COMPUTER SCIENCES AND INFORMATION TECHNOLOGIES

UDC 629.3.025.2(045)
DOI:10.18372/1990-5548.84.20186

[1]**O. O. Tkhai,**
[2]**N. V. Shapoval**

OPTIMIZING KUBERNETES AUTOSCALING WITH ARTIFICIAL INTELLIGENCE

National Technical University of Ukraine “Ihor Sikorsky Kyiv Polytechnic Institute,” Kyiv, Ukraine
E-mails: [1]darwinwoda@gmail.com, [2]shovgun@gmail.com, ORCID 0000-0002-8509-6886

Abstract—*This study explores how to improve Kubernetes auto-scaling using artificial intelligence based forecasting. The authors emphasize the limitations of traditional, reactive auto-scaling methods that lag behind rapid changes in demand and propose a proactive approach that predicts future resource requirements. The paper presents a framework for integrating artificial intelligence based predictions into the Kubernetes ecosystem to improve operational efficiency and resource utilization. To address the main challenges, the authors focus on improving workload forecasting and mitigating the impact of random fluctuations in Kubernetes performance. To address this issue, they use time-series forecasting models combined with data preprocessing techniques to predict future CPU utilization and thus inform scaling decisions before peaks or troughs in demand occur. The results show that artificial intelligence based forecasting can significantly improve scaling accuracy, reduce latency, and optimize resource utilization in Kubernetes environments. Time-series models are developed and evaluated using real CPU utilization data from a Kubernetes cluster, including RNN, LSTM, and CNN-GRU. The study also explores new architectures such as Fourier Analysis Network and Kolmogorov–Arnold Network and their integration with the transformer model. In general, the proposed approach aims to improve resource efficiency and application reliability in Kubernetes through proactive automatic scaling.*

Keywords—Autoscaling; Kubernets; Kolmogorov–Arnold network; Fourier analysis network; transfomer; time-series forecasting; long short-term memory.

I. INTRODUCTION

Modern cloud-native applications require dynamic resource management to remain both performant and cost-efficient under variable workloads. While Kubernetes offers built-in autoscaling mechanisms, these systems are typically reactive, often lagging behind sudden changes in demand. This paper explores a proactive approach to autoscaling by incorporating AI-driven forecasting into the Kubernetes ecosystem, with the goal of improving responsiveness and resource utilization.

II. PROBLEM STATEMENT

Kubernetes provides robust yet reactive autoscaling strategies through its Horizontal Pod Autoscaler (HPA) and Vertical Pod Autoscaler (VPA) [2] (Fig. 1). The HPA monitors real-time CPU or memory usage and adjusts the number of running pods accordingly, while the VPA adjusts each pod’s resource requests based on observed utilization patterns. These autoscalers operate under the assumption that current or recent resource consumption is an adequate predictor of immediate future demand. In many real-world scenarios, however, this assumption breaks down – leading to delayed reaction to demand surges, cold-start penalties, and inefficient resource use during idle periods.

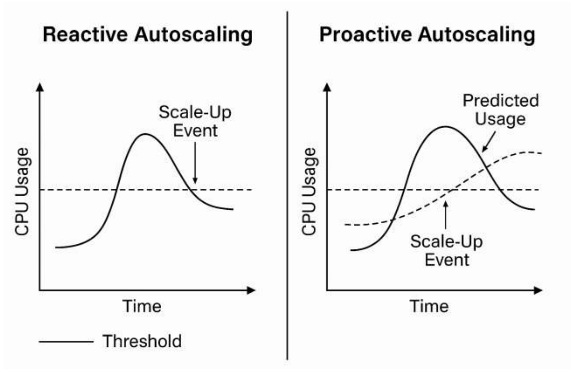

Fig. 1. Reactive vs Proactive autoscaling

Another key limitation lies in the threshold-based nature of these autoscalers. Predefined target values (e.g., 70% CPU utilization) do not dynamically adapt to workload patterns or trends, and tuning them manually is both error-prone and non-scalable across large clusters or microservice architectures [3].

To mitigate these issues, proactive autoscaling has emerged as a promising alternative, aiming to anticipate workload changes and scale resources before thresholds are breached. Integrating artificial intelligence – specifically, time-series forecasting via deep learning – into the Kubernetes autoscaling loop can provide more responsive and efficient scaling strategies. However, this approach presents its own challenges: Kubernetes metrics are inherently noisy and traditional models may struggle with short-term variability [1].

