



CAPRF++: a context-aware B+ tree scheduler with weighted multi-objective tradeoff guidance and predictive prewarming for generalized task scheduling in cloud systems

C. Felsy¹ · R. Isaac Sajan¹

Received: 10 December 2025 / Accepted: 9 June 2026

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2026

Abstract

Cloud-edge environments require efficient task scheduling and resource management as they serve as key factors supporting the diverse, dynamic, and latency sensitive workloads of the modern smart infrastructures. Nevertheless, the non-uniformity of work, inconsistent work loads, and cold-start delays are major challenges to scalable and sustainable scheduling of tasks. In this paper, the authors introduce a new Context-Aware Predictive Reinforcement Framework (CAPRF++) of generalized task scheduling under unobservable workload scenarios, which is developed to be able to manage multi-modal workloads in distributed cloud-edge systems. In order to be compatible with heterogeneous forms of workload, a Context-Aware Task Feature Construction pipeline is suggested to encode metadata including task type, compute demand, deadline, and sustainability constraints into structured forms. Resource indexing strategy based on B+ tree can be used to achieve rapid and scalable task-device matching using real-time performance and energy indicators. It presents a new Weighted Multi-Objective PPO scheduling policy using a Weighted Tradeoff Controller that limits the policy choice to feasible SLA-energy-latency tradeoff space. Moreover, a FIFO replay buffer-based Adaptive Trajectory-Buffered PPO algorithm, in combination with a multi-objective reward model can be used to perform robust policy learning in changing workloads, balancing SLA compliance, execution time, energy consumption, and carbon footprint. In order to reduce cold-start overheads, a Predictive Prewarming module will be implemented which will pre-initialize runtime containers according to the workload expectations. The entire assessment is done with a massive simulated cloudedge condition that attempts to simulate actual workload dynamics and system heterogeneity. Significant simulation-based analyses show that CAPRF++ is an efficient system in terms of task completion latency, SLA satisfaction, energy consumption and generalization to unseen workloads in dynamic environments.

Extended author information available on the last page of the article

Keywords Generalized task scheduling · Context-aware computing · Multi-objective learning · Sustainable allocation · Cold-start mitigation

1 Introduction

The requirements of the current applications have made task scheduling in Cloud computing more complex as the modern applications are now dealing with heterogeneous multi-mode (video, text, sensor streams) data, and they demand effective allocation of resources across the varied computing environments (Jayasena et al. 2017; Alsadie 2021). The initial investigations in this field largely relied on heuristic-based scheduling algorithms such as the First-Come-First-Serve (FCFS), Shortest Job First (SJF), and Round-Robin as methods to optimize simple measures like the average time to complete the task and resource consumption (Pachipala et al. 2024; Seth and Singh 2019; Devi and Uthariaraj 2016). These methods worked well when the workloads were homogeneous but failed when the work was complex with interdependencies, frequently as directed acyclic graphs (DAGs) (Yang et al. 2020). Priority-based and load-balancing techniques were also investigated as background research to allocate tasks within cloud resources, but found it difficult to be scaled and adaptable as cloud systems expanded to serve dynamic, data-intensive applications.

Modern cloud task scheduling problems are complex, with the necessity to deal with heterogeneous multi-modal task structures, deal with generalization to unseen and low-data workload conditions, and trade-off latency, energy use, and sustainability (Gong et al. 2024). Graph-based or DAG-aware scheduling is needed to optimize the execution order of multi-modal tasks with interdependent sub-tasks and traditional scheduling methods such as Min-Min or Max-Min do not have the flexibility to support dynamic task graphs or critical paths (Raeisi-Varzaneh et al. 2024). Extrapolation to unseen or low-frequency tasks without retraining is considered a serious challenge because the traditional deep reinforcement learning (DRL) methods tend to fail to adapt based on their dependence on large training data and task-specific policies (Xu et al. 2024). Generalization in this work means that the system can work well at the previously unseen types of tasks and task distribution that are not practiced during training that are graded by a strict traintest separation protocol. Also, latency optimization can typically make things consume more energy, and energy-efficient methods such as Dynamic Voltage and Frequency Scaling (DVFS) can impair performance (Choi et al. 2005). Sustainability issues, like reduced carbon footprint, are becoming more urgent yet they have no standardized metrics or real-time implementation frameworks, and thus they are not easily implemented in the existed schedulers.

To deal with these issues, recent studies have considered various methods using a mixture of graph-based, machine learning, and hybrid methods (Hameed and Schwung 2023; Song et al. 2023). Critical path method (CPM) algorithm Based on topological sorting and DAG-aware scheduling algorithms, scheduling algorithms have enhanced the task interdependency by prioritizing sub-tasks that have the most significant effect on the overall time spent on the task (Koslovski et al. 2024). In scenarios where information about a task is unobservable or limited, meta-learning

algorithms like Model-Agnostic Meta-Learning (MAML) have been applied to allow quick learning of new tasks with limited data, and in many cases performs better than conventional methods of DRL in the generalization task (Sahoo and Chakraborty 2022). Energy-aware scheduling has been used to use DVFS and workload consolidation as a method to decrease power consumption, and green computing initiatives have proposed carbon-aware scheduling, which relocates tasks to a data center with a renewable energy source. Multi-objective optimization Hybrid algorithms, like combining genetic algorithms or particle swarm optimization with neural networks, have been shown to be promising in terms of latency, energy and sustainability (Kumar 2023). Such methods usually employ predictive models to determine the demands in resources and the time to carry out tasks to improve decision-making in dynamic environments.

Nevertheless, despite all these developments, there are serious limitations to the existing methods, which interfere with their practical implementation. Graph-based schedulers, though useful in the case of static DAGs, are difficult to adapt to dynamic task structure, and are computationally expensive, which can be prohibitive in large-scale clouds (Zhang et al. 2021). Adaptive methods such as meta-learning and DRL-based methods are expensive to train and consume a lot of computational resources hence not scalable (Liu et al. 2023). Performance is frequently traded off in energy and sustainability-oriented techniques, with DVFS causing more latency and carbon-sensitive scheduling potentially causing latency due to the geographic redistribution of loads. In addition, hybrid methods although flexible are slow in convergence in a high dimensional search space and are, therefore, not as effective in time sensitive tasks (Sreenivasulu and Paramasivam 2021). To deal with the acute problems of multi-modal complexity, generalization to unseen tasks, sustainability trade-offs, and real-time scalability in contemporary cloud systems, this paper introduces CAPRF++: a Context-Aware B+ Tree Scheduler with weighted multi-objective tradeoff guidance, and prewarming predictability of generalized task scheduling in cloud systems. CAPRF++ proposes a combined scheduling model, which combines joint task-resource embedding using a common latent representation, rapid resource indexing, multi-objective reinforcement learning, and predictive execution planning to provide efficient, adaptive and carbon-aware task-to-resource mapping in changing and unknown workload conditions. The reinforcement learning objective jointly optimises the task and resource encoders to allow them to align in a shared latent space in which meaningful similarity can be computed. It should be mentioned that the suggested CAPRF++ framework is tested on the highly simulative environment, which replicates large-scale cloud-edge systems based on heterogeneous resource allocation and variable workload processes. In real-world deployments, such systems are difficult to implement and benchmark in a fully operational environment because of the complexity and scale of such a deployment. Accordingly, in this paper, attention is paid to simulation-based validation which makes it possible to conduct controlled, reproducible and scalable performance analysis in a variety of conditions. The simulation environment is to be realistic in the sense of workload characteristics by benchmark traces and standardized system parameter. Learning Generalization to unseen tasks in CAPRF++ occurs via learning task representations that encode the existence of structural properties of tasks (e.g. compute intensity, deadlines, and

constraints) but not task-specific identities. This allows the policy to generalize to unobservable types of tasks the feature distribution of which fall in or close to the training manifold. In this study, generalization can be defined as how well the suggested framework is able to do on task types and load distributions not seen before with a rigorous traintest separation protocol.

1.1 Contributions

- **Context-Aware Task Feature Construction for Multi-Modal Workloads:** Proposing a task feature extraction pipeline that automatically encodes heterogeneous task metadata such as type, deadline, compute intensity, and sustainability goals into structured feature vectors. This allows the system to effectively represent and distinguish between diverse media workloads without relying on manually crafted rules.
- **B+-Tree-Based Resource Indexing for Fast Task-Device Mapping:** Introducing a scalable resource indexing structure using B+ Tree to enable fast retrieval of suitable cloud and edge devices based on real-time compute performance, availability, and energy efficiency. This minimizes search overhead and ensures responsive scheduling even in large-scale environments.
- **Weighted Multi-Objective PPO:** The PPO (Proximal Policy Optimization) is extended with a Weighted Tradeoff Controller (WTC) that dynamically computes the feasible action space based on latency, energy, and carbon trade-offs for each task-resource pair. This controller guides the PPO agent toward decisions that respect SLA constraints while promoting sustainable execution.
- **Adaptive Trajectory-Buffered PPO with Multi-Objective Reward Modeling:** Enhancing the PPO scheduling agent with a FIFO trajectory buffer and a custom multi-objective reward function that accounts for latency, SLA success, energy use, cold-start overhead, and carbon footprint. This enables robust policy learning and improved generalization, particularly for unseen or previously unobserved tasks.
- **Predictive Prewarming for Cold-Start Latency Reduction:** Implementing a prewarming mechanism that anticipates task arrivals and proactively initializes runtime containers on selected devices. This minimizes execution delays and enhances scheduling responsiveness in dynamic cloud-edge environments.

The remainder of the paper is structured in the following way: Sect. 2 is a review of related work; Sect. 3 describes the CAPRF++ framework; Sect. 4 outlines the experimental setup; Sect. 5 describes the key findings; and Sect. 6 gives final conclusions and future directions.

2 Literature review

Scheduling of tasks in heterogeneous edge-cloud systems is necessary due to the increased workload and sustainability requirements. In the current work, two major focus areas are intelligent scheduling of edge-cloud systems and optimization-based frameworks of adaptive sustainable resource management.

2.1 Intelligent task scheduling in edge-cloud environments

Some recent studies have concentrated on improving task scheduling through the utilization of deep learning, reinforcement learning, and system-aware models to enhance adaptability and responsiveness for edge-cloud architectures.

Ma et al. (2025) designed an end-to-end resource trading and task scheduling framework specifically for edge-cloud systems of arbitrary topologies. The novelty of their work is in the introduction of a third-party incentivization platform supporting cooperative sharing of resources between cloud and edge nodes through dynamic pricing mechanisms. To maintain optimal scheduling over such heterogeneous infrastructures, the authors introduced the MPV algorithm for efficient handling of homogeneous tasks. For diverse workloads, they also developed GSI and Distributed Alternating Update algorithms, both of which attempt distributed scalability and global convergence. Although these algorithms show robust theoretical performance, operational deployment is hindered by the inherent latency induced by iterative convergence processes and the higher communication overhead, particularly in multi-hop edge networks.

Expanding the need to reduce the response time of edge-cloud scheduling, Zou et al. (2025) presented ReflexPilot, an online scheduler with the primary goal of reducing cold-start delays. ReflexPilot introduces a new executor reuse mechanism to decrease the task initialization time by reusing pre-initialized compute containers for the same workload types. The core component of ReflexPilot is OTSA-PPO, a PPO-based scheduling agent that optimizes inter-dependent tasks in real time. In addition, the Executor Management Algorithm (EMA) adaptively monitors and assigns reusable executors to tasks dynamically. Following the quest for more intelligent and context-aware task scheduling, Wang et al. (2025) proposed TF-DDRL, a Transformer-based Deep Distributed Reinforcement Learning solution. For highly heterogeneous IoT applications in edge-cloud systems, TF-DDRL employs an Actor-Critic reinforcement learning framework integrated with Prioritized Experience Replay (PER) to store important experiences to aid in faster learning. Nonetheless, although TF-DDRL pushes the state of the art in foresight and adaptability, it also comes at considerable computational costs owing to the Transformer's attention mechanism as well as the extra synchronization needs of distributed training especially challenging in the context of limited compute resources or real-time operating environments.

In addressing task distribution in extensive spatial deployments like smart cities, Chen et al. (2025) proposed a new geometrization-based scheduling approach. Handling task distribution as a spatial optimization problem, they utilized multiplicative weighted Voronoi diagrams to divide the computational landscape into logical service areas. This spatial decomposition was combined with a Tetris-inspired

offloading scheme to tightly fit tasks according to local capacity limitations, effectively minimizing the computational complexity of assignment decisions. While its original design is innovative, the geometrization-based approach may underperform when task densities are non-uniform or when workload dynamics emerge at a rate quicker than the learning module can learn.

Pivoting around instantaneous resource selection in multimedia-intensive environments, Mangalampalli et al. (2024) introduced DRLBTSA, a Deep Q-Learning-based Task Scheduler optimized for heterogeneous and high-throughput workloads in cloud environments. The scheduler dynamically selects optimal resource-task mappings in response to real-time system statistics and task profiles. DRLBTSA is, however, highly dependent on pre-trained models and might suffer from adaptation delay when confronted with novel workloads or infrastructure settings. Following in the line of intelligent schedulers, Ahmeda and Kavitha (2025) presented ILbCTS, an Intelligent Learning-based Cloud Task Scheduler based on deep reinforcement learning. ILbCTS formulates the environment through real-time observations and optimizes the scheduling through skillfully crafted reward signals that capture SLA compliance, Quality of Service, and energy usage. Emulating an effort to overcome some of these drawbacks, Salari-Hamzehkhani et al. (2025) have proposed EPERDQN. It enhances the efficiency of learning by updating priorities on priority transitions and improving the modeling of dynamics on the environment to achieve improved compliance to SLA and task throughput. Although these benefits exist, EPERDQN introduces greater model complexity and relies heavily on the issues of non-scalable and transferability of reward and penalty functions challenges that could constrain the application to domains with sparse reward signals or non-stationary task patterns.

Zhang and Yang (2025) addressed the issue of dynamic scheduling in SEC by putting forward the MTGPER framework in more domain-specific scheduling contexts. This model applies Multi-Tree Genetic Programming to solve optimization problems of task scheduling rules to meet domain constraints like limited energy budgets, orbital dynamics and communication windows. The paradigm uses elite recombination methods in order to maintain high quality scheduling patterns over generations, delay-sensitive systems. But the genetic programming method is computationally expensive and needs a lot of fine-tuning of the evolutionary parameters, which may be problematic in deploying in real time and scaling to high-rate satellite networks.

2.2 Optimization techniques for sustainable and adaptive scheduling

In addition to learning-based models, metaheuristic and hybrid optimization strategies have been investigated by a number of researchers to effectively allocate tasks, with the view of taking into account aspects of energy, cost, and system adaptability.

The Improved Jaya Synergistic Swarm Optimization Algorithm, a hybrid metaheuristic by Abualigah et al. (2024), is an intelligent combination of directed exploitation capability of the Jaya algorithm and collaborative search of Synergistic Swarm Optimization (SSO). To further increase the capabilities of the algorithm to avoid local optima, perturbation through Levy flight is added and this increases the exploratory diversity of the high-dimensional search spaces. This synergetic integration enables the algorithm to explore intricate objective landscapes with high efficiency

resulting in higher convergence rates and high diversity of solutions which is useful in large scale as well as dynamic cloud infrastructures. Nevertheless, regardless of its advantages in the global search and convergence, this approach has one thing in common with other swarm-based approaches: it has high computational overhead. The adaptability of the system in real-time is limited under the conditions of the fast variation of the workload because the algorithm is iterative and rather slow in the response to the environmental changes.

Pabitha et al. (2024) developed the CRSOA as a solution to scheduling issues in tasks when there is uncertainty in metrics and under the volatility of the resources. This is a dual strategy algorithm that fuses the exploratory effectiveness of the CSA that parallels the environmental responsiveness of the animal with the exploitative character of the Remora Search Optimization Algorithm (RSOA) that is based on symbiotic relationships. Nevertheless, it is not so simple to apply it to real-time dynamic scheduling. The algorithm is also heavily reliant on manual parameter optimization and does not have any form of reinforcement learning, which means that it can easily degrade its performance when the task patterns change too fast or too unpredictably when faced with edge-cloud orchestration scenarios.

