PICAB: A Permutation-Invariant Contextual Attention Bandit for Energy-Constrained Edge AI

Sayed Saminur Rahman[1], Victor Balogun[1], Yaser Al Mtawa[1,*]
[1] The University of Winnipeg

Abstract

The deployment of Deep Neural Networks (DNNs) on resource-constrained edge devices presents a fundamental challenge: high-accuracy models often exceed the compute and energy budgets of local hardware, while full cloud offloading incurs unpredictable network latency. Inference splitting has emerged as a promising solution to this trade-off, enabling a DNN to be partitioned layer-wise across the edge-cloud continuum. However, optimizing these split decisions is non-trivial; the action space comprising valid cut points and available target nodes fluctuates dynamically with each request, rendering standard fixed-output Reinforcement Learning (RL) architectures ineffective. In this paper, we propose a Permutation-Invariant Contextual Attention Bandit (PICAB), a lightweight deep learning framework designed for real-time DNN partitioning. Our architecture employs a Multi-Head Attention mechanism to encode variable-sized sets of candidate execution plans, allowing the agent to generalize across diverse network environments without retraining. By incorporating the target node's battery state into the attention mechanism, the agent learns energy-aware and robust offloading decisions. We evaluate our approach using heterogeneous workloads with periodic IoT inference streams. Experimental results demonstrate that our algorithm achieves a 23% reduction in makespan compared to the metaheuristic baselines while maintaining similar Energy-Delay Product (EDP), effectively balancing the trade-off between inference latency and energy sustainability.

Keywords: Split Computing, Contextual Multi-Armed Bandits, Deep Neural Networks, Energy Efficiency, AIoT, DNN Partitioning

1. Introduction

The convergence of Artificial Intelligence and the Internet of Things (IoT) has catalyzed the paradigm of Artificial Intelligence of Things (AIoT) [1]. In this emerging ecosystem, deep learning capabilities are no longer confined to centralized datacenters but are distributed across a continuum of sensors, edge gateways, and cloud servers. While deploying Deep Neural Networks (DNNs) on edge devices reduces transmission latency and enhances privacy, the computational demands of modern architectures frequently exceed the energy and thermal budgets of resource-constrained IoT hardware. To resolve this issue, inference splitting, also known as split computing has emerged as a fundamental computing architecture for AIoT [2].

However, the efficacy of inference splitting relies entirely on the intelligence of the partitioning decision. Seminal frameworks such as Neurosurgeon [3] demonstrated that partitioning DNNs at the layer granularity can reduce end-to-end latency by over 3× compared to cloud-only execution. Yet, these early systems typically rely on offline profiling and static regression models. They assume a stable environment where device capabilities and bandwidth remain constant. In realistic Mobile Edge Computing (MEC) scenarios, this assumption collapses. Wireless channels exhibit highly non-stationary variance [4], and device batteries deplete unevenly, altering the cluster's topology in real-time.

A critical gap in existing literature is the lack of schedulers that can adapt to this volatility without incurring overhead. While Deep Reinforcement Learning (DRL) offers a pathway to

*y.almtawa@uwinnipeg.ca

dynamic adaptation, standard architectures (e.g., fixed-input MLPs) struggle to generalize in edge environments where the number of available nodes fluctuates due to mobility or battery exhaustion. Furthermore, existing granular approaches like horizontal partitioning [5] often introduce synchronization overheads that are detrimental in high-latency IoT networks. Consequently, layer-wise (vertical) partitioning remains the most communication-efficient strategy, provided the scheduler can robustly identify the optimal cut under uncertainty.

To address these challenges, we propose the **P**ermutation-**I**nvariant **C**ontextual **A**ttention **B**andit (**PICAB**), a lightweight decision framework designed for dynamic DNN partitioning. Unlike traditional heuristics that optimize for an instantaneous snapshot, our approach utilizes a Multi-Head Attention mechanism to encode the variable states of the edge cluster including battery health and queue depth into a permutation-invariant representation. This allows the agent to handle dynamic topologies where devices join or leave the network, learning a robust policy that balances inference latency with long-term energy sustainability. Our key contributions are:

- **Architecture-Agnostic Policy:** We introduce a permutation-invariant attention mechanism that encodes the DNN computational graph and the edge cluster state into a fixed-size representation. This enables a single policy to generalize across a range of DNN architectures and heterogeneous edge-cloud deployments without retraining.
- **Communication-Aware Layer Splitting:** To address the high variability of edge–cloud links, we restrict the action space to vertical (layer-wise) partitioning and incorporate communication stability into the reward formulation. The agent is penalized for selecting split points that depend on highly variable or unreliable links, encouraging robust partitioning decisions under fluctuating network conditions.
- **Energy Awareness:** We incorporate battery state into the bandit context, allowing the agent to account for the energy availability of candidate execution nodes. This biases offloading decisions away from energy-constrained devices, reducing premature battery depletion and extending the overall operational lifetime of the IoT network.

2. Related Work

2.1. Edge-Cloud Inference Splitting

As surveyed by Siam et al. [1], AIoT computing architectures are transitioning from centralized cloud processing to collaborative edge-cloud hierarchies. The foundational work in this domain, Neurosurgeon [3], introduced a lightweight scheduler that automatically identifies optimal partition points by profiling layer-wise execution times. It treats the DNN as a directed graph, evaluating every possible cut point to minimize either latency or energy. While effective for single-user scenarios, Neurosurgeon relies on pre-built performance prediction models that assume predictable hardware behavior.

