Batching-Aware Joint Model Onloading and Offloading for Hierarchical Multi-Task Inference

Seohyeon Cha*, Kevin Chan†, Gustavo de Veciana*, Haris Vikalo*
*Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX, USA.
†DEVCOM Army Research Laboratory, Adelphi, MD, USA.

***Abstract*—The growing demand for intelligent services on resource-constrained edge devices has spurred the development of collaborative inference systems that distribute workloads across end devices, edge servers, and the cloud. While most existing frameworks focus on single-task, single-model scenarios, many real-world applications (e.g., autonomous driving and augmented reality) require concurrent execution of diverse tasks including detection, segmentation, and depth estimation. In this work, we propose a unified framework to jointly decide which multi-task models to deploy ("onload") at clients and edge servers, and how to route queries across the hierarchy ("offload") to maximize overall inference accuracy under memory, compute, and communication constraints. We formulate this as a mixed-integer program and introduce J3O (Joint Optimization of Onloading and Offloading), an alternating algorithm that (i) greedily selects models to onload via Lagrangian-relaxed submodular optimization and (ii) determines optimal offloading via constrained linear programming. We further extend J3O to account for batching at the edge, maintaining scalability under heterogeneous task loads. Experiments show J3O consistently achieves over 97% of the optimal accuracy while incurring less than 15% of the runtime required by the optimal solver across multi-task benchmarks.**

***Index Terms*—Hierarchical ML inference, computation offloading and batching, alternating optimization, submodularity**

I. Introduction

The rapid proliferation of edge devices including smartphones, surveillance cameras, and wearables, with possible latency and privacy requirements, has sparked interest in executing Machine Learning (ML)-based inference at the edge [1]. However, as state-of-the-art ML models continue to grow in size and complexity to achieve higher accuracy, their memory and compute requirements often exceed the capabilities of resource-constrained edge hardware [2], [3]. While model compression techniques and lightweight alternatives can help reduce resource usage, they often incur an accuracy drop, particularly in multi-task settings where a compact model must simultaneously support diverse inference tasks [4].

Collaborative inference systems offer a promising alternative by distributing inference workloads across hierarchical computing tiers – end devices, edge servers, and cloud platforms [5]–[7]. These systems typically use lightweight models on device to handle routine or latency-sensitive queries, while selectively offloading more demanding instances to upstream servers with larger models. However, most prior frameworks have focused on single-task, single-model settings (e.g., image classification), which limits their applicability. In contrast, real-world applications such as autonomous driving, augmented reality, and smart surveillance require concurrent execution of multiple tasks (e.g., detection, segmentation, and depth estimation) within the same inference pipeline [8]–[10].

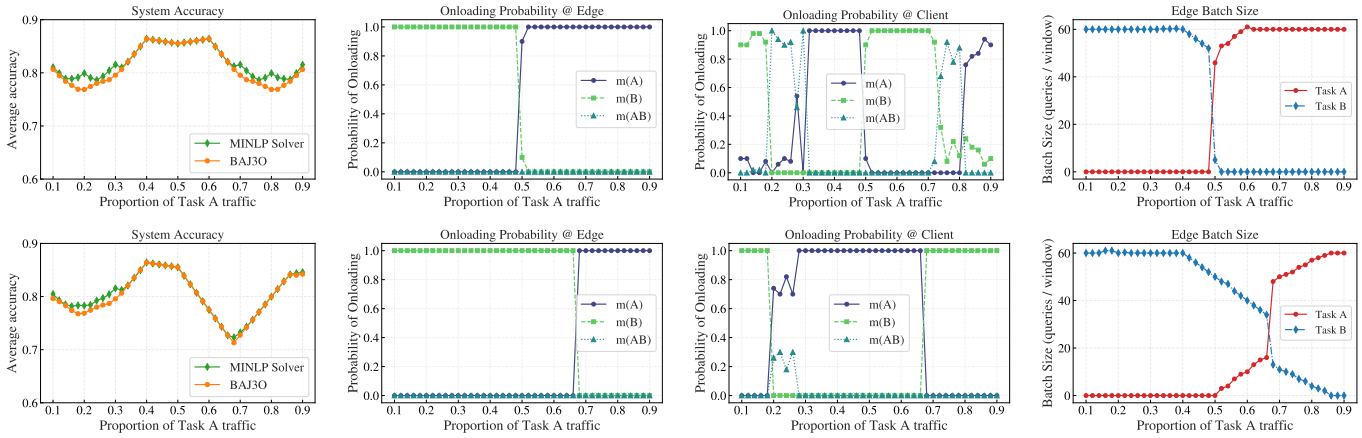
TABLE I: Comparison of related works by objective, model granularity, task scope, and hierarchy of models for inference.

	Objective	Model	Task	Hierarchy
[18]	Accuracy & Latency	Multi	Single	Single-Tier
[21]	Latency & Energy	Multi	Single	Single-Tier
[22]	Latency & Energy	Multi	Single	Two-Tier
[19], [20]	Accuracy	Multi	Single	Two-Tier
[23]	Latency	Multi	Multi	Single-Tier
Ours	Accuracy	Multi	Multi	Multi-Tier

Extending collaborative inference to multi-task scenarios presents new challenges. A single model trained on all tasks may suffer from degraded performance due to task interference and conflicting gradients [11]. On the other hand, maintaining a large pool of single-task models can quickly overwhelm the memory and compute capacity of edge devices. A more scalable strategy is to maintain a small library of specialized multi-task models, each trained on a subset of related tasks, and dynamically select the best model for each inference request [11]–[13]. While this approach has been explored in centralized or cloud-based settings, model selection and deployment under tight system constraints in hierarchical, distributed environments remains largely unaddressed.

The joint model placement and inference offloading has previously been studied primarily in single-task settings, and has considered either a single model or a collection of models. Early efforts in edge intelligence [14]–[17] explored collaborative execution of a single Deep Neural Network (DNN) across edge and cloud, using techniques such as DNN partitioning and early exiting to reduce end-to-end latency by balancing communication and computation. More recent work has extended this to multi-model scenarios, where systems must select, cache, and place multiple candidate models across heterogeneous edge resources [18]–[22]. These approaches aim to optimize inference accuracy, latency, or energy under resource constraints, typically by coordinating model selection and query routing between edge and cloud. However, they remain largely restricted to single-task inference and shallow hierarchies, without jointly optimizing model placement and task routing across clients, edge and cloud. Table I summarizes the distinctions between our approach and these prior methods.

A recent study [23] explores joint multi-task model deployment and task offloading in vehicular edge computing, where vehicles send inference queries to roadside units (RSUs) equipped with shared multi-task models. However, their "one-model-fits-all" approach limits task specialization and may underperform compared to frameworks that rely on tailored models. The framework also lacks hierarchical offloading beyond RSUs, leaving edge device and cloud resources un-



(a) Accuracy (b) Edge Onloading Decision (c) Client Onloading Decision (d) Batch Size

Fig. 1: We compare (top) homogeneous and (bottom) heterogeneous task load distributions across clients. As task A becomes dominant, the optimizer adapts model placement and offloading strategies to maximize batching efficiency and system accuracy.

used. The broader problem of jointly optimizing multi-task model placement and hierarchical task routing under system constraints remains largely unexplored.

To address these challenges, we propose a unified framework for joint model onloading and hierarchical offloading in distributed multi-task inference systems. Each client hosts a tailored subset of multi-task models and selectively offloads queries to higher-tier edge servers. In turn, edge servers may forward difficult requests to the cloud, which stores full-capacity models. The framework jointly determines (i) which models to onload at the client and edge levels and (ii) how to route queries across system tiers to maximize overall accuracy under memory, compute, and communication constraints. To efficiently solve this combinatorial problem, we introduce the **J**oint **O**ptimization of **O**nloading and **O**ffloading (**J3O**) algorithm, an alternating method that combines greedy submodular maximization for model onloading with constrained linear programming for task offloading.

We further extend our formulation to incorporate batching, allowing edge servers to aggregate queries into homogeneous batches to improve GPU utilization. This design aligns with modern accelerator architectures, where batching amortizes fixed inference overhead. In order to bound latency, we impose a time-window constraint on batch formation. To solve the extended problem, we develop **BAJ3O** (**B**atching-**A**ware **J**oint **O**ptimization of **O**nloading, **O**ffloading), which augments J3O with a batching-related latency constraint and integrates it seamlessly into the alternating optimization loop.

