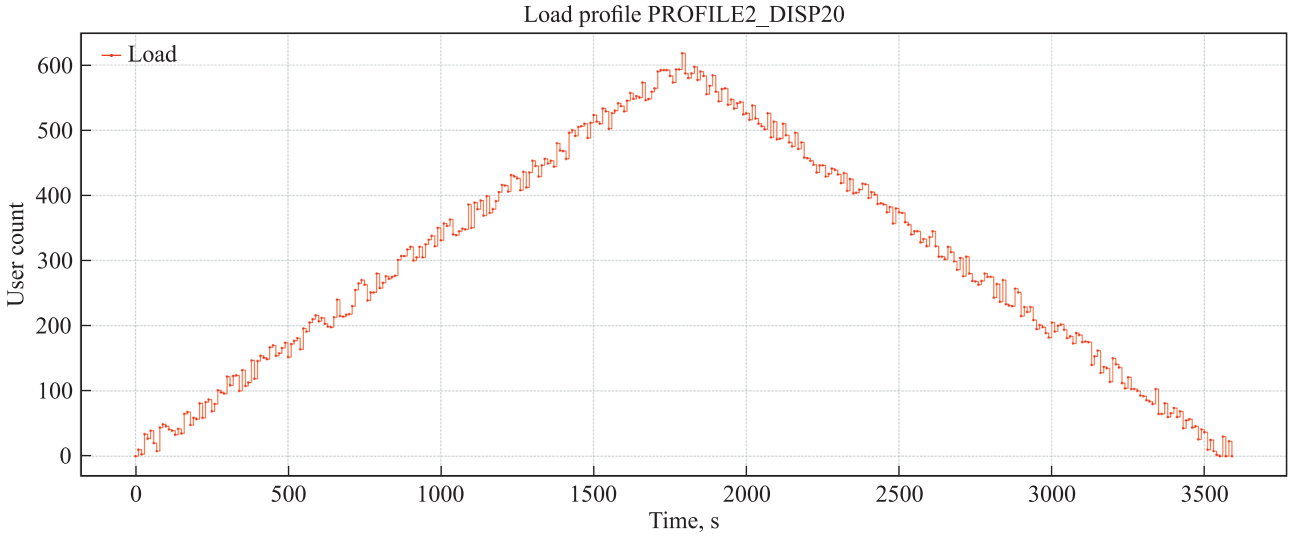

Fig. 4. Load profile «triangle with noise», change the number of users over 1 h

After checking the operation of the control system on synthetic profiles, was chosen a reference load profile, which is often used by other research groups [4]. This profile describes the number of user requests per unit time (*requests per second*, RPS) to various endpoints of the web service of the NASA Kennedy Space Centre in Florida over 24 h from 6 July 1995. The data are presented in public access in the Internet Traffic Archive repository on the website of the Lawrence Berkeley National Laboratory[9].

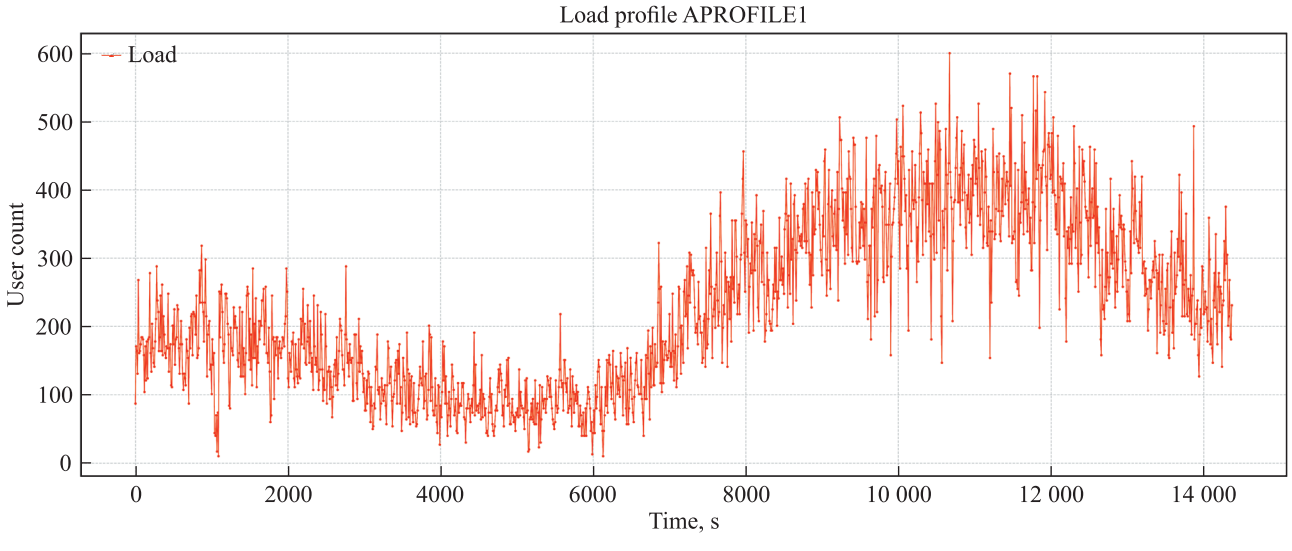

Fig. 5. Load profile «NASA_06.07.1995_4H», change the number of users over 4 h

The original profile was compressed from 24 to 4 h and scaled to 600 users (while the original number of samples was preserved) and converted into a load profile for Locust. In fig. 5 is presented the resulting user load profile «NASA_06.07.1995_4H».

Multi-model automated control system

For managing resources of the model IT system was developed an automated control system (ACS), which in real time receives utilisation data of computational modules and makes a decision on scaling the controlled system. Uses a combination of reactive and proactive control. In this case, for each state of the controlled system automatically is formed its own dataset, is trained its own neural network model, is formed a feature vector of the current pattern based on this dataset, the model and its corresponding vector are saved into the model library, are performed predictions of resource utilisation parameters by different predictors (based on the trained model for the state and model selected from the model library using vector search).

[9]NASA-HTTP // Internet Traffic Archive [Electronic resource]. URL: https://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html (date of access: 09.09.2025).

The ACS compares current data of average load across computational modules and forecast results for a specific system state and makes a decision on state change (scaling), depending on the achievement of threshold values of resource utilisation.

Architecturally, the system contains seven main modules, which are presented in fig. 6.