More recently, SPLIT [6] demonstrated that dividing DNNs into evenly-sized computational blocks can reduce jitter in multi-tenant edge environments by enabling fine-grained task preemption. However, SPLIT relies on offline profiling to generate static partition plans. To mitigate transmission overheads, frameworks like COMSPLIT [4] integrate early-exit strategies with model splitting. While effective, early-exit mechanisms require modifying the DNN architecture and re-calibrating loss functions. In contrast, our approach focuses on pure partitioning of standard architectures, making it applicable to legacy models without invasive changes.

The scope of splitting has also expanded to new architectures. ED-ViT [7] proposed partitioning Vision Transformers across multiple edge devices by creating class-specific submodels. Furthermore, hardware-centric approaches such as Splitwise [8] have proposed phase splitting for Generative AI, separating prompt processing from token generation.

2.2. **Distributed Execution Paradigms**

Beyond vertical layer splitting, researchers have explored alternative execution paradigms. Rodriguez-Conde et al. [5] and Samikwa et al. [9] explored horizontal partitioning, decomposing individual layers across multiple devices to exploit fine-grained parallelism. While effective for localized mesh networks, these methods require frequent synchronization of partial results, making them highly sensitive to inter-node latency characteristic of edge-cloud links. Focusing on execution flow, Mubark et al. [10] proposed ASAP, an asynchronous framework that pipelines inference requests. Similarly, Feltin et al. [11] utilized pipeline parallelism to maximize aggregate throughput. While effective for batch processing, these pipeline approaches often degrade single-request latency. Finally, the utility of splitting extends to privacy; Mai et al. [12] introduced a framework (Split-and-Denoise) for LLMs that partitions models after the embedding layer to inject differential privacy noise.

2.3. **Reinforcement Learning & Optimization**

In the context of 5G-enabled IoT, Karjee et al. [13] proposed a split computing framework integrated with load balancing mechanisms. While effective for distributing aggregate traffic, their approach relies on pre-defined network interaction models. Our Bandit scheduler supersedes this by learning the congestion cost of each node directly from experience, effectively performing implicit load balancing.

Lyu et al. [14] proposed an objective-driven differentiable optimization framework for Split AI. While theoretically superior as it guarantees optimality, embedding optimization solvers within the training loop incurs high computational overhead. In the domain of Generative AI, Chen et al. [15] proposed an adaptive splitting framework for LLMs using Model-Based RL (MBRL). However, MBRL frameworks require iterative planning, bottlenecking real-time inference. Our proposed algorithm circumvents this by directly mapping variable system states to partition decisions in a single forward pass.

Most closely related to our work is SplitPlace [16], which utilizes a bandit-based policy to switch between layer-wise and semantic splitting strategies. However, SplitPlace decouples the partitioning decision from physical node assignment. Our algorithm advances this by formulating a joint action space where each arm represents a specific (Cut, Node) tuple. Alternatively, Jung and Lee [17] proposed a graph-based optimization framework solving partitioning via shortest-path routing. While providing optimal solutions for static snapshots, such deterministic methods lack the exploration capability required to discover performant nodes in partially observable environments which we have considered in our approach.

3. **System Model**

3.1. **Network Topology and Device Heterogeneity**

We model the edge-cloud continuum as a directed graph $\mathcal{G}(t) = (\mathcal{V}(t), \mathcal{E}(t))$, where $\mathcal{V}(t)$ represents the set of available compute nodes at time t and $\mathcal{E}(t)$ represents the set of communication links at time t. The node set consists of a local edge device v_0 (the source of inference requests) and a dynamic set of candidate offloading targets $\mathcal{V}_{\text{cloud}} = \{v_1, \ldots, v_N\}$.

Unlike static topologies assumed in prior works, our model captures the dynamic nature of network nodes; strictly, $|\mathcal{V}(t)|$ is time-varying as devices may join, leave, or deplete their battery reserves. Each node $v_i \in \mathcal{V}$ is characterized by a tuple $\Omega_i = \langle f_i, P_i^{\text{act}}, P_i^{\text{idle}}, E_i^{\text{cap}}, \phi_i(t) \rangle$, where f_i denotes the effective floating-point operations per second (FLOPs), P_i^{act} and P_i^{idle} represent the power consumption in Watts for the active and idle states respectively, E_i^{cap} is the total battery capacity (Joules), and $\phi_i(t)$ represents the instantaneous battery health.

3.2. **DNN Profiling and Partitioning**

We treat a DNN inference task as a sequential Directed Acyclic Graph (DAG) of L layers. Following standard profiling methodology [18], each layer $l \in \{1, \dots, L\}$ is characterized by its computational workload w_l (FLOPs) and output tensor size δ_l (Bytes). A partitioning decision is a tuple $a_t = (k, v_d)$, where $k \in \{0, \dots, L\}$ is the cut index (last layer executed locally) and $v_d \in \mathcal{V}(t)$ is the target node for the remaining layers.