The main contributions of this work are:

- We formulate a new joint model onloading and offloading problem for hierarchical multi-task inference, aiming to maximize system-wide accuracy under memory, compute, and communication constraints. This formulation captures task-aware model specialization, multi-tier coordination, and shared resource budgets.
- We propose an alternating optimization algorithm with provable performance guarantees, **J3O** (**J**oint **O**ptimization of **O**nloading and **O**ffloading), which decomposes the mixed-integer nonlinear program into two tractable subproblems: greedy submodular model selection and LP-based task offloading.
- We extend the formulation to incorporate batching for efficient GPU utilization. The resulting **BAJ3O** algorithm (**B**atching-**A**ware **J**oint **O**ptimization of **O**nloading, **O**ffloading) introduces a linear surrogate for batching constraints and integrates it into the J3O framework.
- We evaluate our approach on standard multi-task benchmarks and show consistent improvements in the accuracy–runtime trade-off over the baselines.

A. Motivating Example

To illustrate how joint model onloading and task offloading interact under system constraints, we present a controlled experiment examining two key factors: (i) the effect of task imbalance and traffic distribution, and (ii) the impact of batching overheads on system behavior. We consider a highly simplified two-tier architecture with one edge server and five clients. Each client and the edge node may onload a single model, and the system supports two tasks: A and B.

Three multi-task models are available: (1) $m(A)$, achieving 90% accuracy on task A, 10% on task B; (2) $m(B)$, 90% on A, 10% on B; and (3) $m(AB)$, 60% on both tasks. Client-side models are compressed and achieve 90% of their original accuracy. The edge can host any of the three models at full accuracy. The overall system is constrained by a shared communication bandwidth across clients, per-client compute budgets, and latency-bounded batching at the edge.

a) Impact of Multi-task Traffic: We first explore how task imbalance affects model deployment. In this setting, the system faces tight communication and client-compute constraints. We vary the global fraction of task A traffic p_A from 0.1 to 0.9 while keeping total query volume fixed.

Balanced vs. skewed task loads. Fig. 1 (top) shows the outcome under a homogeneous client-task distribution, i.e., where all clients observe the same fraction of A and B queries.

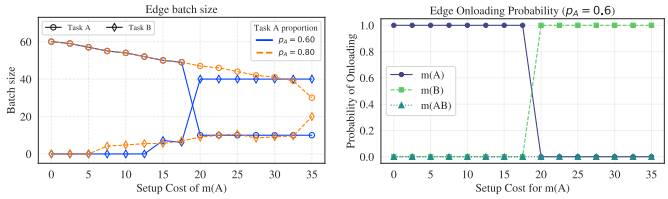


Fig. 2: Higher setup cost for $m(A)$ prompts the system to shift to alternative models or local execution.

When global task traffic is balanced ($p_A = 0.5$), clients offload both tasks to the edge, forming batches of both tasks. As task A becomes dominant, the edge allocates most capacity to A. Once global task A loads exceed the offloading budget, clients begin to onload models to handle residual traffic. Initially, they adopt task-specific models; as traffic becomes more mixed, $m(AB)$ is selected instead. Batch sizes remain stable, but the onloaded model mix adapts to the evolving load.

Fig. 1 (bottom) shows the heterogeneous case, where task B workload is concentrated on a single "hot" client (70% of its queries), while task A remains uniformly distributed. The system continues to prioritize edge offloading of task B due to batching benefits for the hot client. This shifts the transition threshold at which onloading becomes necessary: edge onloading to $m(B)$ persists even when global traffic appears balanced, delaying the need to onload $m(B)$ at clients. When client constraints tighten, the system begins using suboptimal models locally, slightly reducing overall accuracy.

b) Impact of Batching: Next, we consider the impact of batching. Each edge model has a fixed launch cost per batch, better amortized with larger batches, but limiting queries that can be served within a latency bound. We examine how increasing this setup cost influences the optimizer's decisions.

Higher setup cost reduces batching efficiency. Fig. 2 shows the impact of gradually increasing the setup cost for $m(A)$. As the cost rises, batches shrink, reducing the benefit of offloading A. The system compensates by switching to alternative models: at task A proportion $p_A = 0.6$, it adopts $m(B)$ at the edge and relies on client-side model $m(A)$ to handle A tasks. This shows how elevated setup costs can alter onloading strategies to optimize overall system efficiency.

These behaviors highlight the complex interplay between model accuracy, batching efficiency, and system constraints. Even in this simplified setting, optimal decisions emerge from a careful balance of onloading, offloading, and batching, motivating the need for a principled optimization framework which we develop in the next sections.

II. System Model

We consider a three-tier inference system comprising clients $\mathcal{C} = \{1, \ldots, C\}$, edge servers $\mathcal{E} = \{1, \ldots, E\}$, and a centralized cloud. Each client c is connected to a designated edge server e, represented by a binary variable $y^{c,e} \in \{0, 1\}$, and all edge servers connect to the cloud. The system supports a set of ML tasks $\mathcal{T} = \{1, \ldots, T\}$ using a shared library of multi-task models $\mathcal{M} = \{1, \ldots, M\}$, each offering distinct accuracy-cost trade-offs. Clients generate streams of multi-task inference queries and may *onload* a subset of models locally, subject to resource constraints. Queries can also be *offloaded* to the assigned edge server, which can either serve them using its own onloaded models or forward them to the cloud for maximum accuracy. This yields a hierarchical inference strategy, where model placement and task routing decisions jointly determine whether inference occurs at the client, edge, or cloud. Our goal is to maximize the system-wide average inference accuracy, weighted by task load.

A. Inference Model

Each client $c \in \mathcal{C}$ generates a stream of inference queries, each corresponding to a task $t \in \mathcal{T}$. These tasks may include workloads such as edge detection, semantic segmentation, or domain-specific classification. The long-term task demand at client c is represented by a vector $\boldsymbol{\lambda}^c = (\lambda_t^c)_{t=1}^T$, where λ_t^c denotes the average arrival rate of queries for task t, measured in jobs per unit time.

To support client workloads, a library of pre-trained models $\mathcal{M}$ is stored at the cloud. Each model $m \in \mathcal{M}$ may support one or more tasks. Clients and edge servers must select which models to *onload* locally, subject to their resource budgets. We define binary variables $x_m^c, x_m^e \in \{0, 1\}$ to indicate whether model m is onloaded at client c or edge server e, respectively. Given a set of onloaded models, each task query must be served by exactly one model at each tier. To capture this, we introduce binary selection variables $z_{m,t}^c$ and $z_{m,t}^e$, which indicate whether model m is used to serve task t at client c or edge server e, respectively. For convenience, we define the model onloading vectors $\mathbf{x}^c = (x_m^c)_{m \in \mathcal{M}}$ and $\mathbf{x}^e = (x_m^e)_{m \in \mathcal{M}}$, and the task-model selection vectors $\mathbf{z}^c = (z_{m,t}^c)_{m \in \mathcal{M},\, t \in \mathcal{T}}$ and $\mathbf{z}^e = (z_{m,t}^e)_{m \in \mathcal{M},\, t \in \mathcal{T}}$.

Offloading decisions govern how task queries are routed through the system hierarchy. We define continuous variables $o_t^c, o_t^{c,e} \in [0, 1]$, where o_t^c denotes the fraction of task-t queries from client c that are offloaded to its assigned edge server, and $o_t^{c,e}$ denotes the fraction of those edge-level queries that are further offloaded to the cloud. Setting $o_t^c = 0$ corresponds to full local execution at the client, while $o_t^c = 1$ means all task-t queries are sent to the edge. Similarly, $o_t^{c,e} = 1$ implies full offloading from edge to cloud. For convenience, we denote the client- and edge-level offloading vector as $\mathbf{o}^c = (o_t^c)_{t \in \mathcal{T}}$ and $\mathbf{o}^{c,e} = (o_t^{c,e})_{t \in \mathcal{T}}$, respectively.

Each client c is associated with exactly one edge server e, indicated by a binary variable $y^{c,e} \in \{0, 1\}$, such that $\sum_{e \in \mathcal{E}} y^{c,e} = 1$. Clients connected to the same edge server share communication bandwidth, and all edge servers share a common communication link to the cloud. As we describe in the following section, both model onloading and task offloading decisions must respect memory, compute, and communication constraints across all tiers.

