Computer Communications 210 (2023) 45–57

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

RUMP: Resource Usage Multi-Step Prediction in Extreme Edge Computing☆

Ruslan Kain*, Sara A. Elsayed, Yuanzhu Chen, Hossam S. Hassanein

School of Computing, Queen's University, Kingston ON, Canada

ARTICLE INFO

Keywords:
Edge computing
Extreme edge devices
Hidden Semi-Markov model
Dynamic resource usage

ABSTRACT

Extreme Edge Computing that leverages the copious yet underutilized computational resources of Extreme Edge Devices (EEDs) has gained significant momentum lately. Estimating the computational capabilities of EEDs can be strenuously challenging since EEDs are user-owned devices, and are thus subject to a highly dynamic user access behavior (i.e., dynamic resource usage). In this paper, we propose the Resource Usage Multi-step Prediction (RUMP) scheme. RUMP is the first scheme that strives to enable multistep-ahead prediction of the dynamic resource usage of EEDs (i.e., workers) in a computationally efficient way, while providing a relatively high prediction accuracy. Towards that end, RUMP exploits the use of the Hierarchical Dirichlet Process-Hidden Semi-Markov Model (HDP-HSMM). In addition, RUMP uses the Simple and Exponential Moving Average (SMA&EMA) and Savitzky-Golay (SG) filters to improve the prediction accuracy. We scrupulously study the trade-off between computational efficiency, prediction accuracy, practicality, and adaptability of the underlying prediction model by conducting complexity analysis, performing various experiments on a testbed of heterogeneous workers, and comparing the HDP-HSMM model used in RUMP to three other prominent prediction models in different dynamic resource usage scenarios. Extensive evaluations show that RUMP achieves a 91% categorical multi-step prediction accuracy and renders a small performance gap of 6% on average in terms of the Mean Absolute Error (MAE) compared to representatives of state-of-the-art prediction models, while yielding a low computational complexity.

1. Introduction

With the pervasive proliferation of the Internet of Things (IoT), 23.3 billion IoT devices are expected to be connected to the Internet by 2025 [2]. This substantial growth is expected to impose unprecedented demands on computing resources to satisfy the stringent Quality of Service (QoS) requirements associated with delay-critical and/or data-intensive applications, such as the Metaverse, Large AI models, Tactile Internet, virtual and augmented reality, crypto-currency mining, and smart cities [3,4]. Such strict requirements can be challenging to satisfy in Cloud Computing (CC), due to the need to transmit massive amount of data to remote data centers, which significantly increases latency and thrusts a heavy traffic load at backhaul links [5].

Extreme Edge Computing (EEC) has emerged as a propitious computing paradigm that can alleviate the aforementioned issues [6–9]. EEC harvests the profuse yet underutilized computational resources of Extreme Edge Devices (EEDs), such as PCs, Laptops, tablets, smartphones, and connected vehicles [10]. Thus, EEC offers the computing service much closer to end-users, which can drastically curtail the delay and improve the QoS. In addition, parallel processing at EEDs can democratize the edge and permit more players to establish and administer their own edge cloud [1,6–9]. Consequently, EEC paves the way for a new tech market that is people-owned, democratically managed, and accessible/lucrative to all, and it has thus been adopted by various industrial entities [11].

Despite the promising potential of EEC, the viability of such a system is hindered by the heterogeneous nature of EEDs (i.e., workers), and the fact that EEDs are user-owned devices, which subjects them to a highly dynamic user access behavior. In particular, users can access their devices at any given time to run an intensive application, such as streaming a video, playing a video game, running augmented reality, etc. This can dynamically alter and impact the devices' available computational resources. Note that these resources are affected by the demands of the applications run, the operational conditions set, and the limits defined by the device's specifications [12]. Consequently, EEC cannot rely on the same resource characterization/estimation techniques used for infrastructure-based edge nodes or cloud servers. Successful, consistent, and reliable task offloading and resource allocation services that can adapt to the dynamic environment in EEC require new intelligent resource characterization techniques [13].

☆ Some of the work presented in this paper has been published in the proceedings of the 25th International Conference on Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWiM'22) [1].

* Corresponding author.
E-mail address: r.kain@queensu.ca (R. Kain).

https://doi.org/10.1016/j.comcom.2023.07.029
Received 1 April 2023; Received in revised form 8 July 2023; Accepted 21 July 2023
Available online 25 July 2023

Resource characterization, which involves estimating the performance of workers in terms of job completion time or throughput, requires benchmark tasks to be run on the devices [14]. Given the dynamic resource usage of workers in EEC, it is essential to characterize the worker in each possible resource usage state with several benchmark tasks [12]. Thus, it is imperative to identify the resource usage state and run the benchmarks in a timely manner to correctly characterize the worker. Such characterization can enable a dynamic usage-aware scheduler to take into account the resource usage state of the worker when allocating tasks.

Resource characterization that relies on capturing and estimating the highly dynamic resource usage in EEC is mostly overlooked in existing schemes. In addition, predictive models that are proactively used in infrastructure-based edge computing paradigms suffer from various drawbacks, such as high complexity in Machine Learning (ML)-based models [15–17], and the long inference time and inability to predict for multiple related features making pattern matching-based models univariate [18–21]. These drawbacks render such models impractical to cope with the highly dynamic user access behavior of EEDs. In this paper, we propose the Resource Usage Multi-step Prediction (RUMP) scheme. RUMP incorporates the use of the Hierarchical Dirichlet Process-Hidden Semi-Markov Model (HDP-HSMM) to predict the resource usage state of EEDs over multiple steps ahead.

The HDP-HSMM model is used in RUMP due to its unique features that can foster operational efficiency. In particular, HDP-HSMM has the ability to make labeled and numeric multi-step predictions using a single model only, while being trained in a semi-supervised approach, allowing it to identify patterns in the data without requiring intensive labeling efforts [22]. This makes it more practical, effective, and versatile for inference in the context of EEDs. This is as opposed to machine learning models that require a separate model per step size, and require modifying the model to do either labeled or numeric prediction [15,16]. Note that single pattern matching models [21] can be used for multiple step sizes using a sliding window technique that uses prior predicted time steps to predict the next. However, such models are incapable of categorical prediction. In contrast, the HDP-HSMM model possesses such a capability.

Our contributions can be summarized as follows:

- We account for the dynamic user access behavior in EEC by proposing the RUMP scheme to estimate the dynamic resource usage of workers and enable efficient resource characterization. RUMP is the first scheme in EEC that enables practical and computationally efficient multi-step prediction of resource usage.
- We improve the prediction accuracy of resource usage by using the Simple and Exponential Moving Average (SMA&EMA), and Savitzky–Golay (SG) filtering techniques [23] to remove noise, smooth out fluctuations in the input data, and highlight trends over time.
- We study the trade-off between the prediction accuracy of resource usage and the computational efficiency of the underlying prediction model by conducting complexity analysis and comparing RUMP to three prominent prediction models, namely the Hybrid Bi-directions LSTM Encoder–Decoder (HBLED) model [17], the k Nearest Neighbors Time Series Prediction with Invariance (kNN-TSPI) model [21], and the Hybrid Bayesian Particle Swarm Hyper-Parameter Optimization (HBPSHPO) model [16].
- We conduct performance ranking of RUMP and representative state-of-the-art prediction models by using the Multi-Criteria Performance Measure (MCPM) [24], and statistical tests [25], namely, the Friedman test and the Nemenyi post-hoc test.
- We perform extensive evaluations on a realistic testbed of heterogeneous workers in different dynamic resource usage scenarios.

The remainder of the paper is organized as follows. Section 2 presents an overview of the related work. Section 3 introduces RUMP. Section 4 presents the complexity analysis of RUMP and the other models. Section 5 presents the performance evaluation results, analysis, and discusses the performance, complexity, and effectiveness trade-offs. Section 6 concludes the paper and presents future research directions.

2. Related work

Several methods for modeling and predicting resource usage have been proposed for different computing systems, including grid computing [26], volunteer computing [27], cloud computing [28], and more recently, edge computing [16]. However, there is still a lack of methods that can effectively handle the highly dynamic user access behavior and resource usage of workers in EEC.

In infrastructure-based edge computing paradigms, resource characterization can be categorized into reactive and proactive techniques [29]. Reactive techniques rely on modifying resource allocations when shifts in resources are detected, either through pattern-matching [18–21,30], or threshold-based techniques [26]. In contrast, proactive techniques rely on predicting future resource shifts before they occur. While both reactive and proactive techniques enable adaptation to resource usage variability and availability, proactive techniques have been shown to achieve higher prediction accuracy at the expense of higher computational complexity [29].

Sekma et al. [19] develop a reactive technique to predict the CPU resource availability in a volunteer computing grid using Auto-Regressive (AR), Vector Auto-Regressive (VAR), and polynomial fitting models. However, such models assume that the time series is stationary, which is not always the case, since usage may change with behavioral changes of the user. Sorkunlu et al. [26] use an anomaly detection method based on a residual error of a tensor composed of the compute nodes, resource usage information, and time to adapt to abnormal system behavior. Thus, such an anomaly detection method is quintessentially reactive. The problem with reactive techniques is that they tend to ignore the long-term dependencies detected in system-level monitoring signals, which can curtail their predictive abilities [29]. As a result, reactive approaches might repeatedly make the same errors, thus affecting the proper management of the computational resources.

In contrast to reactive techniques, proactive techniques that include the use of ML models and state-based models can indeed capture long-term dependencies and recurrence in the system. In Rusty [29], a Long-Short Term Model (LSTM) is trained to predict lower-level architectural events and system-level characterization under interference to gain insights into the system state and guide the task scheduler. In [16], a combination of LSTM and Convolutional Neural Networks (CNN) are used to predict the resource usage of EEDs, where the architecture of the models is automatically designed to maximize performance using hyper-parameter search optimization methods, thus performing better in comparison to other ML models, such as support vector regression, multiple linear regression, XGBoost, and Deep Neural Networks (DNN). In [17], the Hybrid Bidirectional LSTM Encoder–Decoder (HBLED) model is proposed. HBLED combines bidirectional LSTM layers in the encoder with unidirectional LSTM layers in the decoder. The encoder–decoder model is a way of using recurrent neural networks for sequence-to-sequence prediction problems. It consists of an encoder that transforms the input sequence into a compressed encoding, and a decoder that reconstructs the output sequence from the encoded input. The encoder–decoder model can handle variable-length input and output sequences, which makes it suitable for multi-step and multi-variate prediction tasks. The bidirectional LSTM layers can capture both forward and backward dependencies in the input sequence, while the unidirectional LSTM layers can generate future predictions based on the encoding and previous outputs. Furthermore, Violos et al. [16] use a combination of Bayesian and particle swarm optimization for ML hyper-parameter search to tailor complex models of different architectures to predict EED resource usage.

