УДК 004.94:004.8
DOI: 10.26102/2310-6018/2026.62.11.004

A parallel computational process model for multi-horizon fault prediction in resource-limited conditions

A. Dahe✉, V.V. Stuchilin

National University of Science and Technology "MISIS", Moscow, the Russian Federation

Abstract. The paper addresses multi-horizon fault forecasting for intelligent monitoring of engineering and industrial systems under computational and response-time constraints. The aim is to develop a coordinated computational method in which a common representation of the current system state is used to generate horizon-specific forecasting trajectories and support subsequent technical-state assessment. The proposed method combines shared input preparation with specialized forecasting trajectories: Seq2Seq-LSTM for 6- and 12-hour forecasting and Temporal Fusion Transformer (TFT) for 24-hour forecasting. After the common representation is formed, the trajectories are executed concurrently and their outputs are temporally aligned for unified system-state analysis. The method is experimentally compared with a sequential baseline using identical models, input data, and parameters on two hardware configurations: an NVIDIA Tesla T4 GPU and a CPU-only platform. The evaluation considers inference latency and forecasting quality. On the NVIDIA Tesla T4, concurrent execution reduces the critical model-inference time from approximately 22 ms to 17 ms, a reduction of approximately 23 %, while preserving the forecasting functionality of the considered trajectories. On the CPU-only platform, the sequential organization requires approximately 453 ms and the parallel organization requires approximately 585 ms, corresponding to a 29 % increase in the critical inference path. The results characterize the investigated computational configurations and provide a basis for low-latency multi-horizon monitoring under resource constraints, while highlighting the hardware-dependent nature of the observed latency reduction.

Keywords: computational model, multi-horizon forecasting, parallel computational process, computational optimization, fault prediction, limited resources, time series, intelligent monitoring, cyber-physical systems.

For citation: Dahe A., Stuchilin V.V. A parallel computational process model for multi-horizon fault prediction in resource-limited conditions. *Modeling, Optimization and Information Technology.* 2026;14(11). (In Russ.). URL: https://moitvivt.ru/ru/journal/article?id=2562 DOI: 10.26102/2310-6018/2026.62.11.004

Модель параллельного вычислительного процесса для многогоризонтного прогнозирования неисправностей в условиях ограниченных ресурсов

А. Дахе✉, В.В. Стучилин

Национальный исследовательский технологический университет «МИСИС», Москва, Российская Федерация

Резюме. В статье рассматривается многогоризонтное прогнозирование неисправностей для интеллектуального мониторинга инженерных и промышленных систем в условиях вычислительных ограничений и ограничений по времени отклика. Цель – разработать скоординированный вычислительный метод, в котором общее представление текущего состояния системы используется для генерации траекторий прогнозирования, специфичных для каждого горизонта, и поддержки последующей оценки технического состояния. Предлагаемый метод сочетает общую подготовку входных данных со специализированными траекториями прогнозирования: Seq2Seq-LSTM для прогнозирования на 6 и 12 часов и Temporal Fusion

© Дахе А., Стучилин В.В., 2026

Transformer (TFT) для прогнозирования на 24 часа. После формирования общего представления траектории выполняются параллельно, а их выходы согласуются по времени для единого анализа состояния системы. Метод экспериментально сравнивается с последовательным базовым вариантом при использовании идентичных моделей, входных данных и параметров на двух аппаратных конфигурациях: GPU NVIDIA Tesla T4 и платформе только с CPU. Оценка учитывает задержку инференса и качество прогнозирования. На NVIDIA Tesla T4 параллельное выполнение сокращает критическое время инференса моделей с примерно 22 мс до 17 мс, что соответствует сокращению примерно на 23 %, сохраняя функциональность прогнозирования рассматриваемых траекторий. На платформе только с CPU последовательная организация требует примерно 453 мс, а параллельная – примерно 585 мс, что соответствует увеличению критического пути инференса на 29 %. Результаты характеризуют исследуемые вычислительные конфигурации и создают основу для низколатентного многогоризонтного мониторинга в условиях ограниченных ресурсов, подчеркивая аппаратно-зависимый характер наблюдаемого сокращения задержки.

Ключевые слова: вычислительная модель, многогоризонтное прогнозирование, параллельный вычислительный процесс, вычислительная оптимизация, прогнозирование отказов, ограниченные ресурсы, временные ряды, интеллектуальный мониторинг, киберфизические системы.

Для цитирования: Дахе А., Стучилин В.В. Модель параллельного вычислительного процесса для многогоризонтного прогнозирования неисправностей в условиях ограниченных ресурсов. *Моделирование, оптимизация и информационные технологии.* 2026;14(11). URL: https://moitvivt.ru/ru/journal/article?id=2562 DOI: 10.26102/2310-6018/2026.62.11.004

Introduction

Modern cyber-physical systems operating in continuous monitoring mode are widely used in industry, energy, transport infrastructure, and engineering facilities. Their operation is based on the processing of telemetry data streams generated by distributed sensor networks, which makes it possible to assess the current technical state of equipment and detect signs of degradation [1]. Under these conditions, fault forecasting is an important component of intelligent monitoring systems because it makes it possible to identify potentially hazardous changes in advance and support decisions concerning equipment operation and maintenance [2].

From a mathematical point of view, fault forecasting is associated with the processing of multidimensional time series and the construction of a mapping between the observed history of a system and its future states. Machine learning methods, including recurrent neural networks, transformer architectures, and their hybrid combinations, are actively used for this purpose because of their ability to model complex temporal dependencies [3, 4]. At the same time, practical forecasting systems require not only sufficient prediction accuracy but also an appropriate organization of the computational process, particularly when telemetry must be processed continuously under limited computational resources and strict response-time requirements [5, 6]. Since the efficiency of a computational organization depends on the underlying hardware, the same execution scheme may behave differently on GPU-accelerated and CPU-only platforms, which makes cross-platform evaluation necessary.

Most existing studies primarily focus on the development of forecasting models and improvement of their predictive accuracy, whereas the coordinated computational organization of several forecasting horizons and the subsequent use of their results for technical-state assessment are considered to a lesser extent. In a real-time monitoring system, repeated preparation of the same input data and sequential execution of independent forecasting procedures can increase inference latency even when the underlying models provide accurate predictions [7, 8]. Therefore, the computational organization of multi-horizon forecasting

should be considered together with the formation and interpretation of the resulting forecast trajectories, and its behavior should be assessed across different hardware configurations rather than on a single platform.

