Dynamic Resource Aware Task Scheduling for Mobile Edge Cloud Computing

Indrajeet Roy
College of Engineering
Northeastern University
Boston, MA, United States
roy.i@northeastern.edu

Ajith Kalpathi Ganesh
College of Engineering
Northeastern University
Boston, MA, United States
kalpathiganesh.a@northeastern.edu

Abdulrazaq Surakat
College of Engineering
Northeastern University
Boston, MA, United States
surakat.a@northeastern.edu

***Abstract*—This paper presents an enhanced implementation of the Mobile Cloud Computing (MCC) task scheduling algorithm originally proposed by Lin et al.[1] in Energy and Performance-Aware Task Scheduling in a Mobile Cloud Computing Environment. Although the original algorithm effectively optimized energy consumption under strict completion time constraints, its reliance on static power models, fixed network conditions, and predetermined task characteristics limited its real-world applicability. Our implementation addresses these limitations by integrating dynamic power profiles that adjust to varying device states, modeling realistic network conditions with throughput fluctuations, and adapting task attributes to simulate computational complexity and data intensity. In addition, we extend Lin's binary device-cloud architecture into a three-tier framework that incorporates edge computing resources, providing additional offload options with various energy-performance trade-offs. To further enhance decision-making, we employ a Q-learning approach for task migration, which adaptively explores the solution space to identify optimal offloading patterns. In general, our approach bridges the gap between theoretical MCC scheduling algorithms and practical requirements, enabling efficient energy-efficient task offloading across a range of device power states, network conditions, and computational loads without sacrificing algorithmic efficiency.**

I. Introduction

Mobile Cloud Computing (MCC) augments the limited computational, storage, and battery resources of handheld devices by offloading tasks to remote servers. As applications such as real-time object recognition and on-device language models become prevalent, task placement decisions must optimize across three competing objectives: energy consumption, execution latency, and deadline satisfaction. This optimization is complex, requiring consideration of task dependencies, computational demands, network conditions, and device power states while ensuring application deadlines are met.

Lin et al.[1] introduced a two-phase scheduler for energy and performance-aware task offloading in a binary device-cloud setting. Their approach first generates a minimal-delay schedule using a HEFT-inspired algorithm, then iteratively migrates tasks between device and cloud to minimize energy consumption while respecting deadline constraints. Although effective in controlled settings, their approach relies on simplifying assumptions such as static power models, constant bandwidth, and binary execution options that limit its practical applicability.

Edge computing provides additional options for the task offloading set. Edge devices, positioned closer to end users than traditional cloud datacenters, provide an intermediate execution tier that reduces latency compared to cloud only solutions while typically consuming less energy than local execution. This three-tier architecture introduces new trade-offs and significantly expands the scheduling solution space beyond Lin et al.[1]'s binary model. We present three key extensions to Lin's MCC task scheduling algorithm:

- **Dynamic resource modeling:** Static power models are extended with dynamic profiles that account for device operating states, core-frequency scaling, and workload-dependent power consumption. Network conditions are modeled using time-varying bandwidth and signal-strength-dependent energy costs.
- **Three-Tier Model Integration:** The binary device–cloud scheduler is extended to incorporate edge servers, permitting task placement on the mobile device, an edge node, or the cloud. The original two phase procedure delay optimal initial scheduling followed by energy focused migration is adapted to the expanded architecture, enabling offloading strategies that utilize the low-latency and moderate-energy characteristics of edge resources.
- **Alternative Optimization Strategies:** To efficiently explore the expanded solution space, two refinement approaches are implemented and compared for the energy-optimization phase:
  - A *heuristic migration strategy* extended to three tiers, which deterministically selects task migrations based on estimated energy/time improvements.
  - A *reinforcement learning (Q-learning) migration strategy*, which adaptively explores migration decisions across diverse task graphs and operating conditions to learn policies that may outperform fixed heuristics in complex, dynamic scenarios.

II. Related Work

Prior work in mobile cloud computing (MCC) covers dynamic resource modeling, edge computing, and reinforcement-learning-based task offloading. Lin et al. [1] introduced a two-phase scheduling algorithm that employs static power models and a binary device–cloud architecture. Subsequent studies refined task scheduling under MCC—Guo et al. [5] explored workflow-based offloading, and Wu et al. [6] added context awareness—but none extended the model to a three-tier device–edge–cloud environment.

Power modeling has evolved from simplistic constant-power approaches to sophisticated state-dependent models. Carroll and Heiser [7] conducted empirical measurements of smartphone power consumption, while Huang et al. [8] characterized energy impacts of different network interfaces under variable conditions. Our dynamic resource models incorporate these insights with battery-level sensitivity and workload-dependent energy estimation. Edge computing has transformed mobile offloading from binary decisions to multi-tier architectures. Satyanarayanan et al. [9] introduced cloudlets as trusted, resource-rich computers available for nearby mobile devices, while Shi et al. [10] defined edge computing as computation occurring at the network edge. Our three-tier extension builds on these concepts while addressing algorithmic challenges of task scheduling across heterogeneous tiers.

For decision optimization, Tang and Wong [2] applied deep reinforcement learning to task offloading in edge computing systems, demonstrating adaptability to changing conditions. Huang et al. [11] developed distributed reinforcement learning for multi-user computation offloading. Our Q-learning approach specifically targets energy-time trade-offs in multi-tier environments, integrating reinforcement learning as an enhancement to a proven algorithmic framework.

Task characterization has evolved from simplified execution-time models to nuanced representations. Wang et al. [12] and Flores et al. [13] established taxonomies based on computational and data intensity, informing offloading policies. Our approach incorporates these insights while adding dynamic attributes that drive tier-specific offloading decisions for compute-intensive, data-intensive, and balanced tasks.

III. Foundational Energy and Performance-Aware Task Scheduling Algorithm

Lin et al.[1]'s two-phase task-scheduling algorithm for Mobile Cloud Computing (MCC) serves as the baseline. The following section summarizes the original system model, formalizes the problem, and reviews the scheduling methodology, while identifying limitations that motivate the extensions introduced in this paper.

A. System Model and Problem Formulation

Lin et al.[1] modeled an application as a directed acyclic graph (DAG) $G = (V, E)$, where each node $v_i \in V$ represents a task and each directed edge $(v_i, v_j) \in E$ imposes a precedence constraint requiring task v_i to complete before task v_j can start. The execution environment consists of a mobile device with K heterogeneous local cores and a remote cloud with virtually unlimited computational resources, enabling task offloading and increased parallelism.

The energy consumption model proposed by Lin et al.[1] splits the device's energy into three parts:

- **Local-execution energy**

$$E_i^{\text{device}} \;=\; P_k\, T_{i,k}^{l},$$

  where P_k is the constant power draw of core k and $T_{i,k}^{l}$ is the time required to execute task v_i on that core.

- **Offloading energy**

$$E_i^{\text{RF}} \;=\; P_{\text{d2c}}\, T_i^{s}, \qquad T_i^{s} \;=\; \frac{d_{i,\text{d2c}}^{\text{send}}}{r_{\text{eff}}(\text{d2c})},$$

  where P_{d2c} is the RF-transmitter power, $d_{i,\text{d2c}}^{\text{send}}$ is the input-data size (bytes), and $r_{\text{eff}}(\text{d2c})$ is the effective uplink rate. (Energy for receiving the result is included in background overhead and is not modeled separately.)

- **Cloud-computation energy** Computation energy on the cloud is treated as zero from the mobile device's perspective.

The total energy consumed by the mobile device, $E_{\texttt{total}}$, aggregates individual task energies based on their respective execution assignments (local core or cloud offloading):

$$E_{\texttt{total}} = \sum_{v_i \in V} \begin{cases} E_i^{\texttt{device}}, & \text{if } \texttt{tier}(v_i) = \text{DEVICE} \\ E_i^{\texttt{RF}}, & \text{if } \texttt{tier}(v_i) = \text{CLOUD} \end{cases}$$

The scheduling problem formulated by Lin et al.[1] chooses, for each task $v_i \in V$, both an execution location and schedule to minimize the total mobile-device energy E_{total}, subject to:

1) **Precedence Constraints:** $\forall (v_i, v_j) \in E$, task v_j cannot start until v_i has completed and its results are available.

2) **Deadline Constraint:** The makespan

$$T_{\text{total}} = \max_{v_i \in \text{exit\_tasks}} \big\{\max\big(FT_i^{l},\, FT_i^{wr}\big)\big\}$$

   must satisfy $T_{\text{total}} \le T_{\max}$, where FT_i^{l} is the finish time if v_i runs locally, and FT_i^{wr} is the finish time when v_i is offloaded (including round-trip transmission).

Solving the problem entails three scheduling decisions:

(i) *Task placement*—determine for each task whether it executes on the mobile device or is offloaded to the cloud.

(ii) *Core assignment*—map locally executed tasks to the heterogeneous cores available on the device.

(iii) *Start-time scheduling*—select the exact start time on the assigned resource, i.e., a local core or the device-to-cloud communication channel.

*1) **Phase 1: Initial Scheduling (Minimal Makespan)**:* The goal of this phase is to generate a valid task schedule with the smallest possible makespan T_{total}, satisfying the deadline $T_{\max}$. In practice, $T_{\max}$ is set equal to this minimal makespan. The procedure is based on the HEFT algorithm and consists of three steps:

1) **Primary Assignment (Local vs. Cloud).** Assign each task v_i to the cloud if its estimated remote execution time

$$T_{i,\text{cloud}}^{\text{est}} = T_i^s + T_i^c + T_i^r$$

(upload + compute + download) is less than its minimal local execution time

$$T_{i,\text{device}}^{\text{est}} = \min_k T_{i,k}^l.$$

2) **Task Prioritization (Rank).** Compute an upward rank for each task v_i:

$$\text{rank}(v_i) = \overline{w}_i + \max_{v_j \in \text{succ}(v_i)} \text{rank}(v_j),$$

where

$$\overline{w}_i = \begin{cases} \dfrac{1}{K}\displaystyle\sum_{k=1}^{K} T_{i,k}^l, & \text{if } v_i \text{ is local,} \\ T_{i,\text{cloud}}^{\text{est}}, & \text{if } v_i \text{ is remote.} \end{cases}$$

Higher rank values indicate tasks on the critical path.

3) **Execution-Unit Selection.** Tasks are scheduled sequentially in descending order of their computed priorities. For each available execution resource (local cores k and the cloud), the Earliest Start Time (EST) and Earliest Finish Time (EFT) are computed, ensuring adherence to task precedence constraints:

$$EST(v_i, k) = \max\left\{RT_i^l,\, FAT_{\text{core}}(k),\, \max_{v_j \in \texttt{pred}(v_i)} FT_{\texttt{final}}(v_j)\right\}$$

$$EST(v_i, C) = \max\left\{RT_i^{ws},\, FAT_{\text{ws}},\, \max_{v_j \in \texttt{pred}(v_i)} FT_{\texttt{final}}(v_j)\right\}$$

RT_i^l and RT_i^{ws} represents the earliest times task v_i can start locally or begin uploading data to the cloud. $FAT_{\text{core}}(k)$ and FAT_{ws} represent the finish available times for the respective resources. $FT_{\texttt{final}}(v_j)$ represents the effective finish time of predecessor tasks, accounting for the location of their results. The resource providing the lowest EFT is selected for scheduling task v_i.

*2) **Phase 2: Energy Reduction (Task Migration)**:* Phase 2 minimizes the total energy consumption E_{total} of the Phase 1 schedule while enforcing the makespan constraint $T_{\max}$. Starting from a minimal-makespan schedule (i.e., $T_{\text{total}} \leq T_{\max}$), tasks are iteratively migrated to alternative execution units to reduce E_{total} without exceeding the deadline.

This process consists of four main steps:

1) **Migration Evaluation:** The algorithm evaluates potential migrations of tasks initially assigned to local cores, considering two migration types: (i) from one local core to another (potentially more energy-efficient) local core, and (ii) from a local core to the cloud.

2) **Selection Criteria:** During each iteration, the algorithm selects the most beneficial migration based on energy savings and makespan implications. The decision follows a two-tier criterion:
   (i) Migrations that yield the highest energy reduction ($\Delta E = E_{\texttt{before}} - E_{\texttt{after}}$) without increasing the current makespan ($T_{\texttt{total}}$) are given priority.
   (ii) If no migrations satisfy the first criterion, the migration that maximizes the energy-to-time improvement ratio ($\Delta E/(T_{\texttt{after}} - T_{\texttt{before}})$) is selected, provided the updated schedule remains within the allowed deadline ($T_{\texttt{total}} \leq T_{\texttt{max}}$).

3) **Linear-Time Rescheduling:** After identifying the optimal migration candidate, a linear-time kernel routine (i.e., $O(|V|)$, where $|V|$ is the total number of tasks) updates the existing schedule by adjusting the sequences of tasks on the affected execution units and recalculating their start and finish times.

4) **Iteration and Convergence:** Steps 1 through 3 are repeated iteratively until no further beneficial migrations can be identified that adhere to the original makespan constraint $T_{\texttt{max}}$.

B. Limitations of the Original Approach

Although Lin et al.'s[1] algorithm establishes a baseline for energy and performance aware task scheduling in MCC, its reliance on static assumptions limits effectiveness under dynamic mobile cloud computing operating conditions:

- **Static Power Consumption Model:** The original method assumes constant power parameters (P_k, P_{d2c}) for device cores and wireless transmission. In practice, device power consumption varies with battery state, CPU-frequency scaling, and workload characteristics,making the static model inaccurate and inflexible.
- **Constant Network Condition Assumption:** Lin et al.'s algorithm uses fixed network parameters—constant communication rates $r_{\text{eff}}(\text{d2c})$, $r_{\text{eff}}(\text{c2d})$—and static transmission energy costs, which do not simulate real-world wire-

less conditions where throughput, latency, and power consumption vary with signal strength, congestion, mobility, and interference.

- **Binary Architectural Constraint:** The original framework considers only a two-tier (device-cloud) architecture and omits edge computing, which can offer lower latency than cloud and lower energy costs than local execution, providing effective intermediate offloading options.
- **Simplified Task Characterization:** The original model considers only execution times ($T^l_{i,k}$, T^c_i) and ignores key factors such as computational complexity (CPU-bound vs. I/O-bound), data intensity (transfer volume), and per-task communication overhead that critically influence offloading decisions.

IV. Dynamic Resource Modeling

The proposed framework addresses the limitations of static resource models by integrating dynamic models that accurately simulate variability in real-world mobile computing scenarios.

A. Mobile Device Power Consumption Modeling

In comparison to Lin et al.'s[1] static power model, the framework utilizes dynamic, state-dependent, load-aware power models for mobile devices.

Mobile Device Core Power Model: The power consumption of a specific mobile device core k is modeled as

$$P_k = (b_k + c_k \cdot \texttt{load})\, bf,$$

where b_k is the baseline idle power, c_k is the incremental active power coefficient, $\texttt{load} \in [0, 1]$ is the normalized utilization, and bf is the battery-state scaling factor.

Mobile Device Battery-Level Sensitivity: Battery charge levels significantly impact mobile device power efficiency, especially at lower charge states.This sensitivity is modeled using a multiplicative battery scaling factor bf, which adjusts both idle and active power consumptions for all cores and RF components as follows:

$$bf = \begin{cases} 1.0, & \text{if } B > 30 \\ 1.0 + 0.01 \times (30 - B), & \text{if } B \leq 30 \end{cases}$$

where B represents the current battery level. This simulated up to a 30% increase in power consumption as the battery level drops below 30%, modeling real-world power inefficiencies at low battery states.