3.3. **Communication Model**

To accurately capture the bursty nature of IoT traffic, we implement a rigorous TCP formulation that accounts for the Round-Trip Time (RTT) dominance in small tensor transfers. The transmission time T_{tx} for a tensor of size δ_k to node v_d is modeled as:

$$T_{\text{tx}} = \gamma \cdot \text{RTT} + \begin{cases} \lceil \log_2(\frac{\delta_k}{\text{IW} \cdot \text{MSS}} + 1) \rceil \cdot \text{RTT} & \text{if } \delta_k \leq D_{\text{ramp}} \\ T_{\text{ramp}} + \frac{\delta_k - D_{\text{ramp}}}{B(t)} & \text{otherwise} \end{cases} \tag{3.1}$$

where γ represents the handshake overhead (in RTTs), IW is the Initial Congestion Window (packets), and MSS is the Maximum Segment Size (bytes). In the saturation case (second condition), D_{ramp} denotes the data capacity of the exponential growth phase, T_{ramp} is the time duration of that phase, and $B(t)$ represents the instantaneous channel bandwidth available. The first case captures the latency of transfers fitting entirely within the slow-start window.

Furthermore, to model Medium Contention in shared wireless channels, we apply a non-linear congestion penalty. We define the link utilization $U(t) = \frac{\Lambda(t)}{B_{\text{cap}}}$, where $\Lambda(t)$ is the current traffic load and B_{cap} is the maximum link capacity. The effective transfer time $T_{\text{effective}}$ is scaled by a factor $\beta(t)$:

$$\beta(t) = \max(1.0, 2^{(U(t)-1)}) \quad \Longrightarrow \quad T_{\text{effective}} = T_{\text{tx}} \cdot \beta(t) \tag{3.2}$$

3.4. **Stochastic Computation and Battery Dynamics**

Real-world edge execution is non-deterministic due to thermal throttling and operating system background processes. We model the computation time T_{cmp} for a workload W (FLOPs) on node v_i as a random variable following a Log-Normal distribution:

$$T_{\text{cmp}} \sim \text{LogNormal}\left(\ln(W/f_i) - \frac{\sigma^2}{2}, \sigma^2\right) \tag{3.3}$$

where f_i is the node's compute capability (FLOPs/sec) and σ captures the execution variance. Finally, the system tracks the cumulative energy depletion to update the battery state $\phi_i(t)$:

$$\phi_i(t) = 1 - \frac{1}{E_i^{\text{cap}}} \sum_{\tau=0}^{t} \left(P_i^{\text{act}} \Delta\tau_{\text{busy}} + P_i^{\text{idle}} \Delta\tau_{\text{wait}} \right) \tag{3.4}$$

where E_i^{cap} is the total battery capacity, P_i^{act} and P_i^{idle} are the active and idle power coefficients, and $\Delta\tau_{\text{busy}}$ and $\Delta\tau_{\text{wait}}$ represent the duration the node spends in active and idle states, respectively. A node is removed from $\mathcal{V}(t)$ if $\phi_i(t) \leq 0$.

4. Methodology

We propose the Permutation-Invariant Contextual Attention Bandit (PICAB), a framework designed to handle the non-stationary action spaces characteristic of mobile edge networks where the number of available nodes fluctuates dynamically.

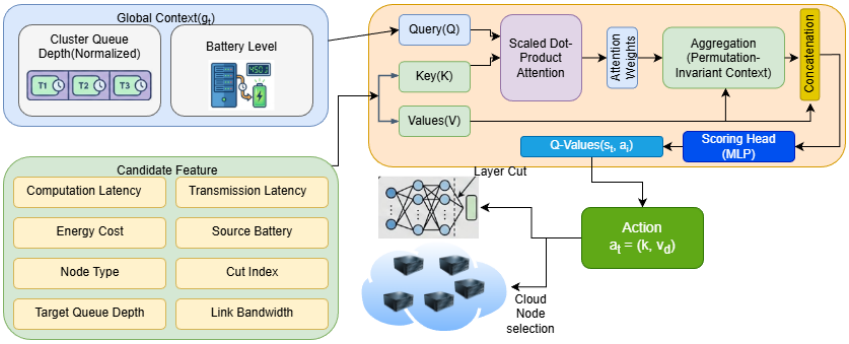


Figure 1. Architecture diagram of PICAB

4.1. Problem Formulation

We formulate the partitioning problem as a Contextual Multi-Armed Bandit (CMAB). At each discrete time step t, the agent observes a system state $\mathcal{S}_t$ and selects an action a_t from a variable set of candidates $\mathcal{A}_t$ to minimize a cumulative cost. Unlike standard Deep Q-Networks (DQN) which assume a fixed action space size $|\mathcal{A}|$, our candidate set size $|\mathcal{A}_t|$ varies as devices deplete batteries or leave the network, necessitating a flexible architecture.

4.2. Permutation-Invariant Architecture

To address the variable action space, we utilize a Permutation-Invariant neural architecture based on the Scaled Dot-Product Attention mechanism.

State Representation: The system state $\mathcal{S}_t = (g_t, \mathcal{X}_t)$ consists of two components:

(1) Global Context ($g_t \in \mathbb{R}^{d_g}$): A vector summarizing cluster-wide metrics, specifically the aggregate queue depth normalized by capacity.