The core problem addressed in this work is the lack of proactive, data-driven autoscaling.

Based on the above, the following problems need to be solved:

1) Predict workload changes before thresholds are breached to implement proactive autoscaling.

2) The metrics (e.g., CPU, memory, latency, etc.) collected by Kubernetes have a lot of random fluctuations (so-called noise), which makes it difficult to identify stable patterns.

By predicting future resource needs rather than reacting to current ones, the system aims to reduce latency in scale-up events, minimize resource overuse, and improve application reliability.

III. Problem Solution

To solve this problem, we propose to utilize time-series forecasting models, enhanced by data preprocessing techniques, to predict future CPU usage and inform scaling decisions before demand spikes or drops occur. The proposed solution can be presented as follows (Fig. 2).

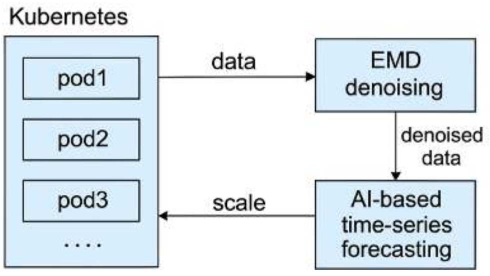

Fig. 2. Proactive autoscaling diagram

A. *Data Preparation and Preprocessing*

We used the anonymized CPU usage dataset collected from a production Kubernetes cluster between January 10, 2022, and January 25, 2022 [4]. This real-world data exhibits the typical characteristics of cloud-native application workloads, including variability, seasonality, and noise.

The dataset underwent a series of processing steps to extract relevant features and improve signal quality.

Timestamp Handling. The raw timestamp data was converted into a standard datetime format. This allowed for the creation of temporal features crucial for capturing cyclical patterns. Specifically, we derived an is_weekend binary feature and a time_of_day categorical feature, later encoded numerically. A normalized timestamp representing elapsed seconds was also included.

One-Hot Encoding. The categorical time_of_day feature was transformed using one-hot encoding, creating distinct binary columns for each time category (morning, day, evening, night). This prevents the models from assuming an ordinal relationship between the categories.

Resampling. The data was resampled to a fixed hourly frequency. For the continuous cpu_usage, the median value within each hour was taken as the aggregate. Median can extract a clean signal reflecting normal behaviour which is suitable for modeling or anomaly detection, where high outliers would affect learning or inference. For the newly created features (is_weekend, one-hot encoded time_of_day, and timestamp_norm), the first value within the hour was used.

Normalization. A normalized version of the timestamp (timestamp_norm) was calculated based on the elapsed seconds from the beginning of the dataset. This provides a continuous, scaled representation of time.

Denoising. In order to handle outliers and noises, we made use of denoising approaches to provide a more smooth forecast. Since we have data that is not periodic in nature, we suggest applying the empirical mode decomposition (EMD) to the cpu_usage time series. Treating the raw CPU usage as a one-dimensional signal, we decomposed it into K intrinsic mode functions (IMFs) and then reconstructed a denoised trace by omitting the first two high-frequency components. This procedure effectively suppresses rapid fluctuations while preserving the underlying trend. The original and denoised signals (first 150 samples) are compared in Fig. 3, demonstrating a noticeably smoother input for our forecasting models [14], [15].

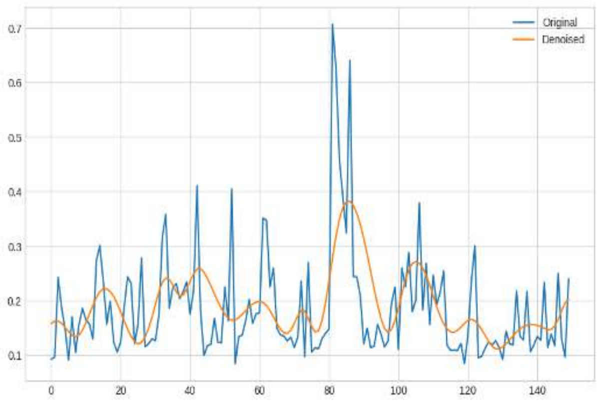

Fig. 3. Original vs Denoised signal

Unknown Periodicity Handling. Empirical mode decomposition is a method that breaks down a signal – like CPU usage – into several underlying patterns, called IMFs, without needing to assume specific time cycles such as hourly or daily trends. Empirical mode decomposition works by repeatedly separating the signal into components that each represent different types of fluctuations, from quick variations to slower, longer-term changes. By analyzing how much energy is present in each of these components, we can identify and retain important repeating patterns that occur at any time scale. This helps forecasting models discover and use hidden timing patterns in the data that don't necessarily follow regular calendar-based cycles.