The proposed implementation models a big.LITTLE architecture with three core classes—high-performance, mid-range, and efficiency—each exhibiting distinct power–performance trade-offs. These differentiated profiles enable energy-aware scheduling by assigning tasks to cores based on their computational demands and the battery-scaling factor bf. Table I summarizes the power model parameters for each core class.

TABLE I: Core power models differentiated by performance class.

Core Class	Base idle (W)	Total Power Expression
High-performance	0.10	$(0.10 + 0.20 + 1.80\,\texttt{load})\, bf$
Mid-range	0.05	$(0.05 + 0.10 + 1.40\,\texttt{load})\, bf$
Efficiency	0.03	$(0.03 + 0.05 + 0.95\,\texttt{load})\, bf$

B. Mobile Device RF Transmission Power Modeling

The framework defines RF power consumption models for device-to-edge and device-to-cloud transmissions, simulating realistic variations due to available bandwidth, and signal strength. r represents the data rate (Mbps), s the signal strength (dBm), bf the battery-scaling factor, and r_{eff} the RF circuit efficiency. The models are given by

$$P_{\text{d2e}} = \frac{bf}{r_{\text{eff}}}\Big[\,0.1 + 0.4\,\frac{r}{10}\big(1 + 0.02\,(70 - s)\big)\Big]$$

$$P_{\text{d2c}} = \frac{bf}{r_{\text{eff}}}\Big[\,0.15 + 0.6\,\frac{r}{5}\big(1 + 0.03\,(70 - s)\big)\Big]$$

These models extend beyond the constant transmission power assumption in Lin et al.[1], enabling a more detailed simulation of mobile RF energy consumption. The coefficient values are derived from empirical measurements and vendor specifications to simulate realistic operating conditions:

- **Baseline Offsets (0.1 for device-to-edge, 0.15 for device-to-cloud):** Simulates the RF transceiver's idle power and control-plane overhead during connected standby (beacon signaling, channel maintenance, handshakes) even when little or no user data are sent. Essentially, the keep-alive cost of staying attached to the network.
- **Data-Rate Scaling Factors (0.4 for device-to-edge, 0.6 for device-to-cloud):** Simulates the additional RF power demanded by higher throughput.The coefficients convert requested data rate into the incremental power budget, with the larger cloud value reflecting the higher effective-isotropic-radiated-power (EIRP) and modulation complexity of wide-area links.
- **Signal-Strength Adjustment Factors (0.02 for device-to-edge, 0.03 for device-to-cloud):** Model automatic transmit-power-control (TPC) responses to path loss and fading. These factors represent the extra power required to maintain the target signal-to-noise ratio as received-signal strength degrades, with the larger cloud coefficient accounting for longer propagation distances and harsher transmission environments.

The RF power models explicitly differentiate between short-range edge offloading and long-range cloud transmissions, enabling the scheduler to generate more accurate and realistic energy-cost estimates for task migration decisions.

C. Edge Device and Cloud Server Power Modeling

Load-dependent power consumption models are utilized for edge devices and cloud servers to simulate hardware heterogeneity and dynamic utilization characteristics.

*1) **Edge Server Power Model**:* Power consumption for each edge server node $m \in \{1, \ldots, M\}$ and each core $c \in \{1, \ldots, C_m\}$ is defined by assigning a core-specific efficiency factor, $\texttt{efficiency}_{m,c}$, modeling hardware diversity:

$$\texttt{efficiency}_{m,c} = \max\left(0.5,\ [1.0 - 0.05(m-1) - 0.02(c-1)] \times \delta\right),$$

where $\delta \sim \mathcal{U}(0.9, 1.1)$ introduces a small random variation, and the linear terms represent gradual efficiency degradation across edge devices and cores. A lower bound of 0.5 ensures realistic operational limits.

From this, a power scaling factor is defined as:

$$\kappa_{m,c} = \frac{1.0}{\texttt{efficiency}_{m,c}},$$

where higher values correspond to lower efficiency (higher power consumption). The power consumption of each edge core is computed as:

$$P_{\texttt{edge}}(m, c, \ell) = P_{\texttt{idle}}(m, c) + P_{\texttt{dynamic}}(m, c, \ell),$$

where:

$$P_{\texttt{idle}}(m, c) = 5.0 \times \kappa_{m,c},$$
$$P_{\texttt{dynamic}}(m, c, \ell) = (3.0 + 12.0\,\ell) \times \kappa_{m,c},$$

with $\ell \in [0, 1]$ representing normalized core utilization.

The numerical coefficients used in the models simulate standard server benchmarks and typical industry data:

- Edge idle power baseline (5.0 W): Simulates low-power edge hardware commonly deployed in close-proximity infrastructures.
- Edge dynamic power scaling (3.0 + 12.0 ℓ W): Simulates realistic edge-server CPU power consumption ranges from idle (minimal load) to maximum load scenarios, consistent with low-to-medium performance edge CPUs.

*2) **Cloud Server Power Model**:* Cloud servers are modeled as high-capacity infrastructure, with increased idle power consumption and load-dependent scaling.The power model is:

$$P_{\texttt{cloud}}(\ell) = P_{\texttt{idle}}(\texttt{cloud}) + P_{\texttt{dynamic}}(\texttt{cloud}, \ell),$$

where:

$$P_{\texttt{idle}}(\texttt{cloud}) = 50.0,$$
$$P_{\texttt{dynamic}}(\texttt{cloud}, \ell) = 20.0 + 180.0\,\ell,$$

and $\ell \in [0, 1]$ representing normalized utilization.

The numerical coefficients used in the models simulate standard server benchmarks and typical industry data:

- Cloud idle power (50.0 W): Simulates a typical idle power draw for standard dual-socket cloud servers according to SPECpower benchmarks.
- Cloud dynamic power scaling (20.0 + 180.0 ℓ W): Simulates realistic incremental power consumption for cloud servers, scaling with workload intensity and aligning with standard 2-socket high-performance CPU server configurations.

These parameter values enable accurate, realistic evaluation of task-offloading strategies in the three-tier MCC framework, addressing the limitations imposed by the static and simplified modeling assumptions.

D. Network Variability Modeling

Lin et al.'s[1] model assumes fixed communication rates, which fails to capture real-world bandwidth fluctuations and congestion. The proposed framework defines separate upload and download base rates $r_{\text{base,up}}(\ell),\ r_{\text{base,down}}(\ell)$ for each link type ℓ in the three-tier architecture. Table II lists example values.

TABLE II: Base upload/download bandwidths for link types

Link type ℓ	$r_{\text{base,up}}(\ell)$ [Mbps]	$r_{\text{base,down}}(\ell)$ [Mbps]
Device $\rightarrow$ Edge ($\texttt{d2e}_m$)	10	12
Device $\rightarrow$ Cloud (d2c)	5	6
Edge $\leftrightarrow$ Edge	30	30
Edge $\rightarrow$ Cloud	50	60

Device $\rightarrow$ Edge links: Represent short-range connectivity (e.g., Wi-Fi, local 5G microcell) between a mobile device and a nearby edge server. Uplink speeds are modeled slightly lower than downlink to reflect common access-point asymmetry.

Device $\rightarrow$ Cloud links: Model wide-area cellular or public internet connections from the mobile device to a remote cloud data center. These links exhibit lower uplink throughput (e.g., due to shared spectrum and ISP traffic shaping) compared to local offloading paths.

Edge $\leftrightarrow$ Edge links: Simulates high-capacity, symmetric backhaul connections (e.g., fiber or metro Ethernet) interconnecting edge nodes supporting low-latency, peer-to-peer task migration.

Edge $\rightarrow$ Cloud links: Simulates network links between edge infrastructure and central cloud regions to accommodate high symmetric bandwidth bulk data exchange and inter-datacenter synchronization.

By distinguishing these link types and their characteristic rates, the scheduler selects the appropriate parameters for each offloading path, improving the accuracy of transfer time and energy cost estimates under heterogeneous, time varying network conditions.

Scenario Scaling: A global bandwidth scaling factor f_{BW} is applied to each base rate to simulate different network-quality scenarios:

$$r_{\mathrm{eff}}(\ell) = f_{\mathrm{BW}}\, r_{\mathrm{base}}(\ell).$$

Values $f_{\mathrm{BW}} > 1$ simulate favorable conditions (e.g., 5G or wired backhaul), while $f_{\mathrm{BW}} < 1$ represent degraded environments. This approach enables simulation across a range of network states without modifying link-specific parameters. In the three-tier architecture, distinct base bandwidths are defined for device–edge ($\texttt{d2e}_m$), edge–cloud, and device–cloud (`d2c`) links. Scaling these rates by f_{BW} simulates symmetric and asymmetric variations, allowing the scheduler to opportunistically exploit higher-bandwidth paths. Such dynamic network modeling is critical for the design and evaluation of effective scheduling strategies in MCC environments.

E. Task Characterization

Lin et al.'s[1] original model characterizes each task only by its execution times ($T^l_{i,k}$, T^c_i), ignoring important factors that affect offloading efficiency. The proposed framework augments task representations with:

- **Computation Category:** Classifies tasks as compute-bound, data-bound, or balanced, enabling the scheduler to prioritize appropriate resources for each task type. This differentiation allows compute-intensive tasks to be directed to high-performance cores or cloud resources, while keeping data-intensive tasks closer to their data sources to minimize transfer overhead.
- **Data Payload Size:** Quantifies the input/output data volumes for each task, allowing precise calculation of energy and time costs for data transfers. This enables the scheduler to make intelligent tradeoffs between local execution (avoiding transfer costs) and remote execution (gaining computational benefits) based on actual data requirements.
- **Communication Overhead:** Models path-specific transfer times based on data sizes and bandwidth rates for each network link, enabling the scheduler to account for heterogeneous network conditions when making migration decisions. This enhances energy efficiency by avoiding costly transfers over constrained or power-hungry channels.

These heterogeneous attributes collectively enable more accurate, context-aware scheduling that dynamically adapts to both the computational nature of tasks and the current state of the three-tier network environment.

Tasks are classified into three categories—*Compute-Intensive*, *Data-Intensive*, and *Balanced*. Categories are assigned probabilistically according to $P(\text{Compute}) = 0.3$, $P(\text{Data}) = 0.3$, $P(\text{Balanced}) = 0.4$, to simulate mixed application workflows. Examples include encryption and image processing for Compute-Intensive tasks, and database queries and large file transfers for Data-Intensive tasks.

Task Attributes: Complexity and Data Intensity: Each task v_i carries two stochastic attributes:

- **Computational complexity** $\texttt{complexity}(v_i) \in [\gamma_{\min}, \gamma_{\max}]$
- **Data intensity** $\texttt{intensity}(v_i) \in [\delta_{\min}, \delta_{\max}]$

Task class	$\texttt{complexity}(v_i)$ range	$\texttt{intensity}(v_i)$ range
Compute-Intensive	$[0.7\,\gamma_{\max},\ \gamma_{\max}]$	$[\delta_{\min},\ 2\,\delta_{\min}]$
Data-Intensive	$[\gamma_{\min},\ 2\,\gamma_{\min}]$	$[0.7\,\delta_{\max},\ \delta_{\max}]$
Balanced	$[\gamma_{\min},\ \gamma_{\max}]$	$[\delta_{\min},\ \delta_{\max}]$

- **Task specialization classification:**
  For *compute-intensive* tasks, $\texttt{complexity}(v_i) \sim U\big[0.7\,\gamma_{\max},\ \gamma_{\max}\big]$; for *data-intensive* tasks, $\texttt{intensity}(v_i) \sim U\big[0.7\,\delta_{\max},\ \delta_{\max}\big]$. This ensures compute-heavy tasks typically demand more CPU than balanced or data tasks of equal size, and vice versa.
- **Offloading incentive structure:**
  The non-dominant attribute is drawn from its lower range: $\texttt{intensity}(v_i) \sim U\big[\delta_{\min},\ 2\,\delta_{\min}\big]$ for compute tasks, and $\texttt{complexity}(v_i) \sim U\big[\gamma_{\min},\ 2\,\gamma_{\min}\big]$ for data tasks.
  - CPU-heavy tasks incur low transmission cost but high local compute cost, promoting offloading when bandwidth permits.
  - Data-heavy tasks incur low compute cost but high transmission cost, favoring local or edge execution under limited uplink.
- **Energy–delay scaling alignment:**
  The 30%/70% splits keep attribute multipliers within the linear regions of the mobile-core power model P_k and the RF-energy models (P_{d2e}, P_{d2c}), ensuring proportional scaling of simulated energy and delay without saturating either model.

By sampling tasks as either strongly CPU-bound or strongly data-bound, the scheduler receives clear energy-versus-delay trade-off signals, simplifying evaluation of the three-tier offloading logic and preventing mixed workloads from obscuring validation of the dynamic power and network models.

V. Three-Tier Heuristic Extension

This section extends Lin et al.'s[1] two-phase scheduling algorithm to a three-tier (device–edge–cloud) architecture. The heuristic framework employs the dynamic resource models from Section IV and minimizes mobile-device energy consumption E_{total} under the makespan constraint $T_{\max}$. The algorithm retains the original two phases:

(i) *Makespan minimization:* A HEFT-inspired kernel algorithm produces a minimal-delay schedule.
(ii) *Energy optimization:* A linear-time ($O(|V|)$) kernel algorithm iteratively migrates tasks to reduce E_{total} while ensuring $T_{\mathrm{total}} \le T_{\max}$.

A. *Phase 1: Initial Scheduling (Minimal Makespan)*

This phase extends the HEFT algorithm to a three-tier (Device–Edge–Cloud) setting to produce a schedule minimizing the total completion time T_{total}, subject to task precedence constraints. The resulting makespan is used to set $T_{\max}$ for Phase 2.

*1) **Step 1.1: Preliminary Tier Estimation**:* For each task $v_i \in V$, we three completion times assuming no contention are computed for each tier, and the tier with the minimum estimate is selected:

a) Device estimate:

$$T_{i,\text{dev}}^{\text{est}} = \min_{0 \le k < K} T_{i,k}^{l}$$

b) Cloud estimate:

$$\begin{aligned} T_{i,\text{cloud}}^{\text{est}} = T_{\text{transfer}}(\texttt{d2c},\, d_{i,\texttt{d2c}}^{\text{send}}) + T_i^c \\ + T_{\text{transfer}}(\texttt{c2d},\, d_{i,\texttt{c2d}}^{\text{recv}}) \end{aligned}$$

c) Edge estimate:

$$\begin{aligned} \widehat{T}_{i,\text{edge}}(m,c) = T_{\text{transfer}}(\texttt{d2e}_m,\, d_{i,\texttt{d2e}_m}^{\text{send}}) + T_{i,m,c}^{e} \\ + T_{\text{transfer}}(\texttt{e2d}_m,\, d_{i,\texttt{e2d}_m}^{\text{recv}})\,, \\ T_{i,\text{edge}}^{\text{est}} = \min_{m,c} \widehat{T}_{i,\text{edge}}(m,c) \end{aligned}$$

d) Tier selection:

$$\text{Tier}_i^{\text{pre}} = \arg \min_{\tau \in \{\text{dev,edge,cloud}\}} T_{i,\tau}^{\text{est}}$$

The preliminary tier $\text{Tier}_i^{\text{pre}}$ is used to compute the average cost $\overline{w}_i$ in the prioritization step (Step 1.2). This assignment is non-binding: the final execution unit selection (Step 1.3) may reassign v_i based on actual resource contention and dependencies to minimize its finish time.