B. Accuracy Modeling

We define the system-wide inference accuracy as the average task accuracy, weighted by the long-term task demands across all clients. Let $\mathbf{x} = ((\mathbf{x}^c)_c, (\mathbf{x}^e)_e)$ denote the model

onloading decisions at the client and edge levels, and let $\mathbf{z} = ((\mathbf{z}^c)_c, (\mathbf{z}^e)_e)$ capture the selected inference model for each task at each client and edge server. To ensure that only onloaded models are used for inference, we enforce that

$$z^c_{m,t} \le x^c_m, \quad \sum\nolimits_{m\in\mathcal{M}} z^c_{m,t} = 1,\ \forall m,t,c, \tag{1a}$$

$$z^e_{m,t} \le x^e_m, \quad \sum\nolimits_{m\in\mathcal{M}} z^e_{m,t} = 1,\ \forall m,t,e. \tag{1b}$$

Let $\mathbf{o} = ((\mathbf{o}^c)_c, (\mathbf{o}^{c,e})_{c,e})$ denote the offloading decisions from clients to edges and from edges to the cloud. Define the total system demand as $\lambda_{\text{tot}} := \sum_{t\in\mathcal{T}} \sum_{c\in\mathcal{C}} \lambda^c_t$, and the normalized task demand for client c as $\bar{\lambda}^c_t := \lambda^c_t / \lambda_{\text{tot}}$. The overall system-wide accuracy is then

$$\begin{aligned} F(\mathbf{x},\mathbf{z},\mathbf{o}) = &\sum_{c\in\mathcal{C}} F_{\text{client}}(\mathbf{x}^c,\mathbf{z}^c,\mathbf{o}^c) \\ &+ \sum_{e\in\mathcal{E}}\sum_{c\in\mathcal{C}_e} [F_{\text{edge}}(\mathbf{x}^e,\mathbf{z}^e,\mathbf{o}^c,\mathbf{o}^{c,e}) + F_{\text{cloud}}(\mathbf{o}^{c,e})], \end{aligned} \tag{2}$$

where $\mathcal{C}_e := \{c \in \mathcal{C} \,|\, y^{c,e} = 1\}$ is the set of clients assigned to edge server e. This objective captures average task accuracy across client, edge, and cloud levels:

(1) Client-side accuracy reflects the accuracy of locally served queries,

$$F_{\text{client}}(\mathbf{x}^c,\mathbf{z}^c,\mathbf{o}^c) = \sum_{t\in\mathcal{T}} \bar{\lambda}^c_t (1 - o^c_t) A^c_t(\mathbf{x}^c,\mathbf{z}^c), \tag{3}$$

where $A^c_t(\cdot)$ denotes the accuracy for task t under client-side execution,

$$A^c_t(\mathbf{x}^c,\mathbf{z}^c) = \sum_{m\in\mathcal{M}}\sum_{e\in\mathcal{E}} y^{c,e} a^e_{m,t} x^c_m z^c_{m,t}. \tag{4}$$

Here, $y^{c,e}$ indicates the client's edge assignment, and $a^e_{m,t}$ denotes the accuracy of model m on task t under edge-e data distribution. This captures location-dependent accuracy variations in mobile settings. Tasks unsupported by a model are assigned zero accuracy.

(2) Edge-side accuracy accounts for queries handled at the edge but not forwarded to the cloud,

$$F_{\text{edge}}(\mathbf{x}^e,\mathbf{z}^e,\mathbf{o}^c,\mathbf{o}^{c,e}) = \sum\nolimits_{t\in\mathcal{T}} \bar{\lambda}^c_t (o^c_t - o^{c,e}_t) A^e_t(\mathbf{x}^e,\mathbf{z}^e), \tag{5}$$

where

$$A^e_t(\mathbf{x}^e,\mathbf{z}^e) = \sum\nolimits_{m\in\mathcal{M}} a^e_{m,t}\, x^e_m z^e_{m,t} \tag{6}$$

denotes the accuracy of edge-level inference.

(3) Cloud-side accuracy captures queries ultimately served by the cloud,

$$F_{\text{cloud}}(\mathbf{o}^{c,e}) = \sum\nolimits_{t\in\mathcal{T}} \bar{\lambda}^c_t o^{c,e}_t A^s_t, \tag{7}$$

where A^s_t is the oracle-level accuracy for task t at the cloud.

C. Resource Constraints

1) Memory and Computation Constraints: Clients and edge servers have limited memory capacity, denoted by μ^c and μ^e, respectively. Each model $m \in \mathcal{M}$ requires s_m bytes of memory. Onloaded models must fit within the available memory, i.e.,

$$\sum\nolimits_{m\in\mathcal{M}} x^c_m s_m \le \mu^c, \tag{8a}$$

$$\sum\nolimits_{m\in\mathcal{M}} x^e_m s_m \le \mu^e. \tag{8b}$$

We consider only model parameter memory, assuming input data is small relative to model size.

Inference also consumes compute resources. Let w_m denote the per-query compute cost of model m. For client c, the expected compute load, weighted by task arrival rate λ^c_t and local execution ratio $1 - o^c_t$, must not exceed its compute capacity β^c, i.e.,

$$\sum_{m\in\mathcal{M}}\sum_{t\in\mathcal{T}} \lambda^c_t (1 - o^c_t) \cdot w_m \cdot x^c_m z^c_{m,t} \le \beta^c. \tag{9}$$

Similarly, edge server e must serve its assigned load within compute budget β^e, that is

$$\sum_{m\in\mathcal{M}}\sum_{t\in\mathcal{T}}\sum_{c\in\mathcal{C}_e} \lambda^c_t (o^c_t - o^{c,e}_t) \cdot w_m \cdot x^e_m z^e_{m,t} \le \beta^e. \tag{10}$$

We assume the cloud has sufficient capacity to store the entire model library and handle any forwarded queries.

2) Shared Offloading Constraint: Communication bandwidth is shared across multiple system tiers. Let d_t denote the input data size for task t. For each edge server $e \in \mathcal{E}$, the total data offloaded from its assigned clients must not exceed its uplink bandwidth budget κ^e (in bytes per unit time),

$$\sum_{t\in\mathcal{T}}\sum_{c\in\mathcal{C}_e} \lambda^c_t o^c_t d_t \le \kappa^e. \tag{11}$$

Similarly, the cloud server is subject to a global communication constraint κ^s. The total traffic forwarded from all edge servers must satisfy

$$\sum_{t\in\mathcal{T}}\sum_{e\in\mathcal{E}}\sum_{c\in\mathcal{C}_e} \lambda^c_t o^{c,e}_t d_t \le \kappa^s. \tag{12}$$

These constraints regulate network-level data transmission and implicitly bound system latency. As bandwidth approaches saturation, queuing and congestion may increase, causing delay. Limiting offloading volumes helps maintain timely task execution by preventing overload-induced latency.

III. Joint Onloading and Offloading

A. Problem Formulation

We formulate the joint model onloading and offloading problem as the maximization of system-wide average inference accuracy across all tasks,

$$P1: \max_{\mathbf{x},\mathbf{z},\mathbf{o}} F(\mathbf{x},\mathbf{z},\mathbf{o}) \tag{13a}$$

$$\text{s.t. } (1), (8) - (12), \tag{13b}$$

$$o^{c,e}_t \le o^c_t\, y^{c,e}, \quad \forall t,c,e \tag{13c}$$

$$x^c_m, z^c_{m,t} \in \{0,1\}, \quad \forall m,t,c \tag{13d}$$

$$x^e_m, z^e_{m,t} \in \{0,1\}, \quad \forall m,t,e \tag{13e}$$

$$o^c_t, o^{c,e}_t \in [0,1], \quad \forall t,c,e. \tag{13f}$$

Here, the objective $F(\cdot)$ captures the accuracy contributions from client-, edge-, and cloud-level inference as previously defined. The constraints enforce valid model selection (1), resource limits (8–12), hierarchical offloading consistency (13c), and variable domains (13d–13f).

This is a *mixed-integer nonlinear program* (MINLP). Even a simplified version reduces to the 0-1 knapsack problem, which is NP-hard. Hence, the full joint onloading–offloading problem is also NP-hard.

B. Problem Decomposition

To address the complexity of the original MINLP, we decompose it into two tractable subproblems – model onloading and inference offloading. The onloading selects models to deploy and forms a constrained submodular maximization. The offloading assigns inference queries across the hierarchy and reduces to a constrained linear program.

Model Onloading Subproblem. Given fixed offloading decisions $\mathbf{o}$, model onloading decomposes across clients and edges, each forming a constrained submodular maximization. For client c, the problem is

$$\max_{\mathbf{x}^c,\mathbf{z}^c} F_{\text{client}}(\mathbf{x}^c,\mathbf{z}^c) := \sum\nolimits_{t\in\mathcal{T}} \bar{\lambda}_t^c(1-o_t^c)A_t^c(\mathbf{x}^c,\mathbf{z}^c). \quad (14)$$
$$\text{s.t.} \quad (1a), (8a), (9), (13d).$$

This maximizes average client-side accuracy subject to memory, compute, and assignment constraints. Let $\mathcal{M}^c \subseteq \mathcal{M}$ be the models selected by client c. The objective simplifies to

$$f(\mathcal{M}^c) = \sum\nolimits_{t\in\mathcal{T}} \bar{\lambda}_t^c(1-o_t^c) \max_{m\in\mathcal{M}^c} a_{m,t}^e, \quad (15)$$

where $a_{m,t}^e$ captures the model's task accuracy under the data distribution at edge e, to which client c is assigned.

Proposition 1. *For a fixed offloading decision $\mathbf{o}^c$, the client-side objective $f(\mathcal{M}^c)$ is monotone and submodular.*

Proof. Monotonicity follows directly – adding a model to $\mathcal{M}^c$ cannot reduce the maximum accuracy for any task.