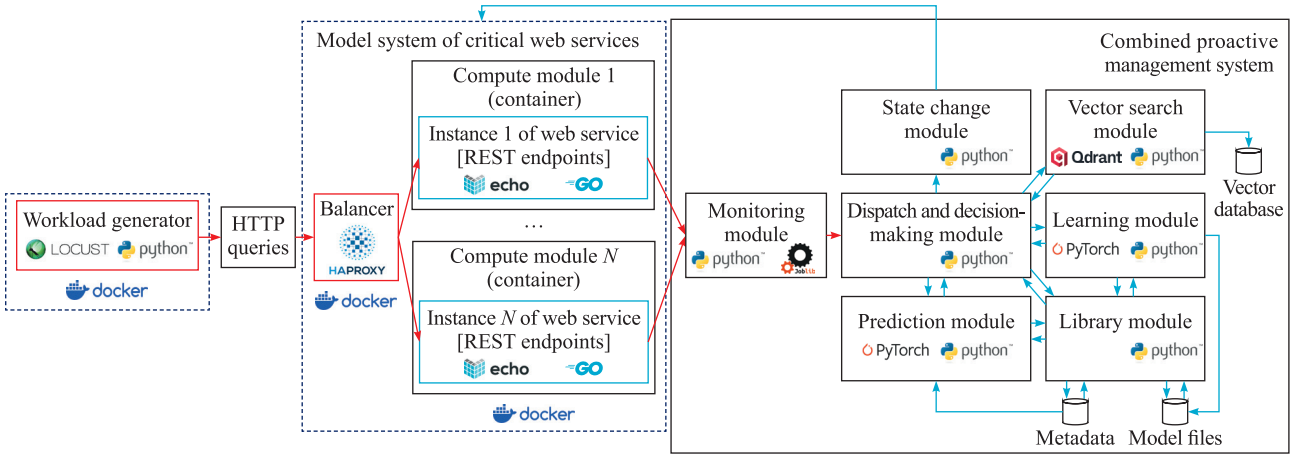

Fig. 6. Diagram of the ACS and the model system

The model IT service, load system, and ACS are described in detail by the authors in the article [2]. As the database is used SQLite[10]. The approach with vector search adds new components to the scheme – a vector database and a vector search module. As the vector database is used either a custom lightweight solution in Python, or the `sqlite-vec` extension[11], or a ready-made vector database Qdrant[12].

Monitoring module. The monitoring module is responsible for collecting performance metrics of the system's computational modules, as well as adding and removing new metrics (when the system state changes). To correctly collect metrics, the Joblib library[13] was used, which allows for parallel queries and timely retrieval of metric values for all modules.

State change module. The state change module is responsible for sending control commands (which change the state of the controlled system) and monitoring the correctness of the state change.

Dispatch and decision-making module. The dispatch and decision-making module is the central one. It and the other modules are implemented in Python. At the initialisation stage, resource utilisation thresholds (system SLOs) are set, the achievement of which leads to a change in the state of the controlled system. Thresholds for adding (`A`) and removing (`D`) computational resources are set separately. Three control signals are used – the current value of resource utilisation, the forecast based on the model library, and the forecast based on the model trained for the current state. The resulting signal is determined as the maximum of the three signals. An activation mechanism for signals is introduced, which blocks or activates signals in the required sequence. This process is controlled by the parameters `cool_reactive`, `cool_proactive`, `cool_proactive_lib`. This mechanism determines the number of cycles during which no actions to change the state are performed in the system and which signals influence the change of the system state. A parameter is set that determines the number of data accumulation iterations for one state to create a model (`M`). A lead time (`Z`) is set – the number of forecast points into the future. In the main process, after the initialisation stage, a cycle is implemented, at each iteration of which the set of the computational modules of the controlled system is updated, the current values of computational module utilisation are obtained, and a decision is made to change the state of the controlled system. Metrics are collected with a period of ~2 s. Historical data on the utilisation of each computational module for different system states are saved in the form of CSV files. In RAM, data on module utilisation for the current state are accumulated in a dictionary. Based on the most recently obtained data, in the main process, the average value across the set of computational modules is calculated. If the maximum of the three active signals exceeds the threshold for addition (`A`), a decision is made to add resources to the controlled system. If it is less than the removal threshold (`D`), a decision is made to remove resources. After a decision is made, a command is sent to the state change block. This process implements mechanisms for non-blocking interaction with processes that run in parallel to the main process, responsible for creating and training neural networks and

[10]SQLite: C-language library [Electronic resource]. URL: https://www.sqlite.org (date of access: 09.09.2025).

[11]SQLite-Vec: a vector search SQLite extension [Electronic resource]. URL: https://alexgarcia.xyz/sqlite-vec (date of access: 09.09.2025).

[12]Qdrant: vector search engine [Electronic resource]. URL: https://qdrant.tech (date of access: 09.09.2025).

[13]Embarrassingly parallel for loops // Joblib: running Python functions as pipeline jobs [Electronic resource]. URL: https://joblib.readthedocs.io/en/latest/parallel.html (date of access: 09.09.2025).

forecasting utilisation parameters for a specific state with a given lead time. This mechanism is implemented using the `multiprocessing` library[14].

Training and prediction modules. All modules are implemented using Python, using the PyTorch library[15]. The training module creates neural network models based on datasets for different states of the controlled system and saves the models and their corresponding vectors into the neural network model library. Each model is encoded by a state index. The prediction module loads models from the neural network model library by index and performs utilisation forecasts with a certain lead time for different states of the controlled system. When forecasting, we assume that each subsequent term of the series depends on a certain number of previous elements of the series, i. e. shifted in time previous values of the original series are used as independent autoregression parameters. The number of such parameters determines the sliding window for the series. We choose the window size (`tw`) equal to 30 elements (determined empirically). We assume that the main contribution to the approximating function for the next element of the series is made by a combination of the 30 previous elements of the series.

A neural network with 1 hidden layer and 1 output neuron is used. The number of neurons in the hidden layer (90) is 3 times greater than the temporal window size. In the output layer, there is 1 neuron, as a sum element. A fully connected linear layer `nn.Linear` is used. The activation function is `ReLU`. For regularisation, dropout of part of the neurons is used (dropout with a neuron dropout probability of 0.015). The loss function used is `nn.MSELoss()` (mean squared error). The optimisation method is `torch.optim.Adam`, learning rate 0.002. Preprocessing of the original series with scaling to the interval $[0, 1]$ is not performed, as it introduces additional error into the original data and leads to additional overhead for scaling before and after training (we utilise the property of the original data, which are bounded from below and above). A short time series is used – 60 samples. The interval between samples is 2 s. The duration of the series is 128 s. The original series is divided into two datasets – train (42 samples, or 70 %) and test (18 samples, or 30 %). Considering that the window size (`tw`) is 30, the number of examples for training is 13. A total of 100 training epochs are used. The batch size is 1. The training time for a neural network with the specified architecture for a dataset is ~2 s. The speed of performing a prediction with a lead time of 40 samples is $\ll 1$ s.

