Journal of Technology Research (JTR)
Volume 4, Special Issue, (2026), pp 880-897, ISSN 3005-639X
The 7th Conference of Engineering Sciences and Technology
(CEST-2025)

Cross-Platform Performance Prediction of Software Applications Using Machine Learning Models for Optimized Resource Utilization

Abraheem Mohammed Alsubayhay *[1] , Mohamed A. E. Abdalla [2] , Geber Khalifa Gaber [3] , Faitouri A Aboaoja[4]

[1] Dept. of Computer Science, Faculty of Arts and Sciences, University of Benghazi, Solouq, Libya,

[2] Dept. of Software Engineering, Faculty of Information Technology University of Benghazi, Benghazi, Libya,

[3] Dept of Computer technologies, Higher Institute to science and technology, suluq, Libya,

[4] Dept. of Computer Science, Faculty of Sciences, University of Derna, Derna, Libya,

*Corresponding author email: abraheem.alsubayhay@uob.edu.ly.

Received: 16-10-2025 | Accepted: 20-11-2025 | Available online: 03-01-2026| DOI:10.26629/jtr.2026.**

ABSTRACT

Efficient cross-platform performance prediction and resource optimization are critical for deploying software applications in heterogeneous computing environments. Existing models often struggle to capture complex software–hardware dependencies and evolving workload dynamics, resulting in limited prediction accuracy and poor adaptability. To overcome these limitations, this study introduces a unified intelligent framework that seamlessly integrates Platform-Aware Graph Attention (PAGA), Cross-Platform Temporal Memory (CPTM), and NSGA-II optimization. This tri-layer integration uniquely combines structural dependency learning, temporal sequence understanding, and multi-objective optimization to deliver adaptive and generalizable performance prediction across diverse hardware platforms. The model is implemented in Python using PyTorch for deep learning components and NumPy/Matplotlib for analysis. Experiments are conducted on a cloud performance dataset (CPU, memory, network, energy, execution time, instruction count) from Kaggle. The proposed framework achieves high prediction accuracy with MAE = 0.087, RMSE = 0.132, and $R^2 = 0.97$, marking a 20–25% improvement over baseline models. In the optimization stage, NSGA-II achieves 26.8% execution time reduction, 23.5% energy saving, and 17.4% memory utilization improvement. These results highlight the novelty and effectiveness of the integrated PAGA–CPTM–NSGA-II architecture, demonstrating its potential for scalable, resource-efficient, and cross-platform software performance management in real-world deployments.

Keywords: Cross-Platform Performance Prediction, Energy-Efficient Computing, Meta-Learning, Resource Utilization Optimization, Software Performance Modelling

التنبؤ بأداء تطبيقات البرمجيات عبر المنصات باستخدام نماذج التعلم الآلي لتحقيق استخدام أمثل للموارد

إبراهيم محمد سليمان الصبيحي[1]، محمدعبد الله المنفي[2]، جبر خليفة جبر [3] ، فيتوري عوض بوعوجة[4]

[1]قسم علوم الحاسوب ، كلية الآداب والعلوم ، جامعة بنغازي ، سلوق، ليبيا

[2]قسم هندسة البرمجيات ، كلية تقنية المعلومات ، جامعة بنغازي ، بنغازي ، ليبيا

3 قسم تقنيات الحاسوب ، المعهد العالي للعلوم والتقنية ، سلوق ، ليبيا

4 قسم علوم الحاسوب ، كلية العلوم، جامعة درنة ، درنة ، ليبيا

ملخـــــص البحــــــث

التنبؤ بكفاءة الأداء عبر المنصات المتنوعة وتحسين الموارد يعدان من العوامل الحاسمة لنشر تطبيقات البرمجيات في بيئات الحوسبة غير المتجانسة. تعتبر الطرق الحالية بما في ذلك نماذج التعلم الآلي التقليدية مثل: **Random Forest، LSTM، CNN-LSTM**. غالبًا ما تفشــل في التحقق من التفاعلات المعقدة بين البرمجيات والأجهزة وديناميكيات عبء العمل الزمنية، مما يؤدي إلى الحصــول على دقة تنبؤ غير مثالية. لمعالجة هذه القيود، نقترح في هذه الورقة إطار عمل جديد PAGA–CPTM–NSGA-II يدمج طبقة Platform-Aware Graph Attention (PAGA) لنمذجة الاعتمادات الهيكلية وطبقة الذاكرة الزمنية العبر-منصات **(CPTM)** لالتقاط التطور الزمني الديناميكي لمقاييس الأداء، وخوارزمية الترتيب الجينية غير المُهيمنة الثانية (NSGA-II) لتحســـين متعدد الأهداف لوقت التنفيذ، اســـتخدام الذاكرة، واســـتهلاك الطاقة. يُنفذ النموذج باســـتخدام Python مع PyTorch للمكونات العميقة و NumPy/Matplotlib لمعالجة البيانات والتصـــور. أُجريت التجارب على مجموعة بيانات لمقاييس أداء الحوســـبة الســـحابية: (**CPU**، الذاكرة، الشـبكة، الطاقة، وقت التنفيذ، عدد التعليمات). يحقق الإطار المقترح دقة تنبؤ عالية بمعدل خطأ مطلق (MAE) قدره 0.087، وجذر متوسـط مربع الخطأ (RMSE) قدره 0.132، ومعامل تحديد **(R^2)** يبلغ 0.97، مما يمثل تحسـناً بنسـبة تتراوح بين 20% و25% مقارنة بالنماذج الأساسية. وفي مرحلة التحسين، يحقق خوارزم NSGA-II انخفاضاً في زمن التنفيذ بنسبة 26.8%، وتوفيراً في استهلاك الطاقة بنسبة 23.5%، وتحسناً في استخدام الذاكرة بنسبة 17.4 **%.** تُظهر هذه النتائج قدرة الإطار على التنبؤ بكفاءة الأداء وتوجيه نشـر الموارد بشـكل واعٍ resource-aware **عبر** المنصــات غير المتجانســة. توفر الطريقة المقترحة حلاً قويًا وقابلًا للتوسع وقابلًا للتفسير، مع خلق فرص للبحث المستقبلي في تحسين البرمجيات في الوقت الحقيقي بشكل تكيفي.

الكلمات الدالة: التنبؤ بأداء الأنظمة عبر المنصات ، الحوسبة ذات الكفاءة في استهلاك الطاقة، التعلم فوقي (الميتا - تعلم) ، تحسين استخدام الموارد ، نمذجة أداء البرمجيات .

1. INTRODUCTION

The rapid growth of cloud computing and non-homogeneous hardware has rendered forecasting software performance in the variety of platforms extremely complex [1]. Change in CPU capacity, memory hierarchy, and network architecture leads to variation in the execution time, energy consumption and usage of resources [2]. Conventional rule-based prediction systems do not represent such dynamic software-hardware interactions and therefore fail to give accurate predictions and waste resources [3], [4]. Dependence on other modules and temporal variation are additional complicates of performance analysis as the workloads change and modules of software interact with various hardware components [5], [6]. Machine learning can provide a powerful answer through learning an irregular dependence between system behaviour and workloads [7], [8]. Inter-module dependencies can be represented by graph-based and temporal models and the changing performance trends with time, enhancing the accuracy of prediction [9], [10]. Furthermore, to make deployment optimal, one should balance several goals, e.g., to minimize the time of execution, the use of memory, and usage of energy [11]. Multi-objective optimization algorithms are able to find effective configurations that optimize performance and limits overhead [12], [13]. The generalization of models to unseen environments can also be guaranteed through cross-platform adaptability.