A particular feature of fault forecasting is the need to analyze the evolution of the technical state at different time scales. Short-term forecasting is required for the early identification of rapidly developing deviations and the generation of operational warning signals, whereas longer-term forecasting is useful for assessing degradation trends and supporting maintenance planning. These tasks have different temporal characteristics and can therefore be represented by specialized forecasting models. In the considered study, the Seq2Seq-LSTM architecture is used for the 6- and 12-hour forecasting trajectories, while the Temporal Fusion Transformer (TFT) is used for the 24-hour trajectory. All trajectories operate on a common representation of the current system state obtained from the same incoming observation window.

A straightforward implementation executes the horizon-specific forecasting procedures sequentially. In this case, the common input preparation is performed once, after which the forecasting trajectories are processed one after another. Such organization makes their execution times part of the same sequential inference path. Because the effectiveness of concurrent execution depends on the underlying parallelism mechanism, the proposed organization is evaluated on two hardware configurations: a GPU-accelerated platform, where concurrency is realized through CUDA kernel scheduling, and a CPU-only platform, where concurrency is realized through thread- or process-level parallelism. However, the use of parallel execution itself is not considered the scientific contribution of this study. The scientific problem is the coordinated formation of multi-horizon forecast trajectories from a common representation of the current system state and their subsequent temporal organization for unified technical-state assessment. This requires the computational process to preserve the correspondence between the common observation, the different forecasting horizons, and the resulting technical-state information.

The proposed approach therefore represents multi-horizon fault forecasting as a coordinated computational process comprising three functionally related stages. First, the incoming multidimensional telemetry data are transformed into a common representation of the current system state. Second, this representation is supplied to specialized forecasting trajectories corresponding to different temporal horizons, which can be executed concurrently. Third, the obtained trajectories are temporally aligned and transformed into a unified representation of the predicted technical state. This representation can then be used to identify the development of potentially hazardous conditions and support operational or maintenance decisions. Thus, computational organization, multi-horizon forecasting, and technical-state assessment are considered as interconnected elements of a single monitoring process, whose structural organization is hardware-independent but whose latency characteristics depend on the execution platform.

The aim of the study is to develop and experimentally evaluate a computational method for coordinated multi-horizon fault forecasting and technical-state assessment that uses a shared representation of the current system state, concurrent execution of horizon-specific forecasting trajectories, and unified organization of their results while reducing inference latency and preserving forecasting quality. The evaluation is performed under identical models, input data, and parameters on two hardware configurations (an NVIDIA Tesla T4 GPU and a CPU-only platform) in order to characterize how the proposed computational organization behaves when the underlying parallelism mechanism changes.

To achieve this aim, the following research tasks are formulated:

1. to formalize the computational process of multi-horizon fault forecasting for multidimensional telemetry data under real-time and computational-resource constraints;

2. to develop a computational scheme that separates common input preparation from horizon-specific forecasting and enables coordinated parallel inference;

3. to define the interaction between the common input representation and the Seq2Seq-LSTM and TFT forecasting trajectories;

4. to implement sequential and parallel versions of the computational process and define the corresponding baseline for experimental comparison;

5. to evaluate the computational efficiency of the proposed organization using latency measurements under identical input and model conditions on two hardware configurations: an NVIDIA Tesla T4 GPU and a CPU-only platform;

6. to evaluate the forecasting quality of the short- and long-term trajectories and determine whether the reduction in computational latency is achieved without a material deterioration in prediction quality;

7. to compare the behavior of the proposed computational organization across GPU-accelerated and CPU-only platforms and to determine whether the observed latency reduction is hardware-dependent.

The object of the research is a computational model for intelligent monitoring of the technical condition of engineering and industrial systems based on multidimensional telemetry data obtained from distributed sensor networks. The subject of the research is the organization of the computational process for coordinated multi-horizon fault forecasting and technical-state assessment, including the formation of a common system-state representation, execution of specialized forecasting trajectories, and unified organization of their results.

The computational problem is defined as obtaining forecasts for several time horizons and transforming them into information suitable for technical-state assessment while satisfying the following constraints:

– continuous processing of multidimensional time-series data;
– limited computational resources within a single computing platform;
– simultaneous availability of forecasts for different time horizons;
– single execution of common data-preprocessing operations for a given input;
– preservation of the forecasting quality of the individual trajectories;
– compliance with the response-time requirements of real-time monitoring;
– applicability across hardware platforms with different parallelism mechanisms (GPU-accelerated and CPU-only).

The scientific contribution of the study consists in a coordinated computational organization for multi-horizon fault forecasting in which a common representation of the current system state is generated once and supplied to heterogeneous forecasting trajectories associated with different temporal horizons, followed by their temporal alignment into a unified multi-horizon representation. Unlike a conventional sequential organization of independent forecasting procedures, the proposed organization separates the common state-representation stage from horizon-specific inference and enables concurrent execution of the forecasting trajectories.

The contribution is therefore not limited to the use of parallel execution as a general computing technique. The proposed organization establishes a functional relationship between common state representation, horizon-specific forecasting, and unified organization of the resulting trajectories. This structure makes it possible to reduce the critical sequential inference path while preserving the correspondence between forecasts at different horizons. The effectiveness of the proposed organization is evaluated by comparing sequential and concurrent implementations under identical forecasting conditions in terms of inference latency and forecasting quality on two hardware configurations: an NVIDIA Tesla T4 GPU and a CPU-only platform. This comparison makes it possible to distinguish properties of the computational organization itself from effects caused by a specific parallelism mechanism.

The practical significance of the study is associated with the application of the proposed computational organization in intelligent monitoring systems that require simultaneous short-term and long-term assessment of equipment condition. The resulting multi-horizon information can be used as a basis for operational warning generation and maintenance-oriented decision support, particularly in systems where computational resources are limited and response time is an important operational constraint. Since many industrial monitoring and edge-computing installations operate without dedicated GPU accelerators, the cross-platform evaluation also indicates the extent to which the proposed organization can be applied in CPU-only environments.

Materials and methods

The proposed method considers multi-horizon fault forecasting as a coordinated computational process in which two specialized forecasting trajectories are generated from a common representation of the current system state. The method combines a shared input-preparation stage, horizon-specific forecasting, temporal organization of the obtained trajectories, and formation of a unified multi-horizon representation for technical-state assessment.

The incoming telemetry data are first subjected to common preprocessing, including data cleaning, gap imputation, feature normalization, and formation of a fixed-length observation window. The resulting observation is transformed into a common input representation once for the current system state and is subsequently supplied to both forecasting trajectories.