To show submodularity, let $\mathcal{S} \subseteq \mathcal{T} \subseteq \mathcal{M}^c$ and $m \in \mathcal{M}^c \backslash \mathcal{T}$. For each task $t \in \mathcal{T}$, define $a_{\mathcal{S}}(t) := \max_{m'\in\mathcal{S}} a_{m',t}^e$ and $a_{\mathcal{T}}(t) := \max_{m'\in\mathcal{T}} a_{m',t}^e$, so $a_{\mathcal{S}}(t) \le a_{\mathcal{T}}(t)$. The marginal gain of adding m to set $\mathcal{S}$ is $\Delta_f(t; m \mid \mathcal{S}) = \bar{\lambda}_t^c(1-o_t^c)\big(\max\{a_{\mathcal{S}}(t), a_{m,t}^e\} - a_{\mathcal{S}}(t)\big)$. Hence, by submodularity, the marginal gain must exhibit diminishing returns, i.e., $\Delta_f\big(t; m \mid \mathcal{S}\big) \ge \Delta_f\big(t; m \mid \mathcal{T}\big)$, which we verify as follows:

(1) If $a_{m,t}^e \le a_{\mathcal{S}}(t)$, then the marginal gain is zero for both sets, i.e., $\Delta_f(t; m \mid \mathcal{S}) = \Delta_f(t; m \mid \mathcal{T}) = 0$.

(2) If $a_{m,t}^e > a_{\mathcal{S}}(t)$, then $\Delta_f(t; m \mid \mathcal{S}) = \bar{\lambda}_t^c(1-o_t^c)\big(a_{m,t}^e - a_{\mathcal{S}}(t)\big)$ and $\Delta_f(t; m \mid \mathcal{T}) = \bar{\lambda}_t^c(1-o_t^c)\big(\max\{a_{m,t}^e, a_{\mathcal{T}}(t)\} - a_{\mathcal{T}}(t)\big)$. Since $a_{\mathcal{S}}(t) \le a_{\mathcal{T}}(t)$, the difference $a_{m,t}^e - a_{\mathcal{S}}(t)$ is at least as large as $\max\{a_{m,t}^e, a_{\mathcal{T}}(t)\} - a_{\mathcal{T}}(t)$, proving the inequality.

Summing over t confirms submodularity of $f(\mathcal{M}^c)$. □

Next, define the effective task load handled at edge server e as $\lambda_t^e := \sum_{c\in\mathcal{C}_e} \lambda_t^c(o_t^c - o_t^{c,e})$ and normalize it as $\bar{\lambda}_t^e = \lambda_t^e/\lambda_{\text{tot}}$. Since the cloud-side accuracy is fixed under given offloading rates, the edge-side subproblem becomes

$$\max_{\mathbf{x}^e,\mathbf{z}^e} F_{\text{edge}}(\mathbf{x}^e,\mathbf{z}^e) := \sum\nolimits_{t\in\mathcal{T}} \bar{\lambda}_t^e A_t^e(\mathbf{x}^e,\mathbf{z}^e) \quad (16)$$
$$\text{s.t.} \quad (1b), (8b), (10), (13e).$$

As with the client-side case, this objective defines a weighted coverage function over tasks, and is thus monotone and submodular in the set of selected model-task pairs. With fixed offloading, the full objective in (2) becomes the sum of two monotone submodular functions (i.e., client-side and edge-side terms) plus a constant cloud-side term $F_{\text{cloud}}(\cdot)$. Hence, the overall model onloading problem reduces to a monotone submodular maximization.

Offloading Subproblem Given fixed model selections $\mathbf{x}$ and task assignments $\mathbf{z}$ from the onloading stage, the offloading subproblem determines the optimal offloading rates $\mathbf{o}$ to maximize the average system accuracy while satisfying system-wide compute and communication constraints. The resulting formulation is a constrained linear program

$$\max_{\mathbf{o}} \sum_{t\in\mathcal{T}}\sum_{e\in\mathcal{E}}\sum_{c\in\mathcal{C}_e} \bar{\lambda}_t^c[(1-o_t^c)A_t^c(\mathbf{x}^c,\mathbf{z}^c) + (o_t^c - o_t^{c,e})A_t^e(\mathbf{x}^e,\mathbf{z}^e) + o_t^{c,e}A_t^s] \quad (17)$$
$$\text{s.t.} \quad (9) - (12), (13c), (13f).$$

IV. Algorithm Design

A. Alternating Optimization

To solve the joint model onloading and offloading problem $P1$ in (13), we develop an alternating optimization algorithm, **J3O** (**J**oint **O**ptimization of **O**nloading and **O**ffloading). Alternating optimization (AO) is well suited for large-scale problems where variables decompose naturally into subproblems that are easier to solve individually than jointly [24], [25]. In our case, the problem is split into two interleaved stages: (i) model onloading, addressed via greedy submodular maximization with Lagrangian relaxation, and (ii) offloading, formulated as a constrained linear program. Each subproblem is solved while fixing the variables of the other, and the process iterates until convergence.

1) Greedy Submodular Maximization via Lagrangian Relaxation: The onloading subproblem maximizes a monotone submodular objective over binary variables $\mathbf{x}$ and $\mathbf{z}$, subject to memory and compute constraints. While greedy algorithms guarantee a $(1-1/e)$ approximation under a single knapsack constraint [26], [27], our problem includes compute constraints that couple both $\mathbf{x}$ and $\mathbf{z}$. These interactions preclude direct use of standard greedy methods, motivating a Lagrangian relaxation to decouple the dependencies and enable tractable optimization.

2) Greedy Submodular Maximization via Lagrangian Relaxation: Let $\boldsymbol{\alpha} = ((\alpha^c)_c, (\alpha^e)_e)$ denote the Lagrange multipliers for the client and edge compute constraints. The corresponding dual objective is

$$\mathcal{L}(\mathbf{x},\mathbf{z};\,\boldsymbol{\alpha}) = F(\mathbf{x},\mathbf{z};\mathbf{o}) - \sum_{c\in\mathcal{C}} \alpha^c \left(\sum_{m,t} \bar{\lambda}_t^c(1-o_t^c)w_m x_m^c z_{m,t}^c - \beta^c/\lambda_{\text{tot}}\right) - \sum_{e\in\mathcal{E}} \alpha^e \left(\sum_{m,t,c} \bar{\lambda}_t^c(o_t^c - o_t^{c,e})w_m z_{m,t}^e - \beta^e/\lambda_{\text{tot}}\right). \quad (18)$$

Algorithm 1: J3O Algorithm

Input: Model pool $\mathcal{M}$, tolerance δ; maximum iterations $N_{\max}$.
Output: Onloading decisions $\mathbf{x}^\star, \mathbf{z}^\star$ and offloading rates $\mathbf{o}^\star$.

Initialization: Greedily set $\mathbf{o}^{c,(0)}$ proportional to client task loads, under constraints; $\mathbf{o}^{c,e,(0)} \leftarrow \mathbf{0}$; $F(\mathbf{o}^{(0)}; \mathbf{x}^{(0)}, \mathbf{z}^{(0)}) = F_{\text{best}} \leftarrow -\infty$.
for $k = 1, \ldots, N_{\max}$ **do** // outer AO loop
  /* Greedy Lagrangian onloading */
  Fix $\mathbf{o}^{(k-1)}$ and $(\mathbf{x}^{\text{new}}, \mathbf{z}^{\text{new}}) \leftarrow$ GREEDY-LR $(\mathbf{o}^{(k-1)})$
  if $F(\mathbf{x}^{\text{new}}, \mathbf{z}^{\text{new}}; \mathbf{o}^{(k-1)}) > F(\mathbf{o}^{(k-1)}; \mathbf{x}^{(k-1)}, \mathbf{z}^{(k-1)})$ **then**
    $\mathbf{x}^{(k)} \leftarrow \mathbf{x}^{\text{new}}$, $\mathbf{z}^{(k)} \leftarrow \mathbf{z}^{\text{new}}$
  else
    $\mathbf{x}^{(k)} \leftarrow \mathbf{x}^{(k-1)}$, $\mathbf{z}^{(k)} \leftarrow \mathbf{z}^{(k-1)}$
  /* LP-based offloading */
  Solve the linear program with fixed $\mathbf{x}^{(k)}, \mathbf{z}^{(k)}$ to obtain $\mathbf{o}^{(k)}$.
  if $F(\mathbf{o}^{(k)}; \mathbf{x}^{(k)}, \mathbf{z}^{(k)}) - F_{\text{best}} < \delta$ **then**
    break
  $F_{\text{best}} \leftarrow F(\mathbf{o}^{(k)}; \mathbf{x}^{(k)}, \mathbf{z}^{(k)})$
return $(\mathbf{x}^\star, \mathbf{z}^\star, \mathbf{o}^\star) \leftarrow (\mathbf{x}^{(k)}, \mathbf{z}^{(k)}, \mathbf{o}^{(k)})$.