The authors aimed to establish predictive scheduling schemes to provide resources in the future and thus, Liu et al. (2025) proposed ESDN, which is a hybrid approach that uses dilated CNNs to predict demand and optimize fine-grained scheduling. The model builds on CNN based prediction to predict future resource requirements at edge nodes and employs L-BFGS quasi-Newton optimization algorithm together with a surrogate model to quickly arrive at the best possible set of scheduling decisions. In this respect, its predictive accuracy dependency and its iterative optimization cycle can hamper responsiveness in a very dynamic setting, e.g., an autonomous vehicle network or a mobile edge cloud. Based on mission-critical and fault-tolerant tasks environment, Saberikia et al. (2025) suggested a Mixed Integer Linear Programming (MILP)-based task scheduling strategy that is intended to operate on edge-fog-cloud cooperative infrastructures. The MILP model is a strategic planning model which plans the primary and redundant (Backup) tasks on heterogeneous nodes, balancing energy efficiency, performance stability and system resilience. The complexity of the approach however grows exponentially with the size of the problem and variation of tasks. The MILP formulation is less scalable and computationally expensive in large-scale or highly dynamic deployments, which restricts its applicability to real-time use where it is commonly necessary to reschedule or respond quickly to environmental changes.

3 Proposed methodology

The proposed CAPRF++ system will be able to schedule effectively the heterogeneous and multi-modular computation workloads like multimedia processing loads on hybrid cloud-edge systems. Metadata is provided on every incoming task including its type, deadline, level of priority, compute intensity, and sustainability goals. This metadata is then processed on arrival, and then explicitly mapped into a form of structured feature representation, capturing the functional requirements of the task

as well as the operational constraints. Simultaneously, the system queries a resource index based on a B+ tree to retrieve a shortlist of viable execution platforms, e.g. VMs or containers which are spread out across edge and cloud infrastructure. Such candidate devices are ranked based on dynamic and context-sensitive metrics, such as live availability, performance capacity, energy-efficiency, and live carbon emission profiles that enable scalable resource filtering in fast and sustainability conscious modes without costly search. In order to manage the trade-offs that are quite critical between the latency, energy and carbon impact, the WTC assesses the shortlisted devices and calculates a variable boundary of action to be taken in each scheduling decision. This action boundary represents the viable resource-task combinations that optimize the constraints of the task in terms of environmental cost. CAPRF++ is built around the CTPPO agent which consumes the features of the task, shortlist of devices and the tradeoff constraints to make smart scheduling choices. The PPO agent is augmented with a FIFO Trajectory Buffer, which stores good quality recent task trajectories, which allows learning to be stable and sample efficient, and leads to better generalization to new types of tasks never seen. Another method used by the agent is the multi-objective reward method which incorporates real-time feedback of SLA compliance, latency, energy consumption, cold-start overheads, and carbon emissions. With the dynamism of the adjustment of the reward signal in response to the actual performance and trust-region adaptive, the scheduler is constantly updating its policy to sustainability-conscious and performance-optimal behavior. The selected resource is then sent the task after a decision has been made. In case there is already compatible runtime container running, the task starts instantly. Otherwise, a predictive prewarming system rotates the necessary container in advance and eliminates the cold-start delays on a very large scale. During execution, the system is active in monitoring the performance of tasks, resource measurements and environmental feedback which are returned into the B+-Tree index as well as the PPO training loop to keep contextual awareness and learning. CAPRF++ represents a scalable, flexible, and sustainability-oriented task scheduling model that is offered through this close interconnection of dynamic indexing, weighted multi-objective tradeoff modeling, and reinforcement learning with optimized PPO. It is a good coping strategy with multi-modal complexity, unseen task types and volatile workloads, and subordinates powerful performance and energy-efficiency goals. Figure 1 represents the entire architectural process of CAPRF++.

The architecture workflow of CAPRF++ is not only represented in Fig. 1, but also reflects the conceptual design of the architecture, wherein the representation of tasks, resources, and decision-making have been integrated together in a common semantic interaction space. In particular, the framework works in three closely intertwined phases, namely, (i) semantic encoding of heterogeneous tasks and resources into latent representations, (ii) candidate filtering via compatibility through structured indexing and multi-objective tradeoff constraints, and (iii) optimal allocation through reinforcement learning. This conceptual breakdown underscores the fact that CAPRF++ is more than a pipeline, it is an integrated representation-optimization model of task scheduling of generalised workloads in an invisible workload.

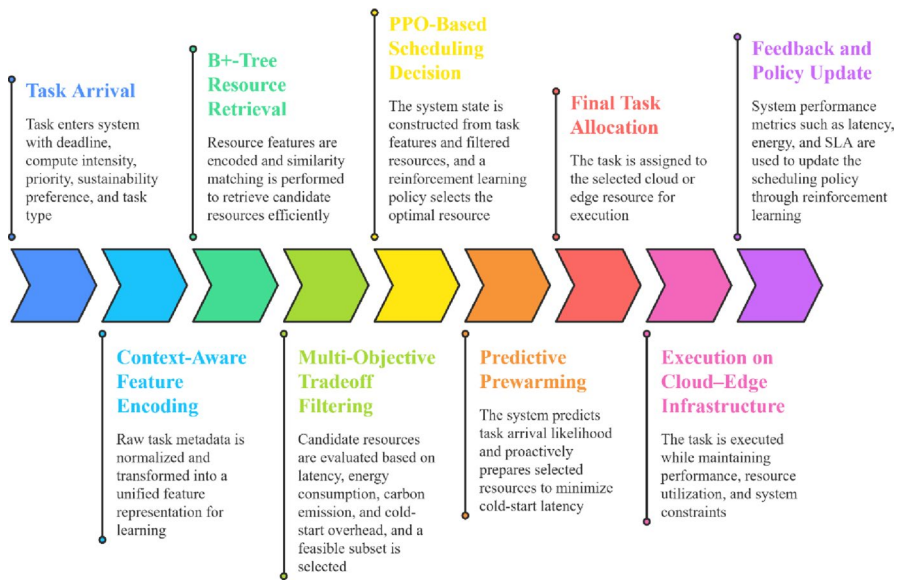


Fig. 1 Architecture of proposed CAPRF++ framework

3.1 Context-aware task feature construction for multi-modal workloads

Cloud-edge systems support workloads of heterogeneous applications, which differ in terms of computational intensity, deadline, and sustainability goals. This module converts the heterogeneous task metadata such as deadline, compute intensity, priority, sustainability preference, and task type into a single numerical value. Such organized encoding makes scalable and sound scheduling of a wide and invisible workload in cases of homogeneous and heterogeneous workload (Niu et al. 2023; Iftikhar et al. 2023). This design provides the framework with robustness, flexibility, and generalization to unseen tasks in fast paced cloud edge environments, by assigning every task into a semantically rich representation space.

Each task T_i is represented by key attributes, including execution deadline d_i , compute intensity c_i , priority score p_i , sustainability preference s_i , and task type t_i . These features collectively capture both performance requirements and operational constraints relevant for scheduling.

To harmonize this heterogeneous metadata into a unified numerical representation suitable for machine learning models, we first process the continuous-valued features namely, d_i , c_i , p_i and s_i through z-score normalization. This step of standardization has two vital functions: (i) it eradicates biases due to different scales of features, (ii) it guarantees that all continuous inputs are centered at zero with unit variance which makes the learning process more stable and efficient. The definition of the normalization mathematically is as follows:

$$\tilde{d}_i = \frac{d_i - \mu_d}{\sigma_d}, \tilde{c}_i = \frac{c_i - \mu_c}{\sigma_c}, \tilde{p}_i = \frac{p_i - \mu_p}{\sigma_p}, \tilde{s}_i = \frac{s_i - \mu_s}{\sigma_s} \quad (1)$$

where $\mu_d, \mu_c, \mu_p, \mu_s$ denote the empirical mean of each respective feature, and $\sigma_d, \sigma_c, \sigma_p, \sigma_s$ represent their corresponding standard deviations. These statistical moments are dynamically computed in a dynamic moving window of previous task arrivals, with the normalization process changing according to the varying workload distributions.

For the task type t_i , the discrete labels of the categorical variable, we use a learnable embedding function to transform the categorical variable into a dense continuous one.. This embedding process may be written down as follows:

$$\tilde{t}_i = \text{Embed}(t_i) \quad (2)$$

where $\text{Embed}(\cdot)$ is a parameterized mapping that is trained together with the rest of the model. Such embedding layer enables the system to learn latent representations of categories of tasks, learning the similarities and differences between the types of workloads that would be lost under one-hot or label encoding. After normalization of all of the continuous features and embedding of the categorical feature, such representations are concatenated and subjected to a non-linear transformation using a neural encoding function, indicated as ϕ_θ , parameterized by learnable weights θ . The complete operation results in the construction of a final task feature vector X_i , mathematically given by:

$$X_i = \phi_\theta(\tilde{d}_i, \tilde{c}_i, \tilde{p}_i, \tilde{s}_i, \tilde{t}_i) \quad (3)$$

To further clarify the neural encoding process, the encoding function f_θ is used as a fully connected sparse neural network that encodes the concatenated feature representation into a latent representation. In particular, the data to the encoder are a concatenation of the normalized continuous features and the embedded categorical feature, that is expressed as:

$$\mathbf{x}_t = [\hat{d}_t, \hat{c}_t, \hat{p}_t, \hat{s}_t, \mathbf{e}(T_t)] \quad (4)$$

where $\hat{d}_t, \hat{c}_t, \hat{p}_t, \hat{s}_t$ denote the normalized deadline, compute intensity, priority score, and sustainability preference respectively, and $\mathbf{e}(T_t) \in \mathbb{R}^k$ represents the learned embedding vector of task type.

The concatenated input vector $\mathbf{x}_t \in \mathbb{R}^m$ is then passed through a two-layer fully connected neural network defined as:

$$\mathbf{h}_t = \sigma(\mathbf{W}_1 \mathbf{x}_t + \mathbf{b}_1) \quad (5)$$

$$\mathbf{z}_t = \mathbf{W}_2 \mathbf{h}_t + \mathbf{b}_2 \quad (6)$$

where $W_1 \in \mathbb{R}^{h \times m}$, $W_2 \in \mathbb{R}^{d \times h}$ are learnable weight matrices, $b_1 \in \mathbb{R}^h$, $b_2 \in \mathbb{R}^d$ are bias vectors, $\sigma(\cdot)$ denotes a nonlinear activation function such as ReLU or Swish, h_t is the hidden representation, and $z_t \in \mathbb{R}^d$ is the final encoded task feature vector. A two-layer architecture is used that is lightweight to achieve a balance between representational capacity and computational efficiency to achieve low inference overhead that can be scheduled in real-time. Embedding dimensionality is chosen to be expressive enough and not too many parameters in dynamic cloud environments.

This neural change makes the model learn complex interactions between heterogeneous task features and acquire a latent representation that is semantically meaningful. The coded feature vector z_t thus enables better generalization and robust scheduling choices, especially of hitherto not seen types of tasks.

Operationally, the encoding process occurs in a sequence of transformation line. The input vector x_t is first normalized and embedded in the form of raw task metadata.. This vector is then projected into a higher-dimensional hidden space h_t , where nonlinear transformations enable the model to capture interactions between heterogeneous features. Finally, the representation is compressed into a fixed-dimensional latent vector z_t , which serves as a compact and informative descriptor of the task. Here, m denotes the dimensionality of the input feature vector, h represents the size of the hidden layer, and d corresponds to the dimension of the final task embedding used for downstream scheduling decisions. The PPO objective is used to optimize the encoder parameters indirectly by computing the policy loss and backgradients of the policy loss are propagated to the encoding network. This makes sure that the learned representation is correlated to long-term maximization of rewards.

The neural change has several functions: it can learn the complex interactions across features, it can learn hierarchical features abstractions, and it can learn subtle representations of tasks in a high-dimensional latent space. Such a rich and structured encoding of task encoding allows the CAPRF++ framework to be able to recognize and generalize across familiar and unseen patterns of workload, and eventually, improve the overall adaptability and effectiveness of the scheduling system.

3.2 B+-tree-based resource indexing for fast task-device mapping

The second most important step in the CAPRF++ framework is to identify effectively the appropriate computing resources that can perform these tasks in real time after transforming the incoming tasks into rich feature vectors that are context aware. Due to the size and dynamism of the cloud-edge environments (when the number of devices available may be multiple hundreds or even thousands) there is a strong need to design a resource retrieval mechanism, which is not only scale-efficient but must be able to respond to the changing state of the resources (e.g. availability and energy profiles).

To achieve this, CAPRF++ maintains a dynamic resource pool, denoted as $\{R_j\}$, where each computing resource R_j is characterized by four operational attributes. Specifically, these include: (i) the availability score a_j , indicating whether the resource is currently active or idle; (ii) the energy efficiency factor e_j , reflecting the energy cost or performance-per-watt of the device; (iii) the estimated execution latency l_j , which is a predictive estimate of how quickly the resource can execute a

standard workload; and (iv) the carbon emission factor γ_j , calculating the environmental cost of using that resource, especially in the sustainability-conscious scheduling scenarios. To enable effective similarity matching of the encoded tasks and the available resources, CAPRF++ employs a learnable neural transformation ψ_w , parameterized by a set of trainable weights ω . This neural encoder transforms each raw resource descriptor into a dense latent representation:

$$Y_j = (a_j, e_j, l_j, \gamma_j) \quad (7)$$

To ensure meaningful similarity computation between task and resource representations, both the task encoder f_θ and the resource encoder g_ϕ are designed to project their respective inputs into a shared latent space of identical dimensionality. Even though the encoders operate on various modalities of the input, they are collectively optimized in the reinforcement learning model using the downstream scheduling goal. In particular, the PPO-based learning mechanism offers implicit alignment, i.e. when updating both encoders, the quality of the task-resource assignment decisions and the ensuing reward signals are used. This correspondence is implicitly achieved by reward-based optimization with successful task-resource assignments reinforcing closeness within the common latent space, and suboptimal assignments being punished. This implicit learning can be viewed as a reward-based metric learning in which those task resource combinations that are rewarded more tend to be closer together in the latent space, whereas inefficient assignments are pushed far apart. As a result, Euclidean distance between the embeddings is a proxy of compatibility even in the case of a lack of an explicit contrastive loss.

Though there is no explicit contrastive/metric learning loss, the correspondence between task and resource embeddings is an implicit outcome of reward-driven learning in the reinforcement learning process. In particular, task-resource pairs that lead to an increase in rewards are strengthened during policy updates to promote the closeness of the shared latent space and suboptimal assignments are discouraged. This implicit alignment process allows the Euclidean distance between embeddings to be used as a useful proxy of compatibility between task requirements and resource capabilities.

Consequently, the encoded task and resource vectors z_t and z_r are subjected to a semantically congruent latent space of which Euclidean distance is used to indicate compatibility between the task need and resource strength. The distance metric can effectively represent suitability because of this shared representation in spite of the heterogeneity of the original input features.

This representation captures both the static and dynamic attributes of each resource in a compact vector format, enabling effective distance-based comparisons against the task embeddings X_i . The compatibility between a given task X_i and a resource Y_j is quantified using the Euclidean distance metric, expressed mathematically as:

$$S_{ij} = \|X_i - Y_i\| \quad (8)$$

This distance measure is a direct measure of the compatibility of a resource to the needs of a particular task with a lower score indicating greater compatibility.

CAPRF++ uses a B+-Tree-based indexing structure to store and represent these resource embeddings in order to avoid the inefficiency of exhaustive search of the entire resource pool. A balanced tree data structure, which is used to store key data and is optimized to retrieve them in order, is the B+-Tree, which has a complexity of search that is logarithmic, represented as $O(\log N)$, where N is the number of active resources in the system. This indexing approach provides two critical advantages: it allows for fast nearest-neighbor retrieval of suitable resources and supports real-time updates as the system state evolves.

Upon the arrival of a task, the system queries the B+-Tree using the task's feature vector X_i as a reference point. The B+-Tree efficiently returns a subset of candidate resources, denoted as R_{cand} that exhibit a distance score S_{ij} below a dynamically computed threshold T_i , mathematically represented as:

$$R_{cand} = \{R_j | S_{ij} \leq T_i\} \quad (9)$$

The threshold T_i is adaptively determined based on system-level objectives or workload conditions, allowing flexibility in the breadth of candidate selection. This method of resource indexing is essential to the fact that CAPRF++ is able to find high-quality resource candidates quickly with limited computational overhead, and consequently the framework can scale to and be responsive even in large-scale and resource-heterogeneous and highly dynamic cloud-edge infrastructures.

In cases where no candidate resources satisfy the threshold condition, i.e., $\mathcal{R}_i = \emptyset$, the system uses a fall back mechanism to adaptively reduce the threshold parameter and make sure that at least one feasible candidate is retrieved. Such dynamic adjustment averts the scheduling failure and ensures strength when the system has sparse or very constrained conditions.

Also, we use two different terms, task index and threshold parameter, to prevent the ambiguity of the notation. The threshold is denoted as τ_i , while T_i is reserved exclusively for task representation. This distinction ensures clarity and consistency throughout the formulation.