Estimation of management quality

To evaluate the quality of management, we consider a problem, where the target function reflects a trade-off between optimising computational resource costs and meeting SLO requirements. The problem variables: $\Omega(t)$ – dynamically changing external system load with uncertainty; $R_k(t)$ – the amount of computational resources allocated to the system at the k^{th} step (system resources change discretely, we assume that the k^{th} step corresponds to state S_k); $\text{Cost}\left(R_k(t)\right)$ – the cost of resource usage at the k^{th} step; $\text{SLO}_k(t)$ – the SLO metric at the k^{th} step; $\text{Violation}\left(\text{SLO}_k(t)\right)$ – the penalty for SLO violation. $\text{Cost}\left(R_k(t)\right)$ and $\text{Violation}\left(\text{SLO}_k(t)\right)$ are normalised in the interval $[0, 1]$ over all states S_k. The target function to be minimised:

$$\arg\min_{A}\left(\lambda\sum_{k=1}^{N}\text{Cost}\left(R_k\left[A\left(\Omega(t)\right)\right]\right)+(1-\lambda)\sum_{k=1}^{N}\text{Violation}\left(\text{SLO}_k\left[A\left(\Omega(t)\right)\right]\right)\right)$$

with the constraint on the rate of dynamic resource changes $\left|R_k - R_{k-1}\right| \le \gamma$, where γ is the maximum allowable resource change.

As a result, the problem reduces to minimising the target function, achieved by selecting the optimal algorithm A, which generates N system states under external load $\Omega(t)$ with uncertainty. An exact solution to this problem cannot be constructed, so we propose a heuristic algorithm that estimates the target function. From the set of algorithms, we select the one that minimises the target function.

Experimental environment

The model system, load balancing service and load testing system operate in Docker containers on a virtual machine running a Linux-based OS – Ubuntu 22.04.1 LTS. The control system is deployed directly in the OS. Tests were conducted on various physical hardware. Final tests were performed on a virtual machine with the following parameters: vCPU – 8 cores, RAM – 24 GB.

We evaluate the performance quality of the combined automatic scaling control system using the previously described methodology. We use the quality metric F with an adjustable weight λ to specify the desired

[14]Multiprocessing – Process-based parallelism // The Python Standard Library [Electronic resource]. URL: https://docs.python.org/3/library/multiprocessing.html (date of access: 09.09.2025).

[15]PyTorch : website. URL: https://pytorch.org (date of access: 09.09.2025).

cost-to-penalty ratio. A threshold value of 24 ms is used for the response time penalty function. For calculating maximum cost, we use $C_{max} = 6$.

In experiments, three load profiles are used, in table 1 is presented a brief description.

Table 1

Description of load profiles

Profile name	Description	Time, h
PROFILE2	Profile «triangle» without dispersion. Linear increasing load from 1 to 600 users over 30 min and linear decreasing load from 600 to 0 users over 30 min	1
PROFILE2_DISP20	Profile «triangle» with random noise. Linear increasing load with random noise from 1 to 600 users over 30 min and linear decreasing load with random noise from 600 to 0 users over 30 min	1
APROFILE1	Profile «NASA_06.07.1995_4H». Based on real load	4

Control algorithms

Each profile was tested on three different algorithms. In the algorithm name are encoded the main operating parameters. Decoding of the code is presented in table 2. In each test group, corresponding to a specific profile, the test with the reactive algorithm is considered as the baseline. The operation of other algorithms is compared by quality metrics with the baseline algorithm.

Table 2

Decoding of algorithm codes

Algorithm code	Type	Main operating parameters	Value
A90.50.R60	Reactive	Addition threshold, `%CPU_Util`	90
		Removal threshold, `%CPU_Util`	50
		Reactive component activation period, samples	60
A90.50.R60P70F	Hybrid (reactive + DLANN)	Addition threshold, `%CPU_Util`	90
		Removal threshold, `%CPU_Util`	50
		Reactive component activation period, samples	60
		Proactive component activation period, samples	70
A90.50.R60P70L60_70FNNL_VEC1	Hybrid (reactive + DLANNLIB_kNM)	Addition threshold, `%CPU_Util`	90
		Removal threshold, `%CPU_Util`	50
		Reactive component activation period, samples	60
		Proactive component activation period, samples	70
		Proactive component activation period (library), samples	60

Let us briefly describe the activation mechanism of control signals for the algorithm with library and vector model search A90.50.R60P70L60_70FNNL_VEC1. A stabilisation period equal to 60 samples (120 s) is used for forced model formation for each state (also possible is the variant used in earlier versions of the ACS, where a model is not formed for each state). After the end of this period, in the decision-making mechanism, signals related to the reactive part (current values of the collected metric) and the forecast based on the model library are activated, and after 10 samples (20 s), the signal related to the forecast using the model trained for the current state is activated, and simultaneously the signal related to the forecast based on the model library is deactivated (we assume that the model trained on data of the current state has greater expertise within this state). Schematically, the activation mechanisms for the algorithms specified in table 2 are shown in fig. 7.

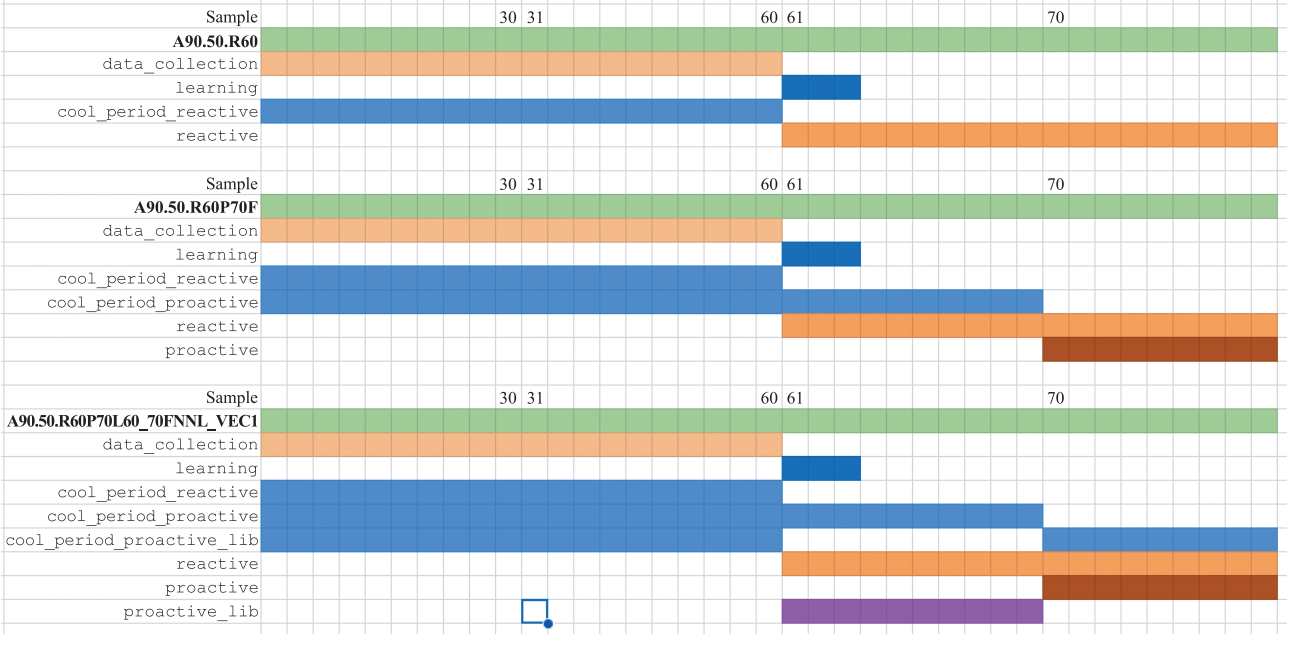

Fig. 7. Diagram of control signal activation

In table 3 are presented the results of experiments on the operation of the ACS for three different load profiles. Quality metrics were calculated after conducting the experiments, based on the obtained statistics of the load system and logs of the control system operation.