Algorithm 1 Primary tier assignment for each task $v_i \in V$

Require: Task set V; execution times $T_{i,k}^{l}, T_{i,m,c}^{e}, T_i^c$; data sizes $d_{i,\ell}^{\text{send}}, d_{i,\ell}^{\text{recv}}$; link rates $r_{\text{eff}}(\ell)$

for all $v_i \in V$ **do**
  $T_{i,\text{device}}^{\text{est}} \leftarrow \min_{k=0}^{K-1} T_{i,k}^{l}$
  $t_{\text{up}} \leftarrow T_{\text{transfer}}(\text{d2c}, d_{i,\text{d2c}}^{\text{send}})$
  $t_{\text{down}} \leftarrow T_{\text{transfer}}(\text{c2d}, d_{i,\text{c2d}}^{\text{recv}})$
  $T_{i,\text{cloud}}^{\text{est}} \leftarrow t_{\text{up}} + T_i^c + t_{\text{down}}$
  $T_{i,\text{edge}}^{\text{est}} \leftarrow \infty$
  for $m \leftarrow 1$ **to** M **do**
    for $c \leftarrow 1$ **to** C_m **do**
      $t_{\text{up}} \leftarrow T_{\text{transfer}}(\text{d2e}_m, d_{i,\text{d2e}_m}^{\text{send}})$
      $t_{\text{down}} \leftarrow T_{\text{transfer}}(\text{e2d}_m, d_{i,\text{e2d}_m}^{\text{recv}})$
      $\widehat{T} \leftarrow t_{\text{up}} + T_{i,m,c}^{e} + t_{\text{down}}$
      $T_{i,\text{edge}}^{\text{est}} \leftarrow \min(T_{i,\text{edge}}^{\text{est}}, \widehat{T})$
  $\texttt{PreliminaryTier}(v_i) \leftarrow \arg\min_{\tau \in \{\text{dev,edge,cloud}\}} T_{i,\tau}^{\text{est}}$

*2) **Step 1.2: Task Prioritization**:* After the primary assignment (Step 1.1) provides an initial estimate $\texttt{PreliminaryTier}(v_i)$ for each task, this step calculates a priority value $\texttt{priority}(v_i)$ for every task $v_i \in V$. This priority determines the order in which tasks are considered during the Execution Unit Selection phase (Step 1.3), giving precedence to tasks on the critical path of the application DAG. The calculation uses the upward rank method, common in list scheduling algorithms like HEFT.

*a) **Compute Task Cost ($\overline{w}_i$)**:* An average or representative computation/execution cost $\overline{w}_i$ is determined for each task v_i. This cost estimate is based on the task's characteristics and its *preliminary tier assignment* $\texttt{PreliminaryTier}(v_i)$ from Step 1.1:

- If $\texttt{PreliminaryTier}(v_i) = \text{DEVICE}$: The cost is the average execution time across all K local device cores.

$$\overline{w}_i = \frac{1}{K} \sum_{k=0}^{K-1} T_{i,k}^{l}$$

- If $\texttt{PreliminaryTier}(v_i) = \text{CLOUD}$: The cost is the total estimated path time $T_{i,\text{cloud}}^{\text{est}}$ for cloud execution calculated in Step 1.1.

$$\overline{w}_i = T_{i,\text{cloud}}^{\text{est}}$$

- If $\texttt{PreliminaryTier}(v_i) = \text{EDGE}$: The cost is the minimum estimated path time $T_{i,\text{edge}}^{\text{est}}$ across all edge cores calculated in Step 1.1.

$$\overline{w}_i = T_{i,\text{edge}}^{\text{est}}$$

This cost $\overline{w}_i$ represents the expected duration contribution of task v_i itself along the critical path, considering its likely execution environment.

*b) **Compute Upward Rank Priority ($\texttt{priority}(v_i)$)**:* The priority of each task v_i is computed recursively, starting from the exit tasks of the DAG and moving upwards towards the entry tasks. The priority indicates the length of the critical path from task v_i to the end of the application, based on the estimated costs $\overline{w}_j$.

- For an **exit task** v_{exit} (a task with no successors, $\text{succ}(v_{\text{exit}}) = \emptyset$):

$$\texttt{priority}(v_{\text{exit}}) = \overline{w}_{\text{exit}}$$

- For any other **non-exit task** v_i:

$$\texttt{priority}(v_i) = \overline{w}_i + \max_{v_j \in \text{succ}(v_i)} \{\texttt{priority}(v_j)\}$$

This recursive calculation ensures that a task's priority incorporates its own cost $\overline{w}_i$ and also the maximum cost $\texttt{priority}(v_j)$ of the path following it through its successors $\text{succ}(v_i)$. Tasks with higher priority values are deemed more critical.

*c) **Create Prioritized Task List (L_{prio}):*** All tasks $v_i \in V$ are stored in a list L_{prio}, sorted in descending order based on their calculated $\texttt{priority}(v_i)$ values. Tasks with equal priority may be ordered arbitrarily or using secondary criteria (e.g., task ID). This list L_{prio} dictates the order in which tasks will be selected for scheduling in the next step.

Algorithm 2 Upward-Rank Prioritization (Step 1.2)

Require: Task set V, DAG $G = (V, E)$, preliminary tiers, execution times, no-contention estimates
Ensure: Priority list L_{prio} (descending)

Stage 1 — Tier-aware cost
for all $v_i \in V$ **do**
  if $\texttt{PreliminaryTier}(v_i) = \text{DEVICE}$ **then**
    $\overline{w}_i \leftarrow \frac{1}{K}\sum_{k=0}^{K-1} T_{i,k}^{l}$
  else if $\texttt{PreliminaryTier}(v_i) = \text{CLOUD}$ **then**
    $\overline{w}_i \leftarrow T_{i,\text{cloud}}^{\text{est}}$
  else
    $\overline{w}_i \leftarrow T_{i,\text{edge}}^{\text{est}}$
Stage 2 — Upward-rank recursion
$R \leftarrow \text{reverseTopo}(G)$
for all $v_i \in R$ **do**
  if $\text{succ}(v_i) = \emptyset$ **then**
    $\texttt{priority}(v_i) \leftarrow \overline{w}_i$
  else
    $\texttt{priority}(v_i) \leftarrow \overline{w}_i + \max_{v_j \in \text{succ}(v_i)} \texttt{priority}(v_j)$
Stage 3 — Sorted priority list
$L_{\text{prio}} \leftarrow \text{sort}(V, \texttt{priority}, \text{desc})$
return L_{prio}

*3) **Step 1.3: Execution Unit Selection (Earliest Finish Time Minimization)**:* This step performs the core resource allocation for initial scheduling by assigning each task v_i to an execution unit u (device core k, edge core (m, c), or cloud server C) and computing its start time. The assignment minimizes the task's effective earliest finish time (EFT) while enforcing DAG precedence constraints and respecting resource availability across all three tiers. Tasks are scheduled sequentially in the order defined by the priority list L_{prio} from Step 1.2.

For each task v_i selected from L_{prio}, the scheduler evaluates its potential execution on every available unit u. For each potential assignment (v_i, u), the scheduler calculates the earliest time v_i could possibly finish its entire execution process, with results becoming available at the device. This involves calculating two key values:

- **Data Ready Time:** The earliest time that all data dependencies for v_i are met, meaning all its immediate predecessors $v_p \in \text{pred}(v_i)$ have completed and their results are available at the device ($FT_{\texttt{final}}(v_p)$).

$$\text{DataReadyTime}(v_i) = \max\left(0, \max_{v_p \in \text{pred}(v_i)} \{FT_{\texttt{final}}(v_p)\}\right)$$

  This corresponds to RT_i^l but is foundational for all tiers.

- **Earliest Start Time ($EST(v_i, u)$):** The earliest moment task v_i can actually start its primary operation (local computation or remote upload) on the first resource required by unit u. This depends on both data readiness and the availability of that specific resource.

$$EST(v_i, u_{\text{initial}}) = \max(\text{DataReadyTime}(v_i), FAT(u_{\text{initial}}))$$

  where u_{initial} is the resource needed first (e.g., core k, channel $\texttt{d2c}$, channel $\texttt{d2e}_m$), and $FAT(u_{\text{initial}})$ is its Finish Available Time. Subsequent phases (compute, download) for remote tiers have their own ESTs determined by the preceding phase's finish time and the availability (FAT) of their respective resources (C or $\texttt{c2d}$ for cloud; edge core (m, c) or $\texttt{e2d}_m$ for edge).

The scheduler computes the final effective EFT for v_i for every possible assignment u. This EFT represents the time the task's results are available back at the mobile device ($FT_{\texttt{final}}(v_i)$). The calculation requires simulating the sequence of operations (upload, compute, download for remote tiers) and considering the availability (FAT) of each involved resource sequentially.

*a) **Detailed Earliest Finish Time (EFT) Calculation and Resource Update Logic**:* The scheduler computes the effective earliest finish time (EFT), representing when task v_i's results are available at the mobile device, for each candidate execution unit u by simulating the necessary steps and respecting resource availability:

1) **Device core $k \in \{0, \ldots, K-1\}$:** For local execution on a device core k, the task can only start after all its predecessors' results are available at the device ($\text{DataReadyTime}(v_i)$) and the core itself is free ($FAT_{\text{core}}(k)$). The Earliest Start Time is the maximum of these two times:

$$EST(v_i, k) = \max\big(\text{DataReadyTime}(v_i),\ FAT_{\text{core}}(k)\big).$$

   The final finish time on the device, which is also the effective EFT for this option, is simply the start time plus the local computation duration:

$$FT_i^l(k) = EST(v_i, k) + T_{i,k}^{l}.$$

2) **Cloud C:** Cloud execution involves three sequential phases. First, the device uploads data. This upload can start at EST_{ws}, which is the later of the data readiness time ($\text{DataReadyTime}(v_i)$) and the upload channel availability (FAT_{ws}). The upload finishes at FT_i^{ws}.

$$EST_{ws} = \max\big(\text{DataReadyTime}(v_i),\ FAT_{\text{ws}}\big),$$
$$FT_i^{ws} = EST_{ws} + T_{\text{transfer}}(\text{d2c},\, d_{i,\text{d2c}}^{\text{send}}).$$

   Second, cloud computation starts at $EST_c = RT_i^c$, which requires the upload to be complete (FT_i^{ws}) and any predecessors also processed on the cloud ($v_p \in \text{pred}_C(v_i)$)

to have finished their computation (FT_p^c). Computation finishes at FT_i^c.

$$RT_i^c = \max\Big(FT_i^{ws},\ \max_{v_p \in \mathrm{pred}_C(v_i)}\{FT_p^c\},\ 0\Big),$$
$$EST_c = RT_i^c,$$
$$FT_i^c = EST_c + T_i^c.$$

Finally, the results are downloaded to the mobile device. This download can start at EST_{wr}, the later of the cloud computation finish time (FT_i^c) and the download channel availability (FAT_{wr}). The effective EFT for the cloud option is the final download finish time FT_i^{wr}.

$$EST_{wr} = \max\big(FT_i^c,\ FAT_{\mathrm{wr}}\big),$$
$$FT_i^{wr} = EST_{wr} + T_{\mathrm{transfer}}(\mathrm{c2d},\ d_{i,\mathrm{c2d}}^{\mathrm{recv}}).$$

3) **Edge core** (m, c) **for all** m, c**:** Edge execution is similar to the cloud path logic but uses edge-specific resources. The device first uploads data to edge node m, starting at $EST_{des} = \max(\mathrm{DataReadyTime}(v_i), FAT_{\mathtt{d2e}}(m))$ and finishing at $FT_{i,m}^{des}$.

$$EST_{des} = \max\big(\mathrm{DataReadyTime}(v_i),\ FAT_{\mathtt{d2e}}(m)\big),$$
$$FT_{i,m}^{des} = EST_{des} + T_{\mathrm{transfer}}(\mathrm{d2e}_m,\ d_{i,\mathrm{d2e}_m}^{\mathrm{send}}).$$

Edge computation on core c can then potentially start at $RT_{i,(m,c)}^e$, which depends on the upload finishing ($FT_{i,m}^{des}$) and any predecessors assigned to the same edge node m ($v_p \in \mathrm{pred}_E(v_i, m)$) completing their computation ($FT_{p,m,c'}^e$). The actual computation starts at EST_e, the later of this readiness time and the specific edge core's availability ($FAT_{\mathrm{edge}}(m, c)$). Computation finishes at $FT_{i,m,c}^e$.

$$RT_{i,(m,c)}^e = \max\Big(FT_{i,m}^{des},\ \max_{v_p \in \mathrm{pred}_E(v_i,m)}\{FT_{p,m,c'}^e\},\ 0\Big),$$
$$EST_e = \max\big(RT_{i,(m,c)}^e,\ FAT_{\mathrm{edge}}(m,c)\big),$$
$$FT_{i,m,c}^e = EST_e + T_{i,m,c}^e.$$

Finally, results are downloaded back to the mobile device starting at $EST_{er} = \max(FT_{i,m,c}^e, FAT_{\mathtt{e2d}}(m))$ and completing at $FT_{i,m}^{er}$. This final time, $FT_{i,m}^{er}$, is the effective EFT for assigning task v_i to edge core (m, c).

$$EST_{er} = \max\big(FT_{i,m,c}^e,\ FAT_{\mathtt{e2d}}(m)\big),$$
$$FT_{i,m}^{er} = EST_{er} + T_{\mathrm{transfer}}(\mathrm{e2d}_m,\ d_{i,\mathrm{e2d}_m}^{\mathrm{recv}}).$$

*b) **Assignment and State Update**:* After evaluating the effective EFT for task v_i on all possible execution units u within the set $\mathcal{U}$ (which includes all device cores k, the cloud C, and all edge cores (m, c)), the scheduler makes the final assignment and updates the system state:

1) **Select Best Unit:** Identify the execution unit u^* that resulted in the minimum effective EFT calculated across all possibilities. This represents the assignment that allows v_i to finish (with results available at the mobile device) earliest, given the current schedule and resource contention.

$$u^* = \arg\min_{u \in \mathcal{U}}\big\{\mathrm{EffectiveEFT}(v_i, u)\big\}.$$

,where $\mathrm{EffectiveEFT}(v_i, u)$ is $FT_i^l(k)$, FT_i^{wr}, or $FT_{i,m}^{er}$ depending on u

2) **Set Task Assignment:** Permanently assign task v_i to the selected unit u^*. This involves updating the task's internal state:
   - Set the numerical assignment index $\mathtt{assignment}(v_i)$ to identify u^*.
   - Set the categorical execution tier $\mathtt{tier}(v_i)$ to DEVICE, EDGE, or CLOUD based on u^*.
3) **Record Scheduled Finish Times:** Store the calculated finish times for all relevant phases associated with the chosen execution path (u^*) with the task v_i. This makes these times available for calculating the ready times of successor tasks. For example:
   - If assigned to device core k, record the final $FT_i^l(k)$.
   - If assigned to cloud C, record FT_i^{ws}, FT_i^c, and the effective finish time FT_i^{wr}.
   - If assigned to edge core (m, c), record $FT_{i,m}^{des}$, $FT_{i,m,c}^e$, and the effective finish time $FT_{i,m}^{er}$.