1.1 Research Motivation

The existing models of performance prediction are not applicable to modern computing environments because they do not emulate the dynamic interaction between software and hardware layers, resulting in a failure to estimate the execution time and allocate resources efficiently. This is a weakness that restricts system optimization and scalability of the heterogeneous platforms. The practical significance of this research is to create the machine learning-based predictive model that will help accurately predict software performance in various platforms to allow optimal use of the resources and could lead to a higher efficiency of operations in the real-life setting of the computing process.

1.2 Significance of the Study

This study is important because it deals with such a severe problem as the necessity of precise cross-platform performance forecasts in contemporary computing ecosystems. The proposed model combines the graph neural networks and the temporal memory layers to help to capture the intricate interactions of software modules and hardware components across time. Multi-objective optimization is built into the deployment decisions to have the effect of balancing the execution time, energy consumption, and memory usage to result in efficient resource utilization. The results of this study can be used to create more sophisticated and responsive performance prediction models and help to improve the field of cloud computing as well as heterogeneous systems management.

1.3 Problem Statement

The fast evolution of heterogeneous computing platforms and complex computing workloads have rendered effective performance prediction to be a challenging task. The current research usually uses a set of static benchmarks or regression-based models, which do not represent dynamic software-hardware interactions and changes in execution patterns over time, resulting in inefficient resource utilization and suboptimal deployment choices [14], [15]. This study manages to address them by introducing a unified machine learning framework that incorporates both graph-based modelling and temporal memory networks, along with cross-platform adaptation and multi-objective optimization. The strategy allows specific forecasting of the execution time, energy, and memory consumption, which will guarantee effective resource management on a variety of computing systems.

1.4 Key Contributions

- A new deep learning-based hybrid framework, PAGA–CPTM–NSGA-II, is proposed for accurate cross-platform performance prediction and optimization of software applications.
- Introduces a PAGA Layer to effectively capture complex software–hardware interaction patterns and spatial dependencies among computational nodes.
- Develops a CPTM Layer to model temporal variations in workload and resource utilization, enhancing predictive adaptability across heterogeneous platforms.
- Employs NSGA-II to optimize performance metrics such as execution time, energy consumption, and memory utilization, yielding Pareto-optimal deployment solutions.

2. RELATED WORK

Amaris et al. [15] and Pintye et al. [14] examined machine learning-based mechanisms of predicting performance and optimizing resources in a heterogeneous computing platform. Amaris et al. used analytical modeling and Random Forest and Support Vector Regression to accurately estimate the time of GPU kernel executions based on performance metrics of benchmarks with RMSE of 0.21 and R- squared equal to 0.94. Pintye et al. proposed a reinforcement learning-based framework of cloud autoscaling which was trained on the CPU, memory, and latency measurements of the public cloud traces and enhanced the utilization of resources by 18% and minimized the overhead by 12%. Taken together, these works demonstrate that hybrid analytical-machine learning models can enhance accuracy and efficiency in the context of both GPU and cloud environments, but it is still difficult to sustain generalization in the case of a wide range of workloads.

De Filippo et al. [16] suggested a machine learning method to estimate the execution time of the COSMO meteorological simulation software. Data on historical performance in simulation and execution logs were used to train regression and ensemble models in the study. Random Forest and Gradient Boosting gave good results, with MAE of 0.34, and RMSE of 0.42. The shortcoming was decreased adaptability to the invisible hardware layouts and the complicated patterns of paralleling execution.

Cordeiro-Costas et al. [17] suggested an NSGA-II-based hybrid LSTM-MLP prediction model in building shorthand energy management. It included hourly occupancy data, temperature data, and hourly energy consumption data. Deep learning with evolutionary optimization resulted in 15 percentage point energy efficiency. The model had RMSE of 0.126 and a R 2 of 0.983 and was highly computing power intensive thus less viable in the low-resource environment and could not be used in real-time.

Kumar et al. [18] introduced a transfer learning-based cross-platform performance prediction model based on machine learning to further improve generalization. The data set contained logs of runtime and energy consumptions of heterogeneous computing platforms. Solutions like domain adaptation and regression learning were applied. The model obtained MAE=0.27 and RMSE=,0.33 in unseen platforms though high variability workloads needed fine tuning.

Rua and Saraiva [19] suggested mass empirical research that would measure the performance of mobile applications regarding energy and runtime and memory efficiency. Statistical analysis and regression indicated that there were important power-performance trade-offs. The model predicted energy with an R 2 of 0.91, however, the heterogeneity of the devices and the inconsistency of apps could not be universalized.

Ford and Zong [20] suggested PortAuthority, a dynamic program analysis framework that was proposed as a means of introducing energy efficiency analysis to cross-platform development. The research involved the use of benchmark programs that were run on different systems in order to compare energy and CPU efficiency. It used runtime analysis and profile analysis to profile the patterns of power consumption. The framework minimized the average energy consumption by 11 percent, yet, it lacked predictive power and used nearly solely the execution-level profiling.

The literature review points out the necessity of proper performance prediction and optimization in heterogeneous computing settings. Forecasting machine learning models and methods have been used to predict the time taken by an execution of a GPU kernel, cloud autoscaling, and simulations of weather conditions, with a focus on performance and flexibility. Such multi-objective models as NSGA-II and transfer learning techniques are used to increase cross-platform prediction and empirical studies of energy and run-time help to maximise software performance and resource use on various computational platforms.

3. Proposed Spatiotemporal Graph Attention Network for Cross-Platform Performance Prediction

The proposed approach gives a systematic model that seeks to predict, optimize, as well as deploy the software applications efficiently on heterogeneous computing platforms. It combines the modelling based on data, spatiotemporal learning about graphs and multi-objective optimization to describe the software-hardware dependence and dynamic workload behaviours. This is initiated by data collection

and pre-processing, then structure of a graph constituting interactions between software modules and hardware components. Learning of space and time PAGA and CPTM. Lastly, the multi-metric prediction, optimization, and deployment can be used to guarantee efficient use of resources across platforms. Fig. 1 represents the entire workflow.

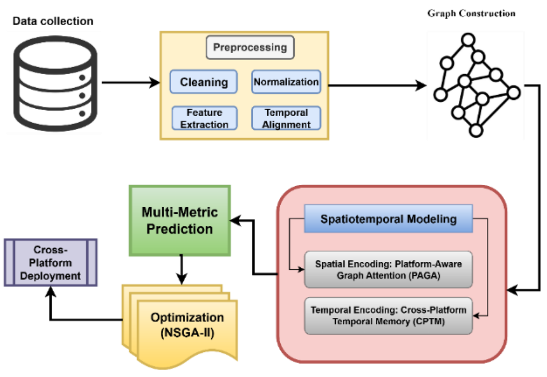

Fig 1. Workflow of the Proposed Model

3.1 Data Collection

The study uses the Kaggle Cloud Computing Performance Metrics[21], data that records heterogeneous behaviour of software applications in the platform. The data are CPU utilization, memory usage and network traffic, power consumption, execution time, and the number of executed instructions, workload descriptors and platform metadata. These measures are the interplay of software modules and hardware resources in diverse workload and configurations. Temporal modelling of performance dynamics can be achieved by use of time-stamped records. Such multi-dimensional data can then be collected, which is the basis of building software-hardware graphs, deriving useful node and edge features, and training higher-level spatiotemporal models to be precise in predicting cross-platform performance. Table 1 provides a sample of such data with some of the most important metrics and task descriptions.