Table 3

Results of experiments

Profile	Algorithm code	Normalised values		Target function		
		C^{norm}	V^{norm}	$2F_{1/2}$	$F_{2/3}$	$F_{1/3}$
PROFILE2	A90.50.R60	**0.367**	0.710 (+204.6 %)	1.077 (+34.6 %)	0.482	0.596
	A90.50.R60P70F	0.403 (+9.7 %)	0.596 (+155.8 %)	1.000 (+24.9 %)	0.468	0.532
	A90.50.R60P70L60_70FNNL_VEC1	0.567 (+54.4 %)	**0.233**	**0.800**	0.456	0.345
PROFILE2_DISP20	A90.50.R60	**0.341**	0.705 (+51.8 %)	1.046 (+26.5 %)	0.463	0.584
	A90.50.R60P70F	0.398 (+16.6 %)	0.595 (+28.2 %)	0.993 (+20.1 %)	0.464	0.530
	A90.50.R60P70L60_70FNNL_VEC1	0.363 (+6.3 %)	**0.464**	**0.827**	0.397	0.431
APROFILE1	A90.50.R60	**0.333**	0.743 (+56.6 %)	1.076 (+14.4 %)	0.470	0.606
	A90.50.R60P70F	0.354 (+6.3 %)	0.673 (+42.0 %)	1.027 (+9.3 %)	0.460	0.567
	A90.50.R60P70L60_70FNNL_VEC1	0.466 (+39.6 %)	**0.474**	**0.940**	0.469	0.471

N o t e. Boldface indicates the best (minimum) normalised value for each profile across the group of algorithms. The values shown in the table represent resource cost, SLO violation penalty, and their sum, respectively. Values in parentheses show the percentage increase relative to the best (minimum) value within the same group for the corresponding profile.

Figures 8–13 are used as an illustration of the differences in algorithm operation for load profiles PROFILE2, PROFILE2_DISP20 and APROFILE1. On the graph of average CPU load across computational modules, data for different states are shown in colour; on the lower graph, the system trajectory (dependence of the number of computational modules on time) with state numbers is shown.

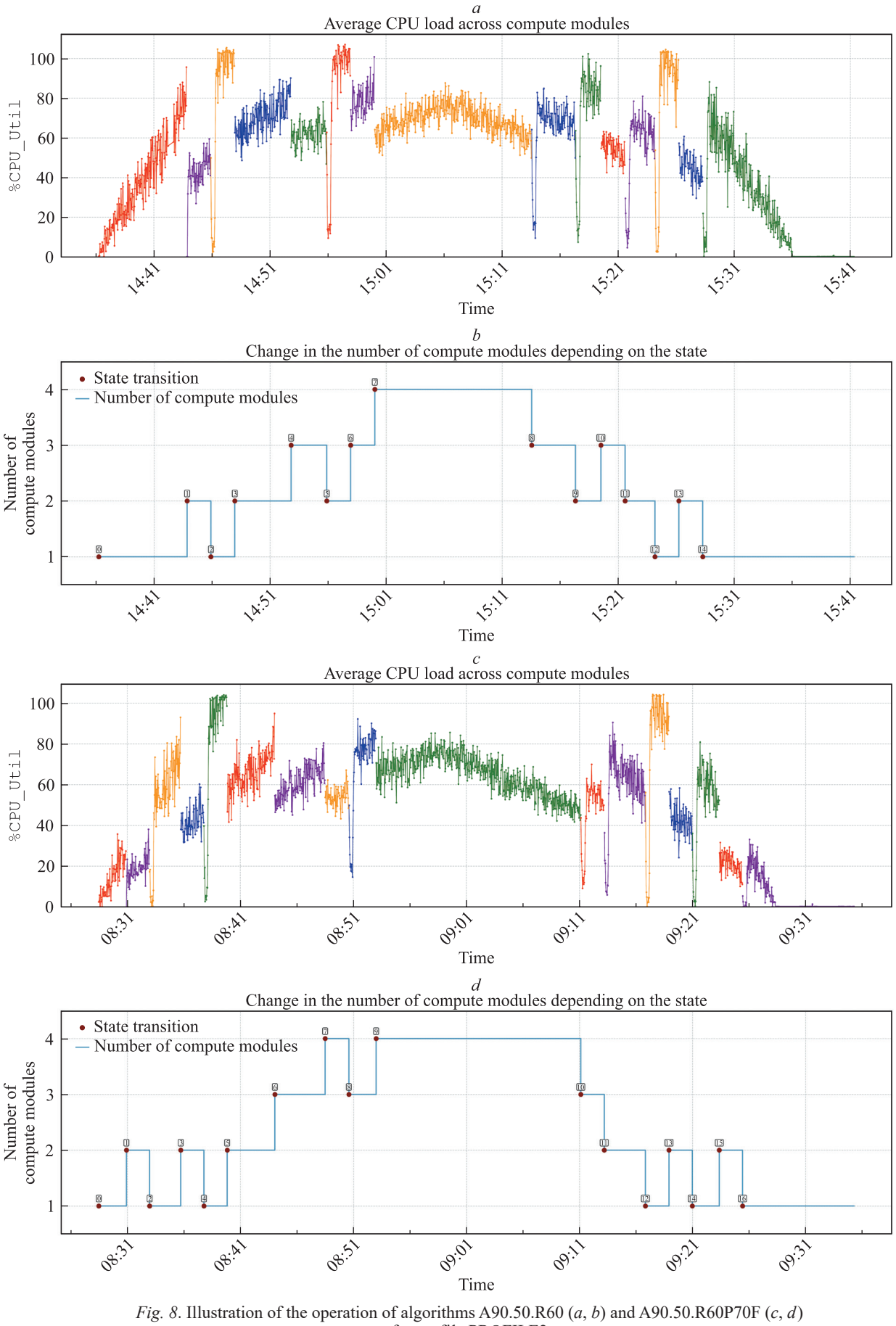

Fig. 8. Illustration of the operation of algorithms A90.50.R60 (*a*, *b*) and A90.50.R60P70F (*c*, *d*) for profile PROFILE2