(2) Candidate Features ($\mathcal{X}_t$): A set of feature vectors $\{x_{t,1}, \ldots, x_{t,N}\}$, where N is the current number of valid partition plans. Each vector $x_{t,i} \in \mathbb{R}^{d_c}$ corresponds to a specific (Cut, Node) candidate:

$$x_{t,i} = [\tilde{T}_{\text{cmp}}, \tilde{T}_{\text{tx}}, \tilde{E}_J, \mathbb{I}_{\text{cloud}}, \tilde{k}_{\text{cut}}, \phi_{\text{src}}, \tilde{Q}_{\text{tgt}}, \tilde{B}_w] \tag{4.1}$$

where $\tilde{T}_{\text{cmp}}$ and $\tilde{T}_{\text{tx}}$ are the normalized estimated computation and transmission latencies, $\tilde{E}_J$ is the energy cost (Joules), $\mathbb{I}_{\text{cloud}}$ is a binary indicator for cloud nodes, $\tilde{k}_{\text{cut}}$ is the normalized cut layer index, and ϕ_{src} is the source device's battery health. $\tilde{Q}_{\text{tgt}}$ and $\tilde{B}_w$ represent the target node's queue utilization and link bandwidth, respectively.

Feature Projection: We first project the global context and candidate features into a shared latent space of dimension d_{model} using learnable linear transformations:

$$h_g = \sigma(W_g g_t + b_g), \quad h_{c,i} = \sigma(W_c x_{t,i} + b_c) \tag{4.2}$$

where W_g, W_c are the weight matrices, b_g, b_c are the bias vectors, and $\sigma(\cdot)$ denotes the ReLU activation function.

Attention Mechanism: To capture the inter-dependencies between the global state and specific candidates, we employ Multi-Head Attention. The global embedding h_g serves as the *Query* (Q), while the candidate embeddings $\{h_{c,i}\}$ serve as both *Keys* (K) and *Values* (V):

$$C_t = \text{MHA}(Q = h_g, K = \{h_{c,i}\}, V = \{h_{c,i}\}) \tag{4.3}$$

Here, C_t is the context vector generated by the attention mechanism, representing the weighted relevance of the available resource pool to the current workload demand.

Late Fusion and Scoring: To estimate the Q-value, $Q(s_t, a_i)$ for a specific candidate i, we concatenate the global context C_t with the candidate's local embedding $h_{c,i}$ (Late Fusion) and pass it through a final regression MLP, f_{out}:

$$z_i = [C_t \parallel h_{c,i}] \quad \Longrightarrow \quad Q(s_t, a_i) = f_{\text{out}}(z_i) \tag{4.4}$$

where $\parallel$ denotes the concatenation operator. This structure ensures the policy is permutation-invariant: the order of candidates in $\mathcal{X}_t$ does not affect the output values, allowing the model to generalize to any cluster size N.

4.3. **Hybrid Reward Function**

To balance latency performance with device survival, we define a composite reward function R_t. The agent maximizes:

$$R_t = -\alpha \tilde{T}_{\text{total}} - \lambda \left(e^{\gamma(1-\phi_{\text{src}})} - 1 \right) - \mathcal{P}_{\text{cong}} \tag{4.5}$$

where $\tilde{T}_{\text{total}}$ is the end-to-end latency and α is a scaling coefficient. The second term imposes a soft survival penalty: $\phi_{\text{src}} \in [0, 1]$ is the source battery fraction, λ controls the penalty magnitude, and γ defines the curvature. This exponential formulation exerts negligible influence when $\phi_{\text{src}} \approx 1$ but dominates the reward as $\phi_{\text{src}} \to 0$, forcing the agent to prioritize offloading when critical. $\mathcal{P}_{\text{cong}}$ is a discrete penalty applied if the request incurs significant queuing delay.

5. **Experimental Evaluation**

To validate the robustness of PICAB, we conducted strictly controlled simulations using high-fidelity synthetic datasets. While public cluster traces (e.g., Alibaba, Google) provide valuable datacenter insights, they lack bursty IOT sensor data required to evaluate split computing in mobile edge environments. For that, we have developed a custom workload traces and published them in a public repository [19]. We utilize real-world profiling data from standard DNN architectures, specifically ResNet-50 (6 blocks) and MobileNetV3 (6 blocks). Layer-wise computational costs (w_l) and activation sizes (δ_l) were extracted from a Raspberry Pi 4 (Edge) and NVIDIA T4 GPU (Cloud). To simulate IoT sensor streams, inference requests are generated via a Poisson Process with arrival rates $\lambda \in [20, 30]$ requests/sec (RPS) distributed across a dynamic cluster of size $N \in [10, 100]$ nodes. To model a realistic battery-constrained IoT cluster, we simulate heterogeneous nodes with compute capabilities $f_i \in \{0.10, 0.18, 0.25\}$ TFLOPS and varying battery capacities of 15 kJ (40% of nodes), 5 kJ (30% of nodes), and 2 kJ (30% of nodes).

5.1. **Simulation Environment**

We evaluate the proposed PICAB against three classes of schedulers:

(1) Stochastic Meta-Heuristics: We select two state-of-the-art metaheuristic algorithms for comparison: the Hybrid Genetic Algorithm/Grey Wolf Optimizer (GA-GWO) [20]

and the Logarithm-Decreasing Particle Swarm Optimizer (Log-PSO) [21]. GA-GWO is included for its proven capability to reduce makespan by 19–21% in cloud environments, while Log-PSO is chosen for its adaptive inertia weight strategy, which has been shown to outperform standard swarm algorithms in dynamic task mapping.