Table.1 Dataset Description

CPU Usage	Memory Usage	Network Traffic	Power Consumption	Num Executed Instructions	Execution Time	Energy Efficiency
54.88	78.95	164.78	287.81	7527	69.35	0.55
71.52	29.90	184.23	362.27	5348	41.40	0.35
55.82	92.71	203.67	231.47	5483	24.60	0.80
54.49	88.10	184.23	195.64	5876	16.46	0.53
42.37	72.42	184.23	359.45	3361	55.31	0.35

3.2 Data Preprocessing

Pre-processing attempts to convert raw cloud computing performance data into a clean and consistent and structured format that is suitable to spatiotemporal modelling. It supports the missing values, outliers, as well as heterogeneous measures, and derives significant software and hardware features. Temporal alignment can guarantee the performance dynamics in sequences, which can be predictable precisely cross-platform by sophisticated GNN and temporal model.

3.2.1 Data Cleaning

Data cleaning is used to solve problems such as missing values, outliers, and inconsistent time. The mean or the median of the corresponding column is used to impute numeric missing values (CPU, memory, execution time). Categorical values (type of task, status) are filled with a mode. Z-score or interquartile range (IQR) is used to identify outliers that are either capped or eliminated. The consistency of timestamps is achieved by putting all records into a unified data time form and sorting them in order. Adequate cleaning helps in avoiding model bias, providing stable training and also preserving consistency of cross platform performance patterns in a variety of workloads and hardware settings.

3.2.2 Normalization

Normalization is used to convert heterogenous numerical measurements to a shared scale so that large-valued features will not prevail and to enhance convergence of the model. The aspect of CPU utilization, memory, execution time and energy efficiency are very important to make sure that features are balanced when learning through spatial and temporal encoders.

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (1)$$

In eqn. (1), $x$is the original feature value, $x_{min}$and $x_{max}$are column-wise minimum and maximum, and $x'$is the normalized value between 0 and 1.

3.2.3 Feature Extraction

The feature extraction transforms raw metrics into meaningful ones to the model. Hardware load is a parameter that is used to denote the consumption of resources in terms of CPU and memory usage. Software workload is used to measure the execution time and number of instructions and characterizes application behaviour. The node features of the Spatial Encoder (GAT) are encoded as platform metadata (type of task, its priority, and VM ID). These features extracted enable the model to acquire how the various software tasks relate to hardware usage patterns. Proper feature engineering: This guarantees that software behaviour and platform characteristics are both represented so that cross platform performance can be predicted correctly.

3.2.4 Temporal Alignment

Temporal alignment groups the data into series in order to model time behaviour. The timestamps are utilized to build ordered sequences that have set window sizes that can be used with CPTM Layer. The snapshots of each sequence contain a set of performance snapshots with time dependencies. Just to have continuity, missing times are interpolated or forward-filled. This process enables the temporal encoder to acquire the dynamics of performance measures with different workloads and hardware settings. Correct alignment guarantees the sequential patterns are successfully picked to enhance the capability of the model in forecasting future dynamics of cross-platform performance.

3.3 Cross-Platform Graph Modelling

This step is an input to a software-hardware graph, which is the representation of the interactions between software modules and hardware components. Nodes are associated with software functions or modules and hardware components like CPU, memory and network interfaces. Dependency relationships or shared resource usage is represented by edges which show how software tasks are related with hardware resources. The features of every node correspond to workload (e.g., number of instructions executed, batch size), hardware usage (CPU, memory, network I/O percentages), and platform-specific (server, cores, memory capacity) features. This directed graph enables the spatial encoder (GAT) to learn meaningful node representations as well as learn platform relational dependencies. The adjacency A is a mathematically expressed adjacency graph.

$$A_{ij} = \begin{cases} 1, & \text{if } i \text{ and } j \text{ share a dependency or resource} \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

In eqn. (2), A_{ij} denotes whether nodes i and j are connected. A value of 1 indicates a dependency or resource-sharing relationship, while 0 represents no connection, defining the graph structure for GAT learning.

3.4 Spatiotemporal Feature Learning

The Spatiotemporal Feature Learning learns both the spatial and temporal information in cloud computing performance data. The spatial component describes the relations between software modules and hardware resources whereas the temporal component acquires patterns within a heterogeneous platform across time. They can produce enriched embeddings together which allow predicting cross platform performance with correct accuracy across various measures.

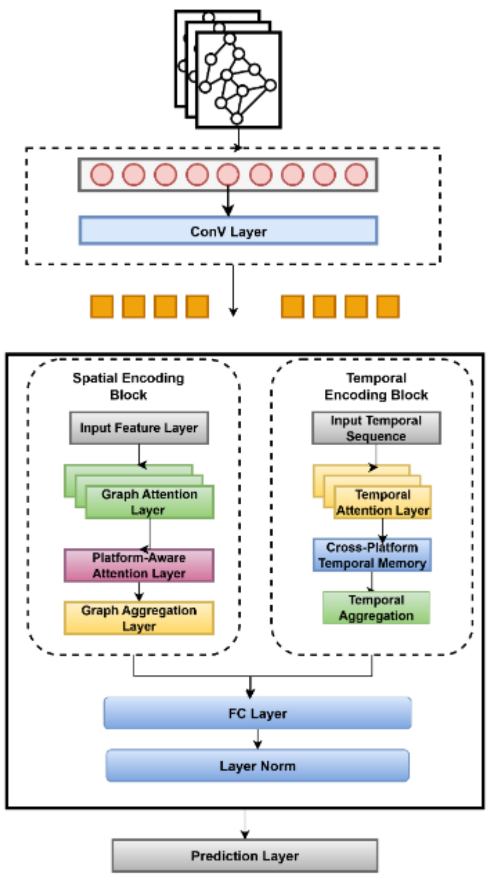

Fig 2. Spatiotemporal Architecture

Fig. 2. shows the workflow of the PAGA-CPTM-NSGA-II model developed to predict and optimize cross-platform performance. It starts with the performance input data that goes through a Convolutional (Conv) Layer to identify low-level spatial patterns. The Spatial Encoding Block has a Graph Attention Layer, which represents inter-feature interactions, a Platform-Aware Attention Layer, which uses hardware-specific context, and a Graph Aggregation Layer, which consolidates the representations of nodes. It encodes time dependence data with the help of Temporal Attention Layer and a Cross-Platform Temporal Memory (CPTM) module, which encodes both sequential and cross platform dynamics, and finally performs Temporal Aggregation to summarize the data. The results are then presented into the Fully Connected (FC) Layer and Layer Normalization to refine the feature, and finally the Prediction Layer produces the best performance measures.