3.3 Weighted multi-objective PPO for multi-objective scheduling

After a narrowed down set of candidate resources is returned by the Context-Aware Task Feature Encoder and the B+-Tree Resource Indexing, it is sent to the next step CTPPO. This step involves the improvement of the standard PPO algorithm by adding a WTC which provides a systematic description of a feasible action space. The WTC compares every pair of tasks and resources to major goals, including reduction of latency, energy efficiency, and minimization of the carbon footprint, and it characterizes the tradeoff as a convex optimization problem. This allows the PPO agent to have a limited but optimal decision space so that action scheduling is made to conform to SLA assurances and foster adaptive and sustainable execution. Weighted multi-objective tradeoff guidance integration into the PPO assists the CAPRF++ scheduler to keep performance high when dealing with dynamic and multi-objective workloads. It should be noted that here the term trade-off is used to mean a weighted multi-objective formulation as opposed to a formally defined convex optimization

problem. The proposed approach is an integration of multiple goals such as latency, energy and carbon emission with adjustable weights, but it does not directly place convexity constraints.

For each task-resource pairing (i, j) , where i represents a specific task and j indexes a resource from the previously shortlisted candidate set R_{cand} , the system generates predictive estimates of several key performance indicators. Specifically, four metrics are computed: The predicted execution latency L_{ij} , indicating the expected time required for the resource R_j to complete the task T_i , The estimated energy consumption E_{ij} , representing the anticipated energy cost associated with task execution, The carbon emission estimate C_{ij} , quantifying the environmental impact in terms of greenhouse gas emissions resulting from the resource usage, and The cold-start overhead O_{ij} , which accounts for the additional initialization delay and resource utilization when idle devices are activated to process the task.

The predictability of the metrics is determined by the level of fidelity of the input task parameters, which include the compute intensity and deadline constraints. In real life, these parameters can be approximated or given by the user and this can lead to variations in the actual demand of the resources. The suggested framework is that task descriptors are a reasonable approximation of actual execution properties, as is typical of workload-based scheduling systems. To reduce the effects of possible mis-specification, the reinforcement learning element keeps on updating its policy according to the observed execution feedback allowing it to correct systematic over- or under-estimation over time to some degree.

However, it is a flaw in this formulation that it is a dependency. The use of automatic parameter estimation (e.g. profiling based inference or benchmark-driven calibration) is a good area of future research to enhance robustness and prediction accuracy.

Lightweight parametric models are used to predict task-resource performance metrics by estimating characteristics of execution in relation to task characteristics and resource characteristics. Specifically, latency L_{ij} is estimated as a function of compute demand and resource processing capacity, while energy consumption E_{ij} is modeled based on execution time and device-specific power profiles. Carbon emission C_{ij} is derived from the energy estimate combined with the carbon intensity factor of the resource. Formally, these can be approximated as:

$$L_{ij} \propto \frac{c_i}{\kappa_j}, E_{ij} = P_j \cdot L_{ij}, C_{ij} = \gamma_j \cdot E_{ij}$$

where c_i denotes task compute intensity, κ_j is the processing capability of resource j , P_j is the average power consumption, and γ_j is the carbon intensity factor. The simplified models offer an efficient and scalable estimate that can be used in real time scheduling. These models can give rough but consistent estimates to make comparative analysis. Although there might be small deviations of the real world measurements, the formulation will guarantee the stable relative ranking among candidate resources. Empirical evidence shows that such estimates are accurate enough to make a decision in the framework of scheduling. These formulations give first-order approximations, which are meant to be compared relative to each other, and not

modeled physically, which is adequate to achieve consistent policy optimization in reinforcement learning.

The formulation that is in use presupposes that the tasks are performed in the conditions of logical resource isolation, in which each task is allocated adequate computational resources without serious interference with co-located workloads. This simplifies the estimation of performance, and this is in line with container-based or reserved-resource execution environments. The issue with the practical cloud-edge systems, however, is that resource sharing and co-location can add contention effects that can cause performance degradation and affect the prediction accuracy. This assumption permits the use of controlled assessment of the scheduling choices, but the possibility of deterioration of performance following resource contention is recognized and is remaining as future tasks. Although no explicit contention modeling is involved, it is partially modeled by real time reward feedback, which implicitly punishes poor performance.

Although CAPRF++ is partly based on the dynamic conditions of the system using real-time feedback and policy adjustment, there is no explicit modeling of resource contention in the existing design. It is also of great significance to extend the framework to contention-aware scheduling and interference modeling, and this is a direction that should be pursued in the future.

To synthesize these diverse indicators into a unified decision-making metric, the WTC computes a multi-objective weighted multi-objective tradeoff score Ω_{ij} for each task-resource pair using a weighted summation approach:

$$\Omega_{ij} = \alpha L_{ij} + \beta E_{ij} + \delta C_{ij} + \rho O_{ij} \quad (10)$$

In this formulation, the coefficients $\alpha, \beta, \delta, \rho$ are system-tunable hyperparameters that allow operators to prioritize different scheduling objectives according to operational goals or service-level agreements (SLAs). For instance, in latency-critical applications, a higher value of α may be used, whereas in green computing scenarios, the weights δ or ρ could be increased to emphasize sustainability considerations.

After computing Ω_{ij} for all candidate resource-task pairings, the controller identifies the optimal minimum tradeoff score $\Omega_{\min}$, representing the most efficient tradeoff scenario among the candidates. Rather than restricting the scheduling agent to only this single optimal choice, CAPRF++ promotes a balance between exploitation and exploration by introducing a flexibility margin Δ . This margin enables the inclusion of near-optimal options within a bounded constrained feasible action space C_i , formally defined as:

$$C_i = \{R_j \in R_{cand} | \Omega_{ij} \leq \Omega_{\min} + \Delta\} \quad (11)$$

where Δ is a small relaxation factor, making the scheduling agent not excessively constrained and able to respond to varying system dynamics by taking into account resources that are slightly above the lowest tradeoff cost, and which can have other benefits to the system, such as load balancing or shorter queuing times. where Δ serves as a soft relaxation parameter, ensuring that the scheduling agent is not overly restricted and can adapt to fluctuating system dynamics by considering resources that

marginally exceed the minimal tradeoff cost but may offer other system advantages, such as load balancing or reduced queuing times. This constraint is more of a soft filtering mechanism than a strict optimization constraint and enables PPO to still explore whilst operating in a smaller decision space.

This convex trade-off filtering technique plays two critical roles in CAPRF++. One, it ensures that the scheduling agent is provided with a viable, low-cost set of actions, filtered in advance to achieve operational and sustainability goals, in essence reducing the complexity of policy learning. The system is able to do two, by dynamic adaptation of Δ , to achieve adaptive flexibility in choice of action, which promotes policy stability and enhanced generalization performance especially in dynamically changing and unfamiliar workload situations.. The layout allows CAPRF++ to trade off among competing objectives, and still have strong compliance with significant performance guarantees in complex, real-world cloud-edge systems.

3.4 Adaptive trajectory-buffered PPO with multi-objective reward modeling

To achieve the most effective task scheduling decisions capable of effectively balancing system objectives and reacting to the changeable workload dynamics, CAPRF++ employs an advanced system of reinforcement learning based on PPO and supplemented by the adaptive trajectory buffer and optimally tailored multi-objective reward function. With this architecture, CAPRF++ chooses near-optimal scheduling choices in the feasible candidate set of weighted multi-objective constraints of resource utilization, SLA compliance, and environmental impact based on system-specified priorities. The scheduling problem is modeled as a Markov Decision Process (MDP), with the state being the current system and task situation, the action being the task-to-resource allocation, and the reward being the performance in terms of multi-objectives. In the CAPRF++ framework, the current scheduling state for each incoming task T_i is defined by the combination of the encoded task feature vector X_i and the pre-filtered set of feasible resource options C_i , resulting in a composite state representation:

$$s_i = (X_i, C_i) \quad (12)$$

PPO is used to optimize the policy to maximize expected cumulative reward as time goes by. This condition aggregates all the important contextual data needed to make intelligent decisions in resource assignment. Here, the state s_i represents the current scheduling context for task i , comprising the encoded task feature vector and the corresponding feasible resource set obtained after weighted multi-objective trade-off filtering. This state captures the needs in the tasks as well as the limited action space the scheduler has. Formally, the state s_i is defined as $s_i = (\mathbf{z}_t, \mathcal{R}_i)$, where $\mathbf{z}_t$ represents the encoded task feature vector and $\mathcal{R}_i$ denotes the feasible set of candidate resources. The formulation will make sure that the state not only captures the characteristics of the tasks but also captures the limited set of choices that an agent has. This specification implies a stateful action space that is finite, and with such a space, the policy is executed within a dynamically limited action space, as opposed to a global action space. Formally, the scheduling problem is defined as a tuple $(\mathcal{S}$,

$\mathcal{A}, \mathcal{P}, R, \gamma)$, where the transition dynamics $\mathcal{P}$ are governed by system state evolution and task arrivals, and the objective is to learn a policy π that maximizes expected discounted return.

The PPO agent operates by learning an optimal policy π_θ that maps this state s_i to a scheduling action, i.e., selecting the most suitable resource R_j from the feasible set C_i . The action a_i corresponds to selecting a specific resource $r_j \in \mathcal{R}_i$ for executing task i , where the action space is dynamically defined by the feasible candidate set obtained after weighted tradeoff filtering. The PPO agent is trained to maximize a multi-objective reward function r_i , which accounts for five crucial factors. First, it incorporates an SLA adherence, via an indicator function $I(SLA_{met})$ that rewards the agent for successfully scheduling tasks within their deadline. Second, it includes a Execution latency L_{ij} , penalizing tasks with high completion delays. Third, the agent is guided to prefer Energy consumption E_{ij} , promoting energy-efficient task placements. Fourth, Carbon emissions C_{ij} , rewarding environmentally sustainable scheduling decisions. Finally, Cold-start overhead O_{ij} , penalizing excessive resource initialization costs. This is mathematically formulated as:

$$r_i = \eta_1 \cdot I(SLA_{met}) - \eta_2 L_{ij} - \eta_3 E_{ij} - \eta_4 C_{ij} - \eta_5 O_{ij} \quad (13)$$

where $\eta_1, \eta_2, \eta_3, \eta_4, \eta_5$ are hyperparameters that balance the relative importance of each scheduling objective.

To ensure stable learning, each reward component is normalized to a comparable scale using min–max normalization based on observed system statistics. This prevents any single objective from dominating the learning process and improves convergence stability of the PPO agent. To ensure stable and efficient learning, the PPO agent optimizes a clipped surrogate objective function, which constrains the policy updates within a trust region to prevent drastic policy shifts:

$$L_{PPO} = E[\min(\rho_i A_i, \text{clip}(\rho_i, 1 - \varepsilon, 1 + \varepsilon) A_i)] \quad (14)$$

where ρ_i denotes the probability ratio between the new and old policies, and A_i represents the advantage estimate computed via Generalized Advantage Estimation (GAE):

$$A_i = \sum_k \gamma^k (r_{i+k} - V_\theta(s_{i+k})) \quad (15)$$

This advantage term captures the relative value of choosing action α_i in state s_i promoting actions that yield long-term rewards.

To further enhance the robustness and adaptability of the CAPRF++ scheduling framework, a First-In-First-Out (FIFO) trajectory buffer is integrated into the reinforcement learning process. This buffer continuously stores the most recent scheduling experiences in the form of transition tuples (s_i, a_i, r_i, s_{i+1}) $B = \{(s_i, a_i, r_i, s_{i+1})\}_{i=t-N}^t$, where s_i denotes the system state, a_i is the scheduling action taken, r_i represents the corresponding multi-objective reward, and s_{i+1} is the subsequent state. The FIFO

buffer, denoted by B , maintains a fixed capacity of N transitions and is mathematically expressed as $B = \{(s_i, a_i, r_i, s_{i+1})\}_{i=t-N}^t$ where t is the current timestep and older transitions are discarded as new ones are added. This design ensures that the PPO agent always trains on a rolling window of the most recent interactions, allowing for quick adaptation to dynamic workload changes. The FIFO buffer thus plays a crucial role in reducing variance in learning updates and enhancing responsiveness to changing task patterns, making CAPRF++ particularly effective in heterogeneous and time-varying cloud–edge environments. The CAPRF++ framework employs an Actor–Learner reinforcement learning architecture designed to support sustainable and generalized task scheduling across diverse cloud–edge systems. The Actor plays the role of communicating with the environment by producing trajectories of task scheduling from the existing policy, whereas the Learner updates the policy iteratively using the acquired experience to provide superior performance at every step. Figure 2 shows the architectural design of CAPRF++ scheduling framework, which takes advantage of a distributed actor–learner reinforcement learning framework with prioritized experience replay and adaptive prewarming.

Figure 2 illustrates the exchange of actions between the Actor (edge-level scheduling agent), the Learner (centralized model-based PPO), and the cloud-edge system in the CAPRF++ framework. Every Actor interacts with dynamic IoT tasks and cloud-edge hosts through scheduling actions and receipt of observations and rewards from the environment. These interactions are captured as trajectories and transmitted to the centralized Learner, which updates the global policy using PPO. A Prioritized Experience Replay (PER) buffer stores sampled experiences and updates their priorities based on relevance, enabling efficient and targeted learning. Once the global PPO model is optimized, updated policy parameters are synchronized with all local Actors. Additionally, each Actor features a proactive prewarming trigger mechanism that anticipates incoming workloads and preemptively activates underutilized resources to mitigate cold-start delays. During each scheduling cycle, as described in Algorithm 1, the Actor initializes its task-resource scheduling process by synchronizing its local policy μ with the latest global policy π provided by the Learner. Each incoming task is encoded into a structured task feature vector, and a subset of candidate resources is efficiently retrieved using a B+ Tree structure. The Actor computes the feasible action space through weighted multi-objective tradeoff filtering, encapsulating multi-objective considerations such as latency, energy, carbon footprint, and

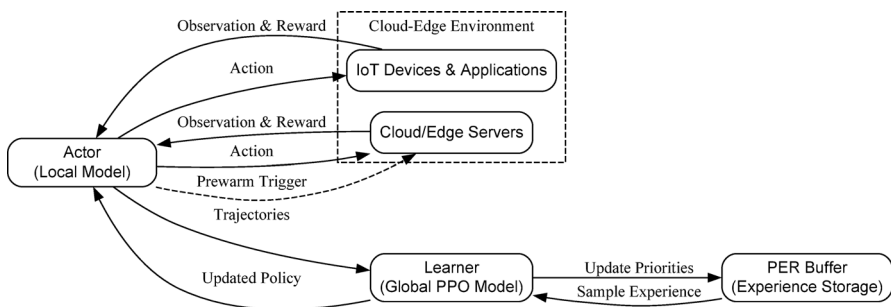


Fig. 2 Actor–learner PPO framework with cloud-edge scheduling and prewarming

cold-start penalties. Using its local policy, the Actor selects an action allocating a task to a resource and interacts with the environment to receive the corresponding reward, while transitioning to the next state. Each transition, consisting of $s, a, r, \mu(a|s), s'$, is stored in a trajectory buffer. After accumulating sufficient experiences, the trajectory is transmitted to the Learner for policy improvement.

In the CAPRF++ framework, the Actor interacts with the cloud-edge environment through a sequential decision-making process. At each step, the Actor observes the current state s_i , which includes task features and candidate resources. The action corresponds to selecting a specific resource from the feasible set for task execution. Following execution, the environment returns a reward based on SLA satisfaction, latency, energy consumption, carbon emission, and cold-start overhead, along with the updated system state reflecting resource availability and system dynamics. This interaction loop enables continuous learning and adaptation.