   The critical value stored is $FT_{\mathtt{final}}(v_i)$, which equals the effective EFT of the chosen path.
4) **Update Resource Availability (FAT):** Update the Finish Available Time (FAT) for each resource that was utilized along the selected execution path. The FAT of a resource is set to the time when task v_i finished using it. This ensures that subsequent tasks consider the correct availability when being scheduled. For instance:
   - If v_i used device core k, update $FAT_{\mathrm{core}}(k) \leftarrow FT_i^l(k)$.
   - If v_i used the cloud path, update $FAT_{ws} \leftarrow FT_i^{ws}$ and $FAT_{wr} \leftarrow FT_i^{wr}$.
   - If v_i used edge core (m, c), update $FAT_{d2e}(m) \leftarrow FT_{i,m}^{des}$, $FAT_{\mathrm{edge}}(m, c) \leftarrow FT_{i,m,c}^e$, and $FAT_{e2d}(m) \leftarrow FT_{i,m}^{er}$.

   This prevents resource over-subscription in future scheduling decisions.

This greedy assignment minimizes each task's finish time under current contention, producing a makespan-optimized initial schedule for the three-tier system.

Algorithm 3 Execution Unit Selection (Step 1.3)

Require: Prioritized list L_{prio}; DAG $G = (V, E)$; current $FT_{\text{final}}(\cdot)$; $FAT(\cdot)$; sets $\mathcal{K}, \mathcal{M}, \mathcal{C}_m$; times $T^l_{i,k}, T^c_i, T^e_{i,m,c}$; transfer times $T_{\text{transfer}}(\ell, d_{i,\ell})$
Ensure: Updated $\texttt{assignment}(v_i)$, $\texttt{tier}(v_i)$, FTs, and FATs

for all $v_i \in L_{\text{prio}}$ **do**
  $DRT \leftarrow \max\big(0, \max_{v_p \in \text{pred}(v_i)} FT_{\text{final}}(v_p)\big)$
  $EFT^* \leftarrow \infty,\ u^* \leftarrow \text{null}$
  ▷ Evaluate device cores
  for all $k \in \mathcal{K}$ **do**
    $EST_k \leftarrow \max(DRT, FAT_{\text{core}}(k))$
    $FT_l \leftarrow EST_k + T^l_{i,k}$
    if $FT_l < EFT^*$ **then**
      $EFT^* \leftarrow FT_l,\ u^* \leftarrow k$
  ▷ Evaluate cloud path
  $EST_{ws} \leftarrow \max(DRT, FAT_{ws})$
  $FT_{ws} \leftarrow EST_{ws} + T_{\text{transfer}}(\text{d2c}, d^{\text{send}}_{i,\text{d2c}})$
  $RT_c \leftarrow \max(FT_{ws}, \max_{v_p \in \text{pred}_C(v_i)} FT^c_p, 0)$
  $FT_c \leftarrow RT_c + T^c_i$
  $FT_{wr} \leftarrow \max(FT_c, FAT_{wr}) + T_{\text{transfer}}(\text{c2d}, d^{\text{recv}}_{i,\text{c2d}})$
  if $FT_{wr} < EFT^*$ **then**
    $EFT^* \leftarrow FT_{wr},\ u^* \leftarrow C$
  ▷ Evaluate edge cores
  for all $m \in \mathcal{M}$ **do**
    for all $c \in \mathcal{C}_m$ **do**
      $EST_{des} \leftarrow \max(DRT, FAT_{d2e}(m))$
      $FT_{des} \leftarrow EST_{des} + T_{\text{transfer}}(\text{d2e}_m, d^{\text{send}}_{i,\text{d2e}_m})$
      $RT_e \leftarrow \max(FT_{des}, \max_{v_p \in \text{pred}_E(v_i,m)} FT^e_{p,m,c'}, 0)$
      $FT_e \leftarrow \max(RT_e, FAT_{\text{edge}}(m,c)) + T^e_{i,m,c}$
      $FT_{er} \leftarrow \max(FT_e, FAT_{e2d}(m)) + T_{\text{transfer}}(\text{e2d}_m, d^{\text{recv}}_{i,\text{e2d}_m})$
      if $FT_{er} < EFT^*$ **then**
        $EFT^* \leftarrow FT_{er},\ u^* \leftarrow (m,c)$
  ▷ Commit assignment and update state
  $\texttt{assignment}(v_i) \leftarrow u^*$; $\texttt{tier}(v_i) \leftarrow \text{tier}(u^*)$
  $FT_{\text{final}}(v_i) \leftarrow EFT^*$
  Update all affected FAT entries to the finish times on the chosen path

B. Phase 2: Task Migration (Energy Optimization)

After Phase 1 produces a minimal-makespan schedule (minimizing T_{total}), Phase 2 iteratively refines this schedule to reduce the mobile device's energy consumption (E_{total}). This optimization is performed while strictly adhering to the application's deadline constraint ($T_{\text{total}} \leq T_{\max}$), which is set to the makespan achieved by the end of Phase 1. The core mechanism involves strategically migrating selected tasks to different execution units (other device cores, edge, or cloud) which will result in a task schedule that reduces mobile device energy consumption.

*1) **Step 2.1: Migration Candidate Generation**:* In each iteration of the optimization loop, the algorithm first identifies a set of potential task migrations worth evaluating. A migration is represented by a pair (v_{tar}, u'), indicating that task v_{tar}, currently assigned to unit $u_{\text{curr}} = \texttt{assignment}(v_{\text{tar}})$, is being considered for moving to a different target unit u' ($u' \neq u_{\text{curr}}$). Since the goal is to minimize energy consumed by the mobile device, the algorithm primarily focuses on migrations that are most likely to reduce the sum of local-core computation energy (E^{device}_i) and RF-upload energy (E^{RF}_i).

1) **Migrations originating from the device tier:** The most promising energy-saving candidates are tasks currently running locally. For every task with $\texttt{tier}(v_{\text{tar}}) =$ DEVICE (currently assigned to device core k), the following potential target units u' are considered:

- *To another device core k' ($k' \neq k$):* This explores utilizing device heterogeneity. Moving v_{tar} might save energy if the target core k' is inherently more power-efficient (i.e., $P_{k'} < P_k$). However, this potential energy gain (ΔE^{device}) must be weighed against any potential increase in the overall makespan T_{total}, which could occur if $T^l_{\text{tar},k'} > T^l_{\text{tar},k}$ or if core k' is less available ($FAT_{\text{core}}(k')$ is later). The resulting schedule's makespan must still satisfy $T_{\max}$.
- *To the cloud C:* This considers offloading the computation entirely. Moving v_{tar} eliminates its local computation energy ($E^{\text{device}}_{\text{tar}}$) but incurs energy cost for the RF upload:

$$E^{\text{RF}}_{\text{tar}} = P_{\text{d2c}} \times T_{\text{transfer}}\big(\text{d2c}, d^{\text{send}}_{\text{tar,d2c}}\big).$$

  This migration is considered a valid candidate only if the saved computation energy is greater than the incurred RF energy ($E^{\text{device}}_{\text{tar}} > E^{\text{RF}}_{\text{tar}}$). The projected final finish time of the task via the cloud path (FT^{wr}_{tar}) after rescheduling must not cause the overall application makespan to exceed $T_{\max}$.
- *To an edge core (m, c):* Similar to cloud offloading, moving v_{tar} to edge core (m, c) eliminates $E^{\text{device}}_{\text{tar}}$ and introduces RF upload energy specific to the edge path:

$$E^{\text{RF}}_{\text{tar}} = P_{\text{d2e}} \times T_{\text{transfer}}\big(\text{d2e}_m, d^{\text{send}}_{\text{tar,d2e}_m}\big).$$

  This migration is considered a valid candidate only if energy saved outweighs the energy spent on upload ($E^{\text{device}}_{\text{tar}} > E^{\text{RF}}_{\text{tar}}$). Additionally, the resulting makespan, influenced by the edge path's final finish time ($FT^{er}_{\text{tar},m}$), must remain within the deadline $T_{\max}$. This evaluation is performed for all available edge cores (m, c).

2) **Exclusion of other migration types:** This heuristic excludes two classes of migrations, since neither reduces mobile-side energy E_{total}:
- **Remote-to-device moves:** Any migration from edge or cloud back to the device reintroduces local computation energy $E^{\text{device}}_{\text{tar}}$, directly contradicting the objective of minimizing mobile energy.
- **Remote-to-remote moves:** Migrations between edge and cloud (or between different edge nodes) do not affect the device's computation or RF-upload energy, and thus have no impact on the primary mobile device energy optimization target E_{total}.

3) **Construct Candidate Set $\mathcal{M}$:** Based on the above criteria, form the set

$$\mathcal{M} = \big\{(v_{\text{tar}}, u') \;\big|\; \texttt{tier}(v_{\text{tar}}) = \text{DEVICE},\ u' \in (\mathcal{K} \setminus \{k\}) \cup \{C\} \cup \{(m,c)\}\big\}.$$

$\mathcal{K}$ is the set of device cores, and (m, c) ranges over all edge cores. Early pruning by the condition

$$E^{\text{device}}_{\text{tar}} > E^{\text{RF}}_{\text{tar}}$$

may be applied at this stage to reduce $\mathcal{M}$ before detailed evaluation.

Each potentially beneficial candidate migration $(v_{\text{tar}}, u') \in \mathcal{M}$ identified in this step is then passed to Step 2.2 for evaluation using the kernel rescheduling algorithm to determine its impact on the overall application execution schedule makespan T_{total} and the mobile device energy consumption E_{total}.

*2) **Step 2.2: Migration Evaluation**:* Once the candidate set $\mathcal{M}$ of migrations (v_{tar}, u') is generated (Step 2.1), we must efficiently predict each move's effect on:

$$T'_{\text{total}} = \max_{v_i \in V} FT_{\text{final}}(v_i), \quad E'_{\text{total}} = \sum_{v_i \in V} \left(E_i^{\text{device}} + E_i^{\text{RF}}\right).$$

Re-running Phase 1 for every candidate is not efficient. Instead, the kernel rescheduling procedure of Lin et al[1] is adapted to the three-tier model. The kernel rescheduling algorithm updates only the affected task and its dependents in time $\mathcal{O}\big(|V| + |E|\big)$ allowing efficient recomputation of all EST/FT values after a single assignment change.

The evaluation for a single candidate migration (v_{tar}, u') involves simulating the change on a temporary copy of the current schedule state (which includes all task attributes like finish times, assignments, and the execution sequences S) using two sub-steps:

1) **Sequence Construction:** The task execution sequences S_u are modified to simulate the proposed migration. This involves:
   - **Removal:** Task v_{tar} is located in and removed from its current execution sequence $S_{u_{\text{curr}}}$, where $u_{\text{curr}} = \texttt{assignment}(v_{tar})$.
   - **Insertion:** Task v_{tar} is inserted into the sequence $S_{u'}$ corresponding to the target unit u'. The insertion point is determined using binary search to maintain an order that respects task readiness, ensuring v_{tar} is placed after tasks likely to start earlier. The key used for this binary search depends on the target unit u':
     - If u' is a device core k', the key is the task's earliest ready time for local execution, RT^l_{tar}.
     - If u' is the cloud C, the key is the earliest ready time for cloud upload, RT^{ws}_{tar}.
     - If u' is an edge core (m, c), the key is the estimated finish time of the device-to-edge upload, $FT^{\text{des,est}}_{\text{tar},m} = RT_{\text{tar},\texttt{d2e}_m} + T_{\texttt{transfer}}(\texttt{d2e}_m, d^{\texttt{send}}_{tar,\texttt{d2e}_m})$. This heuristic approximates the task's readiness at the edge for sequencing purposes before the full reschedule.
   - **Temporary State Update:** On the copied task object for v_{tar}, its `assignment` and `tier` attributes are updated to indicate the target unit u', and its scheduling status $\texttt{state}(v_{tar})$ is reset to UNSCHEDULED, indicating it needs to be rescheduled by the kernel.

   This sub-step produces a new set of task sequences S' simulating the hypothetical post-migration ordering.
2) **Kernel Scheduling:** The kernel algorithm is executed using the modified sequences S' and the temporarily updated task attributes. It recalculates all task timings (EST and FT values: $FT^l_i, FT^{ws}_i, \ldots, FT^{er}_{i,m}, FT_{\texttt{final}}(v_i)$) for the entire DAG in linear time with respect to the number of tasks and edges ($\mathcal{O}(|V| + |E|)$). This process correctly propagates the timing changes resulting from the migration throughout the graph, respecting all dependencies and current resource availabilities (FAT).

After the kernel algorithm completes its run on the temporary schedule copy, the projected makespan T'_{total} is determined by finding the maximum $FT_{\texttt{final}}(v_i)$ among all exit tasks. The projected mobile energy consumption E'_{total} is recalculated based on the new assignments and potentially altered execution/transfer durations seen in the updated FT values. These projected metrics $(T'_{\text{total}}, E'_{\text{total}})$ quantify the expected outcome of the candidate migration and are passed to the selection logic in Step 2.3. This entire two-sub-step evaluation process is repeated for every candidate migration (v_{tar}, u') in the set $\mathcal{M}$.

*3) **Step 2.3: Heuristic Migration Selection**:* After evaluating all candidate migrations $(v_{\text{tar}}, u') \in \mathcal{M}$ via the kernel rescheduling algorithm (Step 2.2) and obtaining their respective projected makespans (T'_{total}) and mobile-energy consumptions (E'_{total}), this step implements the heuristic decision rule to select at most *one* migration to apply permanently in the current optimization iteration. The heuristic prioritizes maximal energy reduction, first considering only options that do not increase the makespan, and then considering energy-efficient time/energy trade-offs, always respecting the overall deadline $T_{\texttt{max}}$.

T_{total} and E_{total} represent the makespan and mobile energy of the schedule before applying any migration in this iteration. For each evaluated candidate (represented by (v, u', T', E') containing the task, target unit, projected time, and projected energy), the change in energy $\Delta E = E_{\text{total}} - E'$ and the change in makespan $\Delta T = T' - T_{\text{total}}$ is calcualated. A positive ΔE indicates energy savings, while a positive ΔT indicates a makespan increase.

The selection follows a two-criteria approach:

1) **Filter Valid Candidates:** The set of evaluated candidates is filtered to include only candidates that are potentially beneficial and feasible. A candidate (v, u', T', E') is considered valid if it strictly reduces mobile energy ($\Delta E > 0$) and its resulting makespan does not violate the deadline ($T' \leq T_{\texttt{max}}$). $\mathcal{M}_{\text{valid}}$ represents this filtered set. If $\mathcal{M}_{\text{valid}}$ is empty, no migration is possible in this iteration (proceed to step 4).
2) **Criteria 1 (Maximize Energy Reduction with No Time Increase):** Prioritize migrations that offer energy savings without negatively impacting performance. Iden-

tify the subset $\mathcal{M}_a = \{(v, u', T', E') \in \mathcal{M}_{\text{valid}} \mid \Delta T \leq 0\}$ containing valid migrations that do not increase the makespan. If $\mathcal{M}_a$ is not empty, the migration $(v_{\text{best}}, u'_{\text{best}})$ is selected from $\mathcal{M}_a$ that provides the largest energy reduction (maximum ΔE):

$$(v_{\text{best}}, u'_{\text{best}}, T'_{\text{best}}, E'_{\text{best}}) = \underset{(v,u',T',E') \in \mathcal{M}_a}{\arg\max} \{\Delta E\}.$$

If a best migration is found under this criterion, it is chosen, and the algorithm proceeds to Step 2.4 to apply it.