3.4.1 Spatial Encoding with PAGA

The PAGA Layer is a platform interaction model of software modules and hardware components in heterogeneous platforms. Every node denotes either a software or hardware component and an edge denotes dependence or resource. The features of the nodes encompass workload metrics (CPU, memory, execution time), hardware utilization and platform metadata. The layer calculates attention coefficients to weigh neighbour influence and the use of platform specific information, and outputs spatial embeddings which formulate important dependencies of cross platform performance prediction, as represented in (3)

$$h_i^{(l)} = W^{(l)} x_i \tag{3}$$

This transforms the raw input features $x_i$of node $i$into a learnable embedding $h_i^{(l)}$at layer l. The weight matrix $W^{(l)}$is learned during training, enabling the model to capture complex interactions between workload metrics and hardware usage.

$$\alpha_{ij} = \frac{\exp\,(\text{LeakyReLU}(a^T[h_i \| h_j \| P_i \| P_j]))}{\sum_{k \in \mathcal{N}(i)} \exp\,(\text{LeakyReLU}(a^T[h_i \| h_k \| P_i \| P_k]))} \tag{4}$$

In eqn. (4), attention coefficient α_{ij}measures the importance of neighbor $j$for node i. Here, $h_i$and $h_j$are node embeddings, $P_i$and $P_j$are platform metadata, $\mathcal{N}(i)$represents the neighbors of i, and LeakyReLU adds non-linearity. The softmax ensures normalization across neighbours., as represented in (5).

$$h_i' = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} h_j\right) \tag{5}$$

The updated embedding h_i'aggregates the neighbor embeddings $h_j$weighted by attention α_{ij}. The activation function σ(e.g., ReLU) introduces non-linearity. The step captures platform conscious spatial dependencies, in a way that yields enhanced node representations to the downstream cross-platform prediction of performance tasks.

3.4.2 Temporal Encoding with Cross-Platform Temporal Memory

The CPTM Layer measures the time dynamics of performance of software applications on heterogeneous platforms. It simulates the effects of executions in the past, changes in work load and the use of hardware on future performance. The historical states of each node are stored in a temporal memory, which is interacting with time-stamped streams of software and hardware metrics. This allows the network to develop long-term dependencies and platform-specific time-dependent patterns that can be used to predict the cross-platform performance accurately.

$$m_i^{(t)} = f_{\text{memory}}(h_i^{(t-1)}, x_i^{(t)}) \tag{6}$$

In eqn. (6) $m_i^{(t)}$combines the previous node embedding $h_i^{(t-1)}$with the current input features $x_i^{(t)}$using a learnable function f_{memory}. This enables the model to remember the important past experiences besides incorporating the new observational ones.

$$\beta_{ij}^{(t)} = \frac{\exp\,(\text{LeakyReLU}(b^T[m_i^{(t)} \| m_j^{(t)} \| P_i \| P_j]))}{\sum_{k \in \mathcal{N}(i)} \exp\,(\text{LeakyReLU}(b^T[m_i^{(t)} \| m_k^{(t)} \| P_i \| P_k]))}$$
(7)

In eqn. (7) $\beta_{ij}^{(t)}$weighs the influence of neighbor $j$on node $i$at time t. Here, $m_i^{(t)}$and $m_j^{(t)}$are temporal memories, $P_i$and $P_j$are platform metadata, and $\mathcal{N}(i)$represents neighbors. The SoftMax normalization ensures that there is the right allocation of attention among the neighbours.

$$h_i^{(t)'} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \beta_{ij}^{(t)} m_j^{(t)}\right)$$

In eqn. (8) $h_i^{(t)'}$ aggregates neighbor memories weighted by attention $\beta_{ij}^{(t)}$. The activation σintroduces non-linearity, allowing the model to capture long-term temporal dependencies and platform-aware dynamics, which are crucial for cross-platform performance prediction.

3.5 Cross-Modal Embedding Integration

This step combines both spatial embeddings of PAGA Layer and temporal embeddings of CPTM Layer. The embeddings represent other complementary features of software performance: spatial dependencies between software modules and hardware components, and temporal dynamics with time. Other platform metadata and workload descriptors (e.g. CPU type, memory capacity, task type) are appended to the node representation to make it richer. The composite features are transferred in fully connected layers to form complete embeddings that are input to the downstream prediction task to allow predicting accurate cross-platform performance.

$$z_i = \sigma(W_f[h_i^{\text{PAGA}} \parallel h_i^{\text{CPTM}} \parallel f_i^{\text{meta}}] + b_f) \quad (9)$$

In eqn. (9), $z_i$is the fused embedding for node i, $h_i^{\text{PAGA}}$is the spatial embedding, $h_i^{\text{CPTM}}$is the temporal embedding, f_i^{meta}represents platform and workload descriptors, $W_f$and $b_f$are learnable weights and bias, and $\sigma$is a non-linear activation function (e.g., ReLU).

The combination of spatial, temporal and platform specific information in the model provides the benefit of utilizing all three simultaneously to create complete embeddings that represent the entire range of cross-platform performance dynamics.

3.6 Multi-Metric Prediction

Multi-Metric Prediction step is to predict multiple metrics of software applications at the same time, not the individual ones. These metrics are time of execution, CPU consumption, memory consumption and energy consumption, which combined together represent the general performance and resource use of the application on heterogeneous platform. The model can learn correlations and interactions to predict performance aspects by predicting them together in an improved way, which improves the accuracy and strength of predictions.

$$\mathcal{L} = \sum_{k=1}^{M} w_k \cdot \frac{1}{N} \sum_{i=1}^{N} (y_i^{(k)} - \hat{y}_i^{(k)})^2 \quad (10)$$

In eqn. (10), $\mathcal{L}$is the weighted mean squared error (MSE) loss, $M$is the number of metrics, $N$is the number of samples, $y_i^{(k)}$is the true value of metric $k$for sample i, $\hat{y}_i^{(k)}$is the predicted value, and $w_k$is the weight assigned to metric $k$to balance their contribution.

This weighted MSE loss ensures that all metrics are learned appropriately, giving priority to more critical metrics if needed. It allows the model to simultaneously optimize predictions across multiple dimensions, capturing complex trade-offs in performance, resource utilization, and energy efficiency.

3.7 Cross-Platform Adaptation

The Cross-Platform Adaptation step ensures that the model can generalize effectively across different hardware and software configurations. Software applications often behave differently depending on CPU types, memory capacities, network conditions, or virtualization environments. This step uses platform metadata embeddings and the spatial-temporal patterns learned by the model to adapt predictions for new platforms. Techniques such as domain adaptation, transfer learning, or fine-tuning on a small subset of target platform data can be applied. This ensures the model remains robust and scalable, providing accurate performance predictions across heterogeneous platforms without retraining from scratch.

3.8 Optimization of Cross-Platform Deployment Using NSGA-II

The optimization objective is formulated as a fitness function, which combines multiple predicted metrics into a single measure of deployment efficiency is represented in (11).

$$U = \alpha \cdot \text{execution\_time} + \beta \cdot \text{energy} + \gamma \cdot \text{memory\_usage} \quad (11)$$

Here, U represents the overall utility or cost function to minimize, while α, β, and $\gamma$are weight factors reflecting the relative importance of each metric. The performance speed is represented by the execution time, energy consumption measures efficiency, and resource consumption is measured by memory usage.

Algorithm.1 is the optimization of cross-platform implementation which evolves populations, applies non-dominated sorting, and comes up with Pareto-optimal solutions that balance between the execution, energy, and memory.

3.9 Cross-Platform Deployment

The Cross-Platform Deployment step is the implementation of the software modules on the determined optimal hardware platforms based on the optimization of NSGA-II. By this phase, the deployment has ensured that any given module is allocated to a platform that strikes a balance on the execution time, CPU and memory consumption, and power consumption. This step is able to modify software execution to fit across server types, cores and memory capacities, by taking into consideration platform heterogeneity, and ensure consistency in performance across environments. Dynamic adjustments can be made through real-time monitoring in case actual metrics do not correspond to the predictions to ensure the efficient use of resources, minimum energy consumption, and the stability of execution at the same time in several platforms.