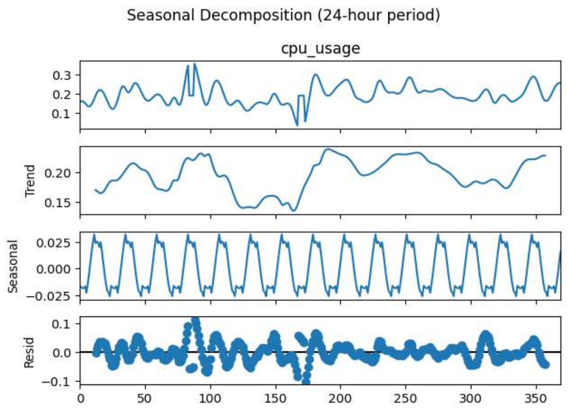

Fig. 4. Seasonal decomposition of CPU usage time series using a 24-hour period

Rolling Statistics. To provide the models with context about recent workload trends and variability, rolling statistics were calculated for the cpu_usage using a 6-hour window. This included the rolling mean, standard deviation, minimum, and maximum CPU usage. These features help the models capture short-term dynamics and volatility.

B. Model Architectures

We implemented and tested a range of time-series forecasting models. The goal was to compare their forecasting accuracy and suitability for proactive autoscaling. We focused on such neural network architectures that have proven to be good at time series forecasting, such as recurrent neural networks (RNN, LSTM, GRU). We also explored such newer architectures as Fourier analysis network (FAN) and Kolmogorov–Arnold network (KAN). These networks show promising results in predicting complex periodic and non-periodic dependencies

Recurrent Neural Network (RNN). The model utilized a standard RNN architecture. While simple, RNNs are capable of modeling sequential data by maintaining a hidden state across time steps [6].

Long short-term memory (LSTM). The LSTM network extended the RNN by introducing memory cells and gating mechanisms, enabling the model to capture long-range temporal dependencies more effectively [7].

Gated recurrent units (GRU). The GRU model provided a more compact alternative to LSTM by combining the forget and input gates into a single update gate, reducing computational overhead while preserving the ability to model long-term dependencies [8].

CNN_GRU. To enhance feature extraction, we combined a 1D convolutional neural network (CNN) with a GRU layer. The CNN component acted as a local feature extractor, identifying short-term patterns in CPU usage. The extracted features were then passed to a gated recurrent unit (GRU) layer to model temporal dependencies [9].

Convolutional fourier analysis network (CFAN). The model extends traditional forecasting architectures by integrating CFAN [11] with a GRU layer to better capture complex CPU usage patterns. The model has three components: CNN for local pattern extraction, a FAN [10] for frequency-domain representation learning, and a GRU for sequential modeling. ConvFAN blocks – convolutional layers with GELU, sine, and cosine activations to extract both nonlinear and periodic features from input sequences. These features are passed to a GRU layer that models temporal dependencies. Static time-based features (e.g., time of day) are processed through FAN layers, which apply linear projections followed by sine, cosine, and GELU activations to encode periodicity. The outputs of the GRU and static branches are concatenated and passed through another FAN-based fully connected layer to produce the final CPU usage prediction.

Transformer FAN. The model integrates FAN with Transformer architecture to capture both periodic and non-periodic patterns. A Transformer encoder analyzes temporal dependencies in CPU data using multi-head attention, identifying critical patterns (e.g., spikes, trends). Instead of standard multi layer perceptron (MLP) layers, the model uses FAN layers (Fig. 5). At its core, the FAN layer decomposes computations into periodic components (sine and cosine transformations) and non-periodic components (GELU activation). The Transformer FAN processes sequential metrics via multi-head attention to identify temporal dependencies, enhanced by positional encoding to preserve time-step relationships. After all layers, the final dynamic representation is concatenated with static features and again passed through a final FAN layer.