The model onloading step is then reformulated as a Lagrangian dual problem

$$\min_{\boldsymbol{\alpha}} \max_{\mathbf{x},\mathbf{z}} \mathcal{L}(\mathbf{x}, \mathbf{z}; \boldsymbol{\alpha}) \quad \text{s.t.} \quad (1), (8).$$

For fixed multipliers $\boldsymbol{\alpha}$, the inner maximization decouples across nodes (clients, edge), yielding a *monotone submodular maximization* under a linear memory constraint. Each resulting subproblem can be efficiently approximated using a *greedy algorithm* with provable approximation guarantees. Specifically, the greedy procedure selects models iteratively by maximizing marginal gain per unit memory $\Delta\mathcal{L}^v(m; \mathcal{M}^v_{(i)})/s_m$, where $\Delta\mathcal{L}^v(m; \mathcal{M}^v_{(i)}) = \mathcal{L}^v(\mathcal{M}^v_{(i)} \cup m) - \mathcal{L}^v(\mathcal{M}^v_{(i)})$ is the gain in Lagrangian objective for node $v \in \mathcal{V} = \mathcal{C} \cup \mathcal{E}$ when adding model m to the current model set $\mathcal{M}^v_{(i)}$ at iteration i. The full procedure is formalized as Algorithm 2.

After solving the inner problem via the greedy algorithm, the Lagrange multipliers are updated using a subgradient method [28] as

$$\begin{aligned} \alpha^c &\leftarrow \left[\alpha^c + \frac{\eta^c}{\sqrt{k}} \left(\sum_{m,t} \bar{\lambda}^c_t (1 - o^c_t) w_m x^c_m z^c_{m,t} - \frac{\beta^c}{\lambda_{\text{tot}}} \right)\right]^+, \\ \alpha^e &\leftarrow \left[\alpha^e + \frac{\eta^e}{\sqrt{k}} \left(\sum_{m,t,c} \bar{\lambda}^c_t (o^c_t - o^{c,e}_t) w_m z^e_{m,t} - \frac{\beta^e}{\lambda_{\text{tot}}} \right)\right]^+. \end{aligned} \tag{19}$$

Here, k is the iteration index, η^c and η^e are step size parameters, and $[\cdot]^+$ denotes projection onto the non-negative orthant. The updates proceed until constraint violations fall below a target threshold, yielding a near-feasible and near-optimal solution to the original problem.

3) Offloading Optimization via Linear Programming: Given fixed onloading decisions $(\mathbf{x}, \mathbf{z})$ and the resulting accuracy matrices $A^c_t(\cdot,\cdot)$, $A^e_t(\cdot,\cdot)$ and A^s_t, we optimize the continuous offloading variables $\mathbf{o}$ by solving a constrained linear program. This can be efficiently handled using standard solvers such as Gurobi or CPLEX, yielding optimal offloading strategies across the hierarchy (client $\rightarrow$ edge $\rightarrow$ cloud).

4) Full Algorithm and Complexity Analysis: Algorithm 1 summarizes the proposed J3O procedure. To ensure convergence and stability, we adopt a conservative update rule: the onloading decision at iteration k is accepted only if it improves the overall objective relative to iteration $k-1$; otherwise, the previous decision is retained. This ensures the objective is non-decreasing over iterations, i.e., $F(\mathbf{x}^{(k)}, \mathbf{z}^{(k)}, \mathbf{o}^{(k)}) \geq F(\mathbf{x}^{(k-1)}, \mathbf{z}^{(k-1)}, \mathbf{o}^{(k-1)})$, $\forall k$. The algorithm terminates once the improvement falls below a predefined threshold δ, or a maximum number of iterations is reached. The final output is a near-optimal joint onloading–offloading configuration that balances accuracy and resource usage.

Algorithm 2: GREEDY–LR$(\mathbf{o})$

Input: Fixed offloading rates $\mathbf{o}$; node sets $\mathcal{V} = \mathcal{C} \cup \mathcal{E}$; tolerance ε.
Output: Binary onloading variables $(\mathbf{x}, \mathbf{z})$

foreach $v \in \mathcal{V}$ **do**
  $\alpha^v \leftarrow 0$, $\mathcal{M}^v_{(0)} \leftarrow \emptyset$ // model set at node v
for $\ell = 1$ **to** $L_{\max}$ **do** // Subgradient iteration
  foreach $v \in \mathcal{V}$ **do**
    while *True* **do** // Greedy set selection
      $m^\star \leftarrow \text{argmax}_{m \in \mathcal{M} \setminus \mathcal{M}^v_{(\ell)}} \Delta\mathcal{L}^v(m; \mathcal{M}^v_{(\ell)}, \mathbf{o})/s_m$
      if adding $m^\star$ keeps memory budget μ^v *feasible* **then**
        $\mathcal{M}^v_{(\ell)} \leftarrow \mathcal{M}^v_{(\ell)} \cup \{m^\star\}$
      else
        break
    Construct (x^v, z^v) from $\mathcal{M}^v_{(\ell)}$
    $\alpha^v \leftarrow [\alpha^v + \eta_\ell \cdot viol_v(\mathcal{M}^v_{(\ell)})]^+$ in (19)
    if $\text{viol}_v(\mathcal{M}^v_{(\ell)}) < \varepsilon$ **then**
      break
return $(\mathbf{x}, \mathbf{z})$

We analyze the runtime of J3O by focusing on the model onloading subproblem, as the LP-based offloading step is efficiently solvable. Let K be the number of subgradient updates to the Lagrange multipliers. In each update, the greedy routine for each node $c \in \mathcal{C}$, $e \in \mathcal{E}$ evaluates the marginal gain of all M candidate models at most B^c, B^e times, respectively, bounded by memory budgets. Summing across nodes, the per-update cost is $\mathcal{O}\big(M(\sum_c B^c + \sum_e B^e)\big)$. Thus, one outer iteration has total cost $\mathcal{O}\big(KM(\sum_c B^c + \sum_e B^e)\big)$, which is linear in the number of models and memory budgets.

B. Optimality Guarantee of Algorithm

To establish a theoretical guarantee, we first analyze a simplified two-tier setting with clients and a central server; the extension to three tiers follows analogously.

a) Optimality Guarantee for Two-Level System: We analyze the solution (x^n, o^n) returned at iteration n of our alternating optimization scheme for a two-level system comprising clients and a central server. The system objective is given by

$$F(x, o) = 1 - \sum\nolimits_t \bar{\lambda}_t (1 - o_t)(1 - A_t(x)), \tag{20}$$

where $x \in \mathcal{X}$ denotes the binary model onloading decision under memory, and $o \in \mathcal{O}(x)$ specifies the offloading rates subject to compute and communication budgets. The global optimum is denoted (x^*, o^*).

Assumption 1 (Monotone Onloading)**.** *At each iteration $k = 1, \ldots, n$, let x' be the onloading solution proposed by the greedy subroutine (given fixed o^{k-1}). The update is accepted only if it improves or maintains the objective, i.e., $F(x', o^{k-1}) - F(x^{k-1}, o^{k-1}) \geq 0$. Otherwise, the previous onloading decision is retained, i.e., $x^k = x^{k-1}$.*

Assumption 2 (Bounded Offloading Gap)**.** *Let o^l denote the offloading solution obtained at iteration l after convergence, and suppose $\|o^l - o^*\|_\infty \le \epsilon$. Then the resulting objective is within ϵ of the global optimum, which follows from*

$$\begin{aligned} F(x^*, o^*) - F(x^*, o^l) &= \sum\nolimits_t \bar{\lambda}_t (o_t^* - o_t^l)(1 - A_t(x^*)) \\ &\le \sum\nolimits_t \bar{\lambda}_t \cdot |o_t^l - o_t^*| \\ &\le \max_t |o_t^l - o_t^*| \le \epsilon. \end{aligned} \tag{21}$$

Lemma 1 (LP-Step Monotonicity)**.** *Given a fixed onloading decision x, the offloading step, solved as an LP, yields a non-decreasing objective across iterations k. That is, $F(x, o^k) \ge F(x, o^{k-1})$, where $o^k = \arg\max_{o \in \mathcal{O}(x)} F(x, o)$.*

Proof. By definition of the maximizer, $o^{k-1} \in \mathcal{O}(x)$ and o^k is the optimal solution over the feasible set $\mathcal{O}(x)$. Therefore, $F(x, o^k) = \max_{o \in \mathcal{O}(x)} F(x, o) \ge F(x, o^{k-1})$. □