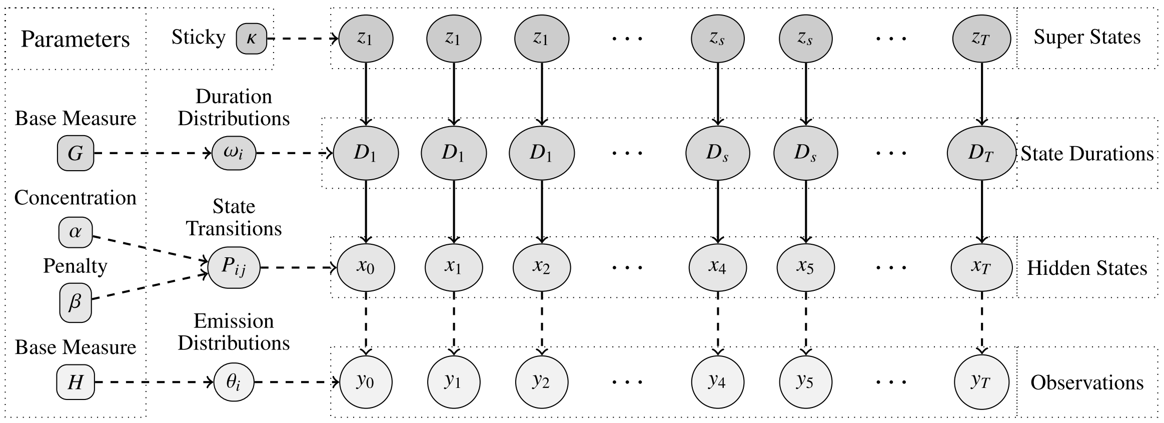


Fig. 1. The Hierarchical Dirichlet Process - Hidden Semi-Markov Model (HDP-HSMM) components, including the state durations (D_s), observations (y_t), hidden states (x_t), and super states (z_s), for $t \in \{1, \dots, T\}$ were T is the length of the observation sequence, and is used to denote the end of a sequence, and $s \in \{1, 2, \dots, S\}$ where S are the number of super-states. In addition to concentration parameters (α), penalty parameter (β), sticky parameter (κ), base measure parameters for emissions (H), and distributions (G), distribution parameters for emissions (θ_i) and durations (ω_i), and lastly, the state transition matrix P_{ij}, where $i, j \in \{1, \dots, N\}$.

ML techniques have a high predictive power. However, they are computationally intensive, and suffer from long training times and low adaptability to resource usage patterns that continuously change in real-time [15]. Consequently, they are rendered impractical for resource usage prediction in EEC paradigms. In addition, most proposed ML-based models have limited usability within the context of EEC, due to their reliance on 1-step ahead predictions that fail to capture the highly dynamic nature of such systems.

Multi-step ahead prediction of CPU loads using a pattern matching technique has been proposed in distributed systems [30]. Parmezan et al. [21] propose the k Nearest Neighbors Time Series Prediction with Invariances (kNN-TSPI) model, which is a local approach used for time series prediction. The kNN-TSPI algorithm works by finding the k examples that are the nearest to an unlabeled example based on a similarity measure, while accounting for various types of invariances in the data. This technique can handle multi-step ahead prediction by recursively applying the model on its own predictions, known as the sliding window approach. It also has the advantage of rendering negligible training time, as well as the ability to use a single model for multi-step ahead predictions. However, it is unfit for multi-variate prediction of resource usage in EEC, due to the explosive number of pattern combinations, which impacts prediction accuracy and computation time [31].

State-based models have been shown to reliably capture the dynamics of worker resource usage [32]. The types of state-based models include the Hidden-Markov Model (HMM) [33] that can accurately model continuous resource usage data, and the Semi-Markov model (SMM) [27] that can predict the time duration of the model's states, thus enabling forward predictions of the worker's dynamics. Pramanik et al. [34] show that Markov-state models are problematic in such dynamic scenarios due to lack of memory. Thus, resource usage patterns have non-Markovian properties. However, the Hidden Semi-Markov model is nonetheless suitable for modeling the relationship between the dynamic run-time of edge-native applications and the resource usage.

In contrast to existing schemes, the proposed RUMP scheme is tailored to model and capture the dynamic resource usage of EEDs in a computationally-efficient way by using the HDP-HSMM model. HDP-HSMM combines the accuracy of HMMs and the time-dimension predictability of SMMs [27]. HSMM is practical and adaptable for EEC systems, since it requires lower complexity and workload, uses a single model for multi-step numeric and label prediction, and adapts to new data input. In an earlier version of RUMP [1], we employed the HDP-HSMM model and investigated the use of the Hybrid Bayesian Particle Swarm Hyper-Parameter Optimization (HBPSHPO) model [16] only. However, the possibility of achieving more performance improvements via rigorous data preprocessing has not been explored, and no other predictive models have been investigated. In this paper, we provide an extended version of RUMP that incorporates data filtering techniques as a preprocessing step to induce performance improvements, particularly in terms of prediction accuracy. In addition, along with the HBPSHPO model, we analyze the use of two additional predictive models in terms of computational complexity and performance, namely, the pattern matching technique, kNN-TSPI [21], and the ML-based technique, HBLED [17]. Moreover, we conduct more extensive experiments and expand the performance evaluation metrics. Furthermore, we conduct performance ranking of RUMP and the other three prediction models by employing the Multi-Criteria Performance Measure (MCPM) alongside statistical tests, including the Friedman test and the Nemenyi post-hoc test. Lastly, as opposed to existing schemes that rely on simulations, we conduct extensive experiments on a realistic testbed of heterogeneous workers.

3. Resource usage multi-step prediction (RUMP)

In RUMP, we model the workers' dynamic resource usage using a Hidden Semi-Markov Model (HSMM) to enable multi-step prediction of the workers' resource usage state. The implementation uses the Hierarchical Dirichlet Process (HDP) for a Bayesian nonparametric extension of HSMM [22]. The extension of HDP-HSMM enables the model to infer the number of hidden states using a weak-limit Gibbs sampling method without requiring prior knowledge. This is since the workers' usage behavior and the applications runs tend to vary from one user to another. Moreover, the use of explicit duration semi-Markov modeling overcomes the limitation of the resource usage data being non-Markovian, since the time when the next resource usage state is reached depends on the time spent in the current state and not just the state itself. To learn the model, the resource usage information of the worker, such as CPU time and memory percent usage, is collected at a fixed interval over a long enough period to capture all the possible resource usage states. A graphical representation of all components of HDP-HSMM is shown in Fig. 1.

3.1. Hierarchical Dirichlet process - hidden semi-Markov model (HDP-HSMM)

In this section, we present the underlying HSMM and HDP components of the prediction model used in RUMP. All the notations used in this section are shown in Table 1.

3.1.1. The HSMM model in RUMP

Much like the Hidden Markov Model, the Hidden Semi-Markov Model is composed of two layers; a hidden state layer and an observation/emission layer, represented by random variables. However, the HSMM additionally expresses the time duration of states, which is limited to a geometric distribution in HMM. An application running on a worker, which is considered as a hidden state from the perspective of an EC service provider, has a similar relation to the corresponding resource usage pattern. Note that the latter can be viewed as emission from a distribution of values previously observed. State transition patterns can be modeled probabilistically, where each entered state is given an explicit duration, also drawn from a distribution. Such a Semi-Markov model is defined as an explicit duration Semi-Markov model. Thus, the model is able to capture different time scales of resource usage patterns, which is particularly important for EEDs with diverse workloads and performance characteristics.

The observations are represented by $y_t \in \mathbb{R}^+$ for $t \in \{1, 2, \dots, T\}$, where T is the length of the observation sequence. In RUMP, the observations are the values of the resource usage information, such as CPU time, memory percent usage, and network rates. Moreover, the observations, drawn from a distribution corresponding to a hidden Semi-Markov state, are represented by a sequence of random variables $x_t \in \{1, 2, \dots, N\}$, forming a Markov chain for the sequence of N possible states. The hidden states represent the resource usage states associated with applications and processes running on an EED. The state transition matrix, which collects the transition probabilities in a row-stochastic matrix, is denoted P_{ij}, where $P_{ij} = p(x_{t+1} = j|x_t = i)$. The emission distribution with parameters $\{\theta_i\}$ is represented by the probability $p(y_t|x_t, \theta_i)$.

The HSMM is augmented with a random variable, denoted D_s, representing state duration time drawn from a distribution specific to the entered state with the probability mass function $p(d_s, |x_t = i, \omega_i)$, where $\{\omega_i\}$ are the duration distribution parameters. Once D_s of an entered state passes, a Markov transition occurs towards a different state. To simplify representation, a sequence of the same hidden state is represented by "super-states" z_s, where $s \in \{1, 2, \dots, S\}$ for S possible super-states. Each super state has associated resource observations, multiple consecutive resource usage states, and avoids self-transition within the time D_s (i.e. a super-state transitioning to another instance of the same super-state). In RUMP, the super-states also represent the resource usage states associated with the hidden-states x_t over the duration D_s. More details of how self-transitions are eliminated using an additional auxiliary variable can be found in [22]. Note that we assume that the models are time-homogeneous. This means that the transition probabilities do not change with time. However, the model may be modified to adapt to changes in the underlying data over time using extensible Markov Models [35]. The worker's dynamic resource usage scenario allows the models to adapt with time to regularly evolving worker usage behaviors. The state transition probabilities of the model are derived from a given sequence of observations/emissions using the standard message-passing inference of the forward–backward algorithm, also called smoothing [22].

3.1.2. State-inference using HDP

The HSMM generation method used in RUMP relies on the Hierarchical Dirichlet Process (HDP) [22]. The HDP allows for Bayesian non-parametric inference of the hidden states (resource usage states) and hidden-state duration distributions, instead of the usual treatment from a non-Bayesian perspective that approximates parameters using the Expectation–Maximization algorithm. The HDP allows the model to automatically determine the optimal number of states and transitions without manual intervention, allowing for a more data-driven and adaptive approach to predicting resource usage. As such, the model parameters are treated as random variables with HDP priors $p(\{P_t\}|\alpha)$, $p(\{\theta_i\}|H)$, and $p(\{\omega_i\}|G)$, where $\alpha > 0$ is a concentration parameter specifying the shape of the distribution, and H and G are the base measures used to parameterize the emission distribution and duration distribution, respectively. The base measures are sampled from the

Table 1
Table of notations for RUMP.