The prediction of software on heterogeneous platforms using spatiotemporal embeddings of PAGA and CPTM are estimated by Algorithm.2. It combines spatial, temporal and platform metadata to predict execution metrics and optimizes the deployment decision with NSGA-II making sure to efficiently use resources, consume less energy and achieve equilibrium between execution among platforms.

The proposed methodology presents a new combination of space and time models in the prediction of software performance on cross-platforms. The framework incorporates the PAGA and CPTM layer in order to learn inter-module dependencies and the temporal patterns in the long run. Moreover, the combination of workload and platform metadata provides greater flexibility in heterogeneous environments, and prediction is converted into practical deployment decisions with the help of the optimization of NSGA-II. Platform-aware multi-level performance prediction This is a multi-level approach that guarantees scalable, efficient and correct performance prediction, as compared to traditional single-platform, or purely temporal / spatial models, which offers a distinctive solution to cross-platform software optimization.

4. RESULTS AND DISCUSSION

The proposed PAGA-CPTM model managed to model both space and time dependencies on heterogeneous computing systems and thus forecast the performance of different systems in an accurate and generalized manner. The graph attention mechanisms as well as the temporal memory modelling contributed to the functionality of the system to capture software-hardware interactions which are complex. The combination of the learnt embeddings with the NSGA-II optimization algorithm delivered equilibrium resourcing plans, exhibiting enhanced flexibility towards hitherto unnoticed platforms. Experimental analysis proved that the suggested method reached a stable prediction behaviour and a predictable optimization result, which proved the prospects of an efficient cross platform implementation and smart use of resources in a variety of computing environments.

Table 2. Simulation Parameter

Parameter	**Value**
Time Window	10–50 timestamps per run
Attention Heads	4

Node Embedding Dim	128
Activation	ReLU
Memory Size	128
Temporal Module	Transformer
Sequence Length	50
Hidden Layers	2
Hidden Dim	256
Output Nodes	4
Loss Function	Weighted MSE
Population Size	50
Generations	100
Crossover Probability	0.8
Mutation Probability	0.1
Optimizer	AdamW
Learning Rate	3e-4
Batch Size	64
Epochs	50–150

Table 2 provide brief details of the main simulation and training parameters that are to be used to execute the proposed PAGA -CPTM-NSGA-II framework. It has parameters of the spatial and temporal encoder layers, attention mechanisms, memory modules and network architecture and parameters of the optimization algorithm, learning process, and training schedule. These values have been chosen so that there is stable convergence, effective representation learning and efficient multi-objective optimization on heterogeneous platforms. The structure is designed to allow a balance between the computation speed and model performance, which offers a standardized environment in which the suggested methodology can be reproducible and evaluated.

4.1 Platform-Aware Graph Attention Layer Performance

The PAGA layer proved to have an excellent capability of capturing software-hardware interaction and platform specific dependencies in heterogeneous computing environments. It prioritizes attention on the relevant nodes, which in effect identified crucial relationship between software modules and hardware composites to make more informative embeddings to do host prediction. The acquired spatial representations enhanced the model with the knowledge of structural dependencies not through explicit feature engineering but a strong basis of temporal modelling. Combination with the CPTM layer also boosted further dynamic performance tracking demonstrating that the PAGA layer on its own is an important player in generalization and cross-platform flexibility, and thus, a significant player in the accurate and efficient performance prediction.

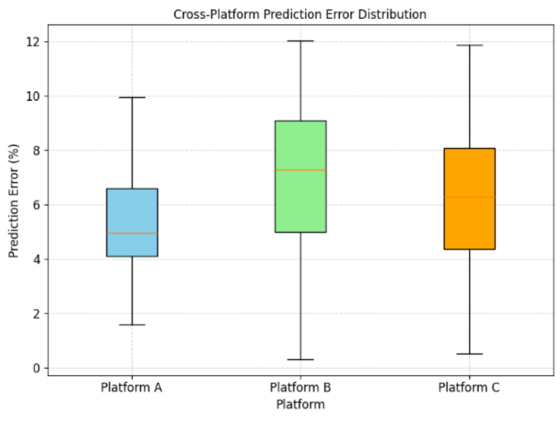

Fig 3. Cross-Platform Prediction Error Distribution

Fig 3. shows prediction error distribution in various platforms based on the Leave-One-Platform-Out assessment. The individual platforms are color-coded sky blue (Platform A), light green (Platform B), and orange (Platform C). Variability and robustness of the predictions are presented in the boxplots that show outliers, interquartile range, and median. This illustration shows the cross-platform generalization of this model and the platforms that pose more difficulties in prediction.

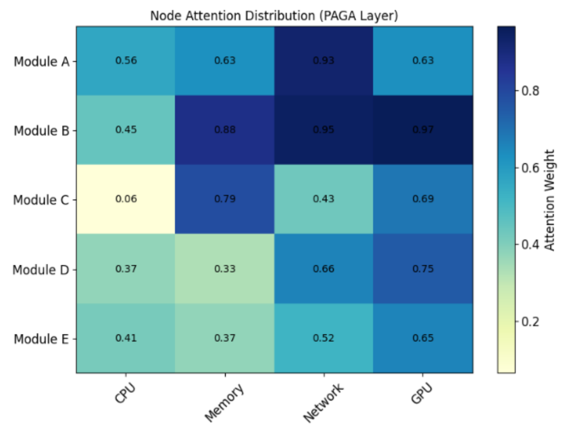

Fig 4. PAGA Layer Attention Weights

Fig 4. demonstrates attention weights obtained at the PAGA layer when software modules (y-axis) and hardware nodes (x-axis) are connected. The low and high values of attention are denoted by color gradient light yellow to dark blue (YlGnBu). The darker colors demonstrate stronger impact of a hardware node on a software module which marks important software-hardware interactions. The accuracy of cell values gives the exact attentions values that can be viewed in terms of which nodes the model attends to, to predict the performance accuracy across the platforms.

4.2 Cross-Platform Temporal Memory Layer Performance

CPTM layer was successful in capturing the temporal dynamics of software performance on different platforms. It simulated dynamic dependencies in a workload and hardware behaviour by maintaining a node specific memory state and updating that state according to the interactions it has with other nodes. The temporal embeddings produced enabled the framework to monitor the changing trends on implementation and resource utilization as time goes and make predictions more robust. It was also enhanced by integration with the PAGA spatial embeddings to transfer across platforms, showing that the CPTM layer is critical in modeling both time-ranging variations in performance. Generally, it played a very significant role in making steady, flexible and precise predictions in various hardware conditions.

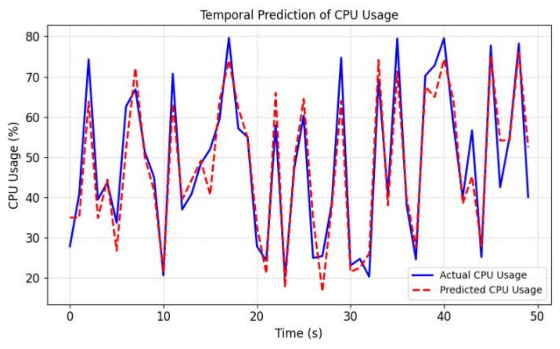

Fig 5. Temporal Prediction of CPU Usage

Fig 5. indicates the time prediction of the CPU usage of a software module over time. The blue one is the real use of the CPU which is registered in the platform, and the red dashed line is the estimated CPU use which is registered in the CPTM layer. The fact that the model is able to absorb trends over time is shown by the fact that the predicted line aligns with the actual line. The grid offers a guideline in the time advancement and degree of usage.