Lemma 2 (Greedy Onloading with Lagrangian Relaxation)**.** *Suppose the onloading problem is solved via greedy submodular maximization under relaxed constraints. For fixed o^k, let $\alpha^* \ge 0$ be the Lagrange multiplier such that the greedy solution x^{k+1} satisfies the relaxed constraint. Then, $F(x^{k+1}, o^k) \ge (1 - 1/e) \max_{x \in \mathcal{X}} F(x, o^k)$.*

Proof. Let $\Delta_k := \beta - \phi(x^{k+1}, o^k) \ge 0$ be the slack in the relaxed constraint. Define the Lagrangian as $\mathcal{L}(x, o; \alpha) = F(x, o) - \alpha[\phi(x, o) - \beta]$. The greedy algorithm ensures

$$\mathcal{L}(x^{k+1}, o^k; \alpha^*) \ge (1 - 1/e) \max_{x \in \mathcal{X}} \mathcal{L}(x, o^k; \alpha^*), \tag{22}$$

$$\max_{x \in \mathcal{X}} \mathcal{L}(x, o^k; \alpha^*) \ge \max_{x \in \mathcal{X}} F(x, o^k). \tag{23}$$

Since $F(x^{k+1}, o^k) = \mathcal{L}(x^{k+1}, o^k; \alpha^*) - \alpha^* \Delta_k$, we conclude

$$F(x^{k+1}, o^k) \ge (1 - 1/e) \max_{x \in \mathcal{X}} F(x, o^k) - \alpha^* \Delta_k. \tag{24}$$

Finally, the dual update $\alpha \leftarrow \max\{0, \alpha - \eta \Delta_k\}$ ensures $|\alpha^* \Delta_k| \le \varepsilon \approx 10^{-5}$, making the additive loss negligible. □

Theorem 1 (Final Guarantee)**.** *Under Assumptions 1 and 2, the final iterate (x^n, o^n) returned by the alternating optimization satisfies $F(x^n, o^n) \ge (1 - 1/e)[F(x^*, o^*) - \epsilon]$.*

Proof. By Lemma 1, the objective is non-decreasing over LP steps, $F(x^n, o^n) \ge F(x^n, o^{n-1}) \ge \cdots \ge F(x^{l+1}, o^l)$. From Lemma 2 and Assumption 2, we have

$$\begin{aligned} F(x^{l+1}, o^l) &\ge (1 - 1/e) \max_x F(x, o^l) \\ &\ge (1 - 1/e) F(x^*, o^l) \\ &\ge (1 - 1/e)[F(x^*, o^*) - \epsilon], \end{aligned} \tag{25}$$

which completes the proof. □

For a system with clients, edges, and a central cloud with oracle accuracy, the offloading gap in Assumption 2 becomes

$$\begin{aligned} &F(x^*, o^*) - F(x^*, o^k) \\ &\quad = \sum\nolimits_t \bar{\lambda}_t^c (A_t^c(x^*) - A_t^e(x^*))(o_t^{c,k} - o_t^{c,*}) \\ &\quad + \sum\nolimits_t \bar{\lambda}_t^c (1 - A_t^e(x^*))(o_t^{c,e,k} - o_t^{c,e,*}). \end{aligned} \tag{26}$$

This shows that the total objective difference is again bounded by deviations in the offloading variables o^c and $o^{c,e}$.

V. Batching-Aware Extension

A. Batching Model for Edge Inference

To capture the operational characteristics of edge accelerators, we extend our framework to incorporate batching, amortizing the per-query inference cost and introducing latency constraints tied to batch formation and execution. Each edge server collects incoming queries over a T_b forming homogeneous batches, each with single-task queries. This design mirrors typical execution patterns in multi-task models, where inputs in the same batch are routed to shared task-specific components (e.g., classifier heads).

The resulting mean batch size for task t at edge server e is $b_t^e = \lambda_t^e T_b$. Empirical studies show batch execution latency grows linearly with batch size [29], [30]. We therefore model the total processing latency for task t at the edge e as

$$\tau_t^e(b_t^e) = \sum_{m \in \mathcal{M}} \left(\nu_m^e \mathbb{1}_{b_t^e} + \tau_m^e \, b_t^e\right) x_m^e z_{m,t}^e, \quad \tau_m^e = \frac{w_m}{\beta^e}, \tag{27}$$

where ν_m^e denotes the model-specific batch setup cost, w_m is the per-query compute cost, and β^e is the compute capacity of edge server e. Here, batching reduces per-query latency by amortizing the fixed ν_m^e over multiple queries within a batch.

To ensure that on average, each batch initiated within an interval is executed before the subsequent interval begins, we impose a per-edge batching-latency constraint:

$$\sum_{t \in \mathcal{T}} \sum_{m \in \mathcal{M}} \left(\nu_m^e \mathbb{1}_{\lambda_t^e} + \frac{w_m}{\beta^e} \lambda_t^e T_b\right) x_m^e z_{m,t}^e \le T_b, \; \forall e, \tag{28}$$

ensuring that the cumulative execution time of all task-specific batches launched on edge e does not exceed the batching interval. This provides a conservative guarantee that the processing delay for each request does not exceed $2T_b$.

B. Batching-Aware Joint Optimization

We extend the J3O framework to incorporate batching-related latency constraints, resulting in the ***B**atching-**A**ware **J**oint **O**ptimization of **O**nloading, **O**ffloading* (**BAJ3O**) algorithm. A core challenge in this setting arises from the nonlinearity introduced by the indicator term $\mathbb{1}_{\lambda_t^e}$, which captures whether task t receives queries at edge e. This term is equivalent to the ℓ_0-norm, $\|\lambda_t^e\|_0$, making the constraint non-convex. Following the surrogate approximation approach in [30], we linearize this term using a first-order Taylor expansion, yielding the relaxed constraint

$$\sum_{t \in \mathcal{T}} \sum_{m \in \mathcal{M}} \left(\nu_m^e (\theta_t^e \lambda_t^e + \psi_t^e) + \frac{w_m}{\beta^e} \lambda_t^e T_b\right) x_m^e z_{m,t}^e \le T_b, \; \forall e, \tag{29}$$

where θ_t^e and ψ_t^e are iteration–specific coefficients derived from the surrogate function and updated to refine the approximation. Further details can be found in Section IV of [30].

Leveraging this relaxation, BAJ3O performs alternating optimization over model onloading and offloading in the presence of batching. At each iteration, the batching constraint is first linearized using the current estimates of the surrogate coefficients θ_t^e and ψ_t^e. We then solve the onloading and offloading subproblems following the same procedure as in the original

TABLE II: Models and input statistics for each benchmark.

Model	# Param	Mem (MB)	GFLOPs	Input Mem (MB)
Xception	18.41M	73.66	9.17	0.79
ResNet34	21.80M	83.15	3.68	0.60

(a) Taskonomy and DomainNet.

Metric	m_1	m_2	m_3	$m_{\{1,2\}}$	$m_{\{1,3\}}$	$m_{\{2,3\}}$	$m_{\{1,2,3\}}$
Mem (GB)-FP32	1.10	1.10	1.14	1.18	1.22	1.22	1.30
Mem (GB)-INT8	0.80	0.80	0.84	0.88	0.91	0.91	0.99
TFLOPs	1.13	1.13	0.73	1.84	1.44	1.44	2.16
Input Mem (MB)	25.17 (shared across models)						

(b) Cityscape3D. Numbers in braces denote supported tasks.

TABLE III: Specifications of client (C1–C5) and edge (E1–E3) devices across benchmarks. Task = Taskonomy, Dom = DomainNet, City = Cityscape3D.

Dataset	Metric	C1	C2	C3	C4	C5	E1	E2	E3
Task/Dom	Mem (MB)	24	24	48	48	96	512	512	1024
	GFLOPs	0.5K	1.0K	1.0K	2.0K	2.0K	10.0K	12.0K	15.0K
City	Mem (GB)	1.0	2.0	2.0	3.0	4.0	5.0	6.0	6.0
	TFLOPs	10.0	10.0	15.0	15.0	20.0	30.0	40.0	50.0

J3O algorithm, with the key difference that the edge compute constraint is now replaced by the batching-latency constraint. Once the model selections and offloading rates are updated, the surrogate coefficients are recalculated based on the updated effective task arrival rates $(\lambda_t^e)_{t\in\mathcal{T},e\in\mathcal{E}}$, reflecting the new batch sizes. This alternating process (consisting of constraint linearization, subproblem optimization, and surrogate update), repeats until convergance.