(2) Deep Reinforcement Learning (DRL-MLP): A standard Multi-Layer Perceptron (MLP) policy with fixed input dimensions serves as the learning-based baseline [22]. This agent utilizes a policy-gradient strategy to directly optimize the joint objective (latency and energy) in an online manner, representing traditional DRL approaches that lack permutation invariance.

(3) Heuristic Baseline: A First-Fit scheduler processes requests in First-Come-First-Served (FCFS) order. It greedily selects the candidate node that maximizes the immediate reward - providing a baseline for varying complexity.

5.2. **Implementation and Hyperparameters**

The policy network π_θ was implemented in PyTorch and trained online using the Adam optimizer with a learning rate of $\eta = 1 \times 10^{-4}$. To ensure stable convergence, we utilized a Replay Buffer of capacity $10,000$ and performed mini-batch updates of size $B = 32$ every 20 decision steps as shown in table 1.

The Permutation-Invariant architecture consists of $H = 4$ attention heads with a hidden embedding dimension of $d_{model} = 128$. The scoring MLP comprises two linear layers with ReLU activation and a hidden dimension of 128. During training, we employed an ϵ-greedy exploration strategy where ϵ decayed linearly from 0.1 to 0.05 over the first $2,000$ steps, balancing exploration with exploitation. As this is a Contextual Bandit formulation, the agent optimizes for the immediate reward r_t.

Table 1. Hyperparameter Settings

Parameter	**Value**
Embedding Dimension (d_{model})	128
Attention Heads (H)	4
Optimizer	Adam
Learning Rate (η)	1×10^{-4}
Mini-Batch Size	32
Replay Buffer Size	10,000
Exploration ($\epsilon_{start} \rightarrow \epsilon_{end}$)	$0.1 \rightarrow 0.05$
Epsilon Decay Horizon	2,000 Steps
Training Frequency	Every 20 Steps

5.3. **Evaluation Metrics**

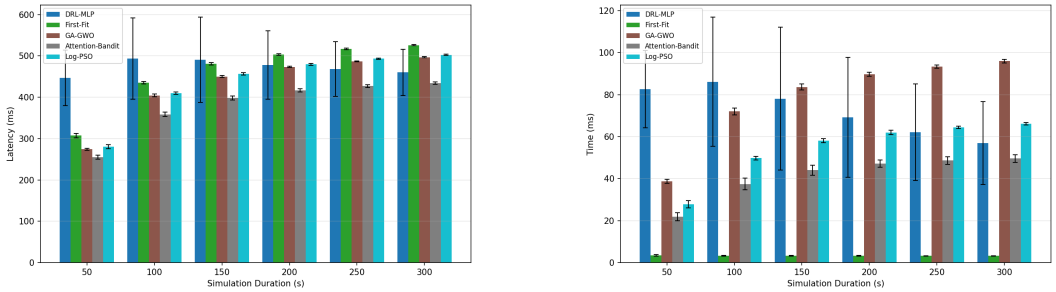
To comprehensively evaluate the proposed PICAB framework, we utilize five key metrics: (1) Total Makespan, defined as the time required to complete the entire batch of inference requests, serving as a proxy for cluster-wide efficiency; (2) Average Latency, the mean end-to-end time comprising computation, transmission, and queueing delays; (3) SLA Failure Rate, the percentage of requests exceeding the strict deadline of $\delta =$ 500 ms; (4) Effective Goodput, a measure of usable throughput counting only successful inferences per second; and (5) Energy-Delay Product (EDP) [23], a composite metric calculated as Latency $\times$ Energy to assess the trade-off between speed and battery conservation.

Statistical Significance: All experiments were repeated 10 times under identical network configurations and traffic traces. This methodology isolates the internal stochasticity of the

algorithms from environmental variance. In the subsequent figures, data points represent the mean value across these runs, and error bars denote the standard deviation ($\pm 1\sigma$) of the algorithmic performance.

5.4. **Simulation Stability over Long Horizon**

The first experiment evaluates system stability over simulation durations increasing from 50s to 300s (averaged over 10 runs).



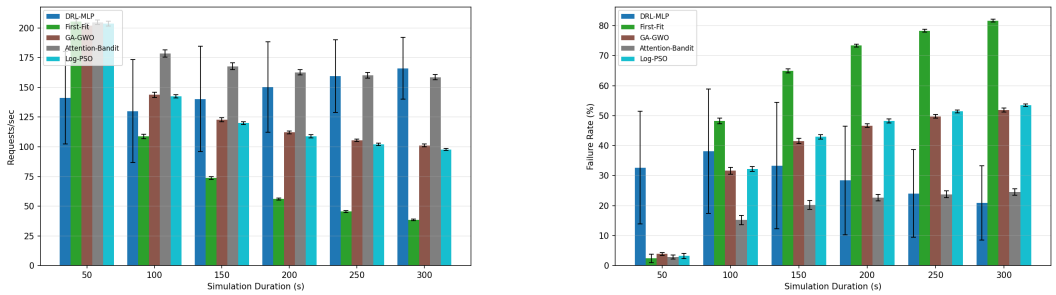
(a) Average Latency vs Duration

(b) Time spent in queue vs Duration

Figure 2. Long Horizon Performance Analysis.