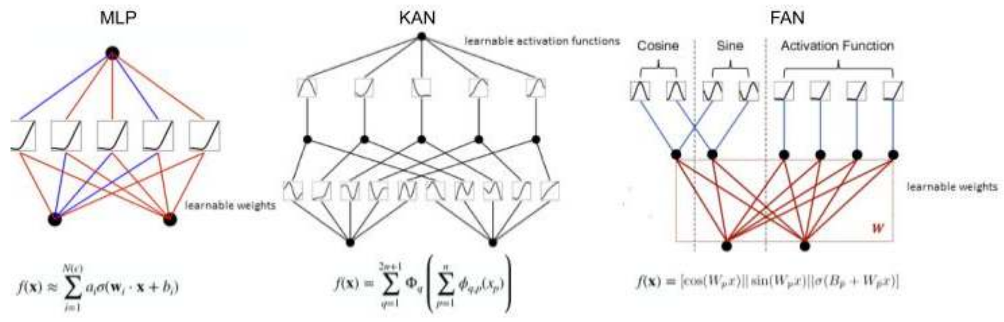
MLP
KAN
FAN
learnable activation functions
Cosine
Sine
Activation Function
learnable weights
learnable weights
W
$f(\mathbf{x}) \approx \sum_{i=1}^{N(\epsilon)} a_i\sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$
$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q\left(\sum_{p=1}^{n} \phi_{q,p}(x_p)\right)$
$f(\mathbf{x}) = [\cos(W_p x)||\sin(W_p x)||\sigma(B_{\bar p} + W_{\bar p} x)]$

Fig. 5. MLP vs KAN vs FAN architecture

By integrating Fourier-based components, the model explicitly learns recurring patterns while retaining flexibility for irregular events, outperforming generic neural networks.

Transformer KAN. In this design, the standard Transformer feed-forward sublayers are replaced by modules from the KAN, a spline-enhanced architecture that learns both smooth trends and sharp nonlinearities [18]. Each encoder layer begins with multi-head self-attention over the embedded CPU usage sequence, followed by a residual connection and layer normalization. After all layers, the time-step embeddings are averaged into a summary vector, concatenated with static features (encoded via a small feed-forward network), and passed through a final head to predict CPU usage. By embedding spline-based nonlinear components into each layer, the Transformer KAN explicitly captures complex, nonperiodic dynamics alongside smooth variations - providing richer expressivity than purely periodic or generic neural approaches [12].

C. *Experiment Results*

The results of the experiment are presented in Tables 1 and 2 and in Figs. 6 and 7.

TABLE I. PREDICTION ON ORIGINAL DATASET

Model	MAE	RMSE
LSTM	0.0431	0.0548
RNN	0.0386	0.0523
GRU	0.0413	0.0532
CNN_GRU	0.0394	0.0508
Transformer FAN	**0.0386**	**0.0498**
Transformer KAN	0.0385	0.0503

TABLE II. PREDICTION ON DENOISED DATASET

Model	MAE	RMSE
LSTM	0.028	0.0339
RNN	0.0212	0.0251
GRU	0.015	0.0188
CNN_GRU	0.0074	0.0095
Transformer FAN	0.0069	0.0084
Transformer KAN	**0.004**	**0.0052**

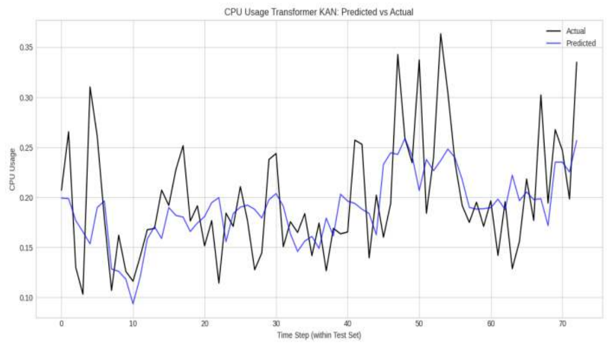

Fig. 6. CPU Predictions of Transformer KAN on original dataset

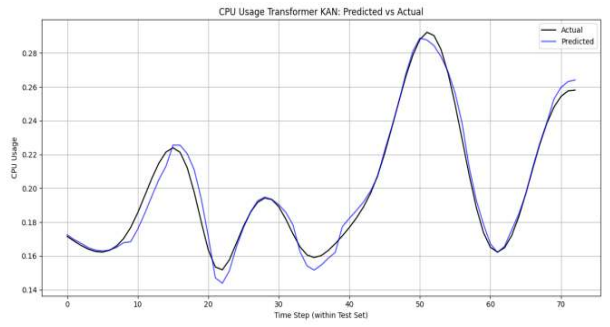

Fig. 7. CPU Predictions of Transformer KAN on denoised dataset

The denoising process enhanced performance of all models. On the original dataset, models like LSTM and GRU struggled with noise, yielding modest MAE scores (> 0.04). On the denoised dataset, the Transformer KAN dominated with near-perfect metrics (MAE = 0.004). Notably, simpler models like the GRU achieved exceptional gains (MAE = 0.015 vs. 0.0413).