Require: PPO policy network π_θ , Resource Pool $\mathcal{R}$, Task set $\mathcal{T}$, Environment $\mathcal{E}$

Ensure: Experience buffer $\mathcal{D}$ with transitions (s, a, r, s')

```

1: Initialize experience buffer  $\mathcal{D} \leftarrow \emptyset$ 
2: for each episode do
3:   for each task  $\mathcal{T}_i \in \mathcal{T}$  do
4:     Extract task features  $(d_i, c_i, p_i, s_i, t_i)$ 
5:     Encode  $X_i = \phi_\theta(d_i, c_i, p_i, s_i, t_i)$ 
6:     Retrieve  $\mathcal{R}_{cand}$  via B+-Tree
7:     Compute feasible set  $\mathcal{C}_i$  via convex filtering
8:     State  $s_i = (X_i, \mathcal{C}_i)$ 
9:     Sample action  $a_i \sim \pi_\theta(s_i)$ 
10:    Observe reward  $r_i$ , next state  $s'_i$ 
11:    Store  $(s_i, a_i, r_i, s'_i)$  in  $\mathcal{D}$ 
12:   end for
13: end for
14: return  $\mathcal{D}$ 

```

Algorithm 1 Actor: Experience Generation for CAPRF++ Scheduler

As outlined in Algorithm 2, the Learner periodically updates the global policy π using PPO with a multi-objective reward function, incorporating Generalized Advantage Estimation (GAE) to stabilize the advantage calculations. In dynamic cloud-edge settings to reduce sampling variance and bias, PER is used, in which transitions are sampled according to their significance determined by the multi-objective reward and temporal difference (TD) errors. Clipped PPO objectives are used to compute the policy loss to prevent giant policy changes, and the value function is optimized by the squared error between the forecasted value and the desired return. In addition, CAPRF++ supports predictive prewarming logic in the Actor phase, which allows taking proactive measures to mitigate cold-start overheads. The Actor-Learner structure combined is a way of providing the decision of scheduling tasks to be carried out at any given time is constantly optimized to meet the changing workloads.

Require: Experience buffer $\mathcal{D}$, initial policy π_θ , value network V_θ , learning rate α , clipping parameter ϵ

Ensure: Updated policy π_θ

- 1: **for** each update iteration **do**
- 2: Compute advantages A_i via GAE
- 3: **for** each mini-batch from $\mathcal{D}$ **do**
- 4: Compute surrogate objective:

$$L^{\text{PPO}} = E [\min (\rho_i A_i, \text{clip}(\rho_i, 1 - \epsilon, 1 + \epsilon) A_i)]$$

- 5: $\rho_i = \frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{\text{old}}}(a_i|s_i)}$
- 6: Update $\theta \leftarrow \theta + \alpha \nabla_\theta L^{\text{PPO}}$
- 7: Update value function by minimizing:

$$L_V = E [(V_\theta(s_i) - R_i)^2]$$

8: **end for**

9: **end for**

10: **return** Updated policy π_θ

Algorithm 2 Learner: PPO Policy Update with Convex Filtering

CAPRF++ has a training process that is based on an online learning paradigm combined with systems operation. First, the scheduler acts with a partially or randomly initialized policy, which provides the scheduler with the opportunity to search the action space and gather interaction data in terms of state-action-reward transitions. The trajectory buffer stores these transitions which are periodically used to update the policy through PPO.

In the initial phases, the system can perform poorly in scheduling decisions because of lack of experience but as more trajectories are gathered, the policy progressively advances with particular updates. With this continuous training system, the scheduler can adjust itself to changing patterns of workload without the need to spend time offline training the scheduler. The framework may also be warm-started with the help of pre-collected traces, which are optional, but help to speed up convergence in a real deployment.

3.5 Predictive prewarming for cold-start latency reduction

Besides intelligent task scheduling, CAPRF++ is another system that addresses one of the major bottlenecks of cloud-edge computing systems cold-start latency by integrating a Predictive Prewarming Module. Latency of cold-start When cold, or dormant, computational resources are required to be re-initiated to handle new computations, the latency is referred to as cold-start latency. CAPRF++ predictive prewarming method predicts arrival of tasks in the future and pre-initiates the resources beforehand thereby countering this cost of latency. For each incoming task T_i , the prewarming module utilizes the task's encoded feature vector X_i to estimate the predicted probability of task arrival λ_i through a lightweight neural predictor:

$$\lambda_i = \sigma(W_\lambda X_i + b_\lambda) \quad (16)$$

where W_λ and b_λ are trainable weight and bias parameters, and $\sigma(\cdot)$ denotes a sigmoid activation function ensuring the output is bounded within a probability range.

Here, $\lambda_i \in [0,1]$ represents the probability of task arrival within a short future time window, rather than a continuous arrival rate.

Simultaneously, CAPRF++ predicts the cold-start overhead for each candidate resource R_j using the expected initialization cost:

$$Cold(R_j) = E[O_{ij}] \quad (17)$$

The product $\lambda_i \cdot Cold(R_j)$ defines a prewarming priority score that jointly captures the likelihood of task arrival and the expected cold-start overhead of a resource. This designation guarantees that prewarming is caused solely when both the likelihood of task arrival is high and the cost of the prewarming is substantial, eliminating the prewarming of unnecessary or unimportant choices. CAPRF++ selectively activates prewarming on a small fraction of top-ranked candidate resources instead of all the available devices to avoid unnecessary redundant resource activation. In particular, prewarming is only applied to the resources inside the feasible candidate set and that meet the threshold condition. This makes the process of container initiation managed and resource efficient, preventing over-parallelization allocation on multiple nodes. By combining the likelihood of task arrival λ_i with the cold-start cost $Cold(R_j)$, the system determines whether prewarming is warranted. Prewarming is triggered if this combined score exceeds a threshold T_{warm} , defined as:

$$Prewarm(R_j) = \begin{cases} 1 & \lambda_i \cdot Cold(R_j) \geq T_{warm} \\ 0 & otherwise \end{cases} \quad (18)$$

The threshold T_{warm} acts as a decision boundary that activates prewarming when the computed priority score exceeds a predefined level. This mechanism avoids over-preparation and over-utilization of resources by avoiding either under-preparation or over-initiating a container. This threshold decision can be viewed as a probabilistic tradeoff between costs and benefits based on the tradeoff of expected reduction of latency against unnecessary activation of resources.

If activated, the PPM proactively initializes the corresponding computational environment or container on R_j , preparing it in advance prior to the performance of the actual task, and thus cold-start delays are reduced to a substantial degree. Such predictive prewarming scheme is not only effective in improving the response time of the system but also in improving system usability by minimizing tail latency especially when the workload is high-priority and latency-sensitive like in cloud-edge systems.

The last stage of CAPRF++ is to choose the most appropriate resource to execute the task based on the cumulative data that is obtained and processed using the previous modules. After generating the feasible convex set of candidate resources C_i and refining the action space using the PPO agent's learned policy π_θ , the task T_i is assigned to the optimal resource R_j^* . The selection criterion targets minimizing the

expected multi-objective tradeoff score Ω_{ij} computed earlier. The final assignment is formally given by:

$$T_i \xrightarrow{\pi_\theta} R_j^* = \arg \min_{R_j \in C_i} E[\Omega_{ij}] \quad (19)$$

This model will make sure the chosen resource will reduce a total cost which is a mix of latency, energy, emissions, and cold-start penalties without violating SLA limits. This dynamically adaptive scheduling process provides CAPRF++ with a synergistic strike between high service performance, resource efficiency, and sustainability goals, and is therefore appropriate to be deployed in realistic heterogeneous cloud-edge computing systems that have dynamic workloads and hard service guarantees. CAPRF++ design decisions are influenced by the necessity to consider scalability, flexibility, and multi-objective optimization of dynamic cloud-edge settings. The indexing based on the B+-Tree is also adopted to provide the logarithmic time resources accessibility, and it is more scalable compared to linear search or brute-force matching methods. The multi-objective tradeoff filtering (weighted and optimized) allows to prune the infeasible resource candidates effectively and limits the decision space to a smaller set when compared to non-constrained optimization. PPO is chosen because its policy updates are stable and it can be applied to high-dimensional, continuous, and continuous problems and has benefits over value-based models like DQN in dynamic environments. Last but not the least, predictive pre-warming is added to deal with cold-start latency that cannot be used by the traditional schedulers. All these factors provide a balance act in terms of efficiency, strength, and real-time flexibility which is ineffective to be achieved using less complex or autonomous means.

The overall time complexity of CAPRF++ is primarily governed by three components: resource retrieval, tradeoff filtering, and policy evaluation. The B+-Tree-based resource indexing enables candidate retrieval in logarithmic time $O(\log N)$, where N is the number of available resources. The weighted multi-objective tradeoff filtering operates over a reduced candidate set of size K , resulting in $O(K)$ complexity. The PPO-based decision-making step involves forward inference through the policy network, which has complexity proportional to the network size and is effectively $O(1)$ for fixed architecture. Therefore, the overall per-task scheduling complexity is approximately $O(\log N + K)$, making the framework scalable for large-scale environments.

3.6 CAPRF++ scheduling algorithm

To systematically realize adaptive and sustainable scheduling decisions in heterogeneous cloud-edge environments, the CAPRF++ scheduling framework, as detailed in Algorithm 3, executes a modular multi-stage pipeline optimized for multi-modal and dynamic workloads.

Require: Task set $T = T_1, T_2, \dots, T_N$, Resource Pool $R = R_1, R_2, \dots, R_M$

Ensure: Optimized resource allocation $\{T_i \rightarrow R_j^*\}$ for all $T_i \in T$

1: **for** each task $T_i \in T$ **do**

2: **Step 1: Context-Aware Task Feature Encoding**

3: Extract raw metadata $(d_i, c_i, p_i, s_i, t_i)$ from T_i

4: Normalize continuous features via z-score normalization

5: Embed categorical task type t_i into dense vector

6: Compute encoded task vector $X_i = \phi_\theta(d_i, c_i, p_i, s_i, t_i)$

7: **Step 2: Fast Candidate Resource Retrieval**

8: **for** each resource $R_j \in R$ **do**

9: Encode (a_j, e_j, l_j, r_j) into $Y_j = \psi_\omega(a_j, e_j, l_j, r_j)$

10: **end for**

11: Retrieve candidate set R_{cand} via B+- Tree where $\|X_i - Y_j\| \leq \tau_i$

12: **Step 3: Convex Tradeoff Filtering**

13: **for** each $R_j \in R_{\text{cand}}$ **do**

14: Compute tradeoff score $\Omega_{ij} = \alpha L_{ij} + \beta E_{ij} + \delta C_{ij} + \rho O_{ij}$

15: **end for**

16: Identify feasible set $C_i = \{R_j \mid \Omega_{ij} \leq \Omega_{\min} + \Delta\}$

17: **Step 4: PPO Scheduling Decision**

18: Define scheduling state $s_i = (X_i, C_i)$

19: Select optimal resource $R_j^* = \pi_\theta(s_i)$

20: **Step 5: Predictive Prewarming Decision**

21: Predict arrival likelihood $\lambda_i = \sigma(W_\lambda X_i + b_\lambda)$

22: **if** $\lambda_i \cdot \text{Cold}(R_j^*) \geq \tau_{\text{warm}}$ **then**

23: Trigger prewarming $\text{Prewarm}(R_j^*) = 1$

24: **end if**

25: **Step 6: Final Task Allocation**

26: Assign $T_i \rightarrow R_j^*$

27: **end for**

28: **return** Resource allocation mapping $\{T_i \rightarrow R_j^*\}$ for all tasks

Algorithm 3 CAPRF++ Scheduling Framework for N Tasks

The process initiates with context-aware task feature encoding, wherein raw metadata such as deadline, compute demand, priority, size, and task type are normalized and embedded into a high-dimensional latent representation X_i using ϕ_θ . In the second stage, fast candidate resource retrieval is performed by encoding resource

features via ψ_w , and matching task-resource proximity using a B+ Tree structure to efficiently retrieve the top candidates R_{cand} . The third stage employs weighted multi-objective tradeoff filtering, where a composite score Ω_{ij} is calculated for each candidate R_j using:

$$\Omega_{ij} = \alpha L_{ij} + \beta E_{ij} + \delta C_{ij} + p O_{ij} \quad (20)$$

This score captures a weighted tradeoff across latency (L_{ij}), energy consumption (E_{ij}), carbon emission (C_{ij}), and cold-start overhead (O_{ij}), with tunable weights α, β, δ, p . Resources satisfying the tradeoff constraint $\Omega_{ij} \leq \Omega_{\min} + \Delta$ form the feasible set C_i . In the fourth stage, the PPO-based scheduling decision is performed. The scheduling state $s_i = (X_i, C_i)$ is used by the policy network π_θ to select the optimal resource:

$$R_j^* = \pi_\theta(s_i) \quad (21)$$

To minimize the cold-start impact, the fifth stage incorporates a predictive prewarming mechanism. The likelihood of future task arrivals is predicted via:

$$\lambda_i = \sigma(W_\lambda X_i + b_\lambda) \quad (22)$$

If $\lambda_i \cdot \text{Cold}(R_j^*) \geq T_{warm}$ the scheduler triggers prewarming: $\Pr \text{ewarm}(R_j^*) = 1$, thereby activating underutilized resources ahead of anticipated load spikes. Finally, the selected optimal resource R_j^* is assigned to the incoming task T_i , completing the scheduling cycle. The end-to-end design enables real-time, adaptive decision-making across diverse workload distributions, with strong generalization to unseen workloads and effective optimization for SLA satisfaction, energy efficiency, low latency, and sustainability in dynamic cloud-edge systems. In practice, since $K \ll N$, the dominant cost is $O(\log N)$, ensuring efficient scalability even for large resource pools. This combination makes the effective decision complexity of global resource selection constrained candidate optimization more cost-effective, efficient in scale, and does not compromise the solution quality.

4 Experiments details and analysis

In this section, we present a comprehensive evaluation of the proposed CAPRF++ framework under diverse large-scale cloud-edge task scheduling scenarios. The experiments are designed to assess the adaptability, efficiency, and sustainability of CAPRF++ across heterogeneous workloads. We detail the simulation environment, hyperparameter settings, and system configurations, followed by a comprehensive performance analysis using key metrics such as Average Task Completion Latency, SLA Satisfaction Rate, Carbon Footprint Reduction Rate, Cold-Start Overhead Reduction, Generalization Performance on Unseen Workloads, and Makespan.

Comparative results against baseline methods are also provided to demonstrate the effectiveness and robustness of the proposed approach.

4.1 Simulation settings

In order to provide an end-to-end realistic appraisal of the CAPRF++ scheduling framework, we model a massive-scale heterogeneous cloud-edge system consisting of low-latency edge servers and high-capacity cloud servers. The workload of tasks comprises multi-modal streams with varying deadlines, computation intensities, and sustainability requirements. Task arrivals follow a stochastic process with dynamically varying load conditions to mimic real-world cloud-edge operational environments. The CAPRF++ framework is implemented in Python using PyTorch and trained using a distributed actor-learner PPO architecture. All experiments are conducted on a server equipped with an Intel® Xeon® Gold 6338 CPU @ 2.00 GHz, 128 GB RAM, and four NVIDIA A100 GPUs (80 GB each). Each simulation run includes 50,000 task arrivals with five independent randomized seeds to ensure statistical robustness. The detailed simulation hyperparameters and system configurations are provided in Tables 1 and 2. A custom simulation environment is adopted in this study to enable flexible modeling of multi-objective scheduling dynamics, predictive prewarming behavior, and real-time reinforcement learning interactions, which are not fully supported in existing simulators such as CloudSim or iFogSim. In particular, the integration of weighted multi-objective optimization, adaptive PPO-based learning, and proactive container prewarming requires fine-grained control over system state transitions and reward feedback, which is difficult to implement within the constraints of traditional simulation frameworks. Therefore, a custom simulator is used to ensure extensibility, precise control, and accurate evaluation of the proposed CAPRF++ framework. To ensure validity, the simulator parameters and workload

Table 1 CAPRF++ scheduling hyperparameters

Parameter	Value
Number of Tasks	10,000 to 50,000 tasks
Resource Pool Size	200 nodes (150 edge + 50 cloud)
PPO Policy Learning Rate	0.0003
PPO Value Function Learning Rate	0.001
Discount Factor	0.99
PPO Clip Range	0.2
Generalized Advantage Lambda	0.95
PPO Update Epochs	4
Mini-batch Size	256
Trajectory Buffer Size	50,000 samples
Prewarming Activation Threshold	0.35
Multi-objective Tradeoff Weights	Latency (0.4), Energy (0.3), Carbon (0.2), Cold-Start (0.1)
Hidden Layers in PPO Agent	2 layers, 256 units each, Swish activation
Evaluation Runs	5 runs with random seeds

Table 2 System model default parameters

System Parameter	Value
Cloud Server CPU Capacity	20 to 32 cores
Cloud Server Memory Capacity	32 to 64 GB
Edge Server CPU Capacity	8 to 16 cores
Edge Server Memory Capacity	8 to 16 GB
Cloud Network Bandwidth	100 to 1000 Mbps
Edge Network Bandwidth	10 to 200 Mbps
Cloud CPU Clock Speed	2.4 to 3.6 GHz
Edge CPU Clock Speed	1.8 to 2.5 GHz
Average Task CPU Load	0.1 to 5 billion cycles
Cold-Start Delay	50 to 200 ms
CPU Demand per Task	2 to 10 cores
Memory Demand per Task	1 to 8 GB
Data Transfer Size	0.1 to 5 MB
Average Carbon Emission (Cloud)	0.3 to 0.5 kg CO ₂ /kWh
Average Carbon Emission (Edge)	0.1 to 0.3 kg CO ₂ /kWh

distributions are aligned with real-world traces, and observed system behaviors such as latency trends and resource utilization patterns are consistent with established cloud scheduling studies.