3) **Criteria 2 (Maximize Energy-per-Time Trade-off Efficiency):** If no migration satisfies Criteria 1 (i.e., $\mathcal{M}_a$ is empty), the algorithm considers valid migrations that do increase the makespan but still meet the deadline. A subset $\mathcal{M}_b = \mathcal{M}_{\text{valid}} \setminus \mathcal{M}_a$ is generated which contains valid migrations where $\Delta T > 0$. If $\mathcal{M}_b$ is not empty,the migration $(v_{\text{best}}, u'_{\text{best}})$ from $\mathcal{M}_b$ that offers the best trade-off is selected, specifically the migration candidate that maximizes the ratio of energy saved per unit of makespan increase ($\Delta E / \Delta T$):

$$(v_{\text{best}}, u'_{\text{best}}, T'_{\text{best}}, E'_{\text{best}}) = \underset{(v,u',T',E') \in \mathcal{M}_b}{\arg\max} \left\{ \frac{\Delta E}{\Delta T} \right\}.$$

If a best migration is found under this criteria, it is chosen, and the algorithm proceeds to Step 2.4.

4) **No Selection / Termination:** If, after checking both criteria, no suitable migration is selected (i.e., $\mathcal{M}_{\text{valid}}$ was initially empty, or both $\mathcal{M}_a$ and $\mathcal{M}_b$ are empty), it indicates that no further energy reduction is possible under the heuristic rules within the deadline constraint. No migration is applied in this iteration, and the optimization process terminates, as described in Step 2.4.

This deterministic selection process ensures that the algorithm greedily pursues the most impactful energy savings available at each step, while performance (effect on makespan) is only traded off when necessary and efficient, always respecting the hard deadline $T_{\texttt{max}}$.

Algorithm 4 SELECTMIGRATION($\mathcal{L}, T_{\text{total}}, E_{\text{total}}, T_{\max}$)

Require: Evaluated list $\mathcal{L} = \{(v, u', T', E')\}$ from Step 2.2;current makespan T_{total};current mobile energy E_{total};deadline $T_{\max}$
Ensure: `best` migration tuple or `null`

for all $(v, u', T', E') \in \mathcal{L}$ **do**
  $\Delta E \leftarrow E_{\text{total}} - E', \quad \Delta T \leftarrow T' - T_{\text{total}}$
  store $(v, u', \Delta E, \Delta T, T', E')$ in $\mathcal{L}'$

$\mathcal{M}_{\text{valid}} \leftarrow \{x \in \mathcal{L}' \mid \Delta E > 0 \ \wedge \ T' \leq T_{\max}\}$

$\mathcal{M}_a \leftarrow \{x \in \mathcal{M}_{\text{valid}} \mid \Delta T \leq 0\}$
if $\mathcal{M}_a \neq \emptyset$ **then**
  return $\arg\max_{x \in \mathcal{M}_a} \{\Delta E\}$

$\mathcal{M}_b \leftarrow \mathcal{M}_{\text{valid}} \setminus \mathcal{M}_a$
if $\mathcal{M}_b \neq \emptyset$ **then**
  return $\arg \max_{x \in \mathcal{M}_b} \left\{ \frac{\Delta E}{\Delta T} \right\}$

return `null`

*4) **Step 2.4: Iteration and Convergence**:* This step concludes one iteration of Phase 2 and determines if the energy optimization loop continues.

1) **Apply Selected Migration:** If a migration $(v_{\text{best}}, u'_{\text{best}})$ was chosen in Step 2.3:
   - The permanent assignment attributes $\texttt{assignment}(v_{\text{best}})$ and $\texttt{tier}(v_{\text{best}})$ are updated to indicate the new unit u'_{best}.
   - The modified execution sequences S' that were generated during the Sequence Construction sub-step (Step 2.2) for the evaluation of the chosen migration $(v_{\text{best}}, u'_{\text{best}})$ are retained.
   - All EST/FT values $(FT_i^l, FT_i^{ws}, \ldots, FT_{\texttt{final}})$ for all tasks v_i are updated to match the results obtained from the Kernel Scheduling sub-step (Step 2.2) for the selected migration.
   - The global task scheduling performance metrics T_{total} and E_{total} are updated based on the newly applied and rescheduled state.

   The algorithm then loops back to Step 2.1 (candidate generation) for the next iteration, starting from this updated schedule state.
2) **Check Convergence:** If Step 2.3 resulted in no migration candidates being selected (either $\mathcal{M}_{\text{valid}}$ was empty or neither criteria identified a suitable migration candidate), it indicates that no further improvements satisfying the heuristic criteria can be found. The optimization loop terminates.

The schedule $\mathcal{S}^\star$ represents when the loop terminates is the final output of the heuristic optimization. It consists of the final task assignments ($\texttt{assignment}(v_i)$, $\texttt{tier}(v_i)$), the corresponding execution sequences S_u, all calculated EST/FT values, and the final makespan T_{total} and mobile energy E_{total}. By design, this final schedule satisfies $T_{\text{total}} \leq T_{\max}$ and represents a local optimum for mobile energy consumption under the defined heuristic migration strategy.

Algorithm 5 Heuristic Energy-Optimization Loop (Phase 2)

Require: Initial schedule $\mathcal{S}_0$; deadline $T_{\max}$; task set V; DAG $G = (V, E)$; resource sets $\mathcal{K}, \mathcal{M}_{\text{nodes}}, \mathcal{C}_m$; execution times $T(\cdot)$; data sizes $d(\cdot)$; rates $r(\cdot)$; power models $P_{\text{core}}(\cdot)$, $P_{\text{RF}}(\cdot)$
Ensure: Schedule $\mathcal{S}^\star$ with $T_{\text{total}} \le T_{\max}$ and locally minimal E_{total}

$\mathcal{S} \leftarrow \mathcal{S}_0$
$T_{\text{total}} \leftarrow$ CALCULATEMAKESPAN($\mathcal{S}$)
$E_{\text{total}} \leftarrow$ CALCULATEMOBILEENERGY($\mathcal{S}$)
repeat
  did_migrate $\leftarrow$ **false**
  $T_{\text{curr}} \leftarrow T_{\text{total}}$; $E_{\text{curr}} \leftarrow E_{\text{total}}$
  Step 2.1 — Generate Migration Candidates
  $\mathcal{M}_{\text{cand}} \leftarrow \emptyset$
  for all $v_i \in V$ **with** unit$(v_i) = \text{core}_k \in \mathcal{K}$ **do**
    $E^l_{i,k} \leftarrow P_{\text{core}}(k)\, T^l_{i,k}$
    for all $k' \in \mathcal{K} \setminus \{k\}$ **do**
      $\mathcal{M}_{\text{cand}} \cup = \{(v_i, \text{core}_{k'})\}$
    $E^{\text{d2c}}_i \leftarrow P_{\text{RF}}(\text{d2c})\, T_{\text{transfer}}(\text{d2c}, d_{i,\text{d2c}})$
    if $E^l_{i,k} > E^{\text{d2c}}_i$ **then**
      $\mathcal{M}_{\text{cand}} \cup = \{(v_i, C)\}$
    for all $m \in \mathcal{M}_{\text{nodes}}$ **do**
      $E^{\text{d2e}}_{i,m} \leftarrow P_{\text{RF}}(\text{d2e}_m)\, T_{\text{transfer}}(\text{d2e}_m, d_{i,\text{d2e}_m})$
      if $E^l_{i,k} > E^{\text{d2e}}_{i,m}$ **then**
        for all $c \in \mathcal{C}_m$ **do**
          $\mathcal{M}_{\text{cand}} \cup = \{(v_i, (m, c))\}$
  Step 2.2 — Evaluate Candidates via Kernel Rescheduling
  $\mathcal{R}_{\text{eval}} \leftarrow \emptyset$
  for all $(v, u') \in \mathcal{M}_{\text{cand}}$ **do**
    $(T', E') \leftarrow$ KERNELEVALUATE($\mathcal{S}, (v, u')$)
    $\mathcal{R}_{\text{eval}} \cup = \{(v, u', T', E')\}$
  Step 2.3 — Select Best Valid Migration
  $\mathcal{M}_{\text{valid}} \leftarrow \{x \in \mathcal{R}_{\text{eval}} \mid E' < E_{\text{curr}} \ \wedge \ T' \le T_{\max}\}$
  $\mathcal{M}_a \leftarrow \{x \in \mathcal{M}_{\text{valid}} \mid T' \le T_{\text{curr}}\}$
  if $\mathcal{M}_a \neq \emptyset$ **then**
    $(v^\star, u^\star, T^\star, E^\star) \leftarrow \arg\max_{x \in \mathcal{M}_a} (E_{\text{curr}} - E')$
  else
    $\mathcal{M}_b \leftarrow \mathcal{M}_{\text{valid}} \setminus \mathcal{M}_a$
    if $\mathcal{M}_b \neq \emptyset$ **then**
      $(v^\star, u^\star, T^\star, E^\star) \leftarrow \arg\max_{x \in \mathcal{M}_b} \big((E_{\text{curr}} - E')/(T' - T_{\text{curr}})\big)$
    else
      $(v^\star, u^\star) \leftarrow$ null
  Step 2.4 — Apply Migration and Check Convergence
  if $(v^\star, u^\star) \neq$ null **then**
    $\mathcal{S} \leftarrow$ APPLYMIGRATIONANDRESCHEDULE($\mathcal{S}, (v^\star, u^\star)$)
    $T_{\text{total}} \leftarrow$ CALCULATEMAKESPAN($\mathcal{S}$)
    $E_{\text{total}} \leftarrow$ CALCULATEMOBILEENERGY($\mathcal{S}$)
    did_migrate $\leftarrow$ **true**
until not *did_migrate*
return $\mathcal{S}$

VI. Q-Learning Enhanced Energy Optimization

After establishing an initial three-tier schedule that minimizes delay (Phase 1), an alternative approach for Phase 2 energy optimization using Q-learning, a model-free reinforcement learning algorithm is tested. Unlike the deterministic heuristic migration strategy (Section V-B), Q-learning adaptively explores the vast solution space of possible task migrations across the device, edge, and cloud tiers. It learns a policy, encoded in a Q-table, to select migrations that optimize long-term rewards, potentially discovering more effective energy-saving strategies, especially in complex scenarios with multiple dependencies and resource interactions.

A. Q-Learning Framework Components

Q-learning models the task migration problem as a Markov Decision Process (MDP), where the agent learns to navigate the scheduling state space by taking migration actions to maximize cumulative rewards related to energy efficiency and maintain makespan deadline constraints.

1) State Representation ($s \in \mathcal{S}$): A state s provides a snapshot of the current scheduling configuration, representing task assignments and scheduling performance metrics and resource characteristics crucial for informed decision-making. A state s incorporates:

- **Task Assignments:** The assigned execution unit `assignment`(v_i) for each task v_i.
- **Current Performance:** The current schedule makespan $T_{\texttt{total}}$ and mobile device energy $E_{\texttt{total}}$.
- **Tier Distribution:** The number of tasks assigned to each tier (DEVICE, EDGE, CLOUD).
- **Tier Characteristics:** Average computational complexity (`complexity`) and data intensity (`intensity`) of tasks within each tier.

To manage the state space, continuous values like $T_{\texttt{total}}$ and $E_{\texttt{total}}$ are discretized and rounded before being used as keys for Q-table lookups.

2) Action Space ($a \in \mathcal{A}$): An action a corresponds to a potential task migration, represented by the pair (v_{tar}, u'), where v_{tar} is the task to be migrated and u' is the target execution unit (a specific device core k', the cloud C, or a specific edge core (m, c)), such that $u' \neq$ `assignment`(v_{tar}). The set of *valid* actions $\mathcal{A}(s)$ from a given state s includes possible migrations (v_{tar}, u') not involving the task's current location.

3) Q-Value ($Q(s, a)$): The core of the learning process is the Q-table, which stores the Q-value, $Q(s, a)$. This value represents the expected cumulative discounted future reward obtainable by taking action a in state s and following the optimal policy. Higher Q-values indicate more promising state-action pairs.

4) Learning Update Rule: The Q-values are learned iteratively using the Bellman equation update rule after observing a transition (s, a, r, s'):

$$Q(s, a) \leftarrow Q(s, a) + \alpha \Big[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \Big]$$

where:

- $\alpha \in (0, 1]$ is the *learning rate*, controlling how much new information overrides old estimates, often decayed over time.
- $\gamma \in [0, 1)$ is the *discount factor*, weighting the importance of future rewards.
- r is the *immediate reward* received after taking action a in state s.
- s' is the successor state following action a.
- $\max_{a'} Q(s', a')$ is the maximum estimated future value from state s'.

This update minimizes the Temporal Difference (TD) error.

5) Reward Function (r): The reward function r guides the learning process by quantifying the immediate effect of a transition from state s to s' resulting from action a.

$$r = \text{EnergyReward} - \text{TimePenalty}$$

where:

- **Energy Reward:** Primarily driven by the energy reduction $\Delta E = E_{\texttt{total}}(s) - E_{\texttt{total}}(s')$. Penalties apply if $\Delta E < 0$.
- **Time Penalty:** Applied if the deadline $T_{\texttt{max}}$ is violated in state s', scaling quadratically with the magnitude of the violation $\max(0, T_{\texttt{total}}(s') - T_{\texttt{max}})$.

This structure encourages energy savings while strongly discouraging deadline violations.

B. Q-Learning Algorithm Execution

The optimization process runs over multiple episodes, with each episode consisting of several iterations (state transitions or steps).

1) **Initialization:** Start with the schedule $\mathcal{S}$ from Phase 1. Initialize the Q-table and replay buffer. Set initial hyper-parameters $(\alpha, \gamma, \epsilon)$.
2) **Episode Loop:** Repeat for a maximum number of episodes:
   a) Reset episode state to the current best-known schedule.
   b) **Iteration Loop:** Repeat for a maximum number of steps per episode or until convergence:
      - Get current state s.
      - **Action Selection (Epsilon-Greedy):** Select action $a \in \mathcal{A}(s)$ either randomly (exploration, probability ϵ) or greedily ($a = \arg\max_{a'} Q(s, a')$, exploitation).
      - **Action Evaluation (Simulation):** Predict the outcome metrics (T', E') of taking action a by performing the sequence construction and kernel rescheduling described in Step 2.2 on a copy of the current state.
      - **Reward Calculation:** Calculate the immediate reward r based on the transition from the current $(T_{\texttt{total}}, E_{\texttt{total}})$ to the predicted (T', E').
      - **Action Execution (State Update):** Permanently apply the chosen migration action a to the actual schedule $\mathcal{S}$ by modifying the task assignment and sequences, then running the kernel rescheduling algorithm to update all timings (FTs) and resource availabilities (FATs). This transitions the system to the next state s'. Update the current $T_{\texttt{total}}$ and $E_{\texttt{total}}$.
      - **Q-Table Update:** Update $Q(s, a)$ using the observed reward r, the resulting state s', and the Bellman equation.
      - **Experience Replay:** Store the transition (s, a, r, s') in the replay buffer. Sample experiences from the buffer and perform additional Q-value updates.
      - Update $s \leftarrow s'$.
      - Check termination conditions.
   c) **Decay Parameters:** Decrease exploration rate ϵ and potentially learning rate α.
3) **Output:** Return the best schedule (lowest valid energy respecting $T_{\max}$) encountered.