The Seq2Seq-LSTM architecture is used for the short-term trajectory, corresponding to the 6- and 12-hour forecasting horizons, whereas the Temporal Fusion Transformer (TFT) is used for the long-term trajectory corresponding to the 24-hour horizon. Both trajectories receive the same prepared representation of the current system state while retaining their respective model architectures and forecasting characteristics.

After the common representation has been formed, the model-specific inference procedures do not depend on the output of one another. They can therefore be executed concurrently. The resulting forecast trajectories are subsequently temporally aligned and organized into a common output structure. This structure preserves the forecast probabilities associated with the different horizons and provides a unified representation for subsequent technical-state assessment.

The proposed computational organization is hardware-independent in its structure but is implemented and evaluated on two hardware configurations in order to characterize how its latency characteristics depend on the underlying parallelism mechanism.

The proposed computational model shown in Figure 1 is implemented as a pipeline consisting of seven functional stages:

1. telemetry data reception and initial validation;
2. data preprocessing;
3. formation of a fixed-length sliding observation window;
4. construction of the common input representation;
5. concurrent execution of the short- and long-term forecasting trajectories;
6. temporal alignment and organization of the obtained forecasting results;
7. formation of a unified multi-horizon output representation for technical-state assessment.

The first stage receives the incoming telemetry stream, performs synchronization, and validates the data. The second stage cleans the data, imputes gaps, and normalizes features, while the third stage constructs the fixed-length observation window required by the forecasting models. The fourth stage transforms this window into a common representation of the current

system state, which serves as the shared input for both forecasting trajectories. At the fifth stage, the process splits into two horizon-specific trajectories: the short-term trajectory uses Seq2Seq-LSTM for 6- and 12-hour forecasts, and the long-term trajectory uses TFT for the 24-hour forecast. Since neither trajectory depends on the other's output during inference, they can be executed concurrently. The concurrency mechanism is platform-dependent: on the GPU configuration, the two trajectories are executed through CUDA kernel scheduling, whereas on the CPU-only configuration, they are executed through thread- or process-level parallelism. Because the trajectories share only the common input representation and exchange no intermediate results, concurrent execution does not alter the forecasting function of either model. The sixth stage temporally aligns the outputs of the two trajectories according to their respective horizons and transfers them to a common output structure without modifying the individual predictions, establishing a unified temporal organization for joint consideration. The seventh stage represents the aligned forecasts as a multi-horizon technical-state vector containing forecast information for the 6-, 12-, and 24-hour horizons, providing a unified output for subsequent technical-state assessment and interpretation. The seven-stage structure thus establishes a complete computational path from incoming telemetry to coordinated multi-horizon technical-state information: common input preparation is performed once, horizon-dependent inference remains specialized, and the resulting forecasts are subsequently organized into a unified representation.

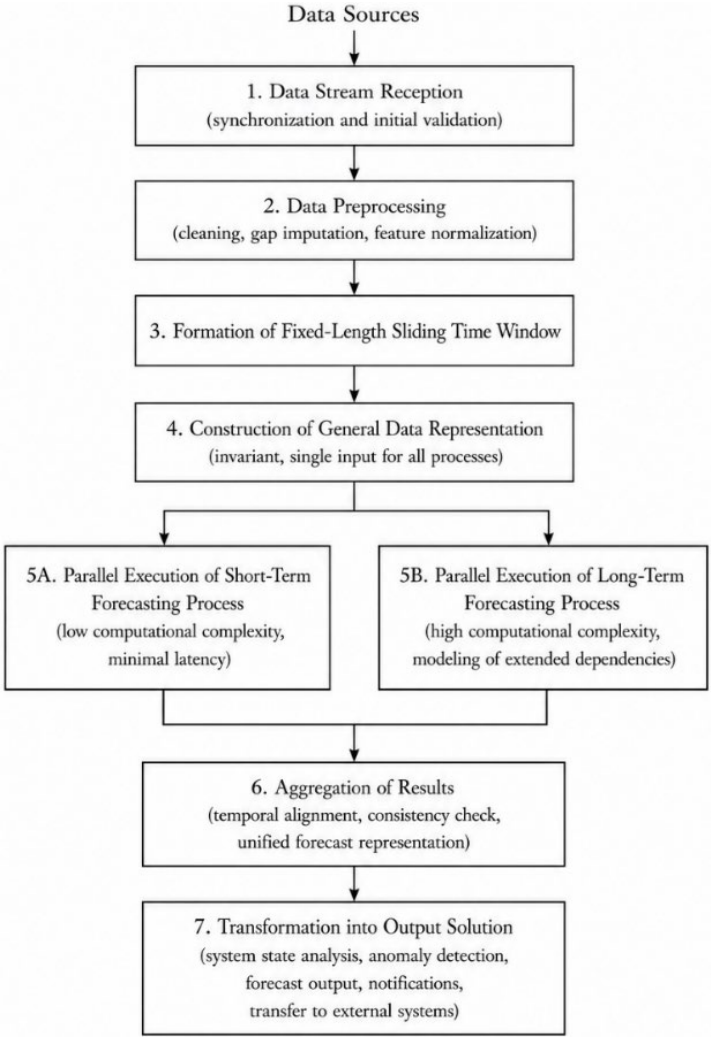


Figure 1 – Computational model for multi-horizon fault prediction
Рисунок 1 – Вычислительная модель для многогоризонтного прогнозирования неисправностей

For experimental evaluation, two computational organizations of the proposed multi-horizon forecasting process are considered: sequential and parallel.

In the sequential organization, the common input preparation is followed by successive execution of the short-term and long-term forecasting trajectories. The Seq2Seq-LSTM and TFT inference procedures therefore belong to the same sequential execution path, and their execution times contribute successively to the total model-inference latency.

In the parallel organization, the common input preparation is performed once, after which the short-term and long-term forecasting trajectories are launched as independent computational tasks. Their outputs are collected after both inference procedures have completed and are then transferred to the common output stage.

The two implementations use identical forecasting models, input data, trained parameters, and output-processing procedures. Within each hardware configuration, the sequential and parallel organizations are executed on the same platform, so the controlled difference between them is the organization of the horizon-specific inference stage rather than a change in the forecasting models. Across the two hardware configurations (GPU and CPU-only), the models, parameters, and input data are also identical; only the execution platform and the mechanism used to realize concurrency differ. The common input preparation and final output organization are identical in both implementations and on both hardware configurations. The comparison therefore isolates the effect of changing the execution organization of the forecasting trajectories within each platform, while the cross-platform comparison characterizes how this effect depends on the underlying parallelism mechanism.