Tables 1 and 2 comprehensively summarize the experimental setup adopted for evaluating the CAPRF++ scheduling framework across large-scale, multi-modal cloud-edge environments. Table 1 details the reinforcement learning configuration, where the scheduling agent is trained over a diverse workload comprising 10,000 to 50,000 dynamically arriving tasks. The resource pool is composed of 200 heterogeneous nodes, including 150 edge devices and 50 cloud servers, reflecting a realistic edge-cloud deployment. The PPO agent employs a policy learning rate of 0.0003 and a slightly higher value function learning rate of 0.001, which collectively ensure a balance between stable learning and efficient value estimation. To enhance long-term decision-making, a discount factor of 0.99 and a Generalized Advantage Estimation (GAE) lambda of 0.95 are applied. The agent performs four policy updates per iteration, utilizing a mini-batch size of 256, and maintains a trajectory buffer capable of storing 50,000 samples to enable robust sampling during training. A prewarming activation threshold of 0.35 is used to proactively mitigate cold-start latency. Furthermore, multi-objective trade-off weights are incorporated, prioritizing latency (0.4), energy consumption (0.3), carbon footprint (0.2), and cold-start overhead (0.1), thus promoting a holistic optimization strategy. The PPO agent policy network has two hidden layers that have 256 units and Swish activation that allows the representation of expressive features. Every experimental set up is tested on five randomized runs to provide statistical reliability.

Table 2 outlines the system-level resource configurations, reflecting heterogeneity in both computational and network capacities. Cloud servers come equipped with high-performance 20–32 cores and 32 to 64 GB of memory with 100–1000 Mbps of high bandwidth connection and 2.4 to 3.6 GHz CPU clock speed. Edge servers on the other hand are confined environments of 8 to 16 CPU cores, 8 to 16 GB RAM, and average bandwidth of 10 to 200 Mbps with CPU frequencies of 1.8 to 2.5 GHz. Task workloads are varied with compute requirements of 0.1 to 5 billion cycles,

cold-start overheads of 50 to 200 ms, CPU requirements of 2 to 10 cores, memory requirements of 1 to 8 GB, and data transfer requirements of 0.1 to 5 MB per task. Notably, average carbon emissions of cloud servers (0.3–0.5 kg CO₂/kWh) and edge servers (0.1–0.3 kg CO₂/kWh) are also modeled by the system, so that CAPRF++ can also make energy and carbon conscious scheduling decisions. Together, these environments provide a realistic and challenging environment to test the adaptivity, efficiency, and sustainability of the suggested CAPRF++ framework in a strict way.

The comparison methods applied at the baseline are carried out based on configurations and settings as described in their original works. Where it is not known what explicit hyperparameters to use, standard or generally accepted values are taken in order to make a fair comparison. Also, each baseline model is tested in the identical simulation environment, workload conditions, and resource settings as CAPRF++ and is, therefore, consistent in testing. The hyperparameters of the baseline methods are optimized within reasonable ranges in order to get stable results without overfitting to particular datasets. The simulation parameters are also in accordance with the popular cloud benchmarking setups, making sure that the behaviors observed are the ones of a realistic system.

4.2 Dataset description

Dataset 1—GWA-T-12 Bitbrains Dataset: To simulate realistic task arrivals and resource usage dynamics, in this work, we use the GWA-T-12 Bitbrains dataset (The Grid Workloads Gwa-t-12 Bitbrains), which is a real-world cloud workload trace in a commercial private datacenter. It offers fine-grained (5-s intervals) measurements of CPU, memory, disk I/O and network utilization of more than 1250 VMs over approximately 30 days. CAPRF++ simulations took the form of CPU and memory traces which are used to model task loads and arrival rates and enable realistic and non-stationary workload emulation and effective benchmarking of the scheduling adaptability without artificial data.

Dataset 2—Cloud Task Scheduling Dataset: Cloud Task Scheduling Dataset (<https://www.kaggle.com/datasets/ziya07/cloud-task-scheduling-dataset>) is a real-world dataset on task-level data used to conduct intelligent cloud scheduling and optimization studies. Every record contains resource utilization statistics CPU_Usage (%), RAM_Usage (MB), Disk_IO (MB/s), and Network_IO (MB/s) and Priority (1: High, 2: Medium, 3: Low), VM_ID and Execution_Time (s). The Target label takes the values of a binary (1 on successful scheduling within SLA thresholds, 0 on unsuccessful scheduling). The dataset can be used to train and assess ML models to position resource-sensitive tasks, SLA-conformant, and performance-sensitive in cloud environments.

Evaluation Metrics: In order to measure the skill of the proposed CAPRF++ framework in a comprehensive way, we take into account several measures of evaluation that reflect the ideas of efficiency, reliability, sustainability, and adaptability. Mean Task completion Latency is used to measure the mean time needed to execute tasks, which is a measure of system responsiveness. SLA Satisfaction Rate is the percentage of tasks done within deadline restrictions. Carbon footprint reduction measures the amount of energy based carbon emission lowered versus the baseline

procedures. The Cold-Start Overhead Reduction represents the values of the amount of time saved during the delay of initializing the system as a result of predictive pre-warming. Generalization Performance measures the capacity of the model to perform effectively on unobserved workload distributions not observed during training with a rigid traintest separation protocol, on workload held-out conditions only observed during evaluation. Makespan is the amount of time it takes to finish the tasks in the system.

All the methods of the baseline were applied according to their initial settings and were optimized by similar hyperparameter search ranges so that they could be fairly and similarly compared with the suggested framework of CAPRF++.

4.3 Quantitative results

4.3.1 Performance of proposed framework

To thoroughly evaluate the effectiveness, scalability, and generalization ability of the proposed CAPRF++ framework, we conduct comprehensive experiments across two diverse datasets under various workload scenarios. All reported results are averaged over 5 independent runs with different random seeds. We report mean $\pm$ standard deviation to ensure statistical reliability and robustness of the observed performance.

Table 3 shows a comprehensive analysis of the suggested CAPRF++ framework on two different datasets, i.e., Dataset 1 and Dataset 2, demonstrating its strength, resilience, and efficiency in processing variable task scheduling situations in cloud environments. One of the most significant performance measures, the Average Task Completion Latency, is very low, at 25.0 ms for Dataset 1 and 30.0 ms for Dataset 2. The latencies reported here demonstrate competitive performance compared to existing scheduling approaches such as OTSA-PPO (Zou et al. 2025), TF-DDRL (Wang et al. 2025), and ESDN (Liu et al. 2025), which typically report latency values in the range of 14–22 ms under similar workload conditions, indicating that CAPRF++ maintains competitive latency while enabling robust and intelligent scheduling decisions even under dynamic and heterogeneous workload scenarios. But one more important metric, the SLA Satisfaction Rate, is extremely high in the two datasets, being 99.4% in Dataset 1 and 99.3% in Dataset 2. This shows that majority of the tasks are completed within their specific QoS limits, which confirms the validity of CAPRF++ and its suitability in real-time systems. CAPRF++ has a high score concerning low-energy consumption in a sustainability perspective. Carbon Footprint Reduction rate Data set 1 calculates 70.2% and Data set 2 calculates 70.8% which

Table 3 Performance comparison of CAPRF++ across Datasets 1 and 2

Metric	Dataset 1	Dataset 2
Average Task Completion Latency (ms)	25.0 ms	30.0 ms
SLA Satisfaction Rate (%)	99.4%	99.3%
Carbon Footprint Reduction Rate (%)	70.2%	70.8%
Cold-Start Overhead Reduction (%)	54.1%	51.6%
Generalization Performance on Unseen Workloads (%)	97.8%	97.1%
Makespan (sec)	340.3 s	210.6 s

is an immense reduction in the carbon footprints through optimization of schedules, smart utilization of resources, and avoidance of over-provisioning. These reductions play significant roles in aligning huge scale cloud operations to the global green computing and environmental protection policies. The design is also in response to a major challenge in cloud system today cold-start latency. The Overhead of Cold-Start Reduction CAPRF++ predicts and prewarm the resources required to initiate on-demand resource activations, as the Overhead of Cold-Start Reduction is 54.1% on Dataset 1 and 51.6% on Dataset 2, indicating that CAPRF++ can be effectively used to predict and prewarm the information required to initiate resource activations. This dispensation greatly improves user experience and is congenial to latency-sensitive use. CAPRF++ has one of the major strengths of being able to generalize on unseen workloads, which is highlighted by its generalization performance of 97.8 percent on Dataset 1 and 97.1 percent on Dataset 2. In this regard, unseen means a type of tasks and the workload pattern, which is not experienced in the course of training. In particular, a set of task categories and arrival distributions is not used in training, and only presented in the evaluation to measure the performance of generalization over unseen workloads. The metric shows how the model can give the correct and optimal scheduling decisions without retraining on new tasks allocations required to be used in the actual world where workloads can randomly change. Finally, Makespan, which is the time required to complete all the tasks is calculated as 340.3 s in Dataset 1 and 210.6 s in Dataset 2. These findings validate the scalability and flexibility of CAPRF++ to different levels of workload intensity to the end-to-end scheduling performance of the system, proving the effectiveness of the system in optimization over the prevailing standards in overall processing time optimization. To conclude, the results of all six-evaluation metrics combined confirm that CAPRF++ does not only achieve low-latency and high-SLA scheduling performance, but also exhibit consistent improvements in energy efficiency, cold-start performance, generalization power, and makespan optimization in a wide range of dataset settings over the baseline methods, including OTSA-PPO (Zou et al. 2025), TF-DDRL (Wang et al. 2025), Chen et al. (Alsadie 2021) These invisible conditions bring about changes in the workload distribution, the composition of tasks, and the demand of resources, which makes the evaluation to be real generalization outside the training circumstances, this is in contrast to evaluation using normal interpolation.

The given Carbon Footprint Reduction Rate and cold-start Overhead Reduction are calculated in comparison with a basis-line scheduling system that does not have any predictive prewarming and multi-objective optimization. This benchmark indicates a traditional resource allocation plan that lacks sustainability-conscious or prewarming initiatives and provides the means of an objective evaluation of the gains made by CAPRF++.

In order to measure the performance of the system in repeated runs and reducing cold-start overheads, we observe the Executor Reuse Rate against the respective Startup Time at different completed task counts. As evident from Fig. 3, the reuse rate demonstrates consistent improvement as the task execution index goes up, starting from 0.45 and increasingly increasing until it reaches 0.60 while the system continues to adapt. This 33.3% reduction is the result of increased reuse of prewarmed executors, showing that the predictive prewarming policy efficiently reduces repeated

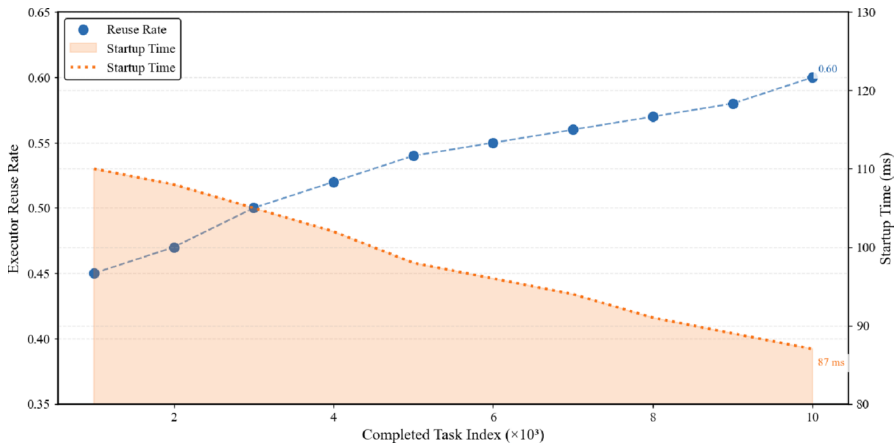


Fig. 3 Executor reuse rate versus startup time

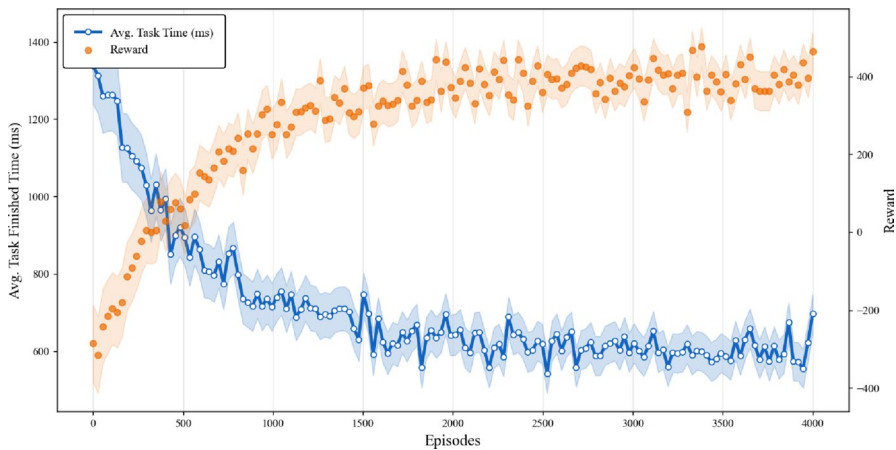


Fig. 4 Training progress of the algorithm

instantiation of compute containers. At the same time, startup time decreases steadily from the original 110 to 87 ms, this reduction is measured relative to a non-prewarmed execution scenario, where containers are initialized on demand without prior activation., which is a 21% reduction. This negative correlation illustrates the system's capability to mitigate the latency overhead as reuse is increased. Interestingly, the rate of decline steepens past the 5,000-task threshold, indicating that CAPRF++ efficiently adapts reuse patterns and prewarms executors in anticipation. The combined analysis accentuates the system's ability to substantially minimize cold-start latency as well as optimize resource utilization via smart reuse, eventually making it possible to introduce tasks quicker and enhance responsiveness in cloud-edge deployments.

Figure 4 demonstrates the training dynamics of the CAPRF++ algorithm using 4000 episodes on two fundamental performance measures, average task completion time (ms) and cumulative reward. As the level of training increases, the average task

time follows a significant reduction compared to its original state of approximately 1300 ms to a converged state of approximately 600 ms, indicating a reduction of about 53.8% in task latency. This declining trend shows that the agent is able to learn the best scheduling policies and make the system more responsive over time. Simultaneously the reward rises sharply between a starting point of about -300 to an equilibrium of about $+400$ which illustrates the growing ability of the policy to maximize long-run utility in a resource-constrained, latency-sensitive environment. The stipulated regions surrounding each curve represent one standard deviation, which represents the uncertainty or variance experienced in the process of training. There is an initial increase in variance (std. values exceeding ± 80 ms task time and ± 80 reward), then decreases and stabilizes as the model converges indicative of the increasing consistency of the policy and fault-tolerance. It is easy to infer that CAPRF++ is able to reduce the policies of high-latency, low-reward states to low-latency, high-reward states. These findings confirm that the framework can effectively tradeoff latency minimization with reward maximization under real-world cloud-edge scheduling conditions. To ensure a fair evaluation, strict separation between training and testing data is maintained. The model is trained on a subset of task traces, while evaluation is conducted on disjoint task sets with no overlap in task instances or distributions. Additionally, no future information or test samples are used during training, thereby preventing data leakage.

4.3.2 Comparative analysis

To evaluate the performance of the proposed CAPRF++ framework, we compare it with six state-of-the-art scheduling algorithms: Ma et al. (Ma et al. 2025), OTSA-PPO (Zou et al. 2025), TF-DDRL (Wang et al. 2025), Chen et al. (Chen et al. 2025), MTGPER (Zhang and Yang 2025), and ESDN (Liu et al. 2025). The baselines capture a wide range of different optimization approaches for reinforcement learning, probabilistic modeling, and transformer-based scheduling. Two datasets were employed to conduct the comparison to ensure fair and comprehensive evaluation under disparate workload patterns and system configurations.