Symbol	Description
y_t	Observations, for $t \in (1, 2, \dots, T)$
T	Length of the observation sequence
x_t	Hidden states $\in \{1, 2, \dots, N\}$
N	Number of possible hidden states
P_{ij}	State transition probability
θ_i	Emission distribution parameters
z_s	Super-state sequence, $s \in (1, 2, \dots, S)$
S	Number of possible super-states
D_s	Duration time of super state s
ω_i	Duration distribution parameters
α	Concentration parameter
H	Base measure for emission distribution
G	Base measure for duration distribution
L	Dimension of the Dirichlet distribution
κ	Sticky parameter

resource usage data. The inference approach enables the modeling of uncertainty over parameters, and the prediction of observations and state sequences by integrating out all possible parameters. In RUMP, H and G are learned from the resource usage data. H serves as a prior distribution of the parameters of the emission distribution (i.e., the distribution of resource usage observations given a particular state), while G serves as a prior distribution of the parameters of the duration distribution (i.e., the distribution of the time spent in a particular state).

The HDP acts as a prior over infinite-state transition matrices biased by a set of states that are consistently re-entered in each smoothing iteration, i.e., the draw of the forward–backward message passing algorithm, which is itself parameterized by a discrete measure β. The β parameter penalizes large numbers of states and reduces them to the states that are consistently visited in the sequence. A weak-limit Gibbs sampler is used to completely represent the transition matrix in a finite form based on L-Dimensional Dirichlet distributions. The L parameter can be sampled over without being set beforehand using the beam sampling technique [36]. The κ parameter allows for some control over duration statistics (i.e., encouraging longer or shorter periods), which is known as the sticky property [22]. With these parameters, the sampler constructs the other parameters by drawing samples from the posterior probability $p(\{x_t\}, \{\theta_t\}, \{P_t\}, \{\omega_t\}|y_t, H, G, \alpha)$. Moreover, the sampler accelerates mixing (i.e., estimating contributions of sources to a mixture) by allowing for block sampling of the entire state sequence simultaneously. Thus, the Gibbs sampling approach iteratively updates the model parameters based on the resource data and the current estimates of the other parameters, allowing the model to converge to a more accurate representation of the underlying data-generating process. Through this iterative process, the model learns the structure of the data, such as the number of resource usage states, the transition probabilities between the states, the resource usage data distributions of each state, and the duration distributions of each state.

3.2. Data filtering

Time-series data often contains unwanted components, such as trend, seasonality, and volatility, which can affect the accuracy of prediction models. To mitigate the impact of these unwanted components on the accuracy of prediction models, various preprocessing techniques can be applied to the time-series data to remove or reduce these components [37,38]. One such technique is the simple moving average (SMA) filter [39], which calculates the average of the data points within a specified window and moves the window across the data to create a smoothed series. The SMA filter is easy to implement and can provide a basic estimate of the future values. However, for time-series data that has trends or seasonality, the exponential moving average (EMA) [40] filter may perform better, since it assigns more weight to the most recent observations and can more effectively capture trends in the data. The EMA filter reduces the lag between the smoothed series and the original series by giving less weight to older data points.

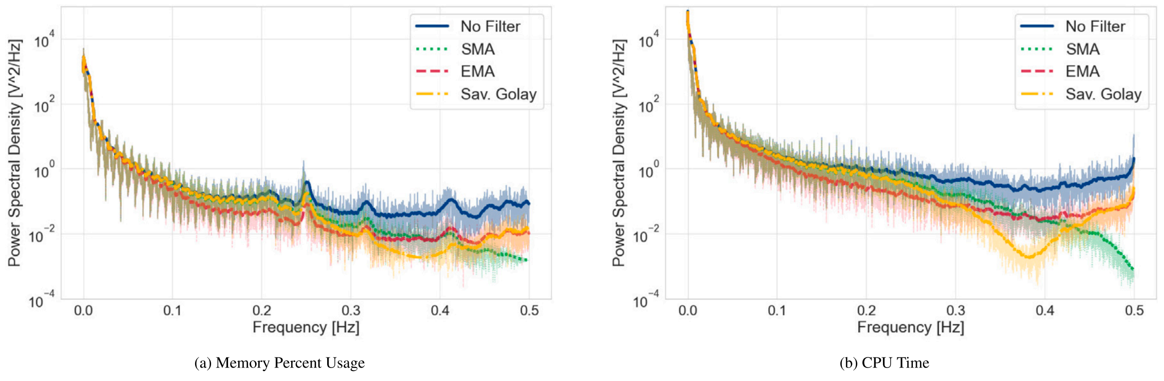


Fig. 2. Periodogram for dynamic resource usage in the dataset, the thin lines show the smooth Power Spectral Density of the corresponding color-matching signal.

The Savitzky–Golay filter [23] is a sophisticated digital filter that can handle data with both trends and noise. It is suitable for data that is highly irregular and that can be smoothed while preserving its overall shape. The Savitzky–Golay filter operates by fitting a polynomial function to a subset of adjacent points in the data and using the function to smooth the data. The degree of the polynomial function and the size of the subset can be modified to balance the trade-off between retaining trends in the data and eliminating noise.

The periodogram in Fig. 2 shows the Power Spectral Density (V^2/Hz) to frequency (Hz) spectrum of the time series data, which helps us understand the impact of the filter on the data. As shown in Fig. 2, the periodogram reveals a strong low-frequency component in the resource data, indicating that there are significant fluctuations over long periods of time. To smooth out these fluctuations and capture the overall trends in the data, we can apply either the Simple Moving Average (SMA) filter or the Exponential Moving Average (EMA) filter. The SMA filter can address autocorrelation at smaller lags by smoothing out short-term fluctuations, but it also reduces the power spectral density at higher frequencies, meaning it dampens the higher frequency oscillations in the data. Moreover, the EMA filter can account for autocorrelation that arises due to trends and seasonality, making the data more stationary (i.e., having constant mean and variance over time).

In contrast to SMA and EMA filters, the Savitzky–Golay filter smooths out the noise in the data, in addition to preserving the overall shape of the time series. The Savitzky-Golay filter can address the autocorrelation that arises from both trends and noise present in the time series data having complex patterns, noise, and trends. This may be observed in the sudden attenuation in the power spectral density for a specific range on frequency components, shown in Fig. 2(b). The frequency range may be controlled with the window size and polynomial degree chosen. Based on our analysis, we recommend using the EMA filter, for two reasons. First, it assigns more weight to recent data points, which makes it more responsive to changes in the data. Second, it follows the unfiltered data more closely, but with a consistent attenuation, which reduces noise and preserves the signal, which is especially crucial for reducing errors that may compound in multi-step ahead resource usage prediction.

3.3. Use cases

In addressing the complexities of resource usage in EEDs, RUMP benefits a broad range of use cases. One key use case is that of resource planning and allocation, where RUMP's predictive capabilities facilitate advanced provisioning of resources in EEDs. By gauging future resource demands, organizations can strategically plan and assign resources, ensuring uninterrupted delivery of offloaded services. This predictive resource allocation presents an innovative approach to contend with the highly dynamic nature of resource utilization, which is a salient research problem inherent in EEC environments.

Another crucial use case lies in task scheduling and load balancing. Harnessing RUMP's understanding of prospective resource usage, scheduling and load distribution can be significantly enhanced. By taking into account the projected resource needs of EEDs, tasks can be appropriately distributed to avoid resource competition and boost system performance. For instance, using a dynamic usage-aware scheduler, benchmark tasks can be allocated based on predicted resource states. This enables not only precise characterization of devices under diverse usage states but also timely execution of tasks, effectively capturing the dynamic essence of resource consumption.

Managing user-requested tasks is another beneficial use case for RUMP. With accurate forecasts of resource usage, tasks requested by users can be scheduled judiciously. This capability is particularly valuable in situations characterized by high resource competition and fluctuating usage patterns. By anticipating the future state of resources, RUMP can guide task allocation to prevent potential resource scarcity or overloads, thereby establishing a balanced and efficient EEC system.

RUMP is also instrumental in energy management and QoS optimization. Predictive insights into resource usage can support energy-efficient strategies by allowing the application of power-saving techniques during low demand periods, optimizing energy consumption. Concurrently, proactive management of QoS requirements is enabled, since service providers can allocate resources based on anticipated computational needs of user-owned devices. This ensures that QoS targets are consistently met and user experiences are continually improved. By addressing these research problems, RUMP presents a promising avenue for the advancement of EEC systems.

4. Complexity analysis

In this section, we showcase the complexity of the HDP-HSMM model used in RUMP in addition to the multiple baseline models: (1) The Hybrid Bayesian Particle Swarm Hyper-Parameter Optimization (HBPSHPO) model [16], (2) The Hybrid Bidirectional LSTM Encoder-Decode (HBLED) [17], and (3) The kNN-Time Series Prediction with Invariance (kNN-TSPI) [21]. All models are used for resource usage prediction, and we analyze their complexities in terms of Big O notation.

4.1. HDP-HSMM

The most computationally intensive part of HDP-HSMM model used in RUMP results from the message passing step using the forward–backward algorithm, i.e. the smoothing process. The algorithm has a complexity of $\mathcal{O}(T^2 N_{\text{States}} + T N_{\text{States}}^2)$, where N_{States} is the number of hidden states (i.e., the resource usage states) and T is the observation sequence length (i.e., the resource usage data). However, not all steps in the sequence need to be considered. Thus, it is enough to consider the steps where a change in the super-state (i.e., the long running resource usage states) is most likely. Such change are referred to as the change points. The complexity is reduced to $\mathcal{O}(T_{\text{Change}}^2 N_{\text{States}} + T_{\text{Change}} N_{\text{States}}^2)$ if change-point detection is used when using the weak-sampler on the observations, where T_{Change} is the number of possible change points, which is much less than T.

4.2. HBPSHPO

The LSTM-CNN based model in HBPSHPO [16] uses the Particle Swarm meta-heuristic (PS) to optimize the model architecture until it stops according to termination criteria with value C. Thus, the number of PS rounds, denoted N_{PSO}, is a function of the termination criteria value $N_{\text{PSO}} = f(C)$. HBPSHPO also uses Bayesian optimization to optimize the selection of activation and loss functions within N_{BO} number of iterations per particle in the PS. Moreover, a separate LSTM-CNN model needs to be trained for each step size, denoted N_{Steps}. Thus, the number of LSTM-CNN models trained is $N_{\text{Models}} = N_{\text{PSO}} \times N_{\text{BO}} \times N_{\text{Steps}}$. Each model has a complexity of $\mathcal{O}(UT)$, where U is the number of LSTM-CNN units and T is the input data sequence length. It follows that the complexity of the HBPSHPO method is $\mathcal{O}(TUN_{\text{Models}})$. Note that the complexity of U is different for each layer type, i.e., LSTM and CNN. For an LSTM layer, the complexity is typically $\mathcal{O}(L^2T)$, where L is the number of LSTM units. For a Conv1D layer, the complexity is $\mathcal{O}(f'kI_N I_S/s')$, where f' is the number of filters, k is the kernel size, I_N is the number of input channels, I_S is the input size, and s' is the stride of the convolution.