This adaptive learning process allows the Q-agent to potentially find superior migration policies compared to the fixed rules of the heuristic approach.

Algorithm 6 Q-Learning-Enhanced Energy Optimization (Phase 2 – RL variant)

Require: Initial schedule $\mathcal{S}_0$; deadline $T_{\max}$; task set V; DAG $G = (V, E)$; Q-learning hyper-parameters $(\alpha, \gamma, \epsilon_{\text{start}}, \epsilon_{\text{end}}, \epsilon_{\text{decay}})$; max episodes N_{eps}; max steps N_{steps}; replay buffer B_{replay}; system parameters $\mathcal{P}$
Ensure: Schedule $\mathcal{S}^\star$ with $T_{\text{total}} \le T_{\max}$ and minimal learned E_{total}

Initialise $Q(s, a) \leftarrow 0$ for all state–action pairs
$\mathcal{S}^\star \leftarrow \mathcal{S}_0$;
$E^\star \leftarrow$ CalculateMobileEnergy($\mathcal{S}^\star$);
$T^\star \leftarrow$ CalculateMakespan($\mathcal{S}^\star$)
$\epsilon \leftarrow \epsilon_{\text{start}}$
for episode $= 1$ **to** N_{eps} **do**
  $\mathcal{S} \leftarrow \text{copy}(\mathcal{S}^\star)$
  $T_{\text{total}} \leftarrow T^\star$; $E_{\text{total}} \leftarrow E^\star$
  for step $= 1$ **to** N_{steps} **do**
Step 2.1 — Candidate Generation
    $s \leftarrow$ GetCurrentState($\mathcal{S}, T_{\text{total}}, E_{\text{total}}$)
    $\mathcal{A}(s) \leftarrow$ GetValidActions(s)
    if $\mathcal{A}(s) = \emptyset$ **then break**
Step 2.2 — Candidate Evaluation
    if rand() $< \epsilon$ **then**
      $a \leftarrow$ random choice from $\mathcal{A}(s)$
    else
      $a \leftarrow \arg\max_{a' \in \mathcal{A}(s)} Q(s, a')$
    $(T', E') \leftarrow$ EvaluateMigration(copy($\mathcal{S}$), $a, \mathcal{P}$)
Step 2.3 — Migration "Selection"
    $r \leftarrow$ CalculateReward($E_{\text{total}}, T_{\text{total}}, E', T', T_{\max}$)
Step 2.4 — Apply Migration & Learn
    $\mathcal{S} \leftarrow$ ApplyMigrationAndReschedule($\mathcal{S}, a, \mathcal{P}$)
    $T_{\text{total}} \leftarrow$ CalculateMakespan($\mathcal{S}$)
    $E_{\text{total}} \leftarrow$ CalculateMobileEnergy($\mathcal{S}$)
    $s' \leftarrow$ GetCurrentState($\mathcal{S}, T_{\text{total}}, E_{\text{total}}$)
    bestQ $\leftarrow \max_{a'} Q(s', a')$
    $Q(s, a) \leftarrow Q(s, a) + \alpha\,[\, r + \gamma \cdot \text{bestQ} - Q(s, a)\,]$
    Add (s, a, r, s') to B_{replay}
    Sample mini-batch $B_{\text{sample}} \subset B_{\text{replay}}$
    for all $(s_j, a_j, r_j, s'_j) \in B_{\text{sample}}$ **do**
      bestQ$_j \leftarrow \max_{a'} Q(s'_j, a')$
      $Q(s_j, a_j) \leftarrow Q(s_j, a_j) + \alpha\,[\, r_j + \gamma \cdot \text{bestQ}_j - Q(s_j, a_j)\,]$
    if $T_{\text{total}} \le T_{\max} \wedge E_{\text{total}} < E^\star$ **then**
      $\mathcal{S}^\star \leftarrow \text{copy}(\mathcal{S})$; $E^\star \leftarrow E_{\text{total}}$; $T^\star \leftarrow T_{\text{total}}$
  $\epsilon \leftarrow \max(\epsilon_{\text{end}}, \epsilon \times \epsilon_{\text{decay}})$
  (optional) decay α
return $\mathcal{S}^\star$

VII. Evaluation

This section evaluates the performance and energy efficiency of the proposed three-tier scheduling frameworks compared to the baseline two-tier heuristic.

1) Quantify the energy savings ($E_{\texttt{total}}$) and makespan ($T_{\texttt{total}}$) impact of incorporating the edge tier and dynamic resource models.

2) Compare the effectiveness of the extended three-tier heuristic migration (3T-H) against the adaptive Q-learning based migration (3T-QL) for energy optimization under deadline constraints ($T_{\texttt{max}}$).
3) Analyze the sensitivity of the algorithms to variations in key system parameters, network conditions, device states, and application characteristics.

A. *Simulation Setup*

Experiments are carried out in a Python based simulator that incorporates the dynamic resource models of Section IV and supports the full three-tier architecture. Application workloads are expressed as DAGs generated with parameters for task count $|V|$, graph density, and task-type distribution (compute-intensive, data-intensive, balanced). Task attributes $\texttt{complexity}(v_i)$ and $\texttt{intensity}(v_i)$ and the per-link data sizes $d_{i,\ell}^{\text{send/receive}}$ are assigned stochastically according to task type.

Evaluated scheduling frameworks:

- **2T-H (baseline).** Lin et al. [1] two-phase heuristic adapted to the proposed dynamic models in a two-tier (device–cloud) setting.
- **3T-H.** Proposed two-phase heuristic operating in the three-tier (device–edge–cloud) environment (Section V).
- **3T-QL.** Three-tier framework that shares the Phase-1 schedule of 3T-H and applies Q-learning for Phase-2 energy optimization (Section VI) with default hyper-parameters.

Parameter sweeps: Each test suite varies a single parameter while other parameters are held at nominal values:

- Battery level B: 20, 50, 80, 100 %.
- Bandwidth scaling factor f_{BW}: 0.5, 0.8, 1.1, 1.4, 1.7, 2.0.
- Number of edge nodes M: 0, 1, 2, 3.
- Number of device cores K: 2, 3, 4, 6, 8.
- Task count $|V|$: 20, 30, 50, 75.

Each configuration is executed three times with independent random seeds; results are reported as mean $\pm$ standard deviation. The deadline is defined as $T_{\text{max}} = 1.5\,T_{\text{min}}$, where T_{min} is the minimal makespan produced by Phase 1 of the 2 step scheduling algorithm.

B. *Results and Analysis*

This section presents the comparative results of the 2T-H, 3T-H, and 3T-QL frameworks, focusing on the insights gained from parameter sweeps. All plots show the mean results over three repetitions, with shaded areas representing ± 1 standard deviation.

*1) **Impact of Edge Tier and Dynamic Models (3T-H vs. 2T-H)**:* Comparing the 3T-H framework against the 2T-H baseline consistently demonstrated the benefits of the edge tier across the tested parameter ranges (Figs. 1–5).

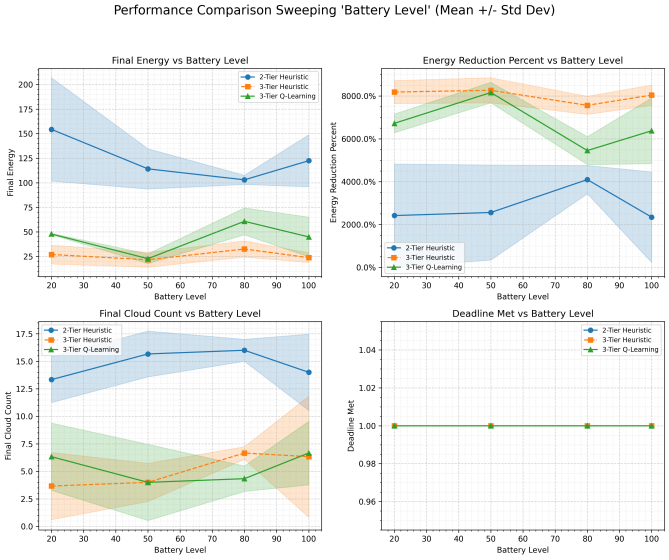

Fig. 1: Performance comparison sweeping Battery Level (%). Plots show Final Energy, Energy Reduction Percentage, Final Cloud Task Count, and Deadline Met Rate as battery level varies from 20% to 100%.

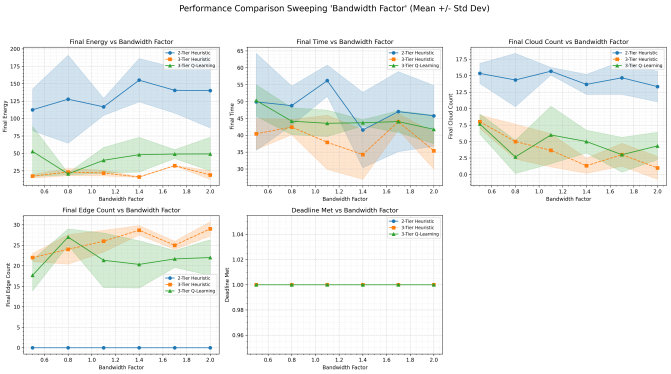

Fig. 2: Performance comparison sweeping Network Bandwidth Factor (f_{BW}). Plots show Final Energy, Final Time, Final Cloud Task Count, Final Edge Task Count, and Deadline Met Rate as bandwidth factor varies from 0.5 to 2.0.

- **Energy Consumption:** 3T-H consistently achieved significantly lower final mobile energy ($E_{\texttt{total}}$) compared to 2T-H. As seen across all sweeps, the orange line (3T-H) lies substantially below the blue line (2T-H). The advantage was particularly highlighted at low battery levels ($B \leq 50\%$, Fig. 1), where 2T-H energy increased sharply while 3T-H remained relatively low. The presence of even one edge node ($M = 1$ vs. $M = 0$ in Fig. 5) caused a significant decrease in energy for 3T-H.
- **Makespan and Deadline Constraint Adherence:** 3T-H often finished faster (lower final $T_{\texttt{total}}$) than 2T-H after energy optimization, particularly noticeable when varying the number of mobile device cores (Fig. 3) or edge nodes (Fig. 5). This indicates the edge tier can improve performance even when the primary goal is energy saving. All frameworks displayed perfect deadline constraint adherence (Deadline Met = 1.0) across all sweeps.

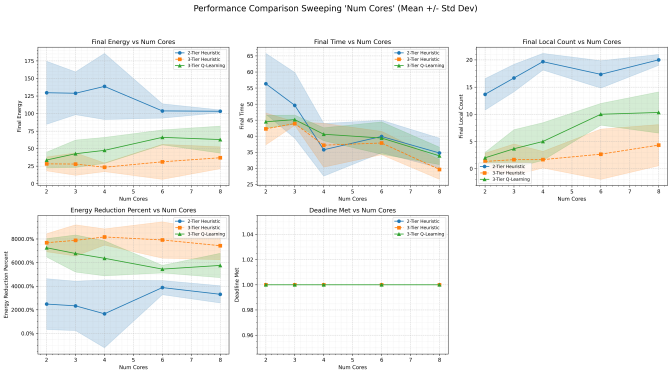

Fig. 3: Performance comparison sweeping Number of Device Cores (K). Plots show Final Energy, Final Time, Final Local Task Count, Energy Reduction Percentage, and Deadline Met Rate as core count varies from 2 to 8.

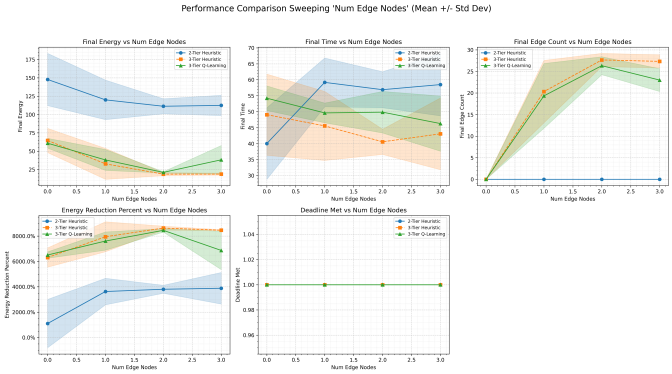

Fig. 5: Performance comparison sweeping Number of Edge Nodes (M). Plots show Final Energy, Final Time, Final Edge Task Count, Energy Reduction Percentage, and Deadline Met Rate as edge node count varies from 0 to 3.

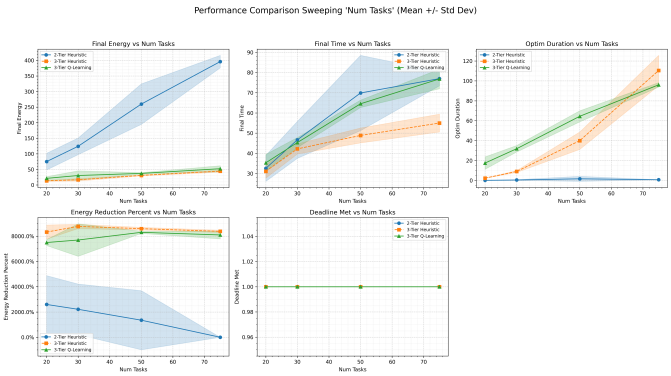

Fig. 4: Performance comparison sweeping Number of Tasks ($|V|$). Plots show Final Energy, Final Time, Optimization Duration, Energy Reduction Percentage, and Deadline Met Rate as task count varies from 20 to 75.

- **Task Distribution:** The results highlight the strategic use of the edge by 3T-H. The final Edge Count or the final count of number of tasks scheduled for execution on the edge tier in Fig. 5 increases sharply from 0 (for 2T-H at $M = 0$) to over 25 tasks by $M = 2$. Conversely, Fig. 2 shows edge tier utilization (Final Edge Count) peaking at lower bandwidth factors ($f_{BW} \approx 0.8$) and then decreasing as bandwidth improves, while the `Final Cloud Count` generally increases, indicating a shift towards cloud when network conditions allow. This adaptability is absent in 2T-H.

*2) **Comparison of Optimization Strategies (3T-QL vs. 3T-H)**:* Comparing the Q-learning optimization (3T-QL, green triangles) against the heuristic migration (3T-H, orange squares) for the energy optimization phase highlighted that under the tested conditions and default hyper-parameters, the heuristic consistently outperformed the Q-learning approach (Figs. 1–4):

- **Energy Savings:** Across all parameter sweeps, the 3T-H heuristic achieved lower final energy ($E_{\texttt{total}}$) than 3T-QL. The orange line consistently sits below the green line in the Final Energy plots. While 3T-QL reduced energy significantly compared to 2T-H, it did not match the effectiveness of the 3T-H heuristic in these experiments. The Energy Reduction Percent plots also show 3T-H generally achieving higher reduction percentages than 3T-QL.
- **Makespan and Deadline Adherence:** Both 3T-H and 3T-QL maintained perfect deadline constraint adherence (Deadline Met = 1.0). 3T-H consistently resulted in a lower final makespan ($T_{\texttt{total}}$) compared to 3T-QL across all sweeps. The difference was noticeable when varying core counts, edge nodes, and bandwidth factors.
- **Computational Overhead:** The Optimization Duration vs Number of Tasks plot (Fig. 4) clearly shows that the optimization phase for 3T-QL required substantially more time than for 3T-H, and this overhead grew rapidly with the number of tasks. The heuristic's optimization duration remained very low and scaled minimally.
- **Task Distribution Patterns:** The final task distributions were similar, with both 3T frameworks utilizing the edge heavily when available (Fig. 5) and shifting load between edge and cloud based on bandwidth (Fig. 2).
- **Potential Q-Learning Challenges:** The consistent underperformance of 3T-QL relative to 3T-H in these tests suggests challenges in its current configuration for this specific problem. Potential reasons include sub-optimal default hyper-parameters, an inadequate state representation that does not fully capture system dynamics impacting the heuristic's effectiveness, or a reward function that does not perfectly align learning with the dual objectives under dynamic conditions.