The input to both forecasting trajectories is represented as a three-dimensional tensor of the form (B, L, F), where B is the batch size, L is the observation-window length, and F is the number of input features.

The observation window contains L=48 time steps. With a sampling interval of 15 minutes, the input represents 12 hours of historical system observations. A batch size of 64 is used during inference.

Three prediction horizons are considered: 6, 12, and 24 hours. Seq2Seq-LSTM is used for the 6- and 12-hour horizons, while TFT is used for the 24-hour horizon. Both forecasting trajectories receive the same prepared representation of the current system state.

The Seq2Seq-LSTM architecture consists of an encoder-decoder structure with an additive attention mechanism. The encoder contains one LSTM layer with 64 hidden units. The attention mechanism calculates relevance scores between the decoder state and the encoder hidden states, applies softmax normalization, and forms a context vector as a weighted combination of the encoder states.

The decoder contains one LSTM layer with 64 hidden units and generates probabilistic forecasts for the 6- and 12-hour prediction horizons. A dropout rate of 0.3 is applied for regularization. During training, teacher forcing is used with a decreasing probability to progressively reduce the dependence of the decoder on the ground-truth previous state.

The output layer is implemented as a temporal fully connected layer with sigmoid activation and produces probabilistic estimates of future fault-related states. Gradient clipping with a maximum norm of 1.0 is used to improve training stability. The model is trained using the Adam optimizer with a learning rate of 10^{-3}. Early stopping is based primarily on validation AUPRC, with F1-score used as an auxiliary evaluation criterion and a patience of 10 epochs.

The Temporal Fusion Transformer is used for the long-term forecasting trajectory. The model receives the same 48-step input representation and is configured for probabilistic forecasting over the 24-hour prediction horizon.

The TFT uses a hidden representation dimension of 64. The variable-selection network operates with a representation dimension of 64, and the gated residual networks use a hidden

dimension of 64. The temporal processing component consists of two attention layers with four attention heads. A dropout rate of 0.10 is used for regularization.

The output layer has one output dimension and uses sigmoid activation to produce the probability of the future fault-related state over the 24-hour prediction horizon. The TFT is trained using the AdamW optimizer with a learning rate of 10^{-3} and weight decay of 10^{-4}. Early stopping is based on validation AUPRC with a patience of 10 epochs.

The same trained Seq2Seq-LSTM and TFT configurations are used in both the sequential and parallel implementations on both hardware platforms. Consequently, the computational comparison does not involve changes in the model architecture, trained parameters, or input representation; only the execution organization and the underlying parallelism mechanism differ. Since the trained models, parameters, and input data are identical across the two hardware configurations, the forecasting-quality evaluation is platform-independent and is reported once for both configurations.

The experimental dataset contains more than 500,000 observations generated over a 10-year period with a sampling interval of 15 minutes. The monitored variables include temperature, pressure, relative humidity, flow rate, energy consumption, carbon dioxide concentration, atmospheric pressure, outdoor temperature, and a binary technical-state indicator.

The preprocessing procedure includes restoration of missing observations, removal of invalid or anomalous values according to the defined data-quality rules, feature normalization, and formation of fixed-length temporal windows.

The data are divided chronologically into training, validation, and test subsets. Temporal windows are constructed after the chronological partitioning to prevent observations from the future test period from entering the training data through overlapping windows.

The binary technical-state indicator is constructed from predefined criteria describing abnormal operating conditions in the monitored system. An observation is assigned to the fault state when at least one of the following conditions is satisfied:

- temperature exceeds 24 °C while pressure is below 0.98 bar;
- relative humidity exceeds 60% while flow rate is below 4 L/min;
- carbon dioxide concentration exceeds 1000 ppm;
- atmospheric pressure is below 990 hPa.

If none of these conditions is satisfied, the observation is assigned to the normal state. The resulting binary indicator is used as the ground-truth technical state for training and evaluating the probabilistic fault forecasts.

The computational pipeline is implemented in Python using the PyTorch framework. The common preprocessing and input-representation stages are executed before the model-specific inference tasks are launched. For the GPU configuration, the standard CUDA-enabled PyTorch build is used. For the CPU-only configuration, the CPU-only PyTorch build is used, and the number of worker threads is explicitly controlled to avoid oversubscription when the two forecasting trajectories are executed concurrently.

In the sequential implementation, the prepared input is passed first to the Seq2Seq-LSTM trajectory and subsequently to the TFT trajectory. This organization is identical on the GPU and CPU-only configurations; only the execution times differ.

In the parallel implementation, the two model-specific inference procedures are submitted as independent computational tasks. On the GPU configuration, concurrency is realized through CUDA kernel scheduling within the same process. On the CPU-only configuration, concurrency is realized through process-level parallelism, with each trajectory assigned a fixed number of worker threads to prevent thread oversubscription. In both cases, the outputs are collected after both tasks have completed and are subsequently transferred to the common output stage.

Both forecasting models use the same prepared input representation, model parameters, and test samples in the two experimental organizations and on both hardware configurations. No changes are made to the forecasting models between the sequential and parallel experiments, and no changes are made to the model architecture or trained parameters across the GPU and CPU-only configurations. Only the execution platform and the mechanism used to realize concurrency differ.

The use of concurrent application-level tasks does not imply that every GPU kernel necessarily executes simultaneously, nor that both CPU branches receive uninterrupted access to all available cores. The reported latency therefore characterizes the actual behavior of the implemented computational organization on the specified hardware rather than an idealized degree of parallelism.

GPU configuration. The GPU experiments are conducted in the Google Colab environment using an NVIDIA Tesla T4 GPU with approximately 16 GB of GPU memory and a CUDA-enabled PyTorch build. The same hardware and software configuration is maintained for all sequential and parallel experiments on this platform.

CPU-only configuration. The CPU-only experiments were run on a local CPU-only platform equipped with an 11th Gen Intel Core i7-1165G7 processor (4 physical cores, 8 logical processors) operating at 2.80 GHz, 16 GB of RAM, and a CPU-only PyTorch build. No GPU was used, and all inference was performed on the CPU. To prevent thread oversubscription, the number of worker threads was fixed in both organizations. In the sequential organization, 4 worker threads were assigned to the single execution path. In the parallel organization, the two forecasting trajectories were launched as separate processes, each assigned 2 worker threads, so that the total number of active threads did not exceed the number of available physical cores. The processor was configured to operate in maximum-performance mode to minimize thermal throttling during measurements. The same trained Seq2Seq-LSTM and TFT models, the same input representation, and the same test inputs were used as in the GPU configuration.