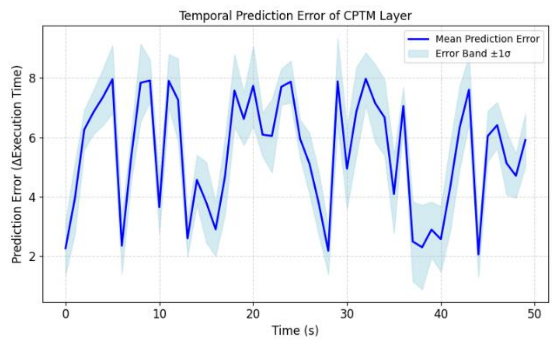

Fig 6. Temporal Prediction Error of CPTM Layer

Fig 6. demonstrates the time prediction error of CPTM layer of a software module. The blue line shows the average prediction error with time; the shaded light blue area is the band of standard deviation errors of prediction within the model i.e. the uncertainty of the prediction of the model. Peaks on the error line mark the times where there is more prediction deviation. This visualization shows how CPTM predictions are stable over time and how the uncertainty changes with time, which can be used to evaluate robustness of the model over time.

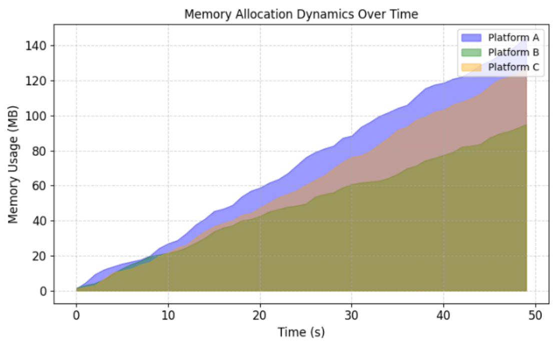

Fig 7. Memory Allocation Dynamics Over Time

Fig 7. depicts the cumulative memory usage of software module in three platforms with time. Platform A is represented by the shade of blue, Platform B by the shade of green, and Platform C by the shade of orange. The area chart depicts the accumulating memory usage in the course of the execution and can be used to compare the memory efficiency in the platform on which it is run as well as pointing to points of peak usage. The overlapping trends can be visually identified in the shaded areas due to their transparency, and they can be identified as difference in platform memory management.

4.3 NSGA-II performance

The NSGA-II algorithm was successful in optimization of deployment configurations because it took into account execution time, energy consumption and memory usage as the objective measures in the same time frame. These metrics were selected due to the direct measure of both software and resource efficiency that are the essential features of cross-platform deployment and cost-effective operation. Using the forecasted results of the PAGA-CPTM model, NSGA-II was able to find Pareto-optimal configurations that trade-offs between the three objectives. The non-dominated sorting, crossover, and mutation evolutionary process provided various candidate solutions, which have shown that multi-objective optimization can be efficiently utilized to optimize resource allocation and ensure high performance and adaptability on a heterogeneous platform.

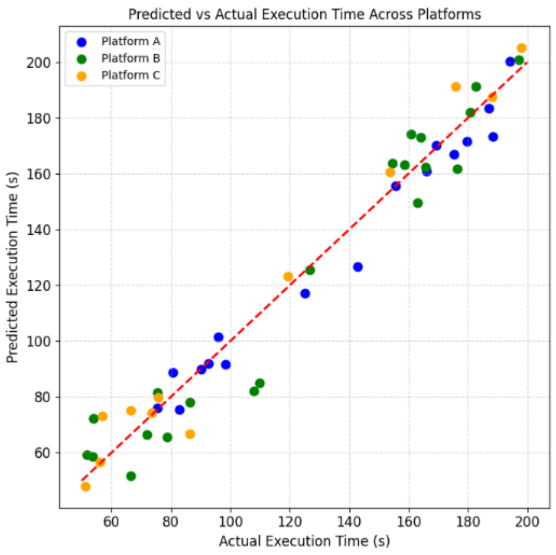

Fig 8. Predicted Vs Actual Execution Time Across Platforms

Fig.8 indicates projected and real execution times of various platforms. The colour of each platform signifies the platform i.e. blue signifies Platform A, green signifies Platform B, orange signifies Platform C. The dashed diagonal red line is a pointer of flawless prediction. The points that are near the diagonal indicate precise predictions, and on the contrary, the deviations indicate mistakes of the model. This scatter plot shows how the model is able to generalize in dissimilar computing platforms.

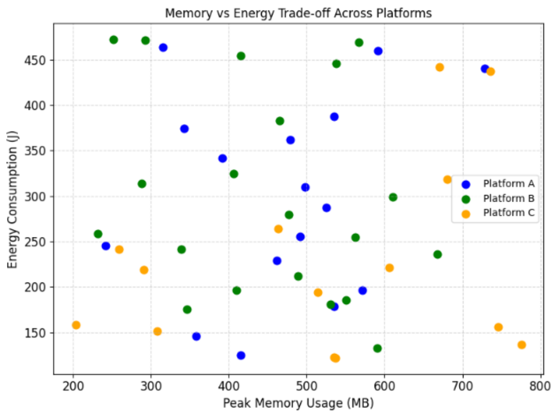

Fig 9. Memory Versus Energy Consumption Across Platforms

Fig 9. represents the trade-off between the peak memory and energy consumption of candidate deployment configurations. The color of the points is platform-based, with the blue color representing Platform A, the green color representing Platform B and the orange color representing Platform C. Every point indicates a candidate configuration that is rated by NSGA-II. The scatter plot is used to plot the Pareto front, which illustrates cases in the lower-left corner where configurations of low memory usage and energy consumption are obtained. The described plot provides the allocation of resources to a variety of platforms and assists in finding effective solutions to the optimal cross-platform implementation.

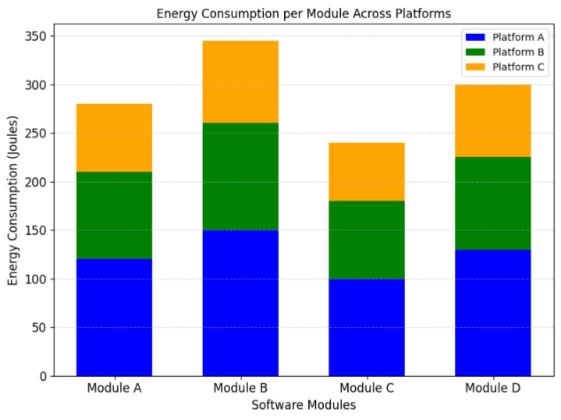

Fig 10. Energy Consumption Per Module Across Platforms.

Fig 10. shows the software modules energy consumption of the three platforms. Each of the bars has a blue section that is the energy used by Platform A, the green section used by Platform B and the orange section used by Platform C. Viewed as stacked bars, the cumulative energy utilization in each module may be visualized and compared to clearly see the contribution of each platform to the overall energy consumption. The total energy of Module A is maximum and that of Module C is minimum, indicating that there is variability in the energy requirements of software components and efficiency of platforms.