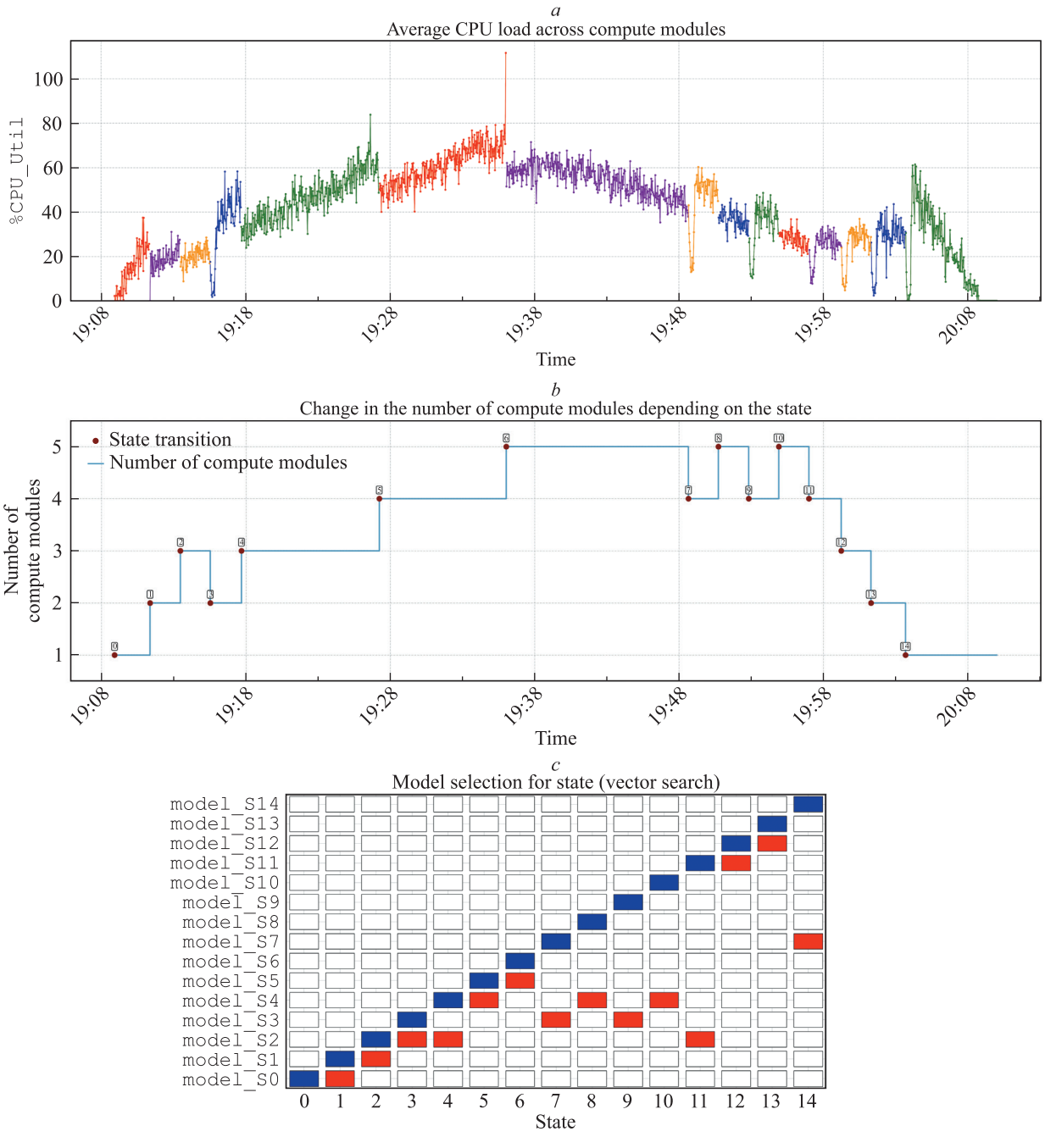

Fig. 9. Illustration of the operation of algorithm A90.50.R60P70L60_70FNNL_VEC1 (*a*, *b*) and model selection using vector search (*c*) for profile PROFILE2

Analysis of the results for profile PROFILE2 shows that in terms of the sum of normalised cost and penalty metric $2F_{1/2}$, the best was the algorithm with library and vector model search A90.50.R60P70L60_70FNNL_VEC1. It is approximately 34.6 % better than the reactive algorithm A90.50.R60 and 24.9 % better than the hybrid algorithm A90.50.R60P70F. If comparing by individual metrics, the most resource-efficient, as expected, is the reactive algorithm A90.50.R60, which is 9.7 % more economical than algorithm A90.50.R60P70F and 54.4 % more economical than algorithm A90.50.R60P70L60_70FNNL_VEC1. In terms of penalties for response time, the best was algorithm A90.50.R60P70L60_70FNNL_VEC1, which is almost 204.6 % better than the reactive algorithm A90.50.R60 and 155.8 % better than hybrid algorithm A90.50.R60P70F. For the target function $F_{1/3}$ with a greater weight towards penalties, the best result was shown by the algorithm A90.50.R60P70L60_70FNNL_VEC1, followed by the algorithm A90.50.R60P70F, which were 73.0 and 54.4 % better than the algorithm A90.50.R60. Figure 9, *c*, shows which models were selected from the library using the DLANNLIB_kNM method for various states under load profile PROFILE2. It is interesting to note that up to state 12, various models trained on increasing load were selected. At the beginning, predominantly the model trained for the previous state was chosen. And only starting from state 12, models trained during decreasing load began to be used.