As observed in Figure 2a, both GA-GWO and Log-PSO maintain reasonable latency levels (∼490ms at 300s), but they exhibit a distinct upward slope compared to the PICAB, which stabilizes at ∼430ms. This performance gap originates from the snapshot optimization nature of GA and PSO. These algorithms optimize the current batch in isolation and fail to account for inter-batch state dynamics, such as the gradual saturation of TCP congestion windows. This is explicitly visualized in Figure 2b, where the baselines exhibit a gradual increase in queuing delay as the simulation progresses, whereas PICAB maintains a stable queue profile as the RL agent learns to penalize actions that lead to future latency spikes.

The DRL-MLP displays significant variance (large error bars in Figure 2a), even though as the time progresses its mean average latency improves. This confirms the hypothesis that fixed-input architectures struggle with dynamic topologies. When nodes deplete batteries and are removed from the action space, the MLP perceives zeroed inputs as valid features, leading to erratic Q-value estimation. Our permutation-invariant attention mechanism effectively masks unavailable nodes, resulting in the tightest confidence intervals among all learning-based methods.



(a) Goodput vs Duration

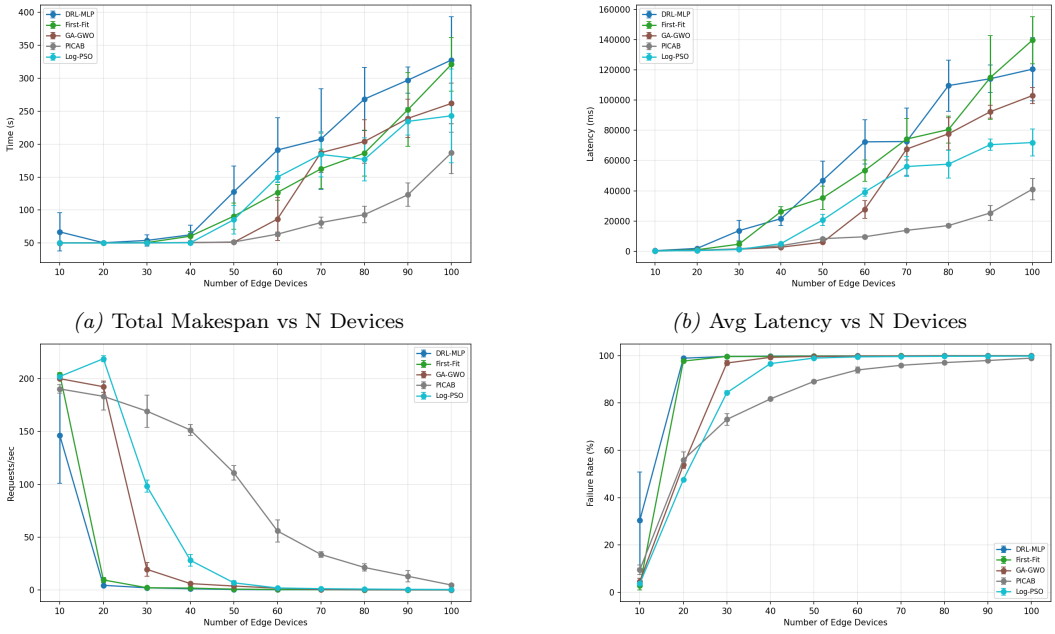
(b) SLA Violation Rate vs Duration

Figure 3. Goodput and SLA violation over longer time period

At $t = 300s$, Log-PSO and GA-GWO suffer failure rates of ∼53% and ∼51% respectively (Figure 3b). Despite their sophisticated search capabilities, these algorithms are energy-blind in the temporal sense; they greedily select the highest-performance node in the current turn, accelerating the depletion of edge nodes. The PICAB, conversely, maintains a failure rate of only ∼24%. By integrating node health directly into the attention context, the agent learns a conservation policy, distributing heavy workloads to robust nodes while shielding energy-critical devices. Consequently, the Bandit achieves the highest sustained Goodput as shown in Figure 3a, stabilizing at ∼160 req/s while the baselines degrade significantly. The stochastic baselines drop to ∼100 req/s because their optimal decisions often target nodes that are moments away from failure, triggering SLA violations.

5.5. **Scalability and Congestion Resilience**

A second experiment evaluated the PICAB's ability to maintain system stability as the cluster size N scales from 10 to 100 heterogeneous devices. This specifically tests the dimensionality problem inherent in standard RL approaches.



(a) Total Makespan vs N Devices

(b) Avg Latency vs N Devices

(c) Goodput vs N Devices

(d) SLA Violation Rate vs N Devices

Figure 4. Scalability Performance Analysis.

As shown in figure 4c, PICAB exhibits superior resilience to saturation. Traditional heuristics and DRL-MLP suffer a catastrophic collapse in goodput as the network size exceeds $N = 20$. Specifically, at $N = 30$, First-Fit and DRL-MLP drop to nearly 0 effective requests/sec due to massive queue overflows. In contrast, our PICAB sustains a high goodput of ≈ 150 req/sec at $N = 40$ and maintains serviceable throughput up to $N = 60$. Figure 4b graph reveals that baseline latency grows exponentially with cluster size. At $N = 80$, the DRL-MLP and GA-GWO baselines incur average latencies exceeding 80,000 ms (80s), rendering the system unusable for real-time inference. The PICAB maintains an average latency of $< 20,000$ ms at the same load point ($N = 80$), a 4x reduction in delay. This directly correlates to the SLA Violation Rate(Figure 4d), where our method maintains a

lower violation curve, delaying the 90% failure threshold until the cluster is fully saturated at $N = 60$, whereas baselines hit this threshold immediately at $N = 20$.