IV. CONCLUSION

The study investigated the potential of using artificial intelligence to optimize Kubernetes autoscaling by forecasting future resource usage. We developed and evaluated a range of time series models, incorporating EMD-based denoising, feature engineering, and rolling statistics. These methods improved prediction accuracy, with the Transformer

KAN model achieving MAE score of 0.004 on the denoised dataset.

These predictions can serve as a foundation for proactive autoscaling strategies within Kubernetes. Instead of reacting to threshold breaches, a system informed by forecasted resource demands can initiate scale-out or scale-in actions in advance, reducing latency during traffic spikes and avoiding unnecessary over-provisioning during idle periods. This approach enables more efficient resource utilization, potentially reducing the infrastructure cost.

REFERENCES

[1] AI-Powered Predictive Scaling in Kubernetes: Reducing Cloud Costs While Maintaining High Availability. URL: https://dev.to/sarthakkarora/ai-powered-predictive-scaling-in-kubernetes-reducing-cloud-costs-while-maintaining-high-4ah0.

[2] Autoscaling Workloads. URL: https://kubernetes.io/docs/concepts/workloads/autoscaling/.

[3] Autoscaling in Kubernetes: Why doesn't the Horizontal Pod Autoscaler work for me? URL: https://medium.com/expedia-group-tech/autoscaling-in-kubernetes-why-doesnt-the-horizontal-pod-autoscaler-work-for-me-5f0094694054.

[4] Saibot. GitHub - saibot94/cpu-dataset-prometheus: Anonymized CPU usage dataset. GitHub. https://github.com/saibot94/cpu-dataset-prometheus.

[5] A Angel, (2024). Denoising data with Fast Fourier Transform - using Python. https://medium.com/@angelAjcabul/denoising-data-with-fast-fourier-transform-277bc84e84a4.

[6] Gábor Petneházi: Recurrent Neural Networks for Time Series Forecasting. *arXiv preprint arXiv:1901.00069,* 2019.

[7] L. Nashold and R. Krishnan, (2020). Using LSTM and SARIMA Models to Forecast Cluster CPU Usage. https://cs229.stanford.edu/proj2020spr/report/Nashold_Krishnan.pdf.

[8] Sharmasaravanan, (2024). Time-Series Forecasting Using GRU: A Step-by-Step Guide. https://sharmasaravanan.medium.com/time-series-forecasting-using-gru-a-step-by-step-guide-b537dc8dcfba.

[9] Ghani Rizky Naufal and Antoni Wibowo, (2023). Time Series Forecasting Based on Deep Learning CNN-LSTM-GRU Model on Stock Prices. https://doi.org/10.14445/22315381/IJETT-V71I6P215

[10] Yihong Dong, Ge Li, Yongding Tao, Xue Jiang, Kechi Zhang, Jia Li, Jinliang Deng, Jing Su, Jun Zhang, and Jingjing Xu, FAN: Fourier Analysis Networks. *arXiv preprint arXiv:2410.02675,* 2024.

[11] Sam Jeong and Hae Yong Kim, Convolutional Fourier Analysis Network (CFAN): A Unified Time-Frequency Approach for ECG Classification. *arXiv preprint arXiv:2502.00497,* 2025.

[12] Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y. Hou, and Max Tegmark, KAN: Kolmogorov-Arnold Networks. *arXiv preprint arXiv:2404.19756*, 2024.

[13] Kim C. Raath, Katherine B. Ensor, Alena Crivello, and David W. Scott, (2023). Denoising Non-Stationary Signals via Dynamic Multivariate Complex Wavelet Thresholding. https://doi.org/10.3390/e25111546

[14] Mina Kemiha, Empirical mode decomposition and normalshrink tresholding for speech denoising. *arXiv preprint arXiv:1405.7895*, 2014. https://doi.org/10.5121/ijit.2014.3203

[15] Bingze Dai and Wen Bai, Denoising ECG by Adaptive Filter with Empirical Mode Decomposition. *arXiv preprint arXiv:2108.08376*, 2021.