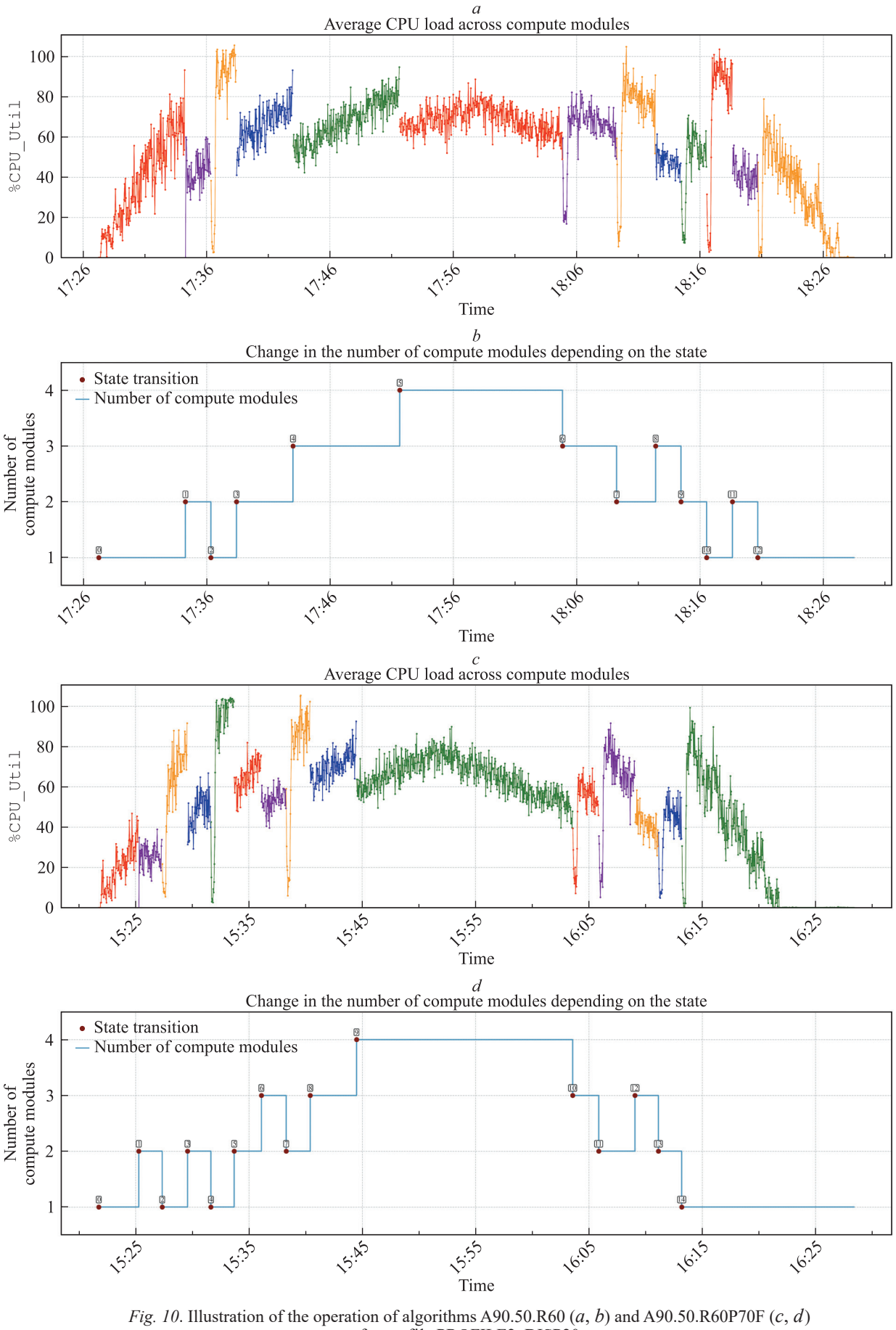

Fig. 10. Illustration of the operation of algorithms A90.50.R60 (*a*, *b*) and A90.50.R60P70F (*c*, *d*) for profile PROFILE2_DISP20

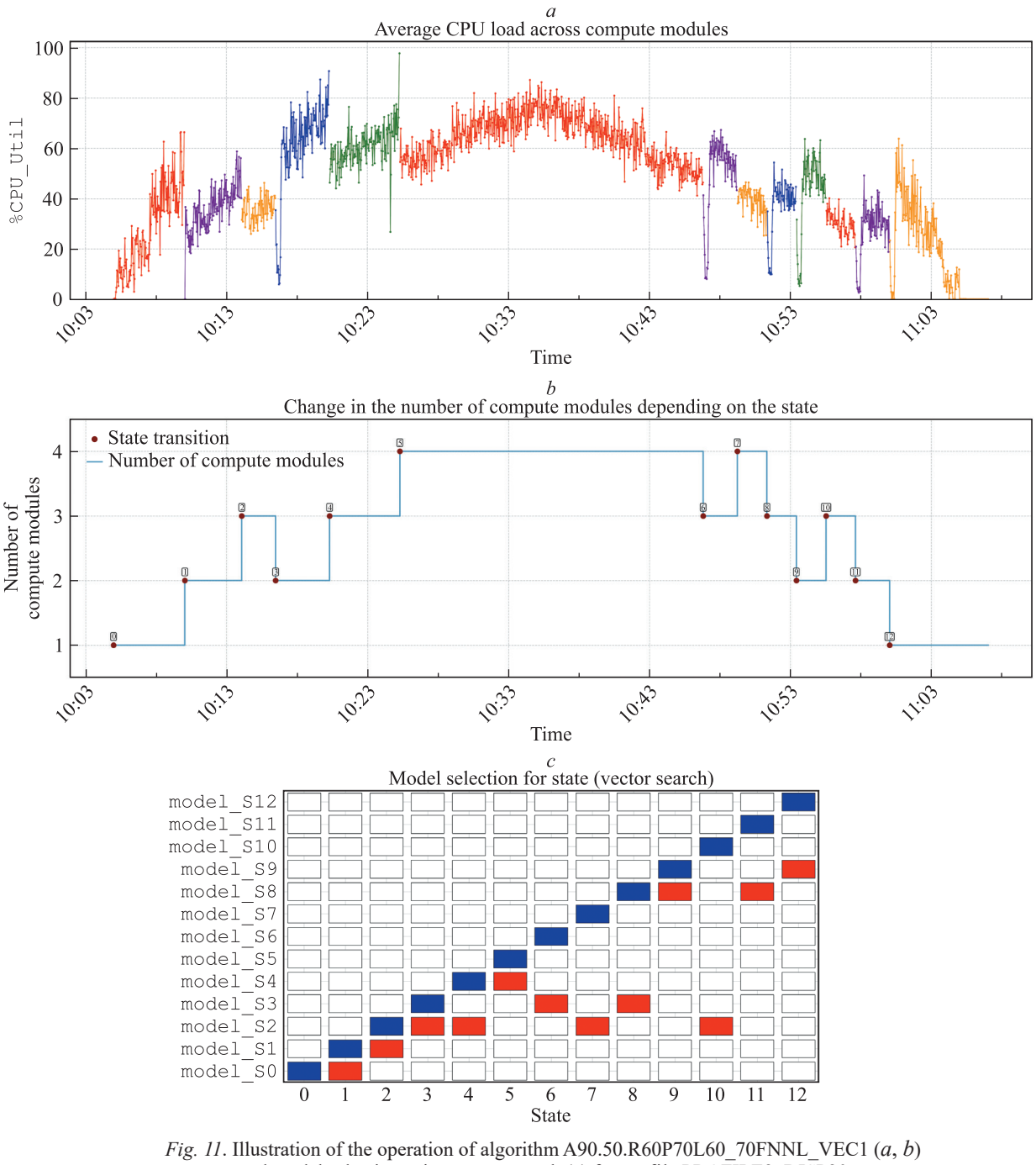

Fig. 11. Illustration of the operation of algorithm A90.50.R60P70L60_70FNNL_VEC1 (*a*, *b*) and model selection using vector search (*c*) for profile PROFILE2_DISP20