4.3. HBLED

The HBLED model's complexity is largely determined by the layers it contains and their individual complexities. The most computationally intensive layers in HBLED, are the Bi-directional LSTM, LSTM and the Time Distributed Dense layer. For each LSTM layer, the complexity is typically $\mathcal{O}(TL^2)$, where T is the input data sequence length and L is the number of LSTM units. Bidirectional LSTM layers have twice the complexity of a single LSTM layer, i.e., $\mathcal{O}(2TL^2)$. The Time Distributed Dense layers have a complexity of $\mathcal{O}(TF^2)$, where F is the number of features. The model's total complexity is the sum of the complexities of each layer, and is multiplied by the number of steps considered, since each step size requires a separate model. Thus, the total complexity is $\mathcal{O}(T(3L^2 + F^2)N_{\text{Steps}})$.

4.4. kNN-TSPI

The kNN-TSPI model has a time complexity of $\mathcal{O}(TQk)$, primarily determined by the similarity search, where k is the number of neighbors, T is the size of the time series, and Q is the length of the subsequences. This means that the time complexity of kNN-TSPI depends on both the size of the time series and the length of sub-sequences used for matching. Compared to traditional kNN, which has a time complexity of $\mathcal{O}(T \log T)$, the kNN-TSPI model can be more computationally expensive, especially for large Q and k. The kNN-TSPI has a linearly scaling time complexity as the length of the sub-sequences Q and the number of neighbors k are of small and fixed numbers compared to T, however, the model is univariate and is incapable of multi-feature prediction. To predict on multiple features, a separate model is required for every feature F, therefore, the time complexity of kNN-TSPI for our use case is $\mathcal{O}(TFQk)$. It should be noted that there are some additional computational overheads due to techniques used for obtaining amplitude and offset invariance ($\mathcal{O}(Q)$), complexity invariance ($\mathcal{O}(Q)$), and treatment of trivial matches ($\mathcal{O}(k)$). These additional overhead complexities are small compared to the main complexity term.

4.5. Comparison

When comparing the time complexities of the different models, we find that they fall into two categories: low complexity and high complexity. These categories are based on the dominant components that make up the different complexities in the context of multi-step prediction of EED resource usage. The HDP-HSMM used in RUMP and the kNN-TSPI fall into the low complexity category, while the HBLED and the HBPSHPO fall into the high complexity category. The HDP-HSMM in RUMP has low complexity when using change point detection, which reduces the dominant term T^2 to T_{change}. The kNN-TSPI has low complexity due to its linear relation to T, but it has an additional term F because it requires a separate model for each predicted feature. As a result, for the kNN-TSPI, predictions of features are made without considering their interdependence, which limits generalization. The kNN-TSPI is unsuitable for our use case for two reasons: (1) it cannot predict resource usage states (labels) because it is only capable of making numeric predictions, and (2) it does not adapt to changes in data patterns over time. As a pattern matching technique, if a similar pattern is not found in historical data, it cannot accurately predict resource usage data.

Both the HDP-HSMM in RUMP and kNN-TSPI are independent of the number of steps considered, making their complexities lower than those of HBLED and HBPSHPO. The complexities of the ML-based models, HBLED and HBPSHPO are higher, since they depend on both the size of the training data and the layer composition of the models. The data used for our experiments have a sizable observation sequence length T, on the order of 10^4. For HBLED, the number of LSTM units (L) is on the order of magnitude of 10^2, while for HBPSHPO, the number of LSTM or Conv1D units is on the order of 10^9. F, k, and Q have the lowest magnitudes compared to the other terms. While other the complexity terms are ranked in order of magnitude as follows: $U \gg T \gg T_{Change} \gg L \gg N_{Models} \gg N_{States}$. In the next section we conduct a performance evaluation to compare the models' accuracy in prediction resource usage data for multiple steps.

5. Performance evaluation

In this section, we evaluate the performance of the HDP-HSMM model used in RUMP compared to three representative state-of-the-art prediction models; the Hybrid Bi-directions LSTM Encoder–Decoder (HBLED) model [17], the k-Nearest Neighbors Time Series Prediction with Invariance (kNN-TSPI) model [21], and the Hybrid Bayesian Particle Swarm Hyper-Parameter Optimization (HBPSHPO) model [16].

We use the following performance metrics: (1) the Mean Absolute Error (MAE), which is calculated as the average magnitude of errors between the predicted and actual values without considering their direction, (2) the Root Mean Square Error (RMSE), which represents the square root of the average squared differences between the predicted and actual values, and is sensitive to outliers, more severely penalizing larger errors compared to smaller ones, (3) the Coefficient of Determination (R^2), specifically $1 - R^2$, and is used to assess the proportion of unexplained variance in the data, where a smaller $1-R^2$ value indicates better model performance due to the model's ability to account for a greater amount of variance, (4) the Mean Absolute Percent Error (MAPE), which is calculated as the average percentage of error between the predicted and actual values, and that is useful for comparing model performance on data with varying scales, (5) the Symmetric Mean Absolute Percent Error (SMAPE), which serves as a variation of MAPE that takes into account the direction of the difference between the predicted and actual values (i.e., positive or negative difference), and (6) the Prediction Accuracy (%) for categorical predictions, which is calculated by dividing the number of accurately predicted labels by the total number of labels.

To compare and rank the performance of the models, we conduct various experiments, and use the Multi-Criteria Performance Measure

Table 2
Workers' specifications and labels.

Worker	Raspberry Pi 4B specifications	
	RAM (GB)	CPU freq. (GHz)
A	8	1.8
B	4	1.5
C	2	1.5
D	2	1.2

(MCPM) [24], as well as two statistical tests [25]. The MCMP is used to compare the different metrics based on direction (the proportions of different metrics) rather than magnitude (the absolute values of metrics). The statistical tests are performed to assess whether there are significant differences among the models and filters. The Friedman's test is used to compare multiple groups across multiple dependent variables. It compares the ranks of the models across all datasets and provides a *p*-value, indicating whether there is a significant difference between the models. When there is a significant difference, the Nemenyi post-hoc test is used to compare the models' ranks pairwise and provides a critical difference value that indicates which models differ significantly from each other [25].

5.1. Experimental setup

In order to assess the multi-step prediction models on heterogeneous workers that accurately reflect EEDs in various usage scenarios, we utilize a group of four devices, specifically the Raspberry Pi (RPi) 4B model, with varying RAM sizes and CPU cycle frequencies. It is important to note that the standard CPU frequency of the RPi 4B model is 1.5 GHz. However, we increase the heterogeneity of workers by overclocking and throttling the RPis. Consequently, we utilized CPU frequencies of 1.8, 1.5, and 1.2 GHz, along with RAM sizes of 8, 4, and 2 GB, respectively. Table 2 summarizes the specifications of the RPis (i.e., workers) used.

We implement dynamic resource usage scenarios by sequentially running a set of applications for different durations. The applications include playing a video game, streaming a YouTube video on a browser, emulating real-time augmented reality, and mining a low-power cryptocurrency called Duino-coin. To emulate augmented reality, we impose ArUco markers on a sequence of image frames, similar to a live video stream. The Duino-coin uses the SHA-1 cryptographic function for encryption on a variant of a blockchain, known as a hash-chain. We also include idle periods where no application is running. To create different usage scenarios (known as resource usage states), we use two types of sequence for each worker: a random sequence and a patterned sequence, which consists of a recurring sequence of applications over a specific duration.

The full resource usage dataset encompasses over 550,000 distinct data points spanning 768 h of applications running on EEDs. The data is gathered from various resource usage scenarios over numerous 48-hour periods, with a monitoring interval of 5 s. Applications run sequentially, and the duration of each is determined by the previously computed state time lengths. The resource usage state time lengths are calculated considering factors such as the total runtime, various application runtime lengths, and the maximum application runtime length. Resource usage is continuously monitored and documented at the specified intervals using the `psutil` Python library [41]. The datasets capture user CPU time, system CPU time, and idle CPU time, percent memory usage, network upload and download size and rates, disk IO, and more. We focus on modeling user, system, and idle CPU time, as well as memory usage. We associate resource usage information with resource usage states using labels, such as "Game", "Stream", "Augmented Reality" (AR), "Mining", and "Idle", which are then given numerical labels for input into the prediction model.

For performance evaluation, we focus on four datasets per worker; two for each sequence type, labeled "random" and "patterned". In terms of resource usage values, we focus on percent memory usage, as well as user, system, and idle CPU times. Fig. 3 illustrates a sample distribution of resource usage for worker A following the random sequence type for the different resource usage states. The figure shows the distinction between the predicted resource usage states and their corresponding actual values. As depicted in Fig. 3(a), Fig. 3(b), and Fig. 3(c), the "Game" state exhibits the highest mean values of user CPU time, idle CPU time, and system CPU time, respectively, followed by the "Mining", "Stream", and "AR" states. In addition, as shown in Fig. 3(d), the "Stream" state leads in terms of the mean value of percent memory usage, followed by the "AR", "Mining", and "Game" states. Whereas the "Idle" state is naturally the lowest in all resource usage values.

In terms of preprocessing, differentiating and filtering are applied to the data. The differentiation process involves calculating the first-order difference for each of the resources to remove running trends, and high frequency-based variations/seasonality. The filtering process involves the EMA filter, where we use a window size of 2. In RUMP, κ is the only parameter that requires tuning based on the distribution of the resource usage states duration. We set κ to 0.05 with filtering and 0.1 without filtering. We conduct different experiments over varying number of steps, set to 1, 2, 5, 10, and 15, corresponding to 5, 10, 25, 50, and 75 s, respectively. We split the datasets into training and testing sets using a 70%–30% split. The code and data used for the experiments are accessible via GitHub.[1]

5.2. Results and analysis

In our experiments, we study the impact of the number of steps and data sequence types, as well as the impact of data filtering on the performance of RUMP, kNN-TSPI, HBLED, and HBPSHPO. We also assess the categorical multi-step prediction for each worker in RUMP. In addition, we discuss the trade-off between complexity, performance, and effectiveness among RUMP, kNN-TSPI, HBLED, and HBPSHPO. Below is a detailed discussion of such experiments and analysis.

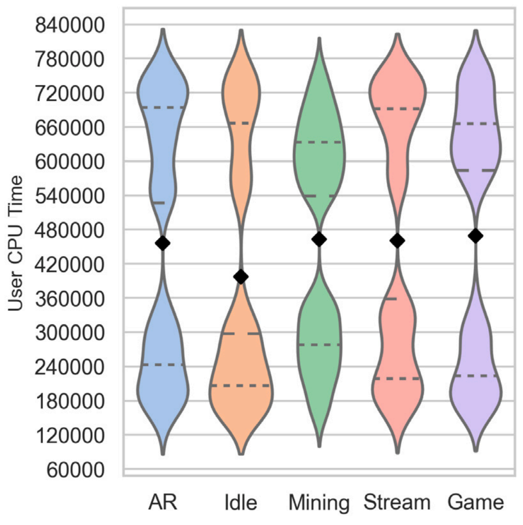
1- The Impact of the Number of Steps and Data Sequence Type

Fig. 4 depicts the performance of RUMP, kNN-TSPI, HBLED, and HBPSHPO in terms of the average MAE of the user CPU time (Fig. 4(a)), idle CPU time (Fig. 4(b)), system CPU time (Fig. 4(c)), and memory percent usage (Fig. 4(d)), for both random (R) and patterned (P) state sequences, over varying step sizes. It can be observed that as the number of steps decreases, all the models can effectively capture finer patterns in the data, due to their use of more frequent data points. However, as the number of steps increases, the models tend to provide a more generalized understanding of data trends, possibly sacrificing some level of detail in the predictions. Thus, all models exhibit various degrees of gradual increase in MAE as the number of steps increases.