The reported latency values characterize the investigated implementation on two hardware configurations (an NVIDIA Tesla T4 GPU and a CPU-only platform) and should be interpreted as platform-specific measurements rather than as hardware-independent performance figures. Additional GPU, CPU, and embedded platforms remain a direction for future work.

Computational latency is measured separately for the sequential and parallel organizations within each hardware configuration, using identical model parameters and test data. Before latency measurement, 20 warm-up inference runs are performed to reduce the influence of one-time model initialization, memory allocation, thread-pool startup, and, on the GPU configuration, CUDA initialization effects. These runs are excluded from the reported statistics. After the warm-up stage, 100 measurement runs are performed for each computational organization on each platform.

Both implementations are evaluated on the same set of test inputs, and the same trained model parameters are used throughout. For each hardware configuration, the hardware and software environment remains unchanged across the sequential and parallel runs.

The measurement interval begins at the model-specific inference stage and ends after the model outputs required for the comparison have been obtained. The common preprocessing stage is measured separately because it is identical in the sequential and parallel organizations. Model loading and one-time initialization operations are excluded from the reported inference latency. For GPU execution, CUDA synchronization is performed before recording the end time of each measurement to ensure that asynchronous GPU operations are included in the measured interval. For CPU execution, the measurement interval ends only after all worker processes of the parallel organization have completed and their outputs have been joined, so that no inference work continues outside the measured interval.

For the 100 measurement runs, the mean latency, median latency, and standard deviation are calculated. The median latency is used as the primary measure of central tendency because it is less sensitive to occasional execution-time outliers, while the mean and standard deviation characterize the variability of the measurements. This procedure is applied identically to the GPU and CPU-only configurations.

The sequential and parallel implementations are compared in terms of their measured model-inference latency within each hardware configuration. The relative reduction in latency is calculated as the percentage difference between the sequential and parallel measurements relative to the sequential latency.

Forecasting quality is evaluated independently for the Seq2Seq-LSTM and TFT trajectories so that computational acceleration can be distinguished from predictive performance. Because the trained models, parameters, and input data are identical across the two hardware configurations, the forecasting-quality evaluation is platform-independent and is reported once for both configurations.

For the binary fault-prediction task, the Seq2Seq-LSTM trajectory is evaluated for the 6- and 12-hour horizons, while the TFT trajectory is evaluated for the 24-hour horizon. Precision, recall, and F1-score are used to evaluate binary fault identification.

Because the forecasting models produce probabilistic estimates, mean absolute error (MAE) and root mean square error (RMSE) are additionally calculated between the predicted fault probability and the corresponding binary technical-state indicator at each forecasted time step. MAE characterizes the average deviation of the predicted probability from the observed technical state, whereas RMSE gives greater weight to larger prediction errors.

The area under the precision-recall curve (AUPRC) is used during model validation because the probabilistic predictions can be evaluated across different decision thresholds. The forecasting-quality metrics are reported separately for the 6- and 12-hour Seq2Seq-LSTM forecasts and the 24-hour TFT forecast.

The parallel and sequential implementations use the same trained forecasting models, and the same trained parameters are used on both hardware configurations. Therefore, no systematic change in forecasting quality is expected as a consequence of changing the execution organization or the execution platform. Any observed difference can be attributed to numerical effects associated with the execution environment rather than to a change in model architecture or parameters. For this reason, forecasting quality is reported once and is not duplicated per platform.

The sequential and parallel implementations use identical datasets, chronological data partitions, input windows, trained model parameters, model architectures, and output-processing procedures. Within each hardware configuration, the only controlled difference between the two experimental organizations is the organization of the horizon-specific inference stage. Across the two hardware configurations, the models, parameters, and input data are also identical, and only the execution platform and the concurrency mechanism differ.

The experimental configuration, including the observation window, prediction horizons, model architectures, optimization parameters, and latency measurement procedure, is kept fixed across the two implementations and the two hardware configurations. Only the execution platform and the concurrency mechanism differ between the GPU and CPU-only configurations.

This experimental design isolates the effect of computational organization within each hardware configuration and prevents changes in predictive modeling, data preparation, or model parameters from being interpreted as effects of concurrent execution. The cross-platform comparison then isolates the effect of the underlying parallelism mechanism, since the computational organization, models, and input data are held constant.

Results and discussion

As described in the previous section, the experiments were performed on two hardware configurations: an NVIDIA Tesla T4 GPU and a CPU-only Intel Core i7-1165G7 platform. Within each configuration, the sequential and parallel organizations were compared under identical experimental conditions. The comparison therefore isolates the effect of the computational organization on inference latency within each platform, while the cross-platform comparison characterizes how this effect depends on the underlying parallelism mechanism.

The computational efficiency was evaluated using the measurement procedure described in the previous section. The per-trajectory inference latency on the GPU configuration is summarized in Table 1.

Table 1 – GPU-only per-trajectory latency
Таблица 1 – Латентность на траекторию только на GPU

Computational trajectory	Horizon (h)	Mean latency (ms)	Median latency (ms)	SD (ms)
Seq2Seq-LSTM	6	2.4	2.3	0.3
Seq2Seq-LSTM	12	2.7	2.6	0.4
TFT	24	16.4	16.1	2.1

To evaluate the effect of the computational organization, the same forecasting trajectories were executed using two schemes: sequential and parallel. In the sequential scheme, the common input representation was first prepared and then the Seq2Seq-LSTM and TFT inference procedures were executed one after another. Therefore, the model-inference latency of the complete forecasting procedure depends on the combined execution time of the two trajectories. In the parallel scheme, the same common input representation was prepared once, after which the Seq2Seq-LSTM and TFT inference tasks were launched concurrently. Consequently, the completion time of the model-inference stage is determined primarily by the longer of the two concurrent trajectories.

Under the GPU configuration, the sequential organization required approximately 22 ms for the complete model-inference stage, whereas the parallel organization required approximately 17 ms. At the same time, the short-term Seq2Seq-LSTM trajectory remained available after approximately 3 ms without waiting for completion of the 24-hour TFT trajectory. Thus, parallel execution reduced the critical inference time required to obtain the complete set of multi-horizon forecasting outputs while preserving the same forecasting models and input data.