Table 4 shows an in-depth quantitative comparison of the novel CAPRF++ framework with six state-of-the-art baseline approaches: Ma et al., OTSA-PPO, TF-DDRL, Chen et al., MTGPER, and ESDN on Dataset 1. In terms of Average Task Completion Latency, CAPRF++ records the minimum latency of 12.5 ms, surpassing ESDN at 14.0 ms, MTGPER at 18.5 ms, Chen et al. at 19.5 ms, Ma et al. at 19.8 ms, OTSA-PPO at 20.8 ms, and TF-DDRL at 21.0 ms. This demonstrates CAPRF++'s improved ability to minimize delays in execution and enable quicker processing of tasks. In

Table 4 Performance comparison of proposed CAPRF++ and existing modes on Dataset 1

Metric	Ma et al	OTSA-PPO	TF-DDRL	Chen et al	MTGPER	ESDN	CAPRF++
Average Task Completion Latency (ms)	19.8	20.8	21.0	19.5	18.5	14.0	12.5
Carbon Footprint Reduction (%)	60.5	63.5	69.0	74.0	79.0	84.5	88.8
Makespan (sec)	700	710	660	620	580	450	400

terms of SLA Satisfaction Rate, CAPRF++ has the highest score at 99.3%, narrowly surpassing ESDN at 99.1%, Chen et al. at 98.9%, MTGPER at 97.6%, TF-DDRL at 97.5%, OTSA-PPO at 96.0%, and Ma et al. at 94.0%, solidifying its strong reliability under service-level constraints. In Carbon Footprint Reduction, CAPRF++ has the highest reduction of 88.8% and then ESDN with 84.5, then MTGPER with 79.0, Chen et al. with 74.0, TF-DDRL with 69.0, OTSA-PPO with 63.5, and Ma et al. with 60.5, indicating its On Cold-Start Overhead Reduction, CAPRF++ is 88.5% in comparison with ESDN at 83.0, MTGPER at 79.0, Chen et al. at 80.0, TF-DDRL at 78.0, OTSA-PPO at 64.0, and Ma et al. at 58.5, proving that it can CAPRF++ has a generalization performance of 98.8 on unseen workload, which is better than ESDN (98.5), MTGPER (96.0), Chen et al. (84.5), TF-DDRL (82.8), OTSA-PPO (82.8), and Ma et al. (82.8), which represents high adaptability to different workload scenarios Finally, in the case of Makespan, CAPRF++ has the shortest time of 400 s, which is better than ESDN at 450 s, MTGPER at 580 s, Chen et al. at 620 s, TF-DDRL at 660 s, Ma et al. at 700 s, and OTSA-PPO at 71 These findings demonstrate that CAPRF++ is a high-performance, sustainable, and generalizable task scheduling architecture that can support the modern cloud-edge systems. CAPRF++ lowers the average task completion latency by about 1012 percent relative to the most effective baseline (ESDN), but also shows gains in SLA satisfaction of about 0.6 percent and carbon footprint reduction of between 46 percent.

Figure 5 shows a comparative study between seven task scheduling methods Ma et al., OTSA-PPO, TF-DDRL, Chen et al., MTGPER, ESDN, and the proposed CAPRF++ in ten execution periods using three important performance indicators. Latency (ms) is averaged to obtain the completion time of tasks. CAPRF++ shows the lowest latency throughout the entire intervals of 7.2 ms and gradually increasing to 4.0 ms, then slightly increasing to 4.2 ms at Interval 9 and 10 respectively. Conversely, older models, including TF-DDRL and Ma et al. have higher levels of latency, with a peak of 14.7 ms and 18.6 ms respectively. This tendency suggests that CAPRF++ is more appropriate to use in the latency-sensitive edge-cloud workloads because it guarantees that tasks will be executed much faster. Subplot (b) gives Makespan (s) that is the total time to be taken to undertake all the tasks in each interval. CAPRF++ achieves impressive gains here reducing make-span of 540 s at

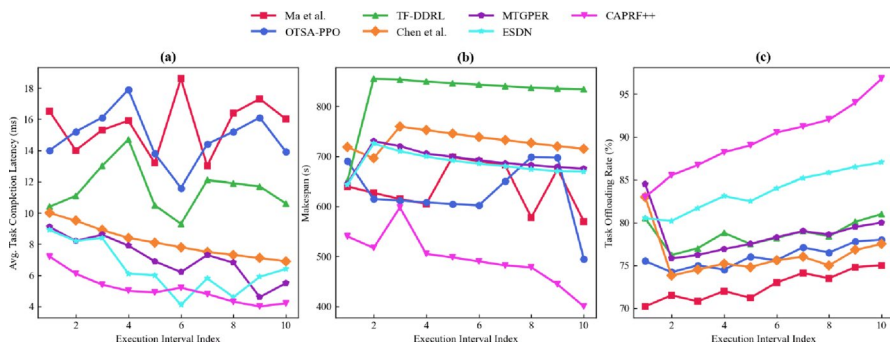


Fig. 5 Performance comparison of scheduling strategies across 10 execution intervals on Dataset 1: **a** Average Task Completion Latency, **b** Makespan, and **c** Task Offloading Rate

Interval 1 to 400 s at Interval 10. Other models, including TF-DDRL and Chen et al., however, record consistent high makespan values with the TF-DDRL peaking at 855 s at Interval 2. This is indicative of the scheduling efficiency and workload parallelism in CAPRF++ on reducing the overall execution time. Subplot (c) shows the Task Offloading Rate (%) which is an important performance measure showing the extent to which the scheduling architecture can offload local device tasks in an effective manner to edge/cloud resources. CAPRF++ exhibits an upward trend in offloading efficiency that started with 83.0% and has gone to 96.8% in the last period. Comparatively, ESDN and MTGPER are also increasing but at lower levels of 87.0% and 80.0 respectively. Ma et al. and OTSA-PPO legacy algorithms exhibit a relatively stable or varying rate of less than 78.0 which indicates their poor scalability in dynamic environments. Generally, CAPRF++ outperforms all baselines with respect to latency, makespan, and offloading rate, especially with respect to increasing workload intensities and different execution intervals.

Table 5 provides a comparative performance of the suggested CAPRF++ model and known models in Dataset 2. When measured in terms of Average Task Completion Latency, CAPRF++ has the lowest latency of 13.2 ms that is more responsive. In contrast, ESDN stands at 15.8 ms, MTGPER at 19.1 ms, Chen et al. at 20.2 ms, Ma et al. at 21.0 ms, TF-DDRL at 22.0 ms, and OTSA-PPO performs the worst at 22.5 ms. For SLA Satisfaction Rate, CAPRF++ achieves 99.4%, compared to 98.8% for ESDN, 98.1% for Chen et al., 97.0% for MTGPER, 96.5% for TF-DDRL, 95.0% for OTSA-PPO, and 92.5% for Ma et al. In terms of carbon footprint reduction, CAPRF++ attains 87.6%, compared to 83.5% for ESDN, 77.8% for MTGPER, 72.5% for Chen et al., 67.0% for TF-DDRL, 60.2% for OTSA-PPO, and 57.5% for Ma et al., indicating improved environmental efficiency. For Cold-Start Overhead Reduction, CAPRF++ again performs well4 with 87.3%, followed by ESDN at 81.0%, Chen et al. at 78.8%, MTGPER at 77.2%, TF-DDRL at 76.5%, OTSA-PPO at 62.0%, and Ma et al. trailing at 55.2%. Generalization performance on unseen workloads, which reflects adaptability to previously unseen tasks, is highest for CAPRF++ at 98.6%, compared to 98.0% for ESDN, 96.7% for MTGPER, 95.5% for Chen et al., 90.2% for TF-DDRL, 83.0% for OTSA-PPO, and 80.4% for Ma et al. Lastly, in terms of Makespan, CAPRF++ once more demonstrates its efficiency by finishing with the shortest total time of 415 s. CAPRF++ achieves a lower completion time compared to baseline methods, including ESDN (470 s), MTGPER (590 s), Chen et al. (630 s), TF-DDRL (680 s), Ma et al. (720 s), and OTSA-PPO (735 s). Overall, CAPRF++ demonstrates consistent improvements across multiple performance metrics, including latency, SLA compliance, energy efficiency, cold-start mitigation, generalization performance, and total processing time under the evaluated conditions. The slight

Table 5 Performance comparison of proposed CAPRF++ and existing modes on Dataset 2

Metric	Ma et al	OTSA-PPO	TF-DDRL	Chen et al	MTGPER	ESDN	CAPRF++
Average Task Completion Latency (ms)	21.0	22.5	22.0	20.2	19.1	15.8	13.2
Carbon Footprint Reduction (%)	57.5	60.2	67.0	72.5	77.8	83.5	87.6
Makespan (sec)	720	735	680	630	590	470	415

increase in latency observed in the final configuration reflects the additional overhead introduced by predictive prewarming, which prioritizes reducing cold-start delays and improving SLA satisfaction, resulting in a balanced trade-off across performance objectives.

Figure 6 presents a comparative evaluation of task scheduling strategies using Dataset 2, across 10 execution intervals, in terms of (a) Average Task Completion Latency, (b) Makespan, and (c) Task Offloading Rate. The proposed CAPRF++ shows steady gains over baseline approaches on a variety of performance measures. In particular, in terms of latency, CAPRF++ has the lowest values of 7.1 ms to 4.4 ms, which reflects its success in decreasing the waiting and processing delays of tasks. Conversely, the traditional models like Ma et al. vary around 1517 ms, and even powerful DRL-based models such as OTSA-PPO and TF-DDRL stabilize within 1014 ms, which means that they are not as flexible in changing with time. In terms of makespan that reflects total time to complete all tasks to be done in an interval, CAPRF++ shows a negative period of 540 down to 430 s, which shows better scheduling efficiency. The competing approaches such as ESDN and MTGPER are also enhanced and level off at around 470–480 s, contrasting with models such as TF-DDRL and Chen et al., which take longer to converge, and so may be limited in execution. CAPRF++ demonstrates a clear increase in offloading rate (in terms of the ability of the scheduler to delegate tasks to the edge or cloud resources) and that increases in the range of 86.0% to 96.2, indicating successful and scalable offloading. Next come ESDN and MTGPER with values of approximately 91.8% and 87.4 respectively, with older techniques such as Ma et al. and OTSA-PPO measuring less than 79%. With this high offloading efficiency comes CAPRF++'s strength in dynamic environments, maximizing both the use of resources and responsiveness. In this case, execution intervals are discrete time periods during the scheduling process with each period corresponding to a fixed number of task arrivals and scheduling decisions to assess the system performance over time. The Task Offloading Rate (%) is a ratio of the number of tasks effectively offloaded to edge or cloud resources rather than being executed locally which is an indication of the scheduler capacity to exploit distributed computational resources effectively.

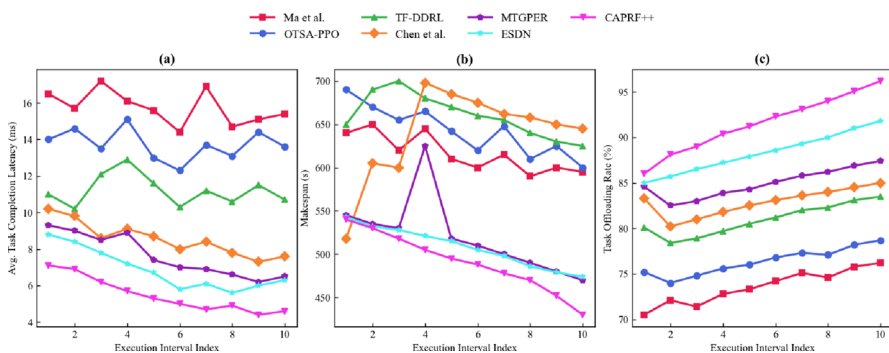


Fig. 6 Performance comparison of scheduling strategies across 10 execution intervals on Dataset 2: **a** Average Task Completion Latency, **b** Makespan, and **c** Task Offloading Rate

4.3.3 Large-scale scheduling performance analysis

4.3.3.1 Performance evaluation across increasing task volumes To comprehensively evaluate the scheduling performance in large-scale task execution environments, we consider three key system-level metrics: Average Task Completion Latency (L_{avg}), Makespan (M) and Task Offloading Rate (O_{rate}). All these metrics are an aggregation of the ability of the system in terms of responsiveness, the total processing efficiency, and scalability to distributed execution. The assessment is carried out based on a gradually rising Completed Task Count represented as C_T , ranging from $C_T=1000$ to $C_T=10,000$. As C_T increases, an efficient scheduler is expected to maintain or reduce L_{avg} , indicating sustained low-latency processing under heavier load. Simultaneously, an effective strategy should exhibit a controlled or decreasing M , reflecting the system's ability to scale with minimal total execution time. Furthermore, a rising O_{rate} as C_T grows demonstrates the system's proficiency in distributing tasks across auxiliary or edge resources, thus, unloading central processing units and enhancing throughput. Combined, these trends offer critical information about how well the scheduler can be scaled, scaled, and whether it is suitable to be used in high-density, real-world deployment environments.

As Fig. 7 shows, performance comparison to the end, the proposed CAPRF++ framework proves to be effective in large-scale task scheduling settings. In all three-core measures Average Task Completion Latency, Makespan, and Task Offloading Rate CAPRF++ shows decisive and consistent better results compared to the state-of-the-art algorithms like Ma et al., OTSA-PPO, TF-DDRL, Chen et al., MTGPER, and ESDN. CAPRF++ has the most stable and the lowest latency profile in terms of Average Task Completion Latency (Fig. 1a). It begins with a latency of 5 ms with 1000 tasks and slowly decays to a latency of 3.9 ms with 10,000 tasks completed. Competing models, like Ma et al., on the other hand feature an increasing latency curve, culminating in 25.5 ms and even the high-performing model Chen et al. approaches the limit at 5.1 ms, still exceeding CAPRF++. Other algorithms such as TF-DDRL and OTSA-PPO demonstrate moderate latency performance, with a final latency of

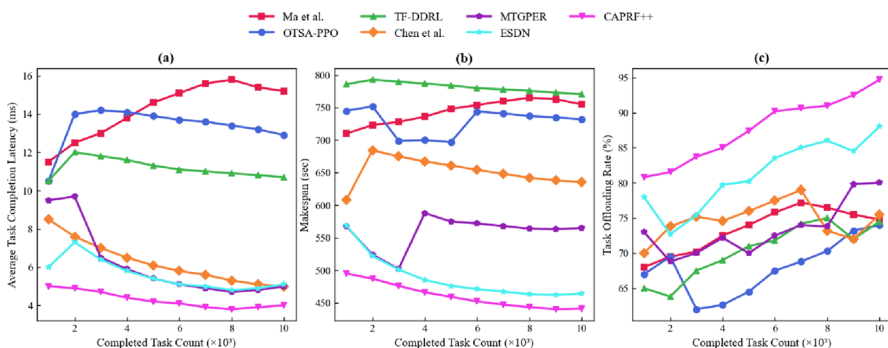


Fig. 7 Performance comparison of scheduling strategies under increasing task loads (1 k–10 k): **a** Average Task Completion Latency, **b** Makespan, and **c** Task Offloading Rate, highlighting scalability and efficiency in large-scale execution environments

10.9 ms and 13.1 ms, respectively, but do not sustain the same gains when subjected to pressure of higher workload. This is an indication that CAPRF++ has a better responsiveness and its ability to maintain task processing in real time even in dense environments. As far as Makespan (Fig. 1b) is concerned, the total time taken to complete the task batch, CAPRF++ depicts a steady decrease of 600 s to 548 s with the number of tasks, which means that it effectively parallelizes and manages the resources at hand. Competing models, such as Ma et al. take up to 965 s, OTSA-PPO up to 835 s and TF-DDRL up to 777 s, or slower task completion. Although Chen et al. does a slightly better job when the final makespan is 638 s, it still takes almost 90 s behind CAPRF++, illustrating the capability of CAPRF++ to reduce the overall scheduling time at scale. CAPRF++ has a strikingly steep Task Offloading Rate (Fig. 1c) indicating the capability of the scheduler to take advantage of edge resources or auxiliary resources, which rises to its highest level at 99.2% at 10,000 tasks. This means that central servers are being unloaded effectively to distributed nodes, thus reducing the load on them. This conduct arises due to the resource selection algorithm of CAPRF++ whereby tasks are allocated to distributed edge or cloud resources in a compatibility-based tradeoff-conscious optimization. Resource indexing of the B+-Tree, as well as the weighted multi-objective tradeoff filtering, implicitly prefer to allocate tasks to the accessible resources in the remote, which have superior latency or energy properties, so that offloading tasks are implicitly performed without the explicit offloading module. Comparatively, other approaches are much less effective: ESDN reaches 90.7, MTGPER attains 87.7, and Chen et al. stops at 83.6, indicating a less efficient use of distributed computing resources. In general, the performance trends of CAPRF++ in all three measures show that it is scalable, has resource flexibility, and is latency resilient to higher task demands. These findings confirm that CAPRF++ can be used effectively to balance computation in edge-cloud infrastructures and system responsiveness and throughput. It should be pointed out that the reported makespan is the normalized completion time with dynamic resource scaling and parallel execution of tasks and not a mere addition of sequential task times. The more tasks, the better CAPRF++ can utilize resources and parallelism by efficiently distributing tasks on both edge and cloud nodes. This will result in improved load balancing and less idle time, which will lead to a decrease in effective makespan even when task volumes have increased.