Analysis of the results for the more complex profile PROFILE2_DISP20 shows that in terms of the target function $2F_{1/2}$, the best results were again shown by the algorithm A90.50.R60P70L60_70FNNL_VEC1. It is approximately 26.5 % better than the reactive algorithm A90.50.R60 and 20.1 % better than the hybrid algorithm A90.50.R60P70F. If comparing by individual metrics, the most resource-efficient, as expected, is the reactive algorithm A90.50.R60, which is 6.3 % more economical than algorithm A90.50.R60P70L60_70FNNL_VEC1 and 16.6 % more economical than algorithm A90.50.R60P70F. In terms of penalties for response time, the best was algorithm A90.50.R60P70L60_70FNNL_VEC1, which is almost 51.8 % better than the algorithm A90.50.R60 and 28.2 % better than the algorithm A90.50.R60P70F. For the target function $F_{1/3}$ with a greater weight towards penalties, the best result was shown by the algorithm A90.50.R60P70L60_70FNNL_VEC1, followed by the algorithm A90.50.R60P70F, whose results were 35.6 and 23.0 % better than the algorithm A90.50.R60. Figure 11, *c*, shows which models were selected from the library using the DLANNLIB_kNM method for various states under load profile PROFILE2_DISP20. It is interesting to note that up to state 6, various models trained on increasing load were selected. Further, during decreasing load, for states 6, 7 and 8, models trained during increasing load were also selected, but for state 9, a model trained during decreasing load was already chosen.

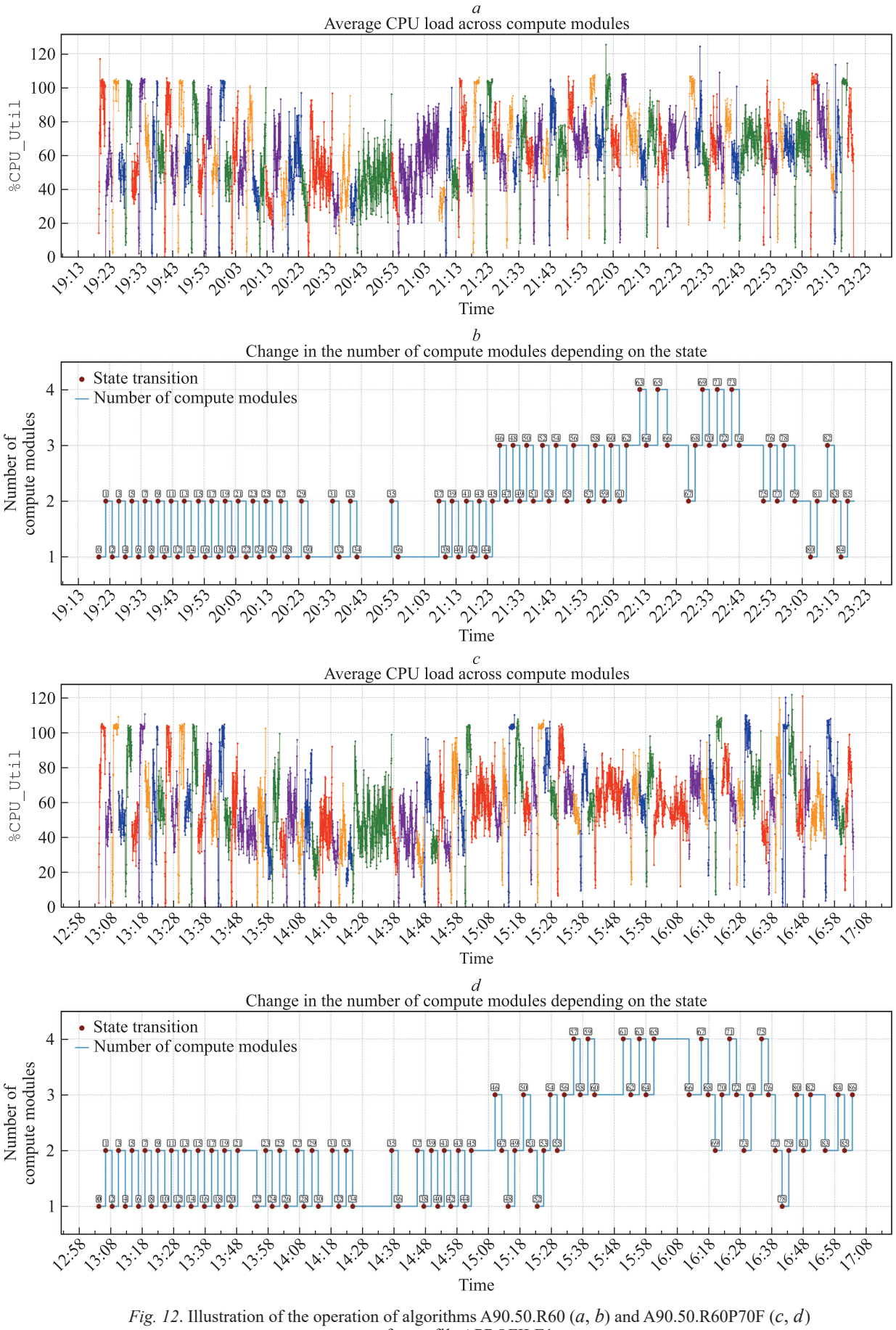

Fig. 12. Illustration of the operation of algorithms A90.50.R60 (*a*, *b*) and A90.50.R60P70F (*c*, *d*) for profile APROFILE1