Within the GPU configuration, the comparison was performed using identical model parameters, test inputs, and output processing for both computational organizations. Therefore, the observed difference in inference latency is attributable to the organization of model execution rather than to changes in the forecasting models or their input representation. The common preprocessing stage was also kept identical and was performed once in both schemes. The comparative results are summarized in Table 2.

The results indicate that the main computational benefit of the proposed organization is obtained from changing the execution order of the horizon-specific forecasting trajectories rather than from modifying the forecasting models themselves. The Seq2Seq-LSTM trajectory can produce the short-term forecast without waiting for completion of the longer TFT trajectory, while both trajectories continue to use the same representation of the current system state. This organization is therefore particularly relevant when forecasts with different temporal horizons must be produced within the same inference cycle.

Table 2 – Comparison of sequential and parallel organization on the GPU configuration
Таблица 2 – Сравнение последовательной и параллельной организации вычислительного процесса на GPU-конфигурации

Characteristic	Sequential organization	Parallel organization
Common input preparation	Performed once	Performed once
Seq2Seq-LSTM inference	Sequential	Concurrent
TFT inference	Sequential	Concurrent
Model-inference latency	22 ms	17 ms
Short-term trajectory latency	3 ms	3 ms
Forecasting models	Identical	Identical
Input data	Identical	Identical
Model parameters	Identical	Identical

To determine whether this benefit depends on the underlying parallelism mechanism, the same comparison was performed on the CPU-only configuration, as described in the following subsection.

The per-trajectory inference latency on the CPU-only platform is summarized in Table 3. Because no GPU accelerator is available, all inference is performed on the CPU, and the absolute latency values are substantially higher than in the GPU configuration. The relative ordering of the trajectories is preserved: the Seq2Seq-LSTM trajectories remain substantially lighter than the TFT trajectory, which dominates the total model-inference time.

Table 3 – CPU-only per-trajectory latency
Таблица 3 – Латентность на траекторию только на CPU

Computational trajectory	Horizon (h)	Mean latency (ms)	Median latency (ms)	SD (ms)
Seq2Seq-LSTM	6	45	44.5	5.5
Seq2Seq-LSTM	12	52.5	52	7
TFT	24	400	395	35

The sequential and parallel organizations were then compared under the same CPU-only conditions. In the sequential organization, the Seq2Seq-LSTM and TFT trajectories are executed one after another on the same execution path. In the parallel organization, the two trajectories are launched as separate processes, each assigned two worker threads, so that the total number of active threads matches the number of available physical cores. The comparison is summarized in Table 4.

Table 4 – Comparison of sequential and parallel organization on the CPU-only configuration
Таблица 4 – Сравнение последовательной и параллельной организации вычислительного процесса на CPU-конфигурации

Characteristic	Sequential organization	Parallel organization
Common input preparation	Performed once	Performed once
Seq2Seq-LSTM inference	Sequential	Concurrent
TFT inference	Sequential	Concurrent
Model-inference latency	453 ms	585 ms
Short-term trajectory latency	53 ms	76 ms
Forecasting models	Identical	Identical
Input data	Identical	Identical
Model parameters	Identical	Identical

On the CPU-only platform, the parallel organization increased the total model-inference latency. Two factors contributed to this outcome: the TFT trajectory dominates the execution time, and process-level parallelism on a four-core CPU introduces thread contention and process overhead that offset the benefit of concurrent execution. This result characterizes the boundary conditions under which the proposed computational organization is beneficial.

These CPU-only results complement the GPU measurements and make it possible to compare the two hardware configurations directly. The cross-platform comparison is presented in the following subsection.

The two hardware configurations make it possible to determine whether the effect of the proposed computational organization depends on the underlying parallelism mechanism. The comparison is summarized in Table 5.

Table 5 – Cross-platform comparison
Таблица 5 – Кроссплатформенное сравнение

Metric	GPU (NVIDIA T4)	CPU-only
Sequential model-inference latency	22 ms	453 ms
Parallel model-inference latency	17 ms	585 ms
Short-term trajectory latency	3 ms	53 ms
Relative latency reduction	23 %	−29 % (increase)
Dominant trajectory	TFT	TFT
Concurrency mechanism	CUDA kernel scheduling	Process-level parallelism

The cross-platform comparison shows that the parallel organization is beneficial on the GPU configuration but does not provide a comparable benefit on the CPU-only configuration. On the GPU, parallel execution reduced the model-inference latency by approximately 23 %, whereas on the CPU-only platform it increased the total latency by approximately 29 % due to resource contention. This difference is attributable to the underlying parallelism mechanism: on the GPU, concurrent execution overlaps the two trajectories at the device level, whereas on the CPU-only platform, the two processes contend for the same physical cores, and the long-term TFT trajectory becomes the limiting factor. These results characterize the boundary conditions under which the proposed parallel organization is effective: it requires sufficient parallel hardware resources, such as those provided by a GPU, to deliver a latency reduction.

The cross-platform comparison characterizes the boundary conditions of the proposed parallel organization. The following subsections compare the proposed organization with previously reported computational schemes and evaluate the forecasting quality of the considered trajectories.

From the forecasting-quality perspective, the parallel organization does not introduce a change in the trained forecasting models, model parameters, or input data, and the same trained models are used on both hardware configurations. Therefore, the forecasting metrics obtained for the sequential and parallel implementations are expected to remain equivalent within numerical precision across both platforms. The forecasting results for the 6-hour and 12-hour Seq2Seq-LSTM trajectories and the 24-hour TFT trajectory are reported separately in the following subsection.

To assess the practical significance of the proposed computational organization, it can be considered in relation to computational approaches described in previous studies. The literature includes several ways of organizing multi-horizon forecasting processes. One approach uses a unified processing function in which the complete forecasting procedure is

implemented within a single computational process [9]. Another approach uses a computationally intensive multi-horizon forecasting model that performs the analysis and prediction for several temporal horizons within an integrated model structure [10]. A further approach decomposes the forecasting task into several specialized computational procedures that are executed sequentially [11].

These approaches address the multi-horizon forecasting problem from different computational perspectives. The characteristics reported indicate differences in the organization of forecasting stages, computational workload, and the degree of separation between temporal forecasting tasks. However, the approaches reported in these studies were not reimplemented in the present work. Therefore, the comparison with them is considered a literature-based conceptual comparison rather than a direct experimental benchmark.