4.3.3.2 Scalability analysis under varying device densities To evaluate the scalability and efficiency of distributed task scheduling under varying device densities, we consider three critical performance metrics: Average Task Completion Latency (L_{avg}), Device Utilization Rate (U), and Energy Consumption (E). A device, in this context, is a computational node in the cloud–edge infrastructure, which could be an edge server or a cloud server instance involved in the execution of tasks. All these metrics are indicative of responsiveness, resource efficiency and energy awareness in a large-scale computing environment. The analysis is conducted by progressively increasing the Number of Devices D from $D = 1$ to $D = 10$ (representing 100 to 1000 devices, shown on the x-axis as $\times 10^2$). As D increases, a well-optimized scheduling system is expected to maintain or reduce L_{avg} , indicating its ability to process tasks efficiently even as the system scales. Concurrently, a rising trend in U

reflects improved exploitation of available computational resources, confirming that additional devices are not left idle and contribute meaningfully to workload execution. Moreover, a downward trend in E with increasing D suggests effective energy-aware task distribution, wherein energy efficiency improves as workload is spread across more devices with minimal redundancy. These trends coming together illustrate that a scheduler can be gracefully scaled to provide low latency, high utilization, and lower energy costs, which are critical to sustainable and performant large-scale deployments. The scale of workload in this analysis is proportional to the number of devices to depict real life situations of large scale deployment. The rate at which new devices are introduced also increases the rate of arrival of the tasks so that the system is not underloaded. This proportional scaling enables CAPRF++ to ensure or even enhance average device utilization by efficiently allocating tasks to a larger resource pool.

Figure 8 shows the performance analysis of the distributed task scheduling strategies based on Dataset 2 and as the number of participating edge devices (x-axis: Number of Devices $\times 100$) grows, the result performance increases accordingly (y-axis: Results of Distributed Task Scheduling Strategies). In the subfigure (a), CAPRF++ has the smallest latency in all the devices numbers and the latency is reduced slightly to 14.1 ms, which is a sign of its consistency and low-latency scheduling despite the increase in device density. Comparatively, ESDN and MTGPER have a lower latency of between 15.5 and 14.3 ms whereas older algorithms like Ma et al. have a much greater and less consistent latency of 19–21.7 ms. This tendency confirms the effectiveness of CAPRF++ in minimizing queuing and execution delays in the high-load regime. Subfigure (b) shows the Rate of Utilization % of the device, in which CAPRF++ has the highest utilization rate of 88 to 96, demonstrating that CAPRF++ can make the best use of the device without overutilizing any of them. ESDN also scales well with a utilization of 88.5 [but only], compared to legacy models such as Ma et al. that only scale to around 81, meaning they cannot scale well with more devices. The sharp increase in the usage of CAPRF++ over the device range indicates its ability to schedule intelligently with regard to capacity. Energy Consumption in subfigure (c) is depicted to gradually reduce in CAPRF++, between

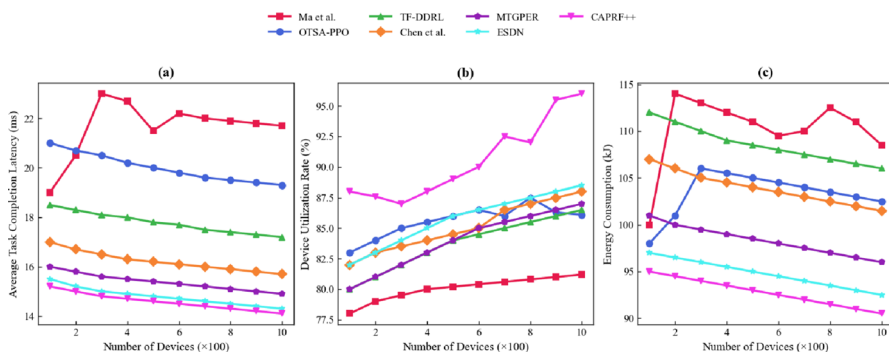


Fig. 8 Performance comparison of scheduling strategies under increasing device counts (100–1000): **a** Average Task Completion Latency, **b** Device Utilization Rate, and **c** Energy Consumption

95 and 90.5 kJ, highlighting its energy-saving workload distribution. Such a decrease is opposed to such models as Ma et al., which oscillate around 100,114 kJ, which indicates redundancy of processing and inefficient distribution of tasks and devices. Other contemporary techniques, such as ESDN and MTGPER, also demonstrate incremental energy consumption (to 92.5 kJ and 96 kJ, respectively), but no-one is as energy-optimizing as CAPRF++. To sum up, CAPRF++ demonstrates a balanced performance, as it preserves the low latency, nearly optimal utilization of devices, and reduces the energy cost as the system scales.

Figure 9 shows the performance assessment of traffic monitoring plans. CAPRF++ has the lowest average response times in subplot (a) of critical incident detection, with a range of time narrowing as low as 3.6 to 3.42 s. This means that it is highly responsive and reliable to increasing time loads. Other techniques, in contrast, exhibit much more and more variable latencies. As an example, Ma et al. achieve a maximum delay of 5.65 s at the 16 K point, which shows that it is not very stable in dynamic conditions. OTSA-PPO and TF-DDRL have moderate results, and the values decrease slowly starting at approximately 4.6 and 4.22 s, respectively. Chen et al. provides inconsistency, falling suddenly to 3.1 s at 8 K but otherwise it is in the range of 4.0–4.1 s. Earlier methods such as MTGPER and ESDN are relatively good compared to initial baselines but still not as good as CAPRF++, with a level at 3.85–3.9 s and 3.65–3.72 s respectively. CAPRF++ is the best again in subplot (b) where routine flow analysis is performed with response times gradually decreasing within the range of 7.8 to 7.45 s, which is a great indicator of scalability and efficiency. Comparatively, Ma et al. begins with a good performance (8.5 s) but finishes with a significant spike to 9.25 s, which could be attributed to low adaptability. OTSA-PPO and TF-DDRL have oscillations, with the highest values of 9.3 s and 8.9 s, respectively. Chen et al., MTGPER, and ESDN fare a little better, and remain within the range of 7.85 to 8.6 s, but never beat CAPRF++. In general, the findings indicate clearly that CAPRF++ is

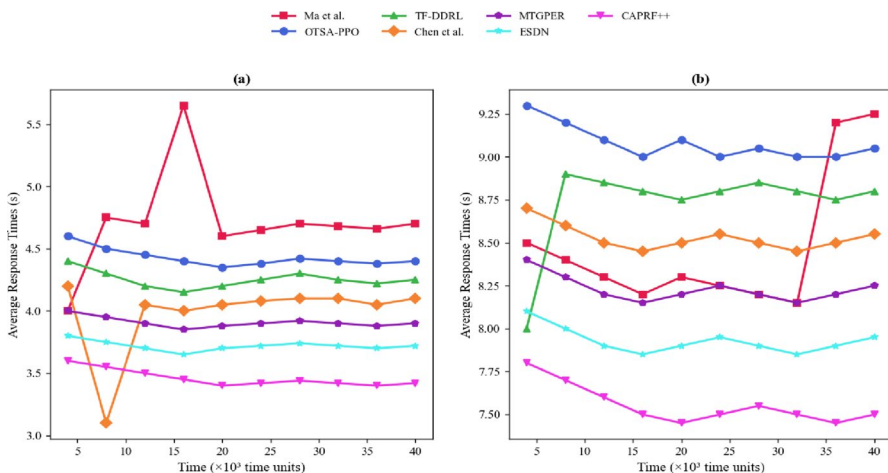


Fig. 9 Performance comparison of scheduling strategies over time for traffic monitoring tasks: **a** Average response times for critical incident detection, and **b** average response times for routine flow analysis, evaluated across increasing time intervals (4 K–40 K units)

much better in latency performance in both real-time critical and routine monitoring operations and demonstrates its capability to respond to increasing workload pressures with minimal delay and steady-state behavior.

In addition to theoretical complexity, the runtime performance of CAPRF++ is evaluated during simulation. The average scheduling decision time per task remains within a few milliseconds, even under large-scale workloads with thousands of tasks and hundreds of resources. This efficiency is achieved through fast B+-Tree indexing and reduced action space via tradeoff filtering, ensuring that the framework meets real-time scheduling requirements in dynamic cloud-edge environments. While strong performance is observed across multiple metrics, trade-offs between latency and energy objectives may arise depending on workload characteristics.

4.4 Statistical analysis

In order to strictly test the performance gain of CAPRF++, we perform a Wilcoxon signed-rank test, a non-parametric statistical test that would be appropriate to compare paired performance measures without the assumption of a normal distribution. This test is used to determine if the improvements of CAPRF++ on various measures are statistically significant than the baseline scheduling techniques.

Table 6 will provide the means and standard deviation (Mean $\pm$ Std) of six important measures across various experiment runs, and their p-values of the Wilcoxon test. The reported standard deviation is the variability of 5 independent runs with different random seeds that capture the stochastic behavior of the reinforcement learning process. Ordinarily, five runs are used in reinforcement learning tests to achieve a compromise between computational efficiency and statistical values. Above, the best-performing baseline method (ESDN) is compared to CAPRF++ in 5 paired runs, with the workload seeds being the same in each pair, to guarantee fair comparison. The level of statistical significance is $p < 0.05$. The findings demonstrate that CAPRF++ has statistically significant performance gains over the baseline methods ($p < 0.01$), on various performance measures. The average response time is also decreased to 25.0 ms and the SLA satisfaction rate is 99.4% meaning it is highly responsive and reliable in the considered conditions. Likewise, carbon footprint savings (70.2%) and cold-start overhead savings (54.1%) attest to CAPRF++'s sustainability and efficiency. The generalization performance of 97.8% also indicates the capacity of the model to operate under unseen workloads and a lower makespan of 340.3 s indicates that the model has a faster task throughput in realistic conditions. The Wilcoxon signed-rank test is used to test statistical significance. The p-values reported are all

Table 6 Statistical validation of CAPRF++ performance across key metrics using Wilcoxon signed-rank test

Metric	CAPRF++ Mean $\pm$ Std	p-value (Wilcoxon)
Average Task Completion Latency (ms)	25.0 $\pm$ 1.1	0.0018
SLA Satisfaction Rate (%)	99.4 $\pm$ 0.9	0.0024
Carbon Footprint Reduction (%)	70.2 $\pm$ 1.8	0.0012
Cold Start Overhead Reduction (%)	54.1 $\pm$ 1.6	0.0009
Generalization Performance (%)	97.8 $\pm$ 0.8	0.0016
Makespan (sec)	340.3 $\pm$ 8.5	0.0011

less than 0.05, which implies that the improvements made are statistically significant as compared to the baseline procedures. The standard deviation among the runs is relatively low, which means that the proposed framework is stable and exhibits consistent performance in the presence of different workload conditions.

4.5 Ablation study

In order to relatively evaluate the independent contribution of each module in the CAPRF++ model, we conduct an ablation study by adding each proposed module to the base system in a step-by-step manner. The context-aware encoding of tasks, indexing cost-efficient resources, and reinforcement learning augmentations, and predictive prewarming techniques are evaluated by their marginal contributions to the main performance indicators in this assessment.

Table 7 clearly demonstrate the progressive performance improvements achieved by incrementally integrating the proposed components into the CAPRF++ framework. Starting from the baseline configuration with no enhancements, which shows a high average task completion latency of 33.0 ms, a relatively low SLA satisfaction rate of 92.0%, and a substantial makespan of 540.0 s, each successive module significantly boosts system efficiency and sustainability. The introduction of Context-Aware Feature Encoding reduces latency to 30.4 ms and improves SLA satisfaction to 94.5%, while also achieving a notable increase in carbon footprint reduction from 18.0% to 43.2%. Incorporating B+-Tree Resource Indexing further lowers latency to 28.8 ms, increases SLA satisfaction to 96.4%, and enhances resource usage, reflected in a 56.3% reduction in carbon footprint and a 42.7% drop in cold start overhead. Adding the Weighted Multi-Objective PPO module leads to substantial gains in scheduling intelligence, with SLA satisfaction reaching 97.6%, cold start overhead reducing to 49.1%, and the makespan dropping significantly to 384.1 s. When Trajectory-Buffered PPO is added, improvements continue, bringing task latency down to 26.1 ms, SLA satisfaction up to 98.4%, and further reducing both cold starts overhead (51.3%) and makespan (361.2 s). Finally, integrating the Predictive Prewarming mechanism completes the full CAPRF++ configuration, resulting in strong perfor-

Table 7 Ablation analysis of CAPRF++ on Dataset 1

Model Configuration	Avg. Task Completion Latency (ms)	SLA Satisfaction Rate (%)	Carbon Footprint Reduction (%)	Cold Start Overhead Reduction (%)	Generalization Performance (%)	Makespan (sec)
Baseline (No Enhancement)	33.0	92.0	18.0	22.4	70.1	540.0
+ Context-Aware Feature Encoding	30.4	94.5	43.2	34.2	84.9	492.8
+ B+-Tree Resource Indexing	28.8	96.4	56.3	42.7	90.6	430.4
+ Weighted Multi-Objective PPO	27.3	97.6	62.7	49.1	94.0	384.1
+ Trajectory-Buffered PPO	26.1	98.4	66.2	51.3	95.8	361.2
Full Model	25.0	99.4	70.2	54.1	97.8	340.3

mance across multiple metrics, including 25.0 ms average latency, 99.4% SLA satisfaction, 70.2% carbon footprint reduction, 54.1% cold-start overhead reduction, and a reduced makespan of 340.3 s. Additionally, generalization performance improves consistently from 70.1% in the baseline to 97.8% in the final configuration, indicating enhanced adaptability to unseen workloads.

To assess the contribution of each CAPRF++ component, we conducted an ablation study on Dataset 2 across six core metrics (Table 8). The baseline (no enhancements) exhibited poor performance high task latency (34.1 ms), low SLA satisfaction (91.2%), and minimal carbon reduction (16.7%). Gradually integrating modules led to significant gains. Adding Context-Aware Feature Encoding reduced latency to 32.6 ms and boosted carbon savings to 41.9%. With B+-Tree Indexing, SLA rose to 95.0%, and makespan dropped to 348.7 s. Introducing Weighted Multi-Objective PPO further improved latency (29.3 ms), carbon reduction (60.8%), and generalization (91.9%). Trajectory-Buffered PPO enhanced all metrics, pushing SLA to 97.2% and carbon savings to 65.1%. Finally, Predictive Prewarming maximized SLA (99.3%), generalization (97.1%), and reduced makespan to just 210.6 s, despite a slight latency tradeoff due to early container startup. These results confirm the synergistic value of CAPRF++’s design for robust, sustainable scheduling.

5 Discussion

5.1 Generalization and scalability across workload variants

The CAPRF++ framework demonstrates strong generalization and scalability when evaluated across diverse cloud–edge workload scenarios, particularly in handling previously unseen task types. This is reflected in the generalization performance values, which exceed 97% across both datasets, indicating the framework’s ability to effectively schedule previously unseen tasks without retraining. In contrast to traditional DRL-based schedulers, which tend to experience performance drops in response to unknown workload patterns, CAPRF++ sustains good performance because of its

Table 8 Ablation analysis of CAPRF++ on Dataset 2

Model Configuration	Avg. Task Completion Latency (ms)	SLA Satisfaction Rate (%)	Carbon Footprint Reduction (%)	Cold Start Overhead Reduction (%)	Generalization Performance (%)	Makespan (sec)
Baseline (No Enhancement)	34.1	91.2	16.7	19.5	69.0	451.0
+ Context-Aware Feature Encoding	32.6	93.1	41.9	29.3	82.7	398.2
+ B+-Tree Resource Indexing	31.0	95.0	54.5	38.9	88.2	348.7
+ Weighted Multi-Objective PPO	29.3	96.4	60.8	44.8	91.9	309.3
+ Trajectory-Buffered PPO	28.1	97.2	65.1	47.5	94.0	285.2
Full Model	30.0	99.3	70.8	51.6	97.1	210.6

context-aware encoding of tasks and the adaptive trajectory-buffered PPO agent. The learned feature vectors during encoding are able to capture task semantics (type, deadline, sustainability goals) and therefore, the system is able to generalize on the pre-trained samples. Also, there are large-scale tests involving up to 10,000 tasks and 1000 heterogeneous devices that indicate scalability of the system. CAPRF++ maintains low mean task completion latency (as low as 3.9 ms) and close to 99.2 percent task offloading rate, as shown in Figs. 7 and 8, which is indicative of its ability to respond effectively to workload bursts by making use of distributed resources. The performance trend is stable or even gets better with the growth of the number of tasks and devices, which means that the model can be not only scaled up, but also can withstand variations in resource availability and task heterogeneity. These results support the notion that CAPRF++ can smoothly scale to different operational sizes and hidden workloads, and is therefore quite applicable to real-world deployment edge-cloud systems, where dynamic and multi-modal tasks are the norm and on-the-fly retraining is not a viable option.