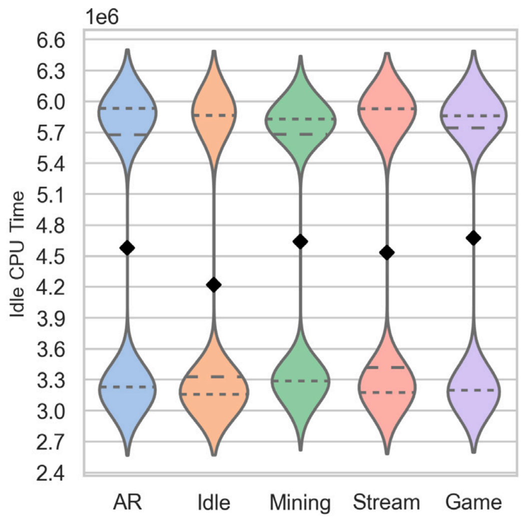
Fig. 4 also shows that the predictions made using the different models more often show better performance when predicting on patterned (P) data in comparison to random (R) data sequence. This indicates that HBPSHPO, HBLED, and kNN-TSPI may be overfitting to the patterned sequence data, leading to a gain in accuracy at the expense of model generalization. In contrast, RUMP follows the same pattern for the user CPU time and idle CPU time, but not for the system CPU time and memory percent usage.

Considering the average percent difference between RUMP and the other models in terms of MAE, for the values in Fig. 4, we observe that RUMP outperforms HBPSHPO for the random data sequence by 3% and 4% at step sizes 5 and 10, respectively. Whereas HBPSHPO outperforms RUMP by 6%–16% for the random and 11%–37% for the patterned data sequence at the other step sizes. It can also be noted that HBLED and

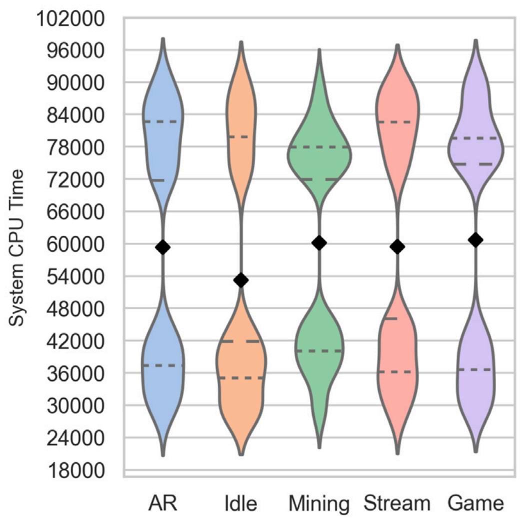
[1] https://github.com/RuslanKain/rump-ec



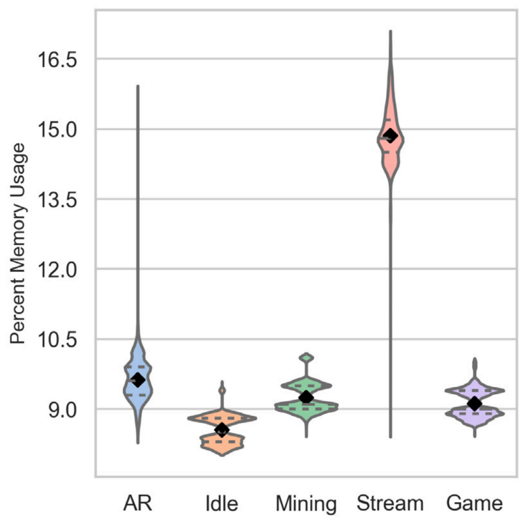
(a) User CPU Time



(b) Idle CPU Time



(c) System CPU Time



(d) Percent Memory Usage

Fig. 3. Distribution of the preprocessed Resource Usage for different states in random sequence for Worker A. The bottom and top dashed lines in the violin plots indicate the 25th and 75th quantiles, respectively, the middle dashed line represents the median, and the black dot indicates the mean.

kNN-TSPI outperform RUMP for both the random and patterned data sequences by 25%–81% and 67%–85%, respectively. This is because of the nature of the HDP-HSMM model used in RUMP, which, unlike the other models, does not rely on a function that maps input to output, but rather draws the output from a probability distribution. The probability distribution is associated with a resource usage state, that is recognized by RUMP based on the input values. Thus, output values in RUMP tend to be more volatile since they are drawn from a probability distribution. This volatility can be seen in Fig. 5, which depicts the actual and predicted system CPU time on worker A for 1-step predictions yielded by RUMP and the other models. As shown in the figure, the predictions made by RUMP tend to fluctuate more than those made by the other models which more closely follow the level of the actual observations. The volatility shown by RUMP does not mean that the inferences made by RUMP are less useful, since the hidden state corresponding to the predicted observation may still be correct, (as explained later). Moreover, the fluctuations in all the models' predictions are also product of the variability of the resource usage data itself, which merits the use of filters to smooth out the signal with the aim of reducing prediction errors. The next experiments show that when incorporating the EMA filtering technique into RUMP, it starts to outperform HBLED.

To show the impact of large stepsizes, Fig. 6 depicts two sets of violin plots to show the percent difference of RMSE and MAE between RUMP and HBPSHPO for both patterned and random data for 1 to 15 steps (Fig. 6(a)), and for 1 to 60 steps (Fig. 6(b)). Note that in scenarios where predictions are made for large step sizes, the kNN-TSPI and HBLED models are rendered impractical. This is because each step size in kNN-TSPI and HBLED requires a new model, and re-training must be applied when new resource usage data is collected. Moreover, HBLED and kNN-TSPI have higher complexity, due to the number of hyper-parameter searches for model architecture optimization technique and the additional kNN-TSPI models needed for each predicted feature. Thus, in this experiment, we focus on RUMP and HBPSHPO.

As shown in Fig. 6(a) and Fig. 6(b), the prediction accuracy of RUMP is comparable to the more computationally complex HBPSHPO model for all step sizes. The additional step sizes, 30 and 60, cause the distribution of the violin plots to have wider bottoms, where both the mean and median values are lower, indicating a relatively improved performance of RUMP for larger step sizes. This means that RUMP is

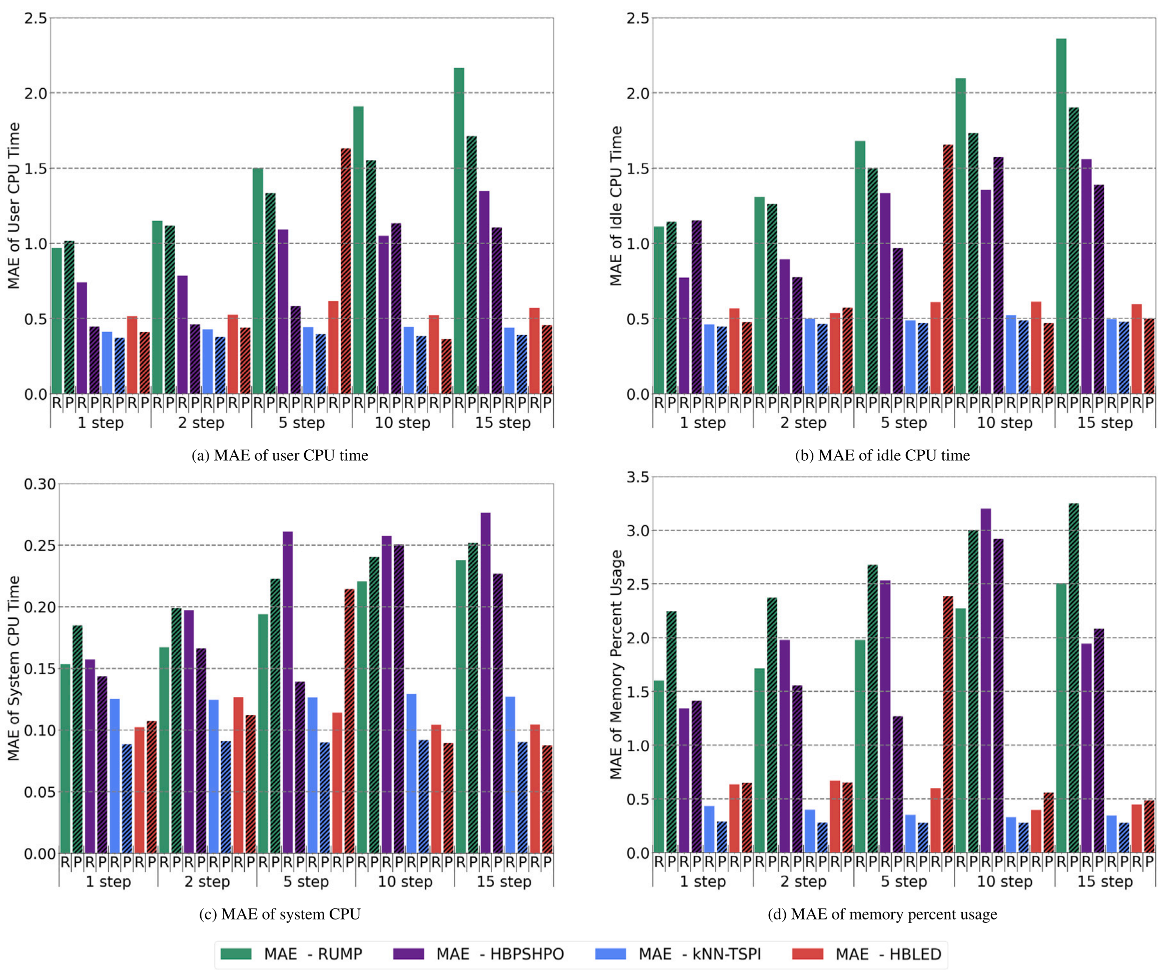


Fig. 4. Average mean absolute error of RUMP, HBPSHPO, kNN-TSPI, and HBLED for all workers for both random (R) and patterned (P) state sequences over varying number of steps.