The proposed organization combines a common input-preparation stage with independent forecasting trajectories corresponding to different temporal horizons. The common representation of the current system state is formed once and subsequently supplied to the Seq2Seq-LSTM trajectory for 6- and 12-hour forecasting and to the TFT trajectory for 24-hour forecasting. Unlike a sequential decomposition in which the forecasting trajectories share the same execution path, the proposed implementation allows the horizon-specific inference procedures to execute concurrently.

The main difference from the approaches is therefore the organization of the computational process rather than the introduction of a new forecasting architecture. The experimental comparison performed in this study uses identical forecasting models, model parameters, and input data for the sequential and parallel implementations within each hardware configuration. Across the GPU and CPU-only configurations, the models and parameters are also identical, and only the execution platform and the concurrency mechanism differ. The measured results consequently provide direct evidence for the effect of changing the execution organization, while the comparison provides contextual information about alternative computational organizations reported in the literature, as shown in Table 6.

Table 6 – Comparative analysis of computational data processing schemes
Таблица 6 – Сравнительный анализ вычислительных схем обработки данных

Computational scheme	Shared preprocessing	Shared input	Parallel exec.	Computational Efficiency	Real time compliance	Multi horizon support
Single unified function	No	No	No	Moderate	Yes	Limited
Computationally intensive multi-horizon model	No	No	No	Moderate	Partial	Yes
Sequential decomposition of computations	Partial	No	No	Moderate	Partial	Yes
Proposed computational model	Yes	Yes	Yes	Conditional	Conditional	Yes

The comparison should not be interpreted as a quantitative ranking of the considered approaches, since the studies use different models, datasets, hardware configurations, and evaluation procedures. Instead, it highlights the structural distinction between the proposed organization and previously described computational arrangements. The table reflects the structural properties of the proposed organization; the quantitative efficiency and real-time

compliance of concurrent execution additionally depend on the hardware platform, as shown by the cross-platform comparison in Table 5.

Forecasting quality was evaluated separately for the three forecasting horizons to determine whether the change in computational organization affected predictive performance as shown in Table 7. The Seq2Seq-LSTM trajectory was evaluated at 6- and 12-hour horizons, while the TFT trajectory was evaluated at the 24-hour horizon. Precision, recall, F1-score, MAE, RMSE, and AUPRC were used to characterize the forecasting performance. Because the trained models, parameters, and input data are identical on both hardware configurations, the forecasting-quality evaluation is reported once and applies to both platforms.

Table 7 – Forecasting quality of the multi-horizon forecasting model
Таблица 7 – Качество прогнозирования многогоризонтной модели прогнозирования

Model	Horizon (h)	Precision	Recall	F1-score	MAE	RMSE	AUPRC
Seq2Seq-LSTM	6	0.987	0.983	0.985	0.041	0.128	0.992
Seq2Seq-LSTM	12	0.976	0.972	0.974	0.057	0.151	0.986
TFT	24	0.961	0.954	0.957	0.084	0.192	0.973

The results show that the Seq2Seq-LSTM trajectory maintained high classification performance at both short-term horizons, with F1-scores of 0.985 and 0.974 for the 6- and 12-hour forecasts, respectively. The decrease at the longer 12-hour horizon is consistent with the increased temporal distance between the observed system state and the predicted fault state. The TFT trajectory achieved an F1-score of 0.957 at the 24-hour horizon, while its MAE and RMSE remained within the range obtained for the considered probabilistic forecasting task.

The AUPRC values provide an additional assessment of the models' ability to distinguish fault-related states under the binary prediction setting. The obtained values indicate that the forecasting trajectories retained high precision-recall performance across the considered horizons. The gradual change in the error measures and classification metrics with increasing forecast horizon is consistent with the increasing difficulty of predicting system-state changes further into the future.

The sequential and parallel implementations use the same trained models, parameters, input data, and output procedures on both hardware configurations. Therefore, neither changing the execution organization nor changing the execution platform alters the forecasting function itself, and the forecasting-quality evaluation is platform-independent and is reported once for both configurations.

On the GPU configuration, sequential execution required approximately 22 ms for the complete model-inference stage, while parallel execution required approximately 17 ms. On the CPU-only configuration, sequential execution required approximately 453 ms, while parallel execution required approximately 585 ms. These results indicate that the benefit of concurrent execution is hardware-dependent. On the GPU, the reduction in critical inference latency is achieved without simplifying or replacing the forecasting models: both organizations use the same Seq2Seq-LSTM and TFT architectures, model parameters, input representation, and test data. The difference therefore arises solely from the way in which the horizon-specific inference procedures are organized after the common input representation has been prepared.

This distinction is important for interpreting the contribution of the proposed method. The contribution is not the use of parallel execution as a general computing technique. Instead, parallel execution is used as part of a coordinated computational organization for heterogeneous forecasting trajectories that operate on the same representation of the current system state but correspond to different temporal horizons. The computational structure therefore links the common representation stage with specialized short- and long-term forecasting trajectories and

allows their execution to be organized according to their different temporal requirements. Unlike a generic parallel pipeline, the proposed organization does not simply aggregate independent branch outputs. The common input representation is a shared state representation, the trajectories are heterogeneous and horizon-specific, and their outputs are temporally aligned into a unified multi-horizon technical-state vector that preserves the correspondence between the common observation and each forecasting horizon. Because the computational reorganization does not require modification of the predictive models, the observed benefit is obtained through execution organization rather than through reducing model complexity or forecasting quality.

The practical relevance of the proposed organization is related to streaming monitoring systems in which information about the current technical state must support forecasts over several temporal horizons. A shared input representation avoids unnecessary reconstruction of the same current-state information for each forecasting trajectory, while concurrent inference allows the trajectories to proceed independently after this representation has been formed. As shown above, the practical benefit of concurrent inference depends on the available hardware: GPU-accelerated platforms can overlap the two trajectories effectively, whereas on CPU-only platforms with limited physical cores, sequential execution may be preferable.

The experimental results should nevertheless be interpreted within the limits of the two investigated configurations. The latency measurements were obtained on an NVIDIA Tesla T4 GPU and on a CPU-only Intel Core i7-1165G7 platform, using the Python and PyTorch implementations described in the Materials and Methods. The reported values therefore characterize these two specific computational environments rather than providing hardware-independent performance measures. In particular, the CPU-only configuration was evaluated on a mobile-class processor with four physical cores, and the observed behavior may differ on CPU platforms with a larger number of cores. Evaluation on additional GPUs, server-class CPUs, and embedded platforms would be required to establish how the computational organization behaves across a wider range of hardware environments.