The Total Makespan result in Figure 4a confirm that better routing leads to faster batch completion. At the extreme scale of $N = 100$, the PICAB finishes the workload in ≈ 186 seconds, whereas the Log-PSO requires > 243 seconds - a 23% improvement in operational speed.

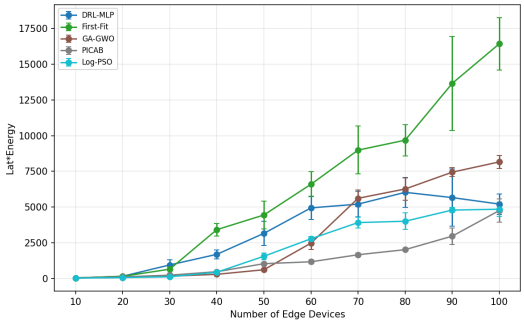


Figure 5. EDP vs N Devices

We analyze the EDP, a composite metric where lower values indicate a better balance between speed and energy consumption. As seen in Figure 5, the PICAB maintains a significantly flatter curve than First-Fit or GA-GWO. Till $N = 50$, GA-GWO has better EDP, however after that, PICAB continue to perform better. At $N = 90$, the EDP for GA-GWO spikes to $> 7,500$, the PICAB keeps the EDP below 3,000 at the same point. This validates that the candidate features ($x_{t,i}$), which include energy cost and battery state, are effectively utilized by the scoring head to penalize energy-expensive nodes even under high load.

6. **Discussion**

Meta-heuristics, particularly Log-PSO and GA-GWO, demonstrated remarkable efficiency in less saturated network ($N \leq 30$). As shown in the Figure 4c, Log-PSO effectively matches and occasionally surpasses the throughput of our PICAB when resources are abundant.

However, their performance degrades strictly as a function of time and scale. Because these algorithms optimize for an instantaneous fitness snapshot, they are energy-myopic. They aggressively target high-performance nodes without accounting for the future cost of battery depletion, leading to the rapid exhaustion of anchor nodes and the massive EDP spike observed in Figure 5 at $N = 90$.

The performance of the standard DRL-MLP illustrates the dimensionality problem in dynamic environments. While DRL-MLP demonstrated its capacity to learn temporal dependencies, it suffers from high variance. This instability arises because fixed-input MLPs must rely on zero-padding to handle variable cluster sizes. As the network scales to $N = 100$, the sparsity of the input vector increases, making it difficult for the gradient updates to credit specific actions correctly. This confirms that the architectural constraint of fixed inputs is a fundamental limiter for scalability.

The PICAB outperforms the advanced baselines by maintaining a consistent policy as the state space explodes. By treating the cluster as a permutation-invariant set, the attention mechanism filters out the noise of scaling, focusing on quality density rather than specific device IDs. This enables it to sustain a Goodput of ≈ 30 req/s at $N = 70$, a point where both Log-PSO and DRL-MLP have effectively collapsed.

A critical distinction between typical cloud scheduling and our edge partitioning scenario is task independence. In cloud datacenters, tasks are often treated as isolated events assigned to fungible resources. In the edge continuum, however, tasks are deeply intertwined with the shared device physics and network medium. A single high-bandwidth partition decision saturates the shared wireless channel, creating a pattern of interference that degrades QoS for all subsequent requests in the batch. Standard heuristics treat requests sequentially, missing these coupled dynamics. The PICAB, through its global context aggregation, effectively uncovers these latent patterns. It learns that a busy state is not just a node property but a cluster-wide constraint, allowing it to throttle back greedy offloading actions that would otherwise cause cascading network contention.

7. Conclusion

In this work, we addressed the scalability and energy-resilience challenges of DNN partitioning in dynamic MEC environments. We identified that while existing meta-heuristics like Log-PSO are highly effective for static load balancing, they lack the temporal foresight to manage intertwined resource dependencies. Similarly, standard DRL architectures struggle to generalize across variable-sized networks due to fixed input dimensionality.

To resolve these limitations, we proposed a Permutation-Invariant Contextual Attention Bandit. By decoupling the state representation from the network topology, our framework enables a single policy to generalize from small ($N = 10$) to hyper-dense ($N = 100$) clusters without retraining. Experimental results demonstrate that our approach yields around 23% reduction in Makespan compared to Log-PSO and prevents premature energy collapse. Crucially, our analysis reveals that the attention mechanism successfully uncovers the latent interference patterns inherent in shared edge networks, validating that architectural invariance is a prerequisite for robust Edge AI orchestration.

This study opens three critical avenues for future research. First, we will expand the action space beyond vertical partitioning to support Hybrid Model Parallelism, enabling the horizontal splitting of large Transformer heads across multiple weak microcontrollers. Second, we plan to integrate monetary costs into the reward function to support hybrid public-private cloud offloading scenarios. Finally, to bridge the gap between simulation and deployment, we aim to validate this policy on a physical testbed comprising NVIDIA Jetson Orin and Raspberry Pi 5 clusters, specifically evaluating the inference overhead of the attention mechanism itself on embedded hardware.

References

[1] S. I. Siam, H. Ahn, L. Liu, S. Alam, H. Shen, Z. Cao, N. Shroff, B. Krishnamachari, M. Srivastava, and M. Zhang. "Artificial intelligence of things: A survey". In: *ACM Transactions on Sensor Networks* 21.1 (2025), pp. 1–75.