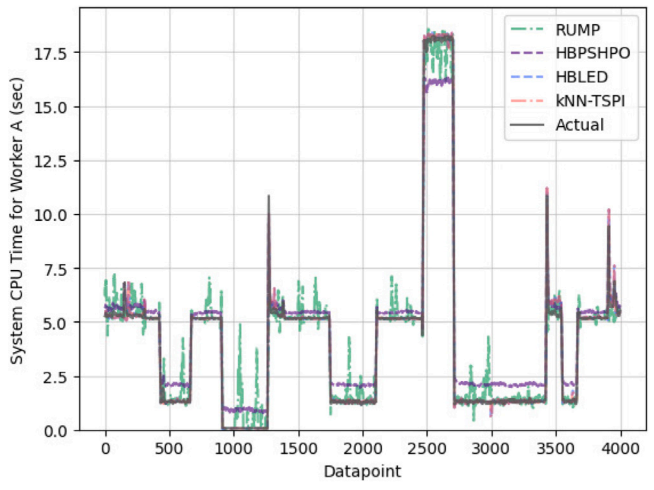


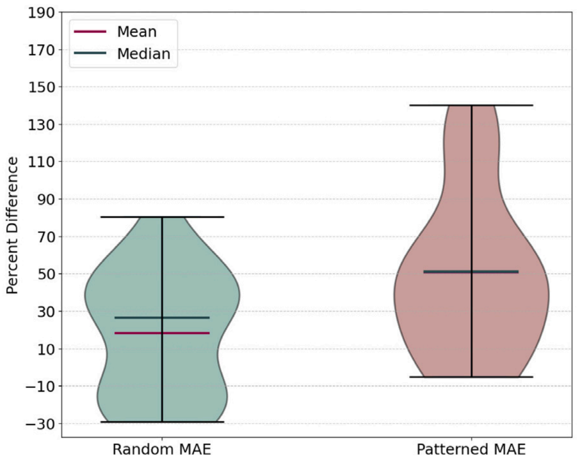
Fig. 5. Actual (gray) and predicted CPU time for worker A by RUMP (green), HBPSHPO (purple), HBLED (blue), and kNN-TSPI (red).

more capable of capturing long-term patterns in the data. In Fig. 6(b), the overall average percent difference is 28% in MAE. Note that the average percent difference in MAE between RUMP and HBPSHPO for patterned sequence data is 39% and for the random sequence data 17%, in favor of the HBPSHPO model. In some cases, RUMP even outperforms HBPSHPO; for example, the minimum difference in MAE for random sequence data is -30% (i.e., in favor of RUMP).

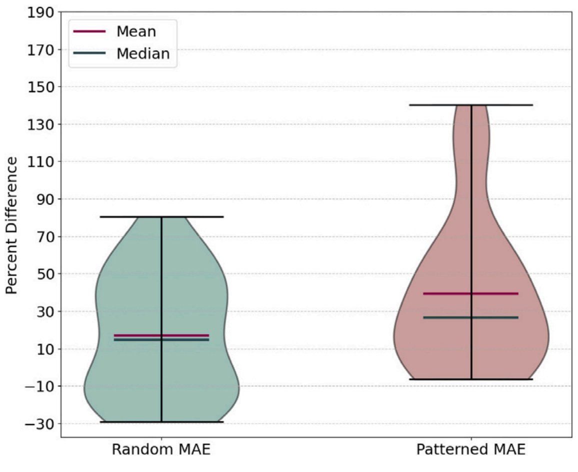
2- The Impact of Filtering

In this experiment, we study the impact of data filtering on the performance of RUMP, HBLED, and kNN-TSPI. Fig. 7 depicts each of these models with and without EMA filters. For simplicity, we refer to them as RUMP-No Filter, RUMP-EMA, HBLED-No Filter, HBLED-EMA, kNN-TSPI-No Filter, and kNN-TSPI-EMA. We conduct the Multi-Criteria Performance Measure (MCPM) [24] to evaluate and rank the performance of these models based on multiple metrics; MAE, RMSE, MAPE, $1 - R^2$, and SMAPE, at different step sizes. Towards that end, the L2 normalized average of the evaluation metrics is considered and mapped on a radar chart to obtain the formed polygon's area. The performance is ranked based on the minimal area size of the associated polygon. The area is calculated using the shoelace formula, an algorithm for computing the area of any simple polygon whose vertices are given by their Cartesian coordinates. L2 normalization is used so that comparisons between data points are based on direction (the proportions of different metrics) rather than magnitude (the absolute values of metrics). As depicted in Fig. 7, the results are visualized using a radar chart, a graphical method of displaying multivariate data in a two-dimensional chart with multiple axes starting from the same point.

We also use the statistical tests [25], namely, the Friedman test and the Nemenyi (post-hoc) tests to assess whether there are significant differences in performance among the models and filters. The Friedman's test is used to compare multiple groups across multiple dependent variables. It compares the ranks of the models across all datasets and

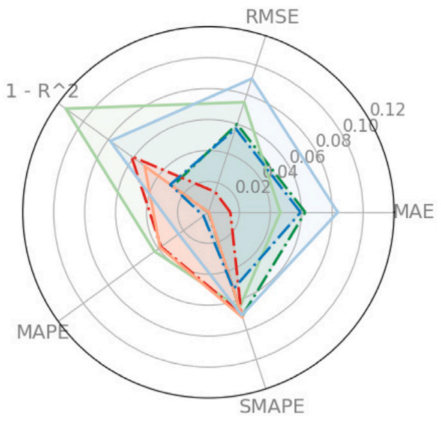


(a) 1 to 15 steps

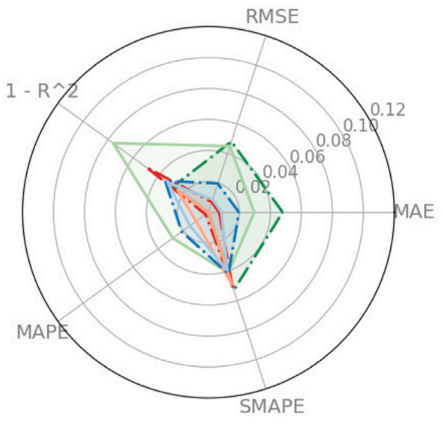


(b) 1 to 60 steps

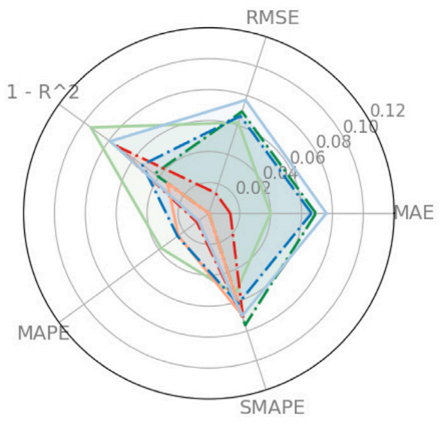
Fig. 6. The distribution of percent error difference in MAE between RUMP and HBPSHPO for different step sizes. Negative values indicate that RUMP outperforms HBPSHPO.



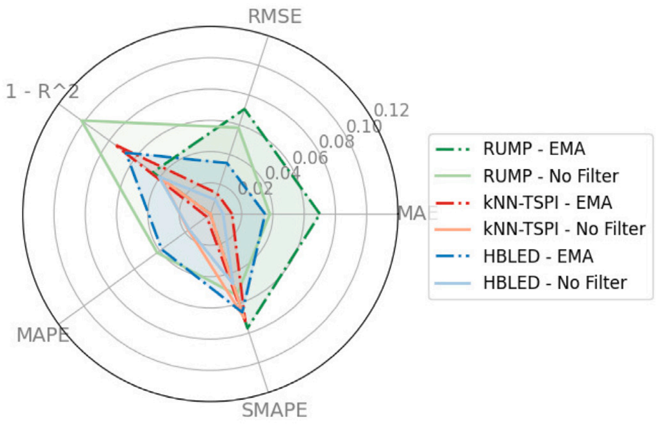
(a) 1 Step



(b) 5 Step



(c) 10 Step



(d) 15 Step

Fig. 7. Multi-Criteria Performance Measure for RUMP, KNN-TSPI, and HBLED, with and without EMA filter for different step sizes.

provides a *p*-value, indicating whether there is a significant difference between the models. When there is a significant difference, the Nemenyi post-hoc test is used to compare the models' ranks pairwise and provides a critical difference value that indicates which models differ significantly from each other [25].

As depicted in Fig. 7(a), the MCMP test for step size 1 shows that kNN-TSPI-No Filter yields the lowest score, followed by kNN-TSPI-EMA, HBLED-EMA, RUMP-EMA, RUMP-No Filter, and HBLED-No Filtering. This shows that EMA filtering renders better prediction accuracy in RUMP and HBLED, whereas filtering yields no improvements in kNN-TSPI, for 1 step prediction. Whereas, the Friedman test results in a *p*-value of 0.0169, which is below the threshold of 0.05, indicating that there are significant differences among the model-filter combinations. The Nemenyi test is computed for various performance metrics, such as MAE, RMSE, MAPE, and SMAPE, for all the considered resource usage values. With a critical difference value of 0.874, the test shows

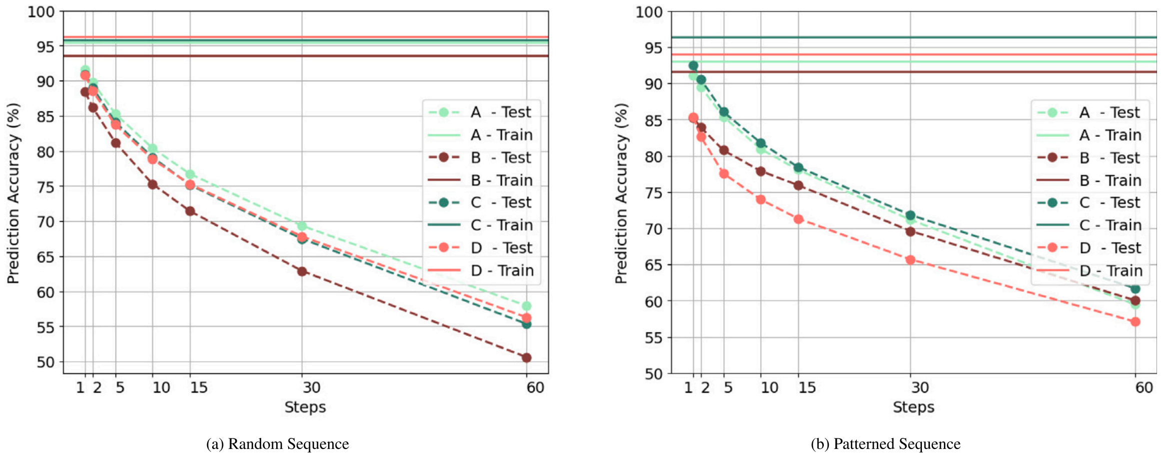


Fig. 8. Multi-step prediction accuracy for training and testing phase for all workers for the random and patterned sequences.

significant performance differences that RUMP-EMA significantly outperforms RUMP-No Filter. Rankings based on the Nemenyi test show that kNN-TSPI-No Filter and kNN-TSPI-EMA consistently perform better across multiple metrics. Although both the MCPM and Nemenyi test rankings show that kNN-TSPI performs well, the Nemenyi test results reveal a more nuanced picture of the performance across different metrics, highlighting the similarities in performance between the kNN-TSPI and HBLED models with the EMA filter.