[16] Kunpeng Xu, Lifei Chen, and Shengrui Wang, Kolmogorov-Arnold Networks for Time Series: Bridging Predictive Power and Interpretability. *arXiv preprint arXiv:2406.02496*, 2024.

[17] Xiao Han, Xinfeng Zhang, Yiling Wu, Zhenduo Zhang, and Zhe Wu, Are KANs Effective for Multivariate Time Series Forecasting? *arXiv preprint arXiv:2408.11306*, 2024.

[18] Xingyi Yang and Xinchao Wang, Kolmogorov-arnold transformer. *arXiv preprint arXiv:2409.10594*, 2024.

Received January 09, 2025

Tkhai Olha. Bachelor's degree student.
Department of Artificial Intelligence. Institute of Applied Systems Analysis, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine.
Research area: deep neural network, distributed systems, cloud computing.
E-mail: darwinwoda@gmail.com

Shapoval Nataliia. ORCID 0000-0002-8509-6886. Candidate of Sciences (Engineering).
National Technical University of Ukraine "Ihor Sikorsky Kyiv Polytechnic Institute," Kyiv, Ukraine.
Education: Kyiv Polytechnic Institute, Kyiv, Ukraine, (2010).

Research interests: computer vision, fuzzy neural network, deep neural network.
Publications: 11
E-mail: shovgun@gmail.com

О. О. Тхай, Н. В. Шаповал. Оптимізація автоскейлінгу в Kubernetes за допомогою штучного інтелекту

У роботі досліджено, як покращити автоматичне масштабування Kubernetes за допомогою прогнозування на основі штучного інтелекту. Автори підкреслюють обмеження традиційних реактивних методів автоматичного масштабування, які відстають від швидких змін попиту, і пропонують проактивний підхід, який передбачає майбутні потреби в ресурсах. У статті представлено фреймворк для інтеграції прогнозів на основі штучного інтелекту в екосистему Kubernetes для підвищення операційної ефективності та використання ресурсів. Щоб вирішити основні проблеми, автори зосереджуються на покращенні прогнозування робочого навантаження та пом’якшенні впливу випадкових коливань на продуктивність Kubernetes. Для вирішення цієї проблеми вони використовують моделі прогнозування часових рядів у поєднанні з методами попередньої обробки даних, щоб передбачити майбутнє завантаження процесорів і, таким чином, інформувати про рішення щодо масштабування до того, як виникнуть піки або спади попиту. Результати показують, що прогнозування на основі штучного інтелекту може значно підвищити точність масштабування, зменшити затримки та оптимізувати використання ресурсів у середовищі Kubernetes. Моделі часових рядів розроблені та оцінені з використанням реальних даних про використання процесорів кластера Kubernetes, включаючи RNN, LSTM та CNN-GRU. Дослідження також досліджує нові архітектури, такі як мережа аналізу Фур’є та мережа Колмогорова–Арнольда, та їх інтеграцію з трансформаторною моделлю. В цілому, запропонований підхід спрямований на підвищення ефективності використання ресурсів та надійності додатків в Kubernetes за рахунок проактивного автоматичного масштабування.

Ключові слова: автомасштабування; Kubernetes; мережа Колмогорова–Арнольда; мережа аналізу Фур’є; трансфомер; прогнозування часових рядів; до́вга короткоча́сна па́м’ять.

Тхай Ольга Олексіївна. Студентка бакалаврату.
Кафедра штучного інтелекту, Інститут прикладного системного аналізу, Національний технічний університет України «Київський політехнічний інститут ім. Ігоря Сікорського», Київ, Україна.
Напрям наукової діяльності: глибокі нейронні мережі, розподілені системи, хмарні обчислення.
E-mail: darwinwoda@gmail.com

Шаповал Наталія Віталіївна. ORCID 0000-0002-8509-6886. Кандидат технічних наук.
Національний технічний університет України «Київський політехнічний інститут ім. Ігоря Сікорського», Київ, Україна.
Освіта: Київський політехнічний інститут, Київ, Україна, (2010).
Напрям наукової діяльності: комп’ютерний зір, нечіткі нейронні мережі, глибокі нейронні мережі.
Кількість публікацій: 11.
E-mail: shovgun@gmail.com