Based on these results, the extended three-tier heuristic (3T-H) emerges as the superior algorithm, delivering better energy efficiency and faster execution times with significantly lower computational overhead compared to the tested Q-learning implementation (3T-QL).

*3) **Evaluation across Parameter Ranges**:* Analysis under varying conditions of the parameter sweeps provide insights into how the dynamic resource modeling (Section IV), real-

istic variations in the system and application characteristics influence scheduling decisions and outcomes. The results from Figs. 1–5, align well with expected real-world behaviors:

- **Battery Level (B, Fig. 1):** Decreasing battery levels below 50% increases final mobile energy ($E_{\texttt{total}}$) for all frameworks. This is a direct consequence of the battery factor (bf) in the power model (Section IV), which realistically simulates increased device power draw at low charge states. The 2T-H framework, reliant only on cloud offloading (which involves energy-intensive RF), is most affected. The 3T frameworks demonstrate greater resilience by shifting tasks towards the edge as indicated by the stable or slightly increasing edge/cloud counts relative to local increases in energy cost, mitigating the impact of low battery. This aligns with the realistic behavior that systems would avoid power-hungry operations or seek lower-energy offload options when battery is critical. The optimal energy point observed between 50–80% indicates a balance where the device is efficient, but offloading still offers benefits.
- **Network Bandwidth (f_{BW}, Fig. 2):** Network quality effects the offloading strategy, as expected in realistic system behavior. Low bandwidth ($f_{BW} \leq 0.8$) makes long-distance cloud transfers slow and energy-intensive due to RF activity captured by E_i^{RF} depending on $T_{\texttt{transfer}}$. Consequently, the 3T frameworks heavily utilize the closer edge tier (the Final Edge Count peaks), minimizing transfer overhead. As bandwidth improves ($f_{BW} \geq 1.4$), the time and energy cost of cloud transfers decrease, making the powerful cloud resources more attractive; both 3T-H and 3T-QL adapt by increasing cloud usage (Final Cloud Count increases) and reducing edge usage. This dynamic shift between edge and cloud based on network conditions is a key realistic behavior enabled by the three-tier model and bandwidth scaling.
- **Edge Nodes (M, Fig. 5):** Increasing the number of edge devices highlights the realistic benefit of edge device availability in a cloud computing environment. Introducing even one edge node ($M = 1$) provides substantial energy and makespan reductions for 3T frameworks compared to the no-edge baseline ($M = 0$, equivalent to 2T-H), by offering a low-latency, moderate-energy offload target. Adding a second node ($M = 2$) yields further, though smaller, improvements as tasks can be better distributed. The observed diminishing returns at $M = 3$, where energy/time benefits plateau and edge task count saturates, realistically simulating that factors like task dependencies, communication bottlenecks to/from the device, or the suitability of some tasks for device/cloud limit or decrease the utility of adding more edge resources.
- **Device Cores (K, Fig. 3):** Increasing local processing power ($K = 2$ to 8) reduces the application makespan ($T_{\texttt{total}}$) due to increased parallelism potential, aligning with realistic system behavior expectations for multi-core devices. The impact on final energy ($E_{\texttt{total}}$): 2T-H sees moderate energy reduction as more tasks can run efficiently locally, reducing costly cloud offloads. For 3T frameworks, energy remains relatively flat or slightly increases; while individual local tasks might be cheaper on more/better cores (modeled via P_k), the overall incentive to offload tasks to edge/cloud for significant energy savings (compared to any local execution) persists, especially as more cores potentially increase total device idle power. The steady increase in Final Local Count confirms increased local execution with higher K.
- **Number of Tasks ($|V|$, Fig. 4):** Increasing application complexity ($|V| = 20$ to 75) leads to near-linear increases in both final makespan ($T_{\texttt{total}}$) and final energy ($E_{\texttt{total}}$) for the heuristic schedulers. The increasing energy gap between 2T-H and the 3T frameworks highlights the importance of efficient tier selection (including edge) for larger, more complex applications, a realistic scenario. The super-linear growth in optimization duration for 3T-QL highlights the scalability challenge inherent in applying reinforcement learning to larger state spaces, contrasting with the efficiency of the heuristics.

The analysis based on the dynamic resource models, produces results that are consistent with mobile cloud computing system behavior expectations and also highlights real-world system behaviors and trade-offs involving mobile device battery life, network quality, available computing resources (local, edge, cloud), and application scale.

VIII. Conclusion

This paper presented a comprehensive enhancement to Lin et al.'s[1] mobile cloud computing task scheduling algorithm by addressing three key limitations: static resource modeling, binary offloading architecture, and simplified task characterization. Our dynamic resource models incorporate battery-level sensitivity, workload-dependent power consumption, and variable network conditions, enabling more realistic simulation of mobile environments. By extending the architecture to include edge computing as an intermediate tier between devices and cloud resources, we provided additional offloading options with diverse energy-performance trade-offs.

Experimental results across different configurations confirm that our three-tier framework consistently outperforms the binary device-cloud architecture, particularly for data-intensive tasks that benefit from edge execution. Furthermore, our Q-learning approach for task migration demonstrated superior energy savings while maintaining completion times within acceptable constraints. The integration of reinforcement learning enables the algorithm to discover non-intuitive optimizations that heuristic approaches might miss, evidenced by oscillatory migration patterns that ultimately converge to more efficient schedules.

IX. Future Work

Several promising research directions emerge from our enhanced three-tier scheduling framework. First, the dynamic power models could be further refined through real-world deployment and measurement studies on diverse mobile devices and network conditions. This would enable more accurate energy estimation and potentially reveal additional optimization opportunities specific to different hardware configurations.

Our Q-learning approach could be extended with more sophisticated reinforcement learning techniques such as deep Q-networks or actor-critic methods to better capture complex relationships between task characteristics, resource states, and optimal offloading decisions. Incorporating transfer learning capabilities would allow the system to leverage knowledge gained from previous applications to accelerate adaptation to new task graphs.

The current framework could be expanded to address multi-device scenarios where multiple mobile devices compete for limited edge and cloud resources. This introduces new dimensions of resource contention, fairness considerations, and potential for collaborative optimization across devices. Game-theoretic approaches might complement our reinforcement learning techniques in such distributed settings.

Incorporating predictive models for network conditions, user mobility, and workload characteristics could enable proactive scheduling decisions rather than purely reactive ones. By anticipating changes in resource availability or network quality, the scheduler could preemptively migrate tasks before conditions deteriorate, further improving both energy efficiency and performance reliability.

APPENDIX A
NOTATION REFERENCE

TABLE III: Comprehensive notation and definitions for the scheduling framework

Symbol	Meaning/Usage	Mathematical Formula/Definition
1. Fundamental Concepts		
$G = (V, E)$	Application represented as a Directed Acyclic Graph	V=task set, E=dependency set
v, v_i	An individual task in the application graph	$v_i \in V$
$\text{pred}(v)$	Set of immediate predecessor tasks whose execution must complete before v's execution can start	$\{v_j \in V \mid (v_j, v) \in E\}$
$\text{succ}(v)$	Set of immediate successor tasks whose execution requires v's execution to complete first	$\{v_j \in V \mid (v, v_j) \in E\}$
k	Index for a local core on the mobile device (Device Tier)	$k \in \{0, \ldots, K-1\}$
(m, c)	Index for core c on edge node m (Edge Tier)	$m \in \{1, \ldots, M\}, c \in \{1, \ldots, C_m\}$
C	Symbol representing the Cloud Tier resource	–
u	Identifier for any potential task execution location (unit)	$u \in \{k, (m, c), C\}$
S_k	Ordered sequence of tasks assigned to device core k	$S_k = \{v_{i_1}, v_{i_2}, \ldots\}$
S_{cloud}	Ordered sequence of tasks assigned to the cloud tier	$S_{\text{cloud}} = \{v_{i_1}, v_{i_2}, \ldots\}$
$S_{(m,c)}$	Ordered sequence of tasks assigned to edge core (m, c)	$S_{(m,c)} = \{v_{i_1}, v_{i_2}, \ldots\}$
S_u	General term for the execution sequence on unit u	$S_u \in \{S_k, S_{\text{cloud}}, S_{(m,c)}\}$
2. Core Input Parameters & Models		
2.1 Task Computation Times & Characteristics		
$T^l_{i,k}$	Duration of v_i computation executed locally on device core k	$T^l_{i,k} > 0, \ \forall i, k$
$T^e_{i,m,c}$	Duration of v_i computation executed on edge server m, core c (Result of estimation model below)	$T^e_{i,m,c} > 0, \ \forall i, m, c$
T^c_i	Duration of v_i computation performed on cloud resources	$T^c_i > 0, \ \forall i$
$\texttt{complexity}(v_i)$	Measure of task v_i's computational requirement	$\texttt{complexity} \in [\gamma_{\min}, \gamma_{\max}]$
$\texttt{intensity}(v_i)$	Measure of task v_i's data handling requirement	$\texttt{intensity} \in [\delta_{\min}, \delta_{\max}]$
2.2 Communication Channels & Data Transfers		
ℓ	Identifier for a specific communication channel type	$\ell \in \{\texttt{d2c}, \texttt{c2d}, \texttt{d2e}_m, \texttt{e2d}_m, \ldots\}$
$\texttt{d2c}$	Channel for device uploading data to cloud (uplink)	–
$\texttt{c2d}$	Channel for device downloading data from cloud (downlink)	–
$\texttt{d2e}_m$	Channel for device uploading data to edge node m (uplink)	$m \in \{1, \ldots, M\}$
$\texttt{e2d}_m$	Channel for device downloading data from edge node m (downlink)	$m \in \{1, \ldots, M\}$
$r_{\texttt{base}}(\ell)$	Base bandwidth provisioned for channel type ℓ (Mbps)	$r_{\texttt{base}}(\ell) > 0$
f_{BW}	Global bandwidth scaling factor simulating overall network conditions	$f_{BW} > 0$

Continued on next page

Table III continued from previous page

Symbol	Meaning/Usage	Mathematical Formula/Definition
$r_{\texttt{eff}}(\ell)$	Effective data transfer rate achieved on channel ℓ	$r_{\texttt{eff}}(\ell) = r_{\texttt{base}}(\ell) \times f_{BW}$
$d_{i,\ell}^{\texttt{send}}$	Data volume (MB) sent for v_i input/initialization over channel ℓ	$d_{i,\ell}^{\texttt{send}} > 0$
$d_{i,\ell}^{\texttt{receive}}$	Data volume (MB) of v_i results received over channel ℓ	$d_{i,\ell}^{\texttt{receive}} > 0$
$T_{\texttt{transfer}}(\ell, d)$	Duration (seconds) required to transfer d MB of data over channel ℓ	$T_{\texttt{transfer}}(\ell, d) = d / r_{\texttt{eff}}(\ell)$
2.3 Mobile Device Power & Energy Models		
B	Current battery level of the mobile device (%)	$B \in [0, 100]$
bf	Battery factor scaling device power based on current battery level B	$bf = \begin{cases} 1.0 & \text{if } B > 30 \\ 1.0 + 0.01(30 - B) & \text{if } B \leq 30 \end{cases}$
b_k, c_k	Base idle power and dynamic power coefficient for device core k	$b_k > 0, c_k > 0$
$\texttt{load}$	Normalized core utilization factor (for device core power calculation)	$\texttt{load} \in [0, 1]$
P_k	Instantaneous power consumption of device core k given $\texttt{load}$ and bf	$P_k = (b_k + c_k \cdot \texttt{load}) \cdot bf$
$rf_{\texttt{eff}}$	Efficiency factor of the mobile device's radio frequency components	$rf_{\texttt{eff}} \in (0, 1]$
s	Received signal strength indicator (RSSI) in dBm, influencing RF power	$s < 0$
r	Data transmission rate (Mbps) used in RF power calculation	$r = r_{\texttt{eff}}(\ell)$ for the relevant channel ℓ
$P_{\texttt{d2e}}$	Mobile device RF power draw during data upload to an edge node at rate r	$P_{\texttt{d2e}} = \frac{bf}{rf_{\texttt{eff}}}[0.1 + 0.4\frac{r}{10}(1 + 0.02(70 - s))]$
$P_{\texttt{d2c}}$	Mobile device RF power draw during data upload to the cloud at rate r	$P_{\texttt{d2c}} = \frac{bf}{rf_{\texttt{eff}}}[0.15 + 0.6\frac{r}{5}(1 + 0.03(70 - s))]$
$E_i^{\texttt{device}}$	Energy consumed by mobile device to compute v_i locally on core k	$E_i^{\texttt{device}} = P_k \cdot T_{i,k}^l$
$E_i^{\texttt{RF}}$	Energy consumed by mobile device RF component to upload v_i's data over channel ℓ	$E_i^{\texttt{RF}} = P_\ell \cdot T_{\texttt{transfer}}(\ell, d_{i,\ell}^{\texttt{send}})$
2.4 Edge & Cloud Server Power Models (Non-Mobile Energy)		
$\texttt{efficiency}_{m,c}$	Performance factor simulating heterogeneity of edge core (m, c)	E.g., $1.0 - 0.05(m - 1) - 0.02(c - 1)$
$\kappa_{m,c}$	Power scaling factor derived from edge efficiency	$\kappa_{m,c} = 1.0 / \texttt{efficiency}_{m,c}$
$P_{\texttt{edge}}(m, c, \texttt{load})$	Total power consumption of edge core (m, c) at a given $\texttt{load}$	$(5.0 + 3.0 + 12.0 \times \texttt{load}) \times \kappa_{m,c}$
$P_{\texttt{cloud}}(\texttt{load})$	Total power consumption of cloud resource at a given $\texttt{load}$	$50.0 + 20.0 + 180.0 \times \texttt{load}$
2.5 Edge Execution Time Estimation Model (Deriving $T_{i,m,c}^e$)		
$T_{\min}^l$	Minimum local execution time for v_i across all device cores	$T_{\min}^l = \min_k T_{i,k}^l$
T_i^s, T_i^r	Simplified cloud send/receive times	Used only in $T_{\texttt{total}}^c$
$T_{\texttt{total}}^c$	Simplified total path time for cloud (input to edge estimate)	$T_{\texttt{total}}^c = T_i^s + T_i^c + T_i^r$
α	Interpolation weight balancing device vs. cloud influence	$\alpha \in [0, 1]$
$T_{\texttt{base}}^{\texttt{edge}}$	Base edge computation time estimate derived from $T_{\min}^l, T_{\texttt{total}}^c$	$T_{\texttt{base}}^{\texttt{edge}} = \alpha T_{\min}^l + (1 - \alpha) T_{\texttt{total}}^c$
$\texttt{adj}_{\texttt{compute}}$	Edge time multiplier for compute-intensive tasks	$= 1.0 + 0.1(\texttt{complexity}(v_i)/3)$