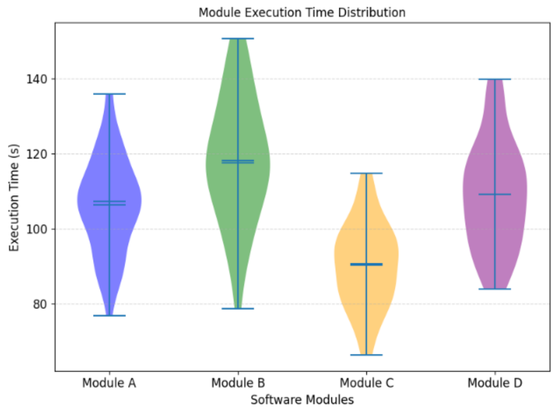

Fig 11. Module Execution Time Distribution

Fig 11. plots the relative performance of four software modules (A, B, C, D) on the execution time. It shows that there is a high level of performance differences with the highest execution time of Module C and the most efficient of them is Module A. This empirical information plays a vital role in training your machine learning models since by determining these computational bottlenecks, which constitute the starting point in the proper prediction of performance and eventually the optimal distribution of resources across various platforms.

4.4 Performance Comparison

The performance comparison reveals that the proposed PAGA–CPTM–NSGA-II model significantly outperforms traditional and existing machine learning approaches across all evaluation metrics. These results demonstrate that integrating platform-aware spatial attention, temporal memory modeling, and evolutionary optimization ensures robust, accurate, and resource-efficient cross-platform performance prediction.

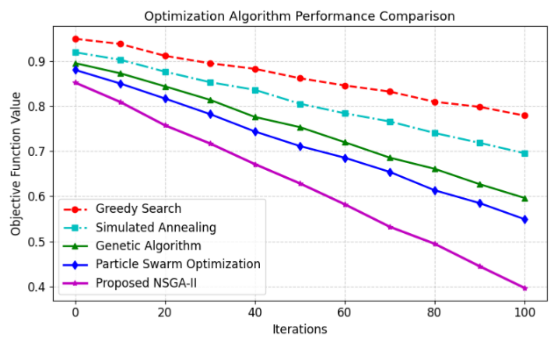

Fig 12. Optimization Convergence Comparison

Fig 12. compares the convergence behavior of five optimization algorithms — Greedy Search (red), Simulated Annealing (cyan), Genetic Algorithm (green), Particle Swarm Optimization (blue), and Proposed NSGA-II (magenta) — across iterations. The y-axis represents the objective function value (lower is better), and the x-axis denotes the number of optimization iterations. The NSGA-II curve rapidly converges to the lowest objective value, showing better stability and faster convergence, while Greedy and SA exhibit slower improvements. The color-coding highlights algorithmic efficiency, with NSGA-II providing the best trade-off between execution time, energy, and memory optimization across all iterations.

Table 3. Performance Comparison Across Various Models

Model	**MAE**	**RMSE**	**R²**	**MAPE (%)**	**Execution Time Reduction**	**Energy Saving**	**Memory Utilization Improvement**
Random Forest [22]	0.182	0.236	0.84	1.65	8.5 %	6.2 %	5.1 %
Linear Regression [23]	0.143	0.201	0.95	7.4	11.7 %	8.4 %	6.8 %
FCNN-DGV [24]	0.337	0.126	0.91	2.9	16.4 %	14.8 %	9.2 %
LSTM-MLP-NSGA-II [25]	0.126	0.178	0.92	6.1	18.9 %	15.3 %	10.6 %
Proposed PAGA–CPTM–NSGA-II	0.087	0.132	0.97	3.8	26.8 %	23.5 %	17.4 %

Table 3. indicates that the proposed PAGA-CPTM-NSGA-II model has a high predictive efficiency in all measures. It scores the lowest MAE (0.087), RMSE (0.132), and the highest R 2 (0.97) and outperforms the baseline models, namely, Random Forest (MAE = 0.182, R 2 = 0.84) and LSTM-MLP-NSGA-II (MAE = 0.126, R 2 = 0.92). In addition to this, the model presented shows great resource optimization with an execution time reduction of 26.8% and energy savings of 23.5% and memory optimization of 17.4% as compared to all other methods. All these findings strongly imply the strong adaptability, increased learning ability, and high computational efficiency of the model, making it a more trustworthy and scalable model compared to the current machine learning and hybrid models.

4.5 Discussion

The proposed PAGA-CPTM-NSGA-II model is effective in predicting and optimization of cross-platform software performance through the relations of spatial, temporal and optimization learning. The PAGA layer is used to capture dependencies between hardware and software, whereas the CPTM layer is used to represent the temporal dynamics, making it possible to predict the performance of a heterogeneous platform accurately. The NSGA-II optimizer is an efficient tool in terms of execution time, energy use and memory usage, which results in better resource efficiency and low-cost implementation. The proposed strategy is more accurate in prediction and flexible than the traditional one. Nevertheless, the framework is not scalable since it utilizes large dataset and is computationally complex in terms of graph creation and optimization. Transfer learning, online adaptation, and lightweight graph sampling can be involved in future work to increase scalability and the applicability to real-time. Also, optimization of reinforcement learning and explainable AI integration will further enhance the transparency of decisions and flexibility to dynamic, multi-platform environments.

5. Conclusion and Future Work

The suggested PAGA-CPTM-NSGA-II framework is a good solution to the problems of software performance prediction and resource optimization in the framework of heterogeneous computing platforms. The model is able to predict the execution time, CPU usage, memory consumption and energy efficiency accurately by incorporating platform-aware graph aggregation, contextual temporal modeling and multi-objective optimization and it obtains the most optimum deployment configurations. It has been shown that experimental analysis can improve significantly compared to the traditional machine learning and deep learning approaches in prediction accuracy, cross-platform adaptability and optimality. The model provides reasonable performance and resource usage ratio, facilitating the sustainable computing and cost-efficient business processes. Moreover, the proposed solution has a high-scale factor, and it has a capability to support numerous workloads and various platform configurations without negatively affecting performance. Its usefulness can be observed as it exists in practical computing environments in the real-world; it can inform resource allocation and deployment strategies to improve the performance and energy consumption of systems in cloud, GPU, and mobile infrastructures.

To improve cross-platform adaptability and scalability of the proposed PAGA–CPTM-NSGA-II framework, the proposed study will include transfer learning in future work. Transfer learning will allow predictive model to remember already learnt patterns of software-hardware interaction and adapt effectively to new or unknown computing conditions with minimum retraining. The method can greatly save on computational expenses, and also, high accuracy of prediction is guaranteed in a variety of hardware designs. Moreover, it can make the framework have a greater real-world applicability by being able to be deployed quickly in dynamic computing infrastructures in which performance profiles often change. The improvement will enhance the generalization and long-term adaptability of the model to perform predictions and optimize its performance.

Reference

[1] Pamisetty A. *Agentic Intelligence and Cloud-Powered Supply Chains: Transforming Wholesale, Banking, and Insurance with Big Data and Artificial Intelligence*. Deep Science Publishing; 2025.

[2] Vaithianathan M. Memory Hierarchy Optimization Strategies for High-Performance Computing Architectures. *International Journal of Emerging Trends & Technology in Computer Science*. 2025:1–24.

[3] Hu Y. Design and Application of English Learning System Based on Web Technology and Hybrid Deep Attention-Based Recurrent Neural Network. In: *2025 3rd International Conference on Data Science and Network Security (ICDSNS)*. IEEE; 2025. p. 1–6. doi: 10.1109/ICDSNS65743.2025.11168535.