As depicted in Fig. 7(c), the MCMP test for step size 10 shows that kNN-TSPI-No Filter performs the best, followed by kNN-TSPI-EMA, RUMP-EMA, HBLED-EMA, RUMP-No Filter, and HBLED-No Filter. the Friedman test shows a statistically significant difference with a p-value of 0.0256, while the Nemenyi test indicates that HBLED-No Filter performs worse than the other models.

The post hoc tests reveal that the EMA filtering method effectively improves the performance of RUMP and kNN-TSPI models for most metrics. RUMP-EMA and kNN-TSPI-EMA show statistically significant differences in several user and idle CPU time metrics compared to the other models. Notably, RUMP-EMA, HBLED-No Filter, kNN-TSPI-No Filter, and kNN-TSPI-EMA have comparable error for memory usage, while HBLED-EMA stands out as the most different from the rest in terms of memory resource usage, implying that it is the least accurate. The Nemenyi test results provide additional insights into the impact of EMA filtering on the performance of RUMP and kNN-TSPI, and highlight the significant differences between RUMP-EMA and kNN-TSPI-EMA and other models for several user and idle CPU time prediction error metrics.

Fig. 7(b) and Fig. 7(d) show the results of the MCMP test for step sizes 5 and 15, respectively. For 5-step predictions, it can be observed that kNN-TSPI-No Filter ranks first, followed by kNN-TSPI-EMA, HBLED-No Filter, HBLED-EMA, RUMP-EMA, and RUMP-No Filter. For step size 15, the kNN-TSPI-No Filter performs the best, followed by kNN-TSPI-EMA, RUMP-EMA, HBLED-No Filter, and RUMP-No Filter. Note that for step sizes 2, 5, and 15, the Friedman test indicates no significant differences between the model-filter combinations, as the p-values are greater than 0.05. The 15-step prediction has a large p-value of 0.3755, while the p-value for the 5-step predictions is 0.052, respectively. The p-value for 5-step predictions is quite close to the threshold, suggesting that the differences in model performance is bordering on statistical significance. Despite the lack of significant differences, the ranking of the model-filter combinations based on their MCMP offers valuable insights into their relative performance.

3- Categorical Multi-step Prediction

In this experiment, we evaluate the ability of RUMP to accurately predict the resource usage state (categorical prediction) of each worker (worker A, worker B, worker C, and worker D) over varying number of steps. Note that only RUMP is evaluated in this experiment since the other models are not designed to predict the resource usage state. For example, kNN-TPSI is not capable of performing categorical prediction, since it is a pattern matching technique. This renders kNN-TPSI impractical and unsuitable to predict the resource usage state of workers. In contrast, RUMP captures the dynamic resource usage state of workers, where the resource usage state of a worker in the HSMM model used in RUMP is represented by the hidden states. Fig. 8 depicts the prediction accuracy of RUMP in each worker for both the training and testing data, and for random sequence data in Fig. 8(a) and patterned sequence data in Fig. 8(b) over different step sizes. Note that the state prediction accuracy for the training data corresponds to 1-step predictions only, and is thus represented by horizontal lines since it remains the same over different step sizes. In contrast, the scattered dots indicate the accuracy achieved when using the model to infer on testing data for each step size. We discuss the results of such inference below.

Table 3
Distribution of 1-step predictions in terms of the percent correct state predictions (✓), incorrect prediction as other known states (×), and prediction of unknown states (?) for the testing-phase patterned sequence data of "Stream", "Game", and "Mining" resource usage states for all workers.

Worker	Resource usage states								
	Stream			Game			Mining		
	✓	×	?	✓	×	?	✓	×	?
A	81.1	9.8	9.1	96.6	0.6	2.8	77.0	2.7	**20.3**
B	77.0	2.0	**21.0**	64.3	17.7	**18.0**	83.4	7.1	9.5
C	85.1	2.9	12.0	92.6	0.4	7.0	92.4	2.5	5.1
D	67.2	18.3	**14.5**	88.5	2.7	8.8	61.6	9.7	**28.7**

As the number of steps increases, the prediction accuracy of RUMP decreases for all workers, where the average training-phase accuracy for all workers is 95% and 93% for random and patterned state sequences, respectively. For multi-step inference on unseen testing data, the average prediction accuracy ranges from 91% to 55%, and 89% to 60% for random and patterned state sequences, respectively. This reduces steepness for larger step sizes, suggesting a non-linear relation with step size due to error propagation from one prediction step to the next. In Fig. 8, the accuracy results of workers B and D show that they have a slightly lower starting accuracy compared to the other workers (i.e., at 1 step).

Table 3 helps explain the source of the aforementioned deviations by presenting the 1-step prediction accuracy on the unseen patterned sequence data for the "Stream", "Game", and "Mining" resource usage states for each worker. The table shows that accuracy is relatively low in the "Stream" state for workers B and D, "Game" for worker B, and "Mining" state for workers A and D. Besides having the state incorrectly classified as a different resource usage state, another source

Table 4
Model yime complexity, accuracy, practicality, and adaptability comparison for all models.

Model	Evaluation criteria			
	Time complexity	Accuracy	Practicality	Adaptability
RUMP	Low	Medium	High	High
HBPSHPO	High	Medium	Low	Medium
HBLED	High	High	Medium	Medium
kNN-TSPI	Low	High	Low	Low

of error is the prediction of a hidden state that does not directly correspond to the labeled resource usage state. This is due to the model generating unknown hidden states that are impacted by some unknown processes instead of the running application. These processes may include automatic operating system processes, which may run at any time, or the application itself uses resources abnormally because the application's operation is affected by an overloaded system. This indicates that the model itself is highly accurate, and prediction errors stem from the fact that the labels in the dataset may fail to completely capture the unknown resource usage states reflected in the resource usage patterns, which also impacts the worker's performance that the model can detect.

5.3. Complexity, performance, and effectiveness trade-offs

In this section, we discuss the trade-offs between the complexity, prediction accuracy, practicality, and adaptability of RUMP, HBPSHPO, HBLED, and kNN-TPSI in multi-step prediction of the workers' resource usage in dynamic EEC systems. Note that by practicality, we refer to the suitability/applicability of the prediction model to EEC systems, whereas adaptability means the ability to adapt the model to changes in data patterns over time, which are common due to the dynamic resource usage of EEDs. Table 4 summarizes the time complexity, accuracy, practicality, and adaptability comparison between RUMP, HBPSHPO, HBLED, and kNN-TPSI. Based on the complexity analysis conducted in Section 4, it can be observed that the complexity profile of kNN-TSPI is linear with respect to the size of the time series (T) and the length of the sub-sequences (Q). However, the time complexity of kNN-TSPI is multiplied for every additional feature (F), which limits its practicality. The time complexities of RUMP and kNN-TSPI models are lower compared to ML-based models, since they do not require several models to predict for multiple step sizes. The kNN-TSPI model does not require multiple models because it relies on the sliding window technique. However, since kNN-TSPI is a pattern matching-based technique, it is much less practical and adaptable than RUMP for EEC systems. In addition, kNN-TSPI is incapable of doing categorical prediction, rendering it unusable for resource usage state prediction. RUMP, HBLED, and HBPSHPO are capable of resource usage state (categorical) prediction. However, unlike RUMP, ML-based models (HBLED and HBPSHPO) require additional modifications and separate models to do categorical prediction, thus hindering their practicality.

In terms of complexity, the complexities of CNN and LSTM-CNN models depend on the input size, as well as other factors, such as the number of filters, the kernel size, and stride. The LSTM-CNN based model used in HBPSHPO also has an additional complexity factor, N_Models, which is influenced by the number of particle swarm optimization rounds and Bayesian optimization iterations. Comparing the two ML models together, the LSTM/CNN based HBPSHPO model and LSTM/Bi-directional HBLED model have different compositions of layers, resulting in distinct complexity profiles. The complexity of both models is affected by the choice of layers, their sizes, and input sequence length. Overall, HBPSHPO has the highest computational complexity compared to other approaches such as CNN-LSTMs or Bidirectional LSTMs, due to its larger number of parameters and more complex architecture.

In terms of adaptability, RUMP is highly adaptable if modified to use an extensible Markov Model [35]. The ML-based models (HBPSHPO and HBLED) are capable of being adapted using online-learning techniques. However, they face issues when the models abruptly and drastically forget previously learned information upon learning new information, which is known as catastrophic interference [42].

In the context of worker resource usage prediction, RUMP strikes a good balance between complexity and accuracy, making it a suitable choice for large-scale and dynamic environments. Although kNN-TSPI is the most accurate predictor, it has limited applicability, since it is univariate, and a combination of models for each feature is required for the prediction application. Moreover, it has a tendency for oversimplification as the algorithm focuses on a single variable and ignores the relationships between features.

In terms of practicality/applicability, RUMP is more practical for EEC because it employs a semi-supervised approach for validating automatically captured resource usage states, utilizing the Hierarchical Dirichlet Process (HDP) with Gibbs Sampling for state inference. This allows RUMP to accurately assess the device's state (reaching a prediction accuracy of up to 95%) unlike kNN-TPSI, which cannot perform categorical prediction, or HBLED and HBPSHPO that require additional modifications and separate models to do categorical prediction. As such, RUMP has the significant advantage of achieving a more comprehensive understanding of the device state changes and resource management, in comparison to the other models. Therefore, RUMP offers a desirable balance between computational complexity and prediction accuracy for multi-step multi-variate time series prediction in the context of worker resource usage, achieving accurate predictions while maintaining manageable computational complexity, which crucial for resource management in large-scale and dynamic environments.

6. Conclusion

In this paper, we have proposed the Resource Usage Multi-step Prediction (RUMP) scheme to model and predict the resource usage of heterogeneous workers in Extreme Edge Computing environments. RUMP deals with the highly dynamic user access behavior in such environments by using a Hierarchical Dirichlet Process-Hidden Semi-Markov Model (HDP-HSMM) to provide a relatively high predictive power without sacrificing computational efficiency or rendering too much complexity. Extensive evaluations on a realistic testbed have shown that RUMP is comparable to a representative of the state-of-the-art machine learning-based models in terms of multi-step and multi-variant prediction accuracy, while rendering a much lower computational complexity. In-depth assessments have demonstrated that RUMP achieves an 91% categorical multi-step prediction accuracy and has a small performance gap of 6% on average in terms of the Mean Absolute Error (MAE) when compared to representatives of state-of-the-art prediction models. Additionally, RUMP yields low computational complexity. In the future, we plan on integrating RUMP as part of an EEC resource management framework and evaluating it on a larger testbed of EDDs to investigate its scalability and performance.