VI. Simulation Results

A. Experimental Setup

1) Datasets and Models: We evaluate our framework on three real-world multi-task benchmarks spanning diverse vision tasks and model architectures. **(1) Taskonomy-5** [31] comprises 5 indoor vision tasks. We evaluate performance using the task-wise losses reported in [11], with Xception backbones [32] and task-specific decoders. **(2) DomainNet-6** [33] is a multi-domain image classification benchmark covering 6 domains and 100 shared classes. Each domain is treated as a separate task. We fine-tune a shared ResNet-34 encoder with domain-specific heads [34] and report test accuracy. **(3) Cityscape3D-3** [35] is a 3-task benchmark for urban 3D perception, covering object detection, semantic segmentation, and depth estimation. We use TaskPrompter [8] models, deploying full-precision variants on edge servers and INT8-quantized versions on clients. Performance is measured using task-specific test loss.

Table II summarizes model configurations. In Taskonomy and DomainNet, models are executed in full precision by default, with INT8 client-side variants cached at 25% memory cost. Due to hard parameter sharing [12], [13], these backbones yield nearly uniform cost profiles across tasks. While this symmetry simplifies system behavior, our framework naturally extends to heterogeneous model profiles, as demonstrated in Cityscape3D. The cloud tier is assumed to host an oracle model with ideal task accuracy or loss.

2) Device and System Parameters: We study performance under heterogeneous hardware configurations in Table III. Taskonomy and DomainNet represent smaller-scale deployments, with clients modeled after low-power devices (e.g., NVIDIA Jetson Nano or TX2) and edge servers with A10-class GPUs. In contrast, Cityscape3D reflects large-scale deployments with higher-end hardware, e.g., clients modeled after NVIDIA RTX 2080 or Mobile 500-class devices and edge servers with H200-class GPUs. Clients and edge servers are allocated 1–3 and 4–6 models, respectively, reflecting realistic storage constraints. This serves as the default configuration; we report results under varied resource budgets.

The simulated system consists of 30 clients and 3 edge servers, with each edge serving 10 clients. For each benchmark, we set the total task arrival rate to $\lambda_{\text{tot}} \in \{100, 2000, 4000\}$ jobs/sec and allocate this load across clients using a Dirichlet distribution with concentration parameter $p_{\text{client}} = 0.5$. Each client distributes its individual load across tasks using a separate Dirichlet distribution with $p_{\text{task}} = 0.5$, resulting in task-specific rates λ_t^c. The offloading budget at each edge server e is given by $\kappa^e = \chi_\kappa^e \sum_{t\in\mathcal{T}} \sum_{c\in\mathcal{C}_e} \lambda_t^c d_t$, where $\chi_\kappa^e = 0.5$ (default), controlling the fraction of traffic that may be offloaded. The cross-edge (i.e., edge-to-cloud) budget is defined as $\kappa^s = \chi_\kappa^s \sum_e \lambda_t^e$ with $\chi_\kappa^s = 0.25$ by default.

3) Baselines: We compare our proposed J3O algorithm against the following five baselines: **(1) MINLP Solver**: Solves the full joint problem optimally using Gurobi. **(2) Greedy-AO**: Performs alternating optimization using greedy onloading (accepted only if the objective improves) and LP-based offloading. **(3) OPT-AO**: Alternates between optimal onloading (via exhaustive search) and LP offloading. **(4) Rand-AO**: Alternates with randomly selected feasible onloading configurations, accepted only if they improve the objective. **(5) Full Local**: Executes all tasks using only client-side models, with optimal model selection per client.

B. Performance and Runtime Comparison

In Fig. 3, our proposed algorithm J3O, evaluated on Taskonomy, shows monotonic objective improvement and converges within a few iterations. Table IV reports average performance (accuracy/loss) and runtime across benchmarks. J3O consistently achieves near-optimal accuracy while significantly reducing runtime compared to the globally optimal MINLP solver. Specifically, J3O reaches 97.7%, 97.5%, and 99.7% of the optimal performance on Taskonomy, DomainNet, and Cityscape3D, respectively, while using only 1.21%, 1.25%, and 14.70% of the MINLP solver's runtime. While MINLP remains tractable on small problems like Cityscape3D (3 tasks, 7 models), its runtime scales poorly with problem size, making it impractical for real-world systems that require frequent, low-latency optimization. In contrast, J3O remains orders of magnitude faster on larger benchmarks.

Compared to the baselines, J3O offers clear advantages. Greedy-AO is faster due to conservative onloading but incurs a noticeable drop in performance. OPT-AO attains slightly higher accuracy on Taskonomy and DomainNet, but at the cost of $2.60\times$ and $2.15\times$ the runtime of J3O, respectively. On Cityscape3D, J3O outperforms OPT-AO on both accuracy

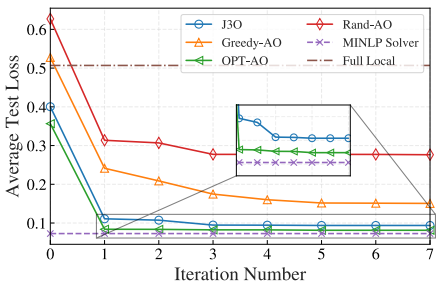


Fig. 3: Convergence of the algorithm.

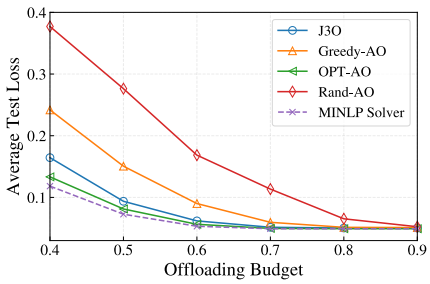


Fig. 4: Varying network capacity with χ_κ^e.

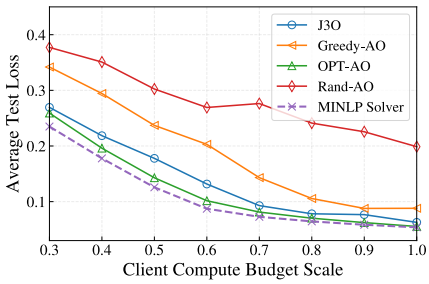


Fig. 5: Varying client compute budget.

TABLE IV: System performance (accuracy/loss) and runtime (ms), averaged over 20 seeds (±std). Taskonomy/DomainNet use 31 models; CityScape3D uses 7.

Method	Taskonomy-5		DomainNet-6		Cityscape3D-3	
	Avg. Loss (↓)	Runtime (ms)	Avg. Acc (↑)	Runtime (ms)	Avg. Loss (↓)	Runtime (ms)
Greedy-AO	$0.151_{\pm 0.081}$	$574.82_{\pm 10.44}$	$0.735_{\pm 0.025}$	$552.27_{\pm 10.74}$	$0.164_{\pm 0.014}$	$98.73_{\pm 5.38}$
Optimal-AO	$0.081_{\pm 0.047}$	$2078.33_{\pm 369.35}$	$0.770_{\pm 0.015}$	$2015.34_{\pm 304.71}$	$0.163_{\pm 0.010}$	$216.62_{\pm 21.20}$
Rand-AO	$0.276_{\pm 0.073}$	$562.67_{\pm 10.53}$	$0.640_{\pm 0.054}$	$537.89_{\pm 13.92}$	$0.221_{\pm 0.044}$	$95.81_{\pm 5.16}$
Full-Local	$0.507_{\pm 0.074}$	$633.71_{\pm 244.23}$	$0.397_{\pm 0.037}$	$597.49_{\pm 237.37}$	$0.439_{\pm 0.098}$	$40.22_{\pm 4.34}$
MINLP Solver	$0.073_{\pm 0.037}$	$65791.68_{\pm 21030.78}$	$0.774_{\pm 0.013}$	$75122.31_{\pm 18706.57}$	$0.159_{\pm 0.008}$	$1123.47_{\pm 303.61}$
J3O	$0.094_{\pm 0.048}$	$797.87_{\pm 8.43}$	$0.754_{\pm 0.018}$	$937.08_{\pm 29.79}$	$0.162_{\pm 0.010}$	$165.10_{\pm 12.02}$

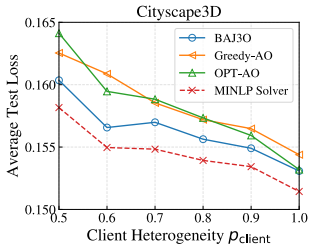


Fig. 6: Client heterogeneity.

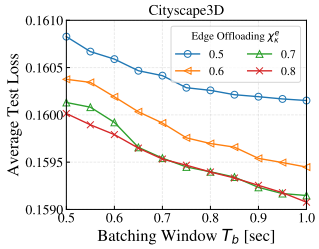


Fig. 7: Batching window.

and runtime. Both Rand-AO and Full-Local degrade significantly due to random model selection and lack of offloading, respectively. In contrast, J3O consistently balances accuracy and efficiency, yielding a robust trade-off.