Continued on next page

Table III continued from previous page

Symbol	Meaning/Usage	Mathematical Formula/Definition
$\texttt{adj}_{\texttt{data}}$	Edge time multiplier for data-intensive tasks	$= 1.0 - 0.1(\texttt{intensity}(v_i)/2)$
$\texttt{adj}_{\texttt{type}}$	Adjustment factor based on v_i's task type	$\in \{\texttt{adj}_{\texttt{compute}}, \texttt{adj}_{\texttt{data}}, 1.0\}$
$\texttt{efficiency}_{m,c}$	Performance factor simulating heterogeneity of edge core (m, c)	$1.0 - 0.05(m-1) - 0.02(c-1)$
$T^{\texttt{final}}_{i,m,c}$	Final estimated edge computation duration before scheduling v_i	$T^{\texttt{final}}_{i,m,c} = T^{\texttt{edge}}_{\texttt{base}} \cdot \texttt{adj}_{\texttt{type}} \cdot \texttt{efficiency}_{m,c} \cdot \mathcal{U}(0.95, 1.05)$
3. Phase 1 (Initial Scheduling) Variables		
3.1 Preliminary Tier Ranking Estimates (No Contention) (Step 1.1)		
$T^{\texttt{est}}_{i,\texttt{device}}$	Estimated time if v_i runs locally (no channel and core contention)	$T^{\texttt{est}}_{i,\texttt{device}} = \min_k T^l_{i,k}$
$T^{\texttt{est}}_{i,\texttt{cloud}}$	Estimated time if v_i runs on cloud (no channel and core contention)	$T^{\texttt{est}}_{i,\texttt{cloud}} = T_{\texttt{transfer}}(\texttt{d2c}, d^{\texttt{send}}_{i,\texttt{d2c}}) + T^c_i + T_{\texttt{transfer}}(\texttt{c2d}, d^{\texttt{receive}}_{i,\texttt{c2d}})$
$T^{\texttt{est}}_{i,\texttt{edge}}$	Estimated time if v_i runs on best edge unit (no channel and core contention)	$T^{\texttt{est}}_{i,\texttt{edge}} = \min_{m,c}(T_{\texttt{transfer}}(\texttt{d2e}_m, d^{\texttt{send}}_{i,\texttt{d2e}_m}) + T^e_{i,m,c} + T_{\texttt{transfer}}(\texttt{e2d}_m, d^{\texttt{receive}}_{i,\texttt{e2d}_m}))$
$\texttt{PreliminaryTier}(v_i)$	Initial tier preference based on minimum $T^{\texttt{est}}_{i,\tau}$	$\arg\min_{\tau \in \{\text{DEVICE,EDGE,CLOUD}\}} T^{\texttt{est}}_{i,\tau}$
3.2 Prioritization Variables (Step 1.2)		
$\overline{w}_i$	Cost weight for v_i based on $\texttt{PreliminaryTier}(v_i)$	$\overline{w}_i = \begin{cases} T^{\texttt{est}}_{i,\texttt{cloud}} & \text{if PT}(v_i) = \text{CLOUD} \\ T^{\texttt{est}}_{i,\texttt{edge}} & \text{if PT}(v_i) = \text{EDGE} \\ \frac{1}{K}\sum_k T^l_{i,k} & \text{if PT}(v_i) = \text{DEVICE} \end{cases}$
$\texttt{priority}(v_i)$	Upward rank (critical path estimate) for scheduling order	$\texttt{priority}(v_i) = \overline{w}_i + \max_{v_j \in \text{succ}(v_i)} \texttt{priority}(v_j)$
L_{prio}	List of tasks sorted by $\texttt{priority}(v_i)$ descending	–
4. Core Scheduling Variables (Phase 1.3 & Kernel)		
4.1 Ready Times (When a task can start)		
RT^l_i	Earliest v_i can start on any device core (data dependency met)	$RT^l_i = \max(0, \max_{v_j \in \text{pred}(v_i)} FT_{\texttt{final}}(v_j))$
RT^{ws}_i	Earliest device can start sending v_i to cloud (data + channel ready)	$RT^{ws}_i = \max(RT^l_i, FAT_{\texttt{ws}})$
RT^c_i	Earliest cloud compute for v_i can start (upload done + cloud preds done)	$RT^c_i = \max(FT^{ws}_i, \max_{v_p \in \text{pred}_C(v_i)} FT^c_p)$
RT^{wr}_i	Earliest cloud results for v_i are ready to send back	$RT^{wr}_i = FT^c_i$
$RT_{i,\texttt{d2e}_m}$	Earliest device can start sending v_i to edge m (data + channel ready)	$RT_{i,\texttt{d2e}_m} = \max(RT^l_i, FAT_{\texttt{d2e}}(m))$
$RT^e_{i,(m,c)}$	Earliest edge core (m, c) can start computing v_i (upload, preds, core ready)	$RT^e_{i,(m,c)} = \max(FT^{\texttt{des}}_{i,m}, \max_{v_p \in \text{pred}_E(v_i,m)} FT^e_{p,m,c'}, FAT_{\text{edge}}(m,c))$
4.2 Scheduled Start & Finish Times (Result of scheduling decision)		
$EST(v_i, u)$	Earliest possible start time for v_i on unit u considering data readiness and resource availability	$EST(v_i,u) = \max(\text{RT}_{\text{for\_unit}}(v_i,u), FAT(u))$ where $\text{RT}_{\text{for\_unit}}$ is: $\begin{cases} RT^l_i & \text{if } u = k \text{ (Device Core)} \\ RT^{ws}_i & \text{if } u = C \text{ (Cloud Upload Start)} \\ RT_{i,\texttt{d2e}_m} & \text{if } u = (m,c) \text{ (Edge Upload Start)} \\ RT^c_i & \text{if } u = C \text{ (Cloud Compute Start)} \\ RT^e_{i,(m,c)} & \text{if } u = (m,c) \text{ (Edge Compute Start)} \\ \ldots & \text{(etc. for download phases)} \end{cases}$
$SST(v_i, u)$	Actual scheduled start time of v_i on its assigned unit u	$SST(v_i,u) = EST(v_i,u)$
FT^l_i	Finish time of v_i if scheduled on a device core k	$FT^l_i = SST(v_i,k) + T^l_{i,k}$

Continued on next page

Table III continued from previous page

Symbol	Meaning/Usage	Mathematical Formula/Definition
FT_i^{ws}	Finish time of sending v_i's data from device to cloud	$FT_i^{ws} = SST_{ws} + T_{\texttt{transfer}}(\texttt{d2c}, d_{i,\texttt{d2c}}^{\texttt{send}})$
FT_i^c	Finish time of v_i's computation within the cloud	$FT_i^c = SST_c + T_i^c$
FT_i^{wr}	Finish time of receiving v_i's results from cloud at the device	$FT_i^{wr} = SST_{wr} + T_{\texttt{transfer}}(\texttt{c2d}, d_{i,\texttt{c2d}}^{\texttt{receive}})$
$FT_{i,m}^{\texttt{des}}$	Finish time of sending v_i's data from device to edge node m	$FT_{i,m}^{\texttt{des}} = SST_{des} + T_{\texttt{transfer}}(\texttt{d2e}_m, d_{i,\texttt{d2e}_m}^{\texttt{send}})$
$FT_{i,m,c}^e$	Finish time of v_i's computation on edge core (m,c)	$FT_{i,m,c}^e = SST_e + T_{i,m,c}^e$
$FT_{i,m}^{\texttt{er}}$	Finish time of receiving v_i's results from edge node m at the device	$FT_{i,m}^{\texttt{er}} = SST_{er} + T_{\texttt{transfer}}(\texttt{e2d}_m, d_{i,\texttt{e2d}_m}^{\texttt{receive}})$
$FT_{\texttt{final}}(v_i)$	Effective finish time: when v_i's results are available at the device	$FT_{\texttt{final}}(v_i) = \begin{cases} FT_i^l & \text{if } \texttt{tier}(v_i) = \text{DEVICE} \\ FT_i^{wr} & \text{if } \texttt{tier}(v_i) = \text{CLOUD} \\ FT_{i,m}^{er} & \text{if } \texttt{tier}(v_i) = \text{EDGE} \end{cases}$
4.3 Resource Availability (Updated after each task scheduling)		
$FAT(u)$	Finish Available Time: Timestamp when resource u completes its last assigned task	$FAT(u) = \max(0, \max_{v_j \in S_u} FT_{\text{on\_unit}}(v_j, u))$ where $FT_{\text{on\_unit}}$ is the relevant finish time on unit u (e.g., FT_j^l if $u = k$, FT_j^{ws} if $u = \text{ws}$, $FT_{j,m,c}^e$ if $u = (m,c)$ etc.)
$FAT_{\text{core}}(k)$	Availability time of device core k	$FAT_{\text{core}}(k) = \max(0, \max_{v_j \in S_k} FT_j^l)$
FAT_{ws}	Availability time of device-to-cloud send channel	$FAT_{\text{ws}} = \max(0, \max_{v_j \in S_{\text{cloud}}} FT_j^{ws})$
FAT_{wr}	Availability time of cloud-to-device receive channel	$FAT_{\text{wr}} = \max(0, \max_{v_j \in S_{\text{cloud}}} FT_j^{wr})$
$FAT_{\text{edge}}(m,c)$	Availability time of edge core (m,c)	$FAT_{\text{edge}}(m,c) = \max(0, \max_{v_j \in S_{(m,c)}} FT_{j,m,c}^e)$
$FAT_{\text{d2e}}(m)$	Availability time of device-to-edge m send channel	$FAT_{\text{d2e}}(m) = \max(0, \max_{v_j \in S_{\text{edge},m}} FT_{j,m}^{\texttt{des}})$
$FAT_{\text{e2d}}(m)$	Availability time of edge m-to-device receive channel	$FAT_{\text{e2d}}(m) = \max(0, \max_{v_j \in S_{\text{edge},m}} FT_{j,m}^{\texttt{er}})$
5. Phase 2 (Energy Optimization) Variables		
$\mathcal{M}$	Set of candidate migrations (v_{tar}, u') generated	–
$E'_{\texttt{total}}, T'_{\texttt{total}}$	Projected Energy/Makespan after a candidate migration	Calculated by rescheduling all tasks using Kernel Rescheduling Algorithm after modifying sequences for the migration
ΔE	Mobile energy reduction from a candidate migration	$\Delta E = E_{\texttt{total}} - E'_{\texttt{total}}$
ΔT	Makespan change from a candidate migration	$\Delta T = T'_{\texttt{total}} - T_{\texttt{total}}$
Criterion 1	Select migration with max ΔE, given $\Delta T \leq 0$	$\arg\max_{\texttt{migration} \in \mathcal{M}_a} \{\Delta E\}$
Criterion 2	Select migration with max $\Delta E/\Delta T$, given $\Delta T > 0$ and $T'_{\texttt{total}} \leq T_{\max}$	$\arg\max_{\texttt{migration} \in \mathcal{M}_b} \{\Delta E/\Delta T\}$
6. Kernel Algorithm Internal State		
$\texttt{ready}_1(v_i)$	Count of unscheduled DAG predecessors of v_i	Integer ≥ 0
$\texttt{ready}_2(v_i)$	Sequence readiness of v_i (0 if ready, 1 if waiting)	$\in \{0, 1\}$
Q	Queue/Stack holding tasks ready for kernel scheduling	$Q \leftarrow \{v_i \mid \texttt{ready}_1(v_i) = 0 \wedge \texttt{ready}_2(v_i) = 0\}$
7. Overall Performance Metrics & Task State		
$T_{\texttt{total}}$	Final application makespan (results available at device)	$T_{\texttt{total}} = \max_{v_i \in \text{exit tasks}} FT_{\texttt{final}}(v_i)$
$E_{\texttt{total}}$	Total mobile device energy consumed	$E_{\texttt{total}} = \sum_{v_i \in V} (E_i^{\texttt{device}} + E_i^{\texttt{RF}})$
$T_{\max}$	Application deadline constraint	$T_{\texttt{total}} \leq T_{\max}$
$\texttt{assignment}(v_i)$	Assigned execution unit index for task v_i	$\in \{0..K-1, K, K+1..K+M \cdot C_m - 1\}$
$\texttt{tier}(v_i)$	Assigned execution tier for task v_i	$\in \{\text{DEVICE}, \text{EDGE}, \text{CLOUD}\}$
$\texttt{state}(v_i)$	Current scheduling phase status of task v_i	$\in \{\text{UNSCHEDULED}, \text{SCHEDULED}, \text{KERNEL\_SCHEDULED}\}$

REFERENCES

[1] Energy and Performance-Aware task scheduling in a mobile cloud computing environment. (2014, June 1). IEEE Conference Publication — IEEE Xplore. http://ieeexplore.ieee.org/document/6973741

[2] Tang, M., & Wong, V. W. S. (2020, April 10). Deep reinforcement learning for task offloading in mobile edge computing systems. arXiv.org. https://arxiv.org/abs/2005.02459

[3] Cong, P., Zhou, J., Li, L., Cao, K., Wei, T., & Li, K. (2020). A survey of Hierarchical Energy Optimization for Mobile Edge Computing. ACM Computing Surveys, 53(2), 1–44. https://doi.org/10.1145/3378935

[4] Ali, F. A., Simoens, P., Verbelen, T., Demeester, P., & Dhoedt, B. (2015). Mobile device power models for energy efficient dynamic offloading at runtime. Journal of Systems and Software, 113, 173–187. https://doi.org/10.1016/j.jss.2015.11.042

[5] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in Proc. IEEE INFOCOM, 2016, pp. 1-9.

[6] H. Wu, W. Knottenbelt, and K. Wolter, "Analysis of the energy-response time tradeoff for mobile cloud offloading using combined metrics," in Proc. 27th Int. Teletraffic Congress (ITC), 2015, pp. 134-142.

[7] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in Proc. USENIX Annual Technical Conference, 2010, pp. 1-14.

[8] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4G LTE networks," in Proc. 10th Int. Conf. Mobile Systems, Applications, and Services, 2012, pp. 225-238.

[9] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," IEEE Pervasive Comput., vol. 8, no. 4, pp. 14-23, Oct.-Dec. 2009.

[10] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," IEEE Internet Things J., vol. 3, no. 5, pp. 637-646, Oct. 2016.

[11] L. Huang, X. Feng, A. Feng, Y. Huang, and L. P. Qian, "Distributed deep learning-based offloading for mobile edge computing networks," Mobile Netw. Appl., vol. 23, pp. 1-8, Nov. 2018.

[12] K. Wang, M. Shen, J. Cho, A. Banerjee, J. Van der Merwe, and K. Webb, "MobiScud: A fast mobile cloud gaming platform with dynamic resource scheduling," in Proc. 7th Int. Conf. Multimedia Systems, 2016, pp. 1-13.

[13] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya, "Mobile code offloading: From concept to practice and beyond," IEEE Commun. Mag., vol. 53, no. 3, pp. 80-88, Mar. 2015.