CRediT authorship contribution statement

Ruslan Kain: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Visualization, Writing – original draft, Writing – review & editing. **Sara A. Elsayed:** Writing – review & editing, Methodology. **Yuanzhu Chen:** Writing – review & editing, Methodology, Resources, Supervision. **Hossam S. Hassanein:** Writing – review & editing, Funding acquisition, Resources, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The link and reference to the data/code are in the submitted manuscript.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used ChatGPT in order to improve readability. After using this service, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Acknowledgment

This research is supported by a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) under grant number ALLRP 549919-20, and a grant from Distributive, ltd.

References

[1] R. Kain, S.A. Elsayed, Y. Chen, H.S. Hassanein, Multi-step prediction of worker resource usage at the extreme edge, in: Proceedings of the 25th International ACM Conference on Modeling Analysis and Simulation of Wireless and Mobile Systems, MSWiM '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 25–32, http://dx.doi.org/10.1145/3551659.3559051.

[2] GSMA, The mobile economy 2022, 2022, The Mobile Economy. URL https://www.gsma.com/mobileeconomy/.

[3] W. Yu, F. Liang, X. He, W.G. Hatcher, C. Lu, J. Lin, X. Yang, A survey on the edge computing for the Internet of Things, IEEE Access 6 (2017) 6900–6919.

[4] S. Tsou, The need for computing power in 2020 and beyond, 2020, Forbes, URL https://www.forbes.com/sites/forbesbusinesscouncil/2020/01/24/the-need-for-computing-power-in-2020-and-beyond/?sh=400e8cb673c5.

[5] T.H. Noor, S. Zeadally, A. Alfazi, Q.Z. Sheng, Mobile cloud computing: Challenges and future research directions, J. Netw. Comput. Appl. 115 (2018) 70–85.

[6] A.J. Ferrer, J.M. Marquès, J. Jorba, Towards the decentralised cloud: Survey on approaches and challenges for mobile, ad hoc, and edge computing, ACM Comput. Surv. 51 (6) (2019) 1–36.

[7] I.M. Amer, S. Sorour, Cost-based compute cluster formation in edge computing, in: 2022 IEEE International Conference on Communications (ICC): IoT and Sensor Networks Symposium, IEEE ICC'22 - IoTSN Symposium, Seoul, Korea, 2022.

[8] D.J. Mays, S.A. Elsayed, H. S. Hassanein, Decentralized data allocation via local benchmarking for parallelized mobile edge learning, in: 2022 IEEE International Wireless Communications and Mobile Computing Conference: IoT and Sensor Networks Symposium, IEEE IWCMC'22, Dubrovnik, Croatia, 2022.

[9] R.F. El Khatib, S. A. Elsayed, N. Zorba, H. S. Hassanein, Optimal proactive resource allocation at the extreme edge, in: 2022 IEEE International Conference on Communications (ICC): IoT and Sensor Networks Symposium, IEEE ICC'22 - IoTSN Symposium, Seoul, Korea, 2022.

[10] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646.

[11] Distributive, Distributed computing made economical, secure, and intuitive, 2023, URL https://distributive.network/.

[12] S. Dey, A. Mukherjee, H.S. Paul, A. Pal, Challenges of using edge devices in IoT computation grids, in: International Conference on Parallel and Distributed Systems, IEEE, 2013, pp. 564–569.

[13] K. Thonglek, K. Ichikawa, K. Takahashi, H. Iida, C. Nakasan, Improving resource utilization in data centers using an LSTM-based prediction model, in: IEEE International Conference on Cluster Computing, CLUSTER, IEEE, 2019, pp. 1–8.

[14] R. Kain, S. Sorour, Worker resource characterization under dynamic usage in multi-access edge computing, in: International Wireless Communications and Mobile Computing, IWCMC, 2022, pp. 1070–1075, http://dx.doi.org/10.1109/IWCMC55113.2022.9824299.

[15] Y. Yao, Z. Xiao, B. Wang, B. Viswanath, H. Zheng, B.Y. Zhao, Complexity vs. Performance: Empirical analysis of machine learning as a service, in: Proceedings of the 2017 Internet Measurement Conference, IMC '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 384–397, http://dx.doi.org/10.1145/3131365.3131372.

[16] J. Violos, T. Pagoulatou, S. Tsanakas, K. Tserpes, T. Varvarigou, Predicting resource usage in edge computing infrastructures with CNN and a hybrid Bayesian particle swarm hyper-parameter optimization model, in: Intelligent Computing, Springer, 2021, pp. 562–580.

[17] T. Theodoropoulos, A.-C. Maroudis, J. Violos, K. Tserpes, An encoder-decoder deep learning approach for multistep service traffic prediction, in: IEEE Seventh International Conference on Big Data Computing Service and Applications, BigDataService, IEEE, 2021, pp. 33–40.

[18] X. Fu, C. Zhou, Predicted affinity based virtual machine placement in cloud computing environments, IEEE Trans. Cloud Comput. 8 (1) (2020) 246–255, http://dx.doi.org/10.1109/TCC.2017.2737624.

[19] N.C. Sekma, N. Dridi, A. Elleuch, Cross-correlation analyses toward a prediction system of CPU availability in volunteer computing system, in: International Conference on Industrial Engineering and Systems Management, IESM, IEEE, 2015, pp. 184–192.

[20] Y. Zhang, Y. Liu, Y. Li, X. Li, CACTA: Collaborative autoregressive-based computation task assignment in edge computing, in: IEEE International Conference on Edge Computing, EDGE, 2018, pp. 1–8, http://dx.doi.org/10.1109/EDGE.2018.00009.

[21] A.R.S. Parmezan, V.M.A. Souza, G.E.A.P.A. Batista, Time series prediction via similarity search: Exploring invariances, distance measures and ensemble functions, IEEE Access 10 (2022) 78022–78043, http://dx.doi.org/10.1109/ACCESS.2022.3192849.

[22] M.J. Johnson, A.S. Willsky, Bayesian nonparametric hidden semi-Markov models, J. Mach. Learn. Res. 14 (1) (2013) 673–701.

[23] X. Fu, C. Zhou, Predicted affinity based virtual machine placement in cloud computing environments, IEEE Trans. Cloud Comput. (2017) http://dx.doi.org/10.1109/TCC.2017.2737624.

[24] A.R.S. Parmezan, V.M. Souza, G.E. Batista, Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model, Inform. Sci. 484 (2019) 302–337, http://dx.doi.org/10.1016/j.ins.2019.01.076, URL https://www.sciencedirect.com/science/article/pii/S0020025519300945.

[25] V. Chourasia, S. Pandey, S. Kumar, Optimizing the performance of vehicular delay tolerant networks using multi-objective PSO and artificial intelligence, Comput. Commun. 177 (2021) 10–23, http://dx.doi.org/10.1016/j.comcom.2021.06.006, URL https://www.sciencedirect.com/science/article/pii/S0140366421002309.

[26] N. Sorkunlu, V. Chandola, A. Patra, Tracking system behavior from resource usage data, in: IEEE International Conference on Cluster Computing, CLUSTER, IEEE, 2017, pp. 410–418.

[27] T.M. Mengistu, D. Che, A. Alahmadi, S. Lu, Semi-Markov process based reliability and availability prediction for volunteer cloud systems, in: IEEE 11th International Conference on Cloud Computing, CLOUD, IEEE, 2018, pp. 359–366.

[28] K. Mason, M. Duggan, E. Barrett, J. Duggan, E. Howley, Predicting host CPU utilization in the cloud using evolutionary neural networks, Future Gener. Comput. Syst. 86 (2018) 162–173.

[29] D. Masouros, S. Xydis, D. Soudris, Rusty: Runtime interference-aware predictive monitoring for modern multi-tenant systems, IEEE Trans. Parallel Distrib. Syst. 32 (1) (2020) 184–198.

[30] D. Yang, J. Cao, J. Fu, J. Wang, J. Guo, A pattern fusion model for multi-step-ahead CPU load prediction, J. Syst. Softw. 86 (5) (2013) 1257–1266.

[31] J. Han, H. Cheng, D. Xin, X. Yan, Frequent pattern mining: current status and future directions, Data Min. Knowl. Discov. 15 (1) (2007) 55–86.

[32] A. Banerjee, H.S. Paul, A. Mukherjee, S. Dey, P. Datta, A framework for speculative scheduling and device selection for task execution on a mobile cloud, in: International Workshop on Adaptive Resource Management and Scheduling for Cloud Computing, Springer, 2014, pp. 36–51.

[33] P. Vrignat, M. Avila, F. Duculty, F. Kratz, Failure event prediction using hidden Markov model approaches, IEEE Trans. Reliab. 64 (3) (2015) 1038–1048.

[34] P.K.D. Pramanik, N. Sinhababu, K.-S. Kwak, P. Choudhury, Deep learning based resource availability prediction for local mobile crowd computing, IEEE Access 9 (2021) 116647–116671.

[35] M.H. Dunham, Y. Meng, J. Huang, Extensible Markov model, in: Fourth IEEE International Conference on Data Mining, ICDM'04, IEEE, 2004, pp. 371–374.

[36] M. Dewar, C. Wiggins, F. Wood, Inference in hidden Markov models with explicit state duration distributions, IEEE Signal Process. Lett. 19 (4) (2012) 235–238.

[37] P. Bellavista, A. Corradi, C. Giannelli, Evaluating filtering strategies for decentralized handover prediction in the wireless internet, in: 11th IEEE Symposium on Computers and Communications, ISCC'06, 2006, pp. 167–174, http://dx.doi.org/10.1109/ISCC.2006.70.

[38] G. Re Calegari, E. Carlino, D. Peroni, I. Celino, Filtering and windowing mobile traffic time series for territorial land use classification, Comput. Commun. 95 (2016) 15–28, http://dx.doi.org/10.1016/j.comcom.2016.04.016, Mobile Traffic Analytics. URL https://www.sciencedirect.com/science/article/pii/S0140366416301529.

[39] O. Serheiev-Horchynskyi, Analysis of frequency characteristics of simple moving average digital filtering system, in: IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology, PIC S&T, IEEE, 2019, pp. 97–100.

[40] D.S. Grebenkov, J. Serror, Following a trend with an exponential moving average: Analytical results for a Gaussian model, Physica A 394 (2014) 288–303, http://dx.doi.org/10.1016/j.physa.2013.10.007, URL https://www.sciencedirect.com/science/article/pii/S0378437113009722.

[41] G. Sforna, Psutil: A cross-platform process and system utilities module for python, 2009, GitHub repository. URL https://github.com/giampaolo/psutil.

[42] R. Ratcliff, Connectionist models of recognition memory: constraints imposed by learning and forgetting functions, Psychol. Rev. 97 (2) (1990) 285.