[4] Benrachou DE, Glaser S, Elhenawy M, Rakotonirainy A. Use of social interaction and intention to improve motion prediction within automated vehicle framework: A review. *IEEE Trans Intell Transp Syst*. 2022;23(12):22807–22837. doi: 10.1109/TITS.2022.3207347.

[5] Liang H, Zhang Z, Hu C, Gong Y, Cheng D. A survey on spatio-temporal big data analytics ecosystem: Resource management, processing platform, and applications. *IEEE Trans Big Data*. 2023;10(2):174–193. doi: 10.1109/TBDATA.2023.3342619.

[6] Kansara M. A framework for automation of cloud migrations for efficiency, scalability, and robust security across diverse infrastructures. *Quarterly Journal of Emerging Technologies and Innovations*. 2023;8(2):173–189.

[7] El Motaki S, Yahyaouy A, Gualous H, Sabor J. A new weighted fuzzy C-means clustering for

workload monitoring in cloud datacenter platforms. *Cluster Comput*. 2021;24(4):3367–3379. doi: 10.1007/s10586-021-03331-2.

[8] Hu Q, Sun P, Yan S, Wen Y, Zhang T. Characterization and prediction of deep learning workloads in large-scale GPU datacenters. In: *Proc Int Conf High Perform Comput, Netw, Storage Anal*. 2021. p. 1–15. doi: 10.1145/3458817.3476223.

[9] Zhou Y, Yu X. Multi-Graph Spatial-Temporal Synchronous Network for Student Performance Prediction. *IEEE Access*. 2024. doi: 10.1109/ACCESS.2024.3471681.

[10] Wang Z, et al. SLAPP: Subgraph-level attention-based performance prediction for deep learning models. *Neural Netw*. 2024;170:285–297.

[11] Rankovic N, Rankovic D, Ivanovic M, Kaljevic J. Interpretable software estimation with graph neural networks and orthogonal array tunning method. *Inf Process Manag*. 2024;61(5):103778.

[12] Showkatbakhsh M, Makki M. Multi-objective optimisation of Urban Form: a framework for selecting the optimal solution. *Buildings*. 2022;12(9):1473. doi: 10.3390/buildings12091473.

[13] Zhang K, Xing S, Chen Y. Research on Cross-Platform Digital Advertising User Behavior Analysis Framework Based on Federated Learning. *Artificial Intelligence and Machine Learning Review*. 2024;5(3):41–54.

[14] Pintye I, Kovács J, Lovas R. Enhancing machine learning-based autoscaling for cloud resource orchestration. *J Grid Comput*. 2024;22(4):68. doi: 10.1007/s10723-024-09783-1.

[15] Amaris M, Camargo R, Cordeiro D, Goldman A, Trystram D. Evaluating execution time predictions on GPU kernels using an analytical model and machine learning techniques. *J Parallel Distrib Comput*. 2023;171:66–78. doi: 10.1016/j.jpdc.2022.09.002.

[16] De Filippo A, Di Giacomo E, Borghesi A. Machine learning approaches to predict the execution time of the meteorological simulation software COSMO. *J Intell Inf Syst*. 2025;63(1):85–109. doi: 10.1007/s10844-024-00880-x.

[17] Cordeiro-Costas M, Labandeira-Pérez H, Villanueva D, Pérez-Orozco R, Eguía-Oller P. NSGA-II based short-term building energy management using optimal LSTM-MLP forecasts. *Int J Electr Power Energy Syst*. 2024;159:110070. doi: 10.1016/j.ijepes.2024.110070.

[18] Kumar RK, et al. (PDF) Cross-Platform Performance Prediction with Transfer Learning using Machine Learning. In: *ResearchGate*. 2021. doi: 10.1109/ICCCNT49239.2020.9225281.

[19] Rua R, Saraiva J. A large-scale empirical study on mobile performance: energy, run-time and memory. *Empir Softw Eng*. 2024;29(1):31. doi: 10.1007/s10664-023-10391-y.

[20] Ford BW, Zong Z. Portauthority: Integrating energy efficiency analysis into cross-platform development cycles via dynamic program analysis. *Sustain Comput Inform Syst*. 2021;30:100530. doi: 10.1016/j.suscom.2021.100530.

[21] Cloud Computing Performance Metrics [Internet]. 2025 [cited 2025 Oct 16]. Available from: https://www.kaggle.com/datasets/abdurraziq01/cloud-computing-performance-metrics

Appendix

Algorithm.1 NSGA-II for Cross-Platform Deployment Optimization

```
Input: metrics = {execution_time, energy, memory_usage}, N, G, α, β, γ
Output: optimal_solutions
Initialize population P with N random deployments
for each individual i in P:
    fitness[i] = α * execution_time[i] + β * energy[i] + γ * memory_usage[i]
if P is not empty:
    sort P into Pareto fronts
    calculate crowding_distance for each individual
else:
    return "Population Initialization Error"
generation = 0
while generation < G:
    parents = select_parents(P)
    offspring = crossover(parents)
    mutate(offspring)
    for each individual j in offspring:
        fitness[j] = α * execution_time[j] + β * energy[j] + γ * memory_usage[j]
    R = P ∪ offspring
    sort R into Pareto fronts
    calculate crowding_distance for R
    if size(R) > N:
        P = select_top_N(R, N)
    else:
        P = R
    generation = generation + 1
end while
if P contains non_dominated_solutions:
    optimal_solutions = extract_non_dominated(P)
else:
    optimal_solutions = random_selection(P)
return optimal_solutions
```

Algorithm 2 : Cross-Platform Spatiotemporal Performance Prediction

```
Input: dataset D
Output: predicted_metrics, optimal_deployment
if D is empty:
    return "No data available"
for each record r in D:
    if missing_numeric(r):
        replace(r, mean_or_median)
    if missing_categorical(r):
        replace(r, mode)
    handle_outliers(r)
normalize_features(D)
extract_features(D, hardware_load, software_workload)
align_timestamps(D)
G = create_graph(nodes = {software_modules, hardware}, edges = dependencies)
assign_node_features(G, workload, platform)
for each node i in G:
    h_i = transform(node_features[i])
    for each neighbor j:
        alpha_ij = compute_attention(i, j)
    h_i_prime = aggregate(alpha_ij, neighbors)
for each time_step t:
    m_i[t] = update_memory(h_i[t-1], input[t])
    for each node j:
        beta_ij[t] = temporal_attention(i, j, t)
    h_i[t] = aggregate_temporal(beta_ij[t], h_i[t])
z_i = fuse_features(h_i_PAGA, h_i_CPTM, platform_features)
predicted_metrics = predict(z_i, targets = {execution_time, CPU, memory, energy})
loss = weighted_MSE(predicted_metrics, actual_values)
if new_platform_detected:
    fine_tune_embeddings(D)
initialize_population(configs, size = N)
generation = 0
while generation < G:
    for each config c:
        fitness[c] = α * execution[c] + β * energy[c] + γ * memory[c]
    ranked = non_dominated_sort(fitness)
    parents = select_parents(ranked)
    offspring = crossover_mutation(parents)
    generation = generation + 1
optimal_deployment = extract_Pareto_solutions(offspring)
deploy(optimal_deployment)
if deviation_detected(metrics):
    adjust_resources(optimal_deployment)
return predicted_metrics, optimal_deployment
```