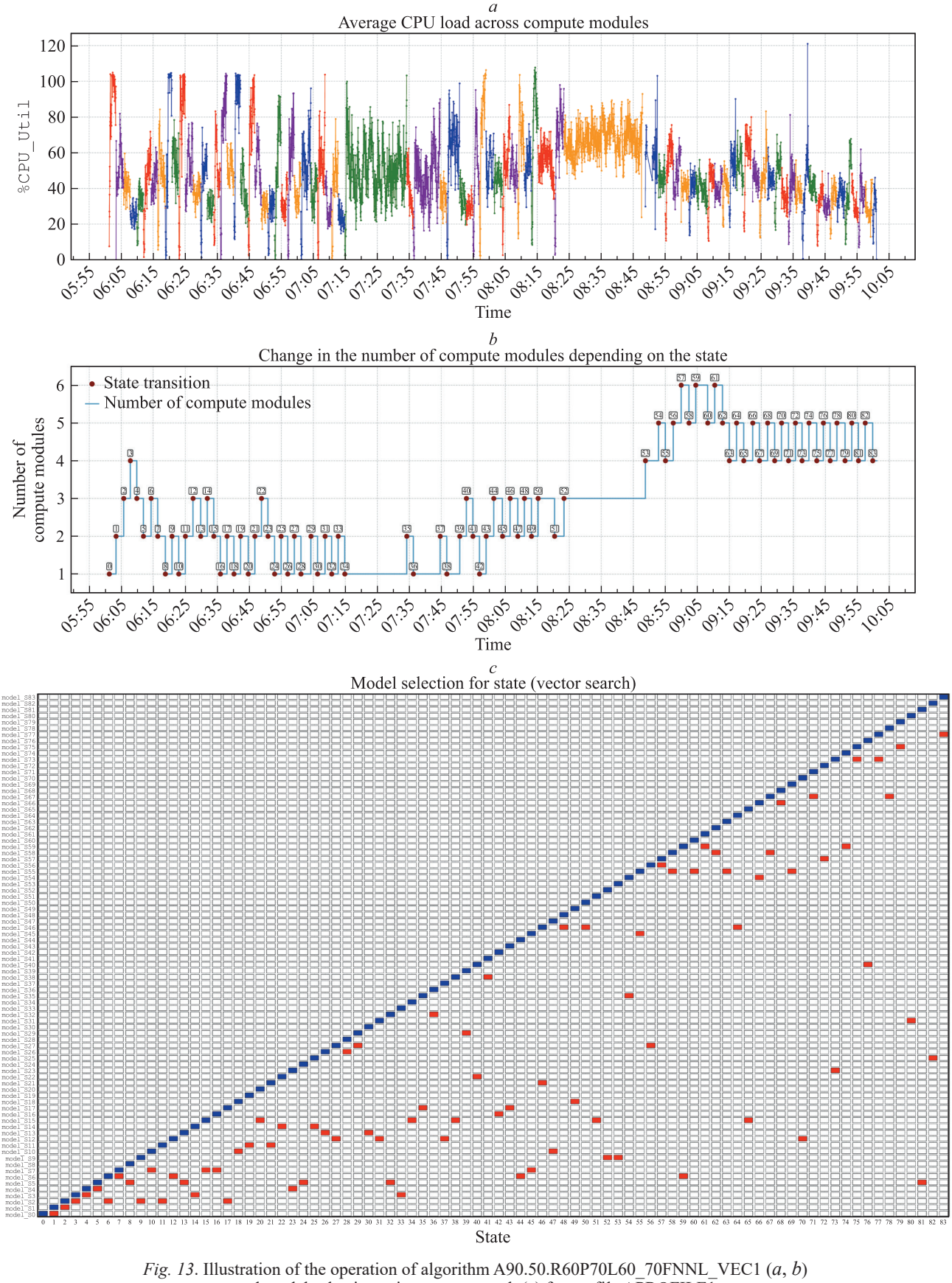

Fig. 13. Illustration of the operation of algorithm A90.50.R60P70L60_70FNNL_VEC1 (*a*, *b*) and model selection using vector search (*c*) for profile APROFILE1

Let us consider the results for the most complex profile APROFILE1. Analysis shows that in terms of the target function $2F_{1/2}$, the best result was shown by the algorithm A90.50.R60P70L60_70FNNL_VEC1. It is approximately 14.4 % better than the reactive algorithm A90.50.R60 and 9.3 % better than the hybrid algorithm A90.50.R60P70F. If comparing by individual metrics, the most resource-efficient is the reactive algorithm A90.50.R60, which is 6.3 % more economical than algorithm A90.50.R60P70F and 39.9 % more economical than algorithm A90.50.R60P70L60_70FNNL_VEC1. In terms of penalties for response time, algorithm A90.50.R60P70L60_70FNNL_VEC1 is the best, 56.6 % better than the reactive algorithm A90.50.R60 and 42.0 % better than the hybrid algorithm A90.50.R60P70F. For the target function $F_{1/3}$ (greater weight towards penalties), the best result is with algorithm A90.50.R60P70L60_70FNNL_VEC1, followed by the algorithm A90.50.R60P70F, whose results were 28.6 and 20.2 % better than the algorithm A90.50.R60. Figure 13, *c*, shows which models were selected from the library using the DLANNLIB_kNM method for various states under load profile APROFILE1. At the beginning, when the library does not contain many models, models for the previous state are predominantly chosen; with further filling of the library, the choice becomes different. It is interesting to note that in earlier studies for the most complex profile APROFILE1, the DLANNLIB_LM method (the simplest variant, where the model for the previous state was selected from the library) was only 5 % better than the reactive one.

The obtained results show that, as in previous studies, algorithms that work well on synthetic data seriously reduce their effectiveness on a complexly structured profile based on real load, but at the same time, new ideas related to complicating the predictor based on using vector search over a model library allow improving the result, even when using the simplest version of the algorithm (where only one model from the library is used to build the predictor, not *k*-nearest models).

Conclusions

The work proposes a new method, increasing the operational efficiency of the control system. The approach implements adaptation to new load patterns, associated with uncertainty, through approximation of data in real time using neural network models. The obtained models are saved into a library together with vector descriptions of load patterns, which allows subsequently promptly selecting suitable models for new system states using vector search by semantic similarity and building an ensemble predictor.

The results of experiments show that algorithms that work well on synthetic loads seriously reduce their effectiveness on a complexly structured profile based on real load. But using a predictor built with vector search allows improving the result, even when using the simplest version of the algorithm.

In terms of the aggregate of indicators when working with irregular and complex load, hybrid algorithms turned out to be more effective than reactive ones. At the same time, the best result was shown by the algorithm using vector search.

References

1. Straesser M, Grohmann J, von Kistowski J, Eismann S, Bauer A, Kounev S. Why is it not solved yet? Challenges for production-ready autoscaling. In: Association for Computing Machinery. *ICPE'22. Proceedings of the 2022 ACM/SPEC International conference on performance engineering; 2022 April 9–13; Bejing, China*. New York: Association for Computing Machinery; 2022. p. 105–115. DOI: 10.1145/3489525.3511680.

2. Krasnoproshin V, Starovoitov A. Real-time management of critical IT service using a multi-model approach. In: Kamil Aida-zade, editor. *2025 6th International conference on problems of cybernetics and informatics (PCI); 2025 August 26–28; Baku, Azerbaijan.* [S. l.]: Institute of Electrical and Electronics Engineers; 2025. p. 1–5. DOI: 10.1109/PCI66488.2025.11219764.

3. Grassberger P, Procaccia I. Measuring the strangeness of strange attractors. *Physica D: Nonlinear Phenomena*. 1983;9(1–2): 189–208. DOI: 10.1016/0167-2789(83)90298-1.

4. Aslanpour MS, Ghobaei-Arani M, Toosi AN. Auto-scaling web applications in clouds: a cost-aware approach. *Journal of Network and Computer Applications*. 2017;95:26–41. DOI: 10.1016/j.jnca.2017.07.012.

Received 24.12.2025 / revised 19.01.2026 / accepted 29.01.2026.