A second limitation concerns the number of forecasting trajectories. The present implementation contains two model-specific trajectories: Seq2Seq-LSTM for the 6- and 12-hour horizons and TFT for the 24-hour horizon. Increasing the number of concurrent trajectories may introduce additional scheduling and resource-management effects that were not investigated in this study. Future work can therefore consider larger sets of forecasting horizons and adaptive allocation of computational resources.

Conclusion

A computational model for multi-horizon fault forecasting in engineering and industrial systems was developed for streaming data under constrained computational conditions. The proposed organization combines a shared current-state representation with specialized forecasting trajectories: Seq2Seq-LSTM for 6- and 12-hour horizons and TFT for 24-hour forecasting, executed concurrently after common input preparation.

The experimental comparison showed that, on the GPU configuration, parallel execution reduced the complete model-inference latency by approximately 23 %, whereas on the CPU-only configuration it increased by approximately 29 %. The computational reorganization preserves the same forecasting models, parameters, and input data on both platforms.

The main contribution is a coordinated computational organization that connects a shared system-state representation with horizon-specific forecasting trajectories and, under sufficient parallel hardware resources, reduces the critical inference path through concurrent execution. The cross-platform evaluation additionally identifies a boundary condition: on

hardware with limited physical cores, concurrent execution of the two trajectories can increase the critical path, and sequential execution may be preferable.

The results characterize the proposed computational organization on two hardware configurations and identify the hardware conditions under which concurrent execution is beneficial: an NVIDIA Tesla T4 GPU and a CPU-only Intel Core i7-1165G7 platform. Future work will evaluate additional GPUs, server-class and embedded CPU platforms, larger sets of forecasting trajectories, and adaptive resource allocation.

REFERENCES / СПИСОК ИСТОЧНИКОВ

1. Dahe A., Stuchilin V.V. Usage of machine learning algorithms for fault detection and diagnostics in life support systems. *Izvestiya Sankt-Peterburgskogo gosudarstvennogo tekhnologicheskogo instituta (tekhnicheskogo universiteta).* 2024;(70):93–99. (In Russ.).

   Дахе А., Стучилин В.В. Использование алгоритмов машинного обучения для обнаружения и диагностики неисправностей в системах жизнеобеспечения. *Известия Санкт-Петербургского государственного технологического института (Технического университета).* 2024;(70):93–99.
2. Mehdiyev Sh. On monitoring the technical condition and technological safety of functional elements of the cyber-physical infrastructure. In: *Cybersecurity for Critical Infrastructure Protection via Reflection of Industrial Control Systems, 27–29 October 2021, Baku, Azerbaijan*. IOS Press; 2022. P. 18–26. https://doi.org/10.3233/NICSP220029
3. Demidova L., Marchev D. Development of the forecasting model for the complex technical systems' failures time during the proactive maintenance using the recurrent neural networks' technology. In: *Proceedings of the 2020 2nd International Conference on Control Systems, Mathematical Modeling, Automation and Energy Efficiency (SUMMA), 11–13 November 2020, Lipetsk, Russia*. IEEE; 2020. P. 370–374. https://doi.org/10.1109/SUMMA50634.2020.9280781
4. Xiao X., Ma Z., Xu D., Liu J. Remaining useful life prediction method of product based on improved transformer and GRU. In: *Proceedings of the 12th International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering (QR2MSE 2022), 27–30 July 2022, Emeishan, China*. IET; 2022. P. 1725–1729. https://doi.org/10.1049/icp.2022.3115
5. Gulomov Sh.R., Yusupov B.K., Kamilov Sh.Sh.U. Models and algorithms for solving problems associated with large amounts of data in the military sphere. In: *Proceedings of the 2020 International Conference on Information Science and Communications Technologies (ICISCT), 04–06 November 2020, Tashkent, Uzbekistan*. IEEE; 2020. https://doi.org/10.1109/ICISCT50599.2020.9351506
6. Benjelloun S., El Aissi M.E.M., Loukili Y., et al. Big data processing: batch-based processing and stream-based processing. In: *Proceedings of the 2020 4th International Conference on Intelligent Computing in Data Sciences (ICDS), 21–23 October 2020, Fez, Morocco*. IEEE; 2020. https://doi.org/10.1109/ICDS50568.2020.9268684
7. Pan H. Lightweight design and implementation of machine learning models in time series forecasting. *International Scientific Technical and Economic Research.* 2025;3(2):160–171. https://doi.org/10.71451/istaer2533
8. Zhadan A.A., Shynkarenko V.I. Modern methods and tools for working with time series. *Science and Transport Progress.* 2025;(4):142–154. (In Ukrainian).
9. Lim B., Arık S.Ö., Loeff N., Pfister T. Temporal Fusion Transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting.* 2021;37(4):1748–1764. https://doi.org/10.1016/j.ijforecast.2021.03.012

10. García-Nava J.-L., Flores J.J., Tellez V.M., Calderon F. Fast training of a transformer for global multi-horizon time series forecasting on tensor processing units. *The Journal of Supercomputing.* 2023;79(8):8475–8498. https://doi.org/10.1007/s11227-022-05009-x
11. Lazcano A., Sandubete J.E., Jaramillo-Morán M.A. A comparative framework for multi-horizon time series forecasting: neural networks with adaptive preprocessing. *Machine Learning with Applications.* 2025;22:100781. https://doi.org/10.1016/j.mlwa.2025.100781

ИНФОРМАЦИЯ ОБ АВТОРАХ / INFORMATION ABOUT THE AUTHORS

Дахе Ахмад, аспирант, Национальный исследовательский технологический университет «МИСИС»», Москва, Российская Федерация.
e-mail: m2112269@edu.misis.ru

Dahe Ahmad, Postgraduate, National University of Science and Technology "MISIS", Moscow, the Russian Federation.

Стучилин Владимир Валерьевич, кандидат технических наук, доцент, заместитель заведующего кафедрой инфокоммуникационных технологий, Национальный исследовательский технологический университет «МИСИС», Москва, Российская Федерация
e-mail: stuchilin.vv@misis.ru

Vladimir V. Stuchilin, Candidate of Engineering Sciences, Docent, Deputy Head of the Department of Infocommunication Technologies, National University of Science and Technology "MISIS," Moscow, the Russian Federation.

Статья поступила в редакцию 03.08.2026; одобрена после рецензирования 27.10.2026; принята к публикации 10.11.2026.

The article was submitted 03.08.2026; approved after reviewing 27.10.2026; accepted for publication 10.11.2026.