[2] Y. Matsubara, M. Levorato, and F. Restuccia. "Split computing and early exiting for deep learning applications: Survey and research challenges". In: *ACM Computing Surveys* 55.5 (2022), pp. 1–30.

[3] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang. "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge". In: *ACM SIGARCH Computer Architecture News* 45.1 (2017), pp. 615–629.

[4] V. Ninkovic, D. Vukobratovic, D. Miskovic, and M. Zennaro. "COMSPLIT: A communication–aware split learning design for heterogeneous IoT platforms". In: *IEEE Internet of Things Journal* (2024).

[5] I. Rodriguez-Conde, C. Campos, and F. Fdez-Riverola. "Horizontally distributed inference of deep neural networks for AI-enabled IoT". In: *Sensors* 23.4 (2023), p. 1911.

[6] D. Luo, T. Yu, Y. Wu, H. Wu, T. Wang, and W. Zhang. "Split: Qos-aware dnn inference on shared gpu via evenly-sized model splitting". In: *Proceedings of the 52nd International Conference on Parallel Processing*. 2023, pp. 605–614.

[7] X. Liu, Y. Song, X. Li, Y. Sun, H. Lan, Z. Liu, L. Jiang, and J. Li. "Efficient Partitioning Vision Transformer on Edge Devices for Distributed Inference". In: *2025 IEEE 45th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2025, pp. 286–296.

[8] P. Patel, E. Choukse, C. Zhang, A. Shah, Í. Goiri, S. Maleki, and R. Bianchini. "Splitwise: Efficient generative llm inference using phase splitting". In: *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2024, pp. 118–132.

[9] E. Samikwa, A. Di Maio, and T. Braun. "Disnet: Distributed micro-split deep learning in heterogeneous dynamic iot". In: *IEEE internet of things journal* 11.4 (2023), pp. 6199–6216.

[10] W. H. Mubark, J. G. Kasula, and M. Y. Sarwar Uddin. "Asap: Asynchronous split inference for accelerated dnn execution". In: *Proceedings of the 25th International Conference on Distributed Computing and Networking*. 2024, pp. 32–44.

[11] T. Feltin, L. Marchó, J.-A. Cordero-Fuertes, F. Brockners, and T. H. Clausen. "Dnn partitioning for inference throughput acceleration at the edge". In: *IEEE Access* 11 (2023), pp. 52236–52249.

[12] P. Mai, R. Yan, Z. Huang, Y. Yang, and Y. Pang. "Split-and-denoise: Protect large language model inference with local differential privacy". In: *arXiv preprint arXiv:2310.09130* (2023).

[13] J. Karjee, P. Naik, K. Anand, and V. N. Bhargav. "Split computing: DNN inference partition with load balancing in IoT-edge platform for beyond 5G". In: *Measurement: Sensors* 23 (2022), p. 100409.

[14] X. Lyu, Y. Li, Y. He, C. Ren, W. Ni, R. P. Liu, P. Zhu, and Q. Cui. "Objective-driven differentiable optimization of traffic prediction and resource allocation for split AI inference edge networks". In: *IEEE Transactions on Machine Learning in Communications and Networking* (2024).

[15] Y. Chen, R. Li, X. Yu, Z. Zhao, and H. Zhang. "Adaptive layer splitting for wireless llm inference in edge computing: A model-based reinforcement learning approach". In: *arXiv preprint arXiv:2406.02616* (2024).

[16] S. Tuli, G. Casale, and N. R. Jennings. "SplitPlace: AI augmented splitting and placement of large-scale neural networks in mobile edge environments". In: *IEEE Transactions on Mobile Computing* 22.9 (2022), pp. 5539–5554.

[17] S. Jung and H.-W. Lee. "Optimization framework for splitting DNN inference jobs over computing networks". In: *Computer Networks* 232 (2023), p. 109814.

[18] I. Korontanis, I. Kontopoulos, A. Zacharia, A. Makris, C. Chronis, M. Pateraki, K. Tserpes, and I. Varlamis. "PEPPER: Profiling-based Edge Placement and Partitioning for Deep Learning Execution". In: *Proceedings of the 15th International Conference on the Internet of Things*. 2025, pp. 228–236.

[19] T. TBD. *IOT Edge-Cloud Traffic Dataset*. 2025. DOI: 10.21227/jar7-cd64. URL: https://dx.doi.org/10.21227/jar7-cd64.

[20] I. Behera and S. Sobhanayak. "Task scheduling optimization in heterogeneous cloud computing environments: A hybrid GA-GWO approach". In: *Journal of Parallel and Distributed Computing* 183 (2024), p. 104766.

[21] X. Huang, C. Li, H. Chen, and D. An. "Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies". In: *Cluster Computing* 23.2 (2020), pp. 1137–1147.

[22] L. Huang, S. Bi, and Y.-J. A. Zhang. "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks". In: *IEEE Transactions on Mobile Computing* 19.11 (2019), pp. 2581–2593.

[23] E. C. de Lima, F. D. Rossi, M. C. Luizelli, R. N. Calheiros, and A. F. Lorenzon. "A neural network framework for optimizing parallel computing in cloud servers". In: *Journal of systems architecture* 150 (2024), p. 103131.