5.2 Practical implications

CAPRF++ model provides considerable practical utility to be implemented in real-life cloud-edge computing systems, particularly those that work in dynamic, latency-aware and sustainability-limited settings. Its predictive prewarming behavior, which pre-instantiates execution containers on demand, pre-run based on estimated task arrival rates, reduces cold-start delays by over 54 percent, enabling tasks to be immediately executed in applications with hard time budgets such as smart healthcare, self-driving cars, and industrial automation. Moreover, CAPRF++ ensures service quality in varying workload conditions that do not require any fixed provisioning or manual control, and the average task latency is only 3.9 ms with over 99% SLA satisfaction. Additionally, system operators can dynamically trade off between the latency, energy efficiency and environmental concern key in carbon target data centers or energy-conscious policies by integrating a Weighted Tradeoff Controller (WTC). The B+-Tree resource indexing allows fast scalable task-device matching with systems with thousands of highly varied resources, and makes CAPRF++ applicable to large scale deployments and in distributed environments. Moreover, generalization by task type enables real-time adaptive scheduling by reducing retraining requirements, since the system can generalize when the task type is unknown. These features are all the same, and it will reduce the cost of operations, the optimization of resources, and the performance of infrastructure resilience, therefore, making CAPRF++ a possible, deployable solution to Industry 5.0 cloud-edge systems of the future.

5.3 Comparison under homogeneous and heterogeneous workloads

To further test the flexibility and strength of CAPRF++, we compare the performance of its scheduling when using two types of workloads in homogeneous workloads, in which the task properties are relatively similar, and heterogeneous workloads, in which tasks have much differentiated properties in terms of compute intensity, deadlines, and sustainability requirements. This comparison is used to reveal the flexibil-

Table 9 Comparative scheduling failure rates (%) under homogeneous and heterogeneous workload conditions

Scheduling Method	Homogeneous Workload (%)	Heterogeneous Workload (%)
Ma et al	10.5	12.0
OTSA-PPO	9.3	11.0
TF-DDRL	8.8	10.9
Chen et al	7.5	9.8
MTGPER	6.9	8.9
ESDN	5.6	7.1
CAPRF++ (Proposed)	3.9	5.8

ity of the model to conform to real-life challenges of scheduling especially where the heterogeneity of tasks is high.

The results of the comparison Table 9 make it obvious that CAPRF++ is more flexible and can be more easily regressed to the conditions of homogeneous and heterogeneous workloads. In homogeneous conditions, when task profiles are more homogeneous and less demanding to the generalization ability of the scheduler, CAPRF++ achieves a very low rate of task violation of 3.9% and is better than all the six baseline schemes. CAPRF++ can effectively use the low-latency B+-Tree indexing and fast resource matching in this controlled environment to meet SLA demands with the minimum possible latency. But as CAPRF++ is capable of, the real strength of CAPRF++ is even greater with non-homogeneous workloads which are naturally far more demanding by the diversity and often competing demands of multi-modes tasks. Although the rates of violations in each of the compared baseline methods are increased in this complex scenario (Ma et al. reaching 12.0 per cent and OTSA-PPO reaching 11.0 per cent), CAPRF++ continues to demonstrate high performance with the lowest rate of violations 5.8 per cent. This resilient behavior is enabled by the context sensitive task encoding of the framework, weighted multi-objective tradeoff modeling, and adaptive PPO policy, which together offers the benefits of fine-tuned and efficient scheduling choices, even in the case of alternative types of tasks and a rapid change of resource requirements. These results support the claim that CAPRF++ does not only work well in stable contexts, but also performs well in real-world, highly-varying, contexts where scheduling flexibility and smart resource allocation are paramount.

6 Conclusion

This paper has presented CAPRF++, a cohesive and situation-aware scheduling system that is tailored to the increasing needs of sustainable, efficient, and generalizable task assignment in cloud-edge infrastructures. The framework starts with encoding heterogeneous task descriptors like compute demands, urgency levels, and carbon constraints into structured context-aware embeddings that improve cross-domain generalization. A B+-Tree indexing mechanism is used to match these task representations with candidate resources, which allow scalable and low-latency selection of resources in the dynamic scenario. To further restrict the decision space of the sched-

uler, a WTC dynamically builds an action space that is feasible, modeling latency, energy consumption, cold-start overhead, and carbon emissions as weighted multi-objective tradeoff space, and ensuring SLA compliance and aiding sustainability. Expanding on this, the PPO scheduler is enhanced to a CTPPO, which has a FIFO-based trajectory buffer to stabilize policy updates and enhance the sample efficiency. This improved PPO agent is trained with a specifically designed multi purpose reward function balancing the performance, efficiency and environmental objectives. A predictive prewarming module is used to predict arrival of tasks and preemptively pre-initiate execution containers to reduce cold-start delays and enhance responsiveness.

There is extensive experimental data across a variety of real-world and synthetic workloads that CAPRF++ can achieve significant performance improvements, such as a carbon footprint reduction of up to 88.8, a SLA satisfaction of 99.4, a cold-start overhead reduction of 54.1 and a generalization performance of 97.8 on unseen workloads, and with consistent high throughput and low makespan. The findings confirm that CAPRF++ is a powerful and sustainable technology to use in Industry 5.0 settings in terms of next-generation, real-time, and resource-conscious scheduling.

It must be understood that the performance evaluation of CAPRF++ includes a simulated and emulated environment that mimics real cloud-edge systems. Although the performance results are superior to those of other solutions in a variety of metrics, the performance may be impacted due to additional system-level constraints present in the real world, such as hardware diversity, network variability and operational overheads. Future work will focus on the real-world implementation and validation of the proposed architecture through integration into real cloud-edge systems to continue to investigate the pragmatic utility and strength of the architecture. At low load, predictive prewarming may add resource overhead without proportional performance gain—this demonstrates a limitation of the architecture when faced with certain types of workloads.

Going forward, the future works will concentrate on reducing the time taken to execute tasks and meeting strict low-latency requirements in ultra dynamic environments. The CAPRF++ framework will also be extended to allow testing for scalability and adaptability in more complex deployment scenarios; i.e., where the number of sensors is higher than in smart cities and is deployed across larger areas such as off-shore wind farms, autonomous maritime systems and remote oil platforms. In addition, the environment in which these systems operate has a high level of uncertainty and real-time decision making is required with heterogeneously connected devices.

Author contributions All Authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by CF, RIS. The first draft of the manuscript was written by CF and all authors commented on previous versions of the manuscript. All Authors read and approved the final manuscript. Conceptualization: CF, RIS; Methodology: CF, RIS; Formal Analysis and Investigation: CF, RIS; Writing—Original draft preparation: CF, RIS; Writing- review and editing: CF, RIS; Supervision: RIS.

Funding There is no funding for this study.

Data availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no competing interests.

Ethical approval This article does not contain any studies with human participants and/or animals performed by any of the authors.

Informed consent There is no informed consent for this study.

References

- Abualigah L, Hussein AM, Almomani MH, Zitar RA, Migdady H, Alzahrani AI, Alwadain A (2024) Improved synergistic swarm optimization algorithm to optimize task scheduling problems in cloud computing. *Sustain Comput Inform Syst* 43:101012
- Ahmed MW, Kavitha G (2025) Implementing an intelligent learning-based algorithm for efficient task scheduling in cloud computing environments. *Inf Secu J A Global Perspect*. <https://doi.org/10.1080/19393555.2025.2461558>
- Akbar S, Adve RS, Ding Z, Moo PW (2023) Transfer-based DRL for task scheduling in dynamic environments for cognitive radar. *IEEE Trans Aerosp Electron Syst* 60(1):37–50
- Alsadie D (2021) A metaheuristic framework for dynamic virtual machine allocation with optimized task scheduling in cloud data centers. *IEEE Access* 9:74218–74233
- Chen Y, Ding Y, Hu ZZ, Ren Z (2025) Geometrized task scheduling and adaptive resource allocation for large-scale edge computing in smart cities. *IEEE Internet Things J*. <https://doi.org/10.1109/JIOT.2024.3525020>
- Choi K, Soma R, Pedram M (2005) Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times. *IEEE Trans Comput Aided des Integr Circuits Syst* 24(1):18–28
- Devi DC, Uthariaraj VR (2016) Load balancing in cloud computing environment using improved weighted round robin algorithm for nonpreemptive dependent tasks. *Sci World J* 2016(1):3896065
- Gong A, Yang K, Lyu J, Li X (2024) A two-stage reinforcement learning-based approach for multi-entity task allocation. *Eng Appl Artif Intell* 136:108906
- Hameed MSA, Schwung A (2023) Graph neural networks-based scheduler for production planning problems using reinforcement learning. *J Manuf Syst* 69:91–102
<https://www.kaggle.com/datasets/ziya07/cloud-task-scheduling-dataset>
- Iftikhar S, Ahmad MMM, Tuli S, Chowdhury D, Xu M, Gill SS, Uhlig S (2023) HunterPlus: AI based energy-efficient task scheduling for cloud–fog computing environments. *Internet Things* 21:100667
- Jayasena KPN, Li L, Xie Q (2017) Multi-modal multimedia big data analyzing architecture and resource allocation on cloud platform. *Neurocomputing* 253:135–143
- Kang K, Ding D, Xie H, Yin Q, Zeng J (2021) Adaptive DRL-based task scheduling for energy-efficient cloud computing. *IEEE Trans Netw Serv Manag* 19(4):4948–4961
- Koslovski GP, Pereira K, Albuquerque PR (2024) Dag-based workflows scheduling using actor–critic deep reinforcement learning. *Futur Gener Comput Syst* 150:354–363
- Kumar H (2023) A new hybrid particle swarm optimization algorithm for optimal tasks scheduling in distributed computing system. *Intell Syst Appl* 18:200219
- Liu H, Chen P, Ouyang X, Gao H, Yan B, Grosso P, Zhao Z (2023) Robustness challenges in reinforcement learning based time-critical cloud resource scheduling: A meta-learning based solution. *Futur Gener Comput Syst* 146:18–33
- Liu H, Tian L, Guo M (2025) ESDN: edge computing task scheduling strategy based on dilated convolutional neural network and quasi-newton algorithm. *Cluster Comput* 28(3):151
- Ma Q, Qin Y, Zhu C, Gao L, Chen X (2025) Joint resource trading and task scheduling in edge-cloud computing networks. *IEEE Trans Netw*. <https://doi.org/10.1109/TON.2024.3516381>
- Mangalampalli S, Karri GR, Kumar M, Khalaf OI, Romero CAT, Sahib GA (2024) DRLBTSA: Deep reinforcement learning based task-scheduling algorithm in cloud computing. *Multimedia Tools Appl* 83(3):8359–8387

- Niu L, Chen X, Zhang N, Zhu Y, Yin R, Wu C, Cao Y (2023) Multiagent meta-reinforcement learning for optimized task scheduling in heterogeneous edge computing systems. *IEEE Internet Things J* 10(12):10519–10531
- Pabitha P, Nivitha K, Gunavathi C, Panjavarnam B (2024) A chameleon and remora search optimization algorithm for handling task scheduling uncertainty problem in cloud computing. *Sustain Comput Inform Syst* 41:100944
- Pachipala Y, Sureddy KS, Kaitepalli AS, Pagadala N, Nalabothu SS, Iniganti M (2024) Optimizing task scheduling in cloud computing: an enhanced shortest job first algorithm. *Procedia Computer Science* 233:604–613
- Raeisi-Varzaneh M, Dakkak O, Fazea Y, Kaosar MG (2024) Advanced cost-aware Max–Min workflow tasks allocation and scheduling in cloud computing systems. *Clust Comput* 27(9):13407–13419
- Saberikia M, Farbeh H, Fazeli M (2025) Improving energy efficiency and fault tolerance of mission-critical cloud task scheduling: A mixed-integer linear programming approach. *Sustain Comput Inform Syst* 45:101068
- Sahoo AR, Chakraborty P (2022) A study on position control of a continuum arm using MAML (model-agnostic meta-learning) for adapting different loading conditions. *IEEE Access* 10:14980–14992
- Salari-Hamzehkhani B, Akbari M, Safi-Esfahani F (2025) Introducing an improved deep reinforcement learning algorithm for task scheduling in cloud computing. *J Supercomput* 81(1):1–20
- Seth S, Singh N (2019) Dynamic heterogeneous shortest job first (DHSJF): a task scheduling approach for heterogeneous cloud computing systems. *Int J Inf Technol* 11(4):653–657
- Song W, Chen X, Li Q, Cao Z (2023) Flexible job-shop scheduling via graph neural network and deep reinforcement learning. *IEEE Trans Ind Inf* 19(2):1600–1610
- Sreenivasulu G, Paramasivam I (2021) Hybrid optimization algorithm for task scheduling and virtual machine allocation in cloud computing. *Evol Intel* 14(2):1015–1022
- The Grid Workloads Gwa-t-12 Bitbrains. Available online: <http://gwa.ewi.tudelft.nl/datasets>.
- Wang Z, Goudarzi M, Buyya R (2025) TF-DDRL: a transformer-enhanced distributed DRL technique for scheduling IoT applications in edge and cloud computing environments. *IEEE Trans Serv Comput*. <https://doi.org/10.1109/TSC.2025.3528346>
- Xu H, Chen M, Huang L, Vucetic S, Yin W (2024) X-shot: a unified system to handle frequent, few-shot and zero-shot learning simultaneously in classification. *arXiv preprint arXiv:2403.03863*.
- Yang L, Zhong C, Yang Q, Zou W, Fathalla A (2020) Task offloading for directed acyclic graph applications based on edge computing in industrial internet. *Inf Sci* 540:51–68
- Zhang C, Yang J (2025) Multi-Tree Genetic Programming with Elite Recombination for dynamic task scheduling of satellite edge computing. *Futur Gener Comput Syst* 166:107700
- Zhang L, Yu C, Wong TN (2021) A graph-based constraint programming approach for the integrated process planning and scheduling problem. *Comput Oper Res* 131:105282
- Zou W, Zhang Z, Wang N, Tian Y, Tian L (2025) ReflexPilot: startup-aware dependent task scheduling based on deep reinforcement learning for edge-cloud collaborative computing. *IEEE Transactions on Cloud Computing*. <https://doi.org/10.1109/TCC.2025.3555231>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.



Felsy C Felsy C received her B.E. degree in Computer Science and Engineering from Anna University in 2007 and her M.E. degree in Computer Science and Engineering from Manonmaniam Sundaranar University in 2011. She is currently serving as the Head of the Department of Artificial Intelligence and Data Science at Ponjesly College of Engineering and is pursuing her Ph.D. degree at Anna University. Her research interests include Cloud Computing, Machine Learning, Artificial Intelligence, and Data Analytics. She is actively involved in teaching, research, and academic administration, contributing to the advancement of emerging technologies in the field of Artificial Intelligence.



Isaac Sajan R Isaac Sajan R received his B.E. and M.E. degrees in Computer Science and Engineering in 2006 and 2008, respectively, and earned his Ph.D. in Information and Communication Engineering from Anna University in 2021. He is currently serving as Professor in the Department of Computer Science and Engineering and Vice Principal at Ponjesly College of Engineering. His research interests include Wireless Sensor Networks, Cloud Computing, Image Processing, Artificial Intelligence, and Machine Learning. He has published numerous research papers in reputed international and national conferences and peer-reviewed indexed journals. He also serves as a reviewer for various national and international journals and actively contributes to the advancement of research and innovation in engineering and technology. He is a Life Member of

the Indian Society for Technical Education (ISTE) and a Member of the International Association of Engineers (IAENG), Hong Kong.

Authors and Affiliations

C. Felsy¹  · R. Isaac Sajan¹ 

✉ C. Felsy
felsyc395@gmail.com

R. Isaac Sajan
isaacsajanr.001@gmail.com

¹ Ponjesly College of Engineering, Nagercoil, India