C. Robustness to Varying Resource Constraints

We evaluate the robustness of J3O under varying system constraints and find that it consistently outperforms all non-optimal baselines. We consider two practical constraint types: (i) the offloading budget, which captures limitations in network bandwidth, and (ii) the client-side compute budget, reflecting the capabilities of resource-constrained devices. Fig. 4 shows the average test loss (↓) on the Taskonomy dataset as the offloading budget $(\kappa^e)_{e\in\mathcal{E}}$ is varied via the scaling parameter χ_κ^e. While all methods improve with greater offloading capacity, J3O remains the closest to the optimal solution except for OPT-AO, which achieves slightly lower loss but at significantly higher runtime. As the offloading budget tightens, J3O's advantage over heuristic baselines becomes more pronounced, demonstrating its robustness under network constraints. A similar trend appears in Fig. 5, where we vary the client compute budgets $(\beta^c)_{c\in\mathcal{C}}$ via χ_β. J3O consistently outperforms the heuristics and remains near-optimal across the full budget range, all while avoiding the computational overhead of exact solvers.

D. Batching-Aware Joint Onloading and Offloading Results

We evaluate BAJ3O on the Cityscape3D benchmark, estimating the batching setup cost ν_m^e via linear regression on measured latency profiles. The decision horizon is set to 10s, with batching intervals of $T_b = 0.5$s. As shown in Fig. 6, BAJ3O outperforms all baselines across varying client heterogeneity levels, achieving the smallest gap to the global optimum. Notably, BAJ3O surpasses OPT-AO, particularly under high heterogeneity. While both methods incorporate batching constraints during model onloading, OPT-AO optimally solves each subproblem based on a fixed batching estimate from the previous iteration. This fails to capture true batching dynamics under diverse client workloads, leading to biased early on/offloading decisions and reduced performance.

In Fig. 7, we vary the batching interval T_b to study its impact on latency and accuracy. As T_b increases, BAJ3O forms larger batches, improving accuracy at the cost of higher delay, highlighting the accuracy–latency trade-off. This effect is stronger with generous edge offloading budgets, where batching is fully leveraged. Under tighter budgets ($\chi_\kappa^e = 0.5$), gains are limited as communication becomes the main bottleneck.

VII. Conclusion

We presented a unified framework for joint model onloading and hierarchical offloading in distributed multi-task inference systems. By coordinating model placement and task routing across clients, edge servers, and the cloud, our J3O and BAJ3O algorithms enable accurate, efficient inference under tight resource constraints. Experiments on diverse benchmarks show our method achieves near-optimal accuracy with significantly reduced runtime. Our approach can be extended to settings where task demands and resource availability evolve slowly or load estimates are noisy. Indeed, our initial investigations indicate that this can be done efficiently, yielding excellent on/offloading decisions over time with minimal overhead.

REFERENCES

[1] Q. Zhou, Z. Qu, S. Guo, B. Luo, J. Guo, Z. Xu, and R. Akerkar, "On-device learning systems for edge intelligence: A software and hardware synergy perspective," *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 11 916–11 934, 2021.

[2] P. Villalobos, J. Sevilla, T. Besiroglu, L. Heim, A. Ho, and M. Hobbhahn, "Machine learning model sizes and the parameter gap," *arXiv preprint arXiv:2207.02852*, 2022.

[3] N. Dhar, B. Deng, D. Lo, X. Wu, L. Zhao, and K. Suo, "An empirical analysis and resource footprint study of deploying large language models on edge devices," in *Proceedings of the 2024 ACM Southeast Conference*, 2024, pp. 69–76.

[4] Y. Matsubara, M. Mendula, and M. Levorato, "A multi-task supervised compression model for split computing," in *Proceedings of the Winter Conference on Applications of Computer Vision (WACV)*, February 2025, pp. 4913–4922.

[5] W. Fan, Z. Chen, Z. Hao, Y. Su, F. Wu, B. Tang, and Y. Liu, "Dnn deployment, task offloading, and resource allocation for joint task inference in iiot," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 1634–1646, 2022.

[6] A. Fresa and J. P. V. Champati, "An offloading algorithm for maximizing inference accuracy on edge device in an edge intelligence system," in *Proceedings of the 25th International ACM Conference on Modeling Analysis and Simulation of Wireless and Mobile Systems*, 2022, pp. 15–23.

[7] H. B. Beytur, A. G. Aydin, G. de Veciana, and H. Vikalo, "Optimization of offloading policies for accuracy-delay tradeoffs in hierarchical inference," in *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, 2024, pp. 1989–1998.

[8] H. Ye and D. Xu, "Taskprompter: Spatial-channel multi-task prompting for dense scene understanding," in *The Eleventh International Conference on Learning Representations*, 2022.

[9] M. Neseem, A. Agiza, and S. Reda, "Adamtl: Adaptive input-dependent inference for efficient multi-task learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 4730–4739.

[10] K. Doshi and Y. Yilmaz, "Multi-task learning for video surveillance with limited data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 3889–3899.

[11] T. Standley, A. Zamir, D. Chen, L. Guibas, J. Malik, and S. Savarese, "Which tasks should be learned together in multi-task learning?" in *International conference on machine learning*. PMLR, 2020, pp. 9120–9132.

[12] C. Fifty, E. Amid, Z. Zhao, T. Yu, R. Anil, and C. Finn, "Efficiently identifying task groupings for multi-task learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 27 503–27 516, 2021.

[13] X. Song, S. Zheng, W. Cao, J. Yu, and J. Bian, "Efficient and effective multi-task grouping via meta learning on task combinations," *Advances in Neural Information Processing Systems*, vol. 35, pp. 37 647–37 659, 2022.

[14] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, "Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, 2018, pp. 671–678.

[15] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[16] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2020.

[17] W. He, S. Guo, S. Guo, X. Qiu, and F. Qi, "Joint dnn partition deployment and resource allocation for delay-sensitive deep learning inference in iot," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9241–9254, 2020.

[18] N. Hudson, H. Khamfroush, and D. E. Lucani, "Qos-aware placement of deep learning services on the edge with multiple service implementations," in *2021 International Conference on Computer Communications and Networks (ICCCN)*, 2021, pp. 1–8.

[19] A. B. Sada, A. Khelloufi, A. Naouri, H. Ning, and S. Dhelim, "Energy-aware selective inference task offloading for real-time edge computing applications," *IEEE Access*, 2024.

[20] W. Zhang, D. Yang, H. Peng, W. Wu, W. Quan, H. Zhang, and X. Shen, "Deep reinforcement learning based resource management for dnn inference in industrial iot," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 8, pp. 7605–7618, 2021.

[21] Y. Chai, K. Gao, G. Zhang, L. Lu, Q. Li, and Y. Zhang, "Joint task offloading, resource allocation and model placement for ai as a service in 6g network," *IEEE Transactions on Services Computing*, 2024.

[22] Z. Chen, S. Zhang, Z. Ma, S. Zhang, Z. Qian, M. Xiao, J. Wu, and S. Lu, "An online approach for dnn model caching and processor allocation in edge computing," in *2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*. IEEE, 2022, pp. 1–10.

[23] Y. Wu, J. Wu, L. Chen, B. Liu, M. Yao, and S. K. Lam, "Share-aware joint model deployment and task offloading for multi-task inference," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 6, pp. 5674–5687, 2024.

[24] M.-H. Chen, B. Liang, and M. Dong, "Joint offloading and resource allocation for computation and communication in mobile cloud with computing access point," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[25] S. Bi, L. Huang, and Y.-J. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 19, no. 7, pp. 4947–4963, 2020.

[26] M. Sviridenko, "A note on maximizing a submodular set function subject to a knapsack constraint," *Operations Research Letters*, vol. 32, no. 1, pp. 41–43, 2004.

[27] R. K. Iyer and J. A. Bilmes, "Submodular optimization with submodular cover and submodular knapsack constraints," *Advances in neural information processing systems*, vol. 26, 2013.

[28] M. L. Fisher, "The lagrangian relaxation method for solving integer programming problems," *Management science*, vol. 27, no. 1, pp. 1–18, 1981.

[29] Y. Inoue, "Queueing analysis of gpu-based inference servers with dynamic batching: A closed-form characterization," *Performance Evaluation*, vol. 147, p. 102183, 2021.

[30] Y. Cang, M. Chen, and K. Huang, "Joint batching and scheduling for high-throughput multiuser edge ai with asynchronous task arrivals," *IEEE Transactions on Wireless Communications*, 2024.

[31] A. R. Zamir, A. Sax, W. Shen, L. J. Guibas, J. Malik, and S. Savarese, "Taskonomy: Disentangling task transfer learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3712–3722.

[32] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[33] X. Peng, Q. Bai, X. Xia, Z. Huang, K. Saenko, and B. Wang, "Moment matching for multi-source domain adaptation," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1406–1415.

[34] M. Wallingford, H. Li, A. Achille, A. Ravichandran, C. Fowlkes, R. Bhotika, and S. Soatto, "Task adaptive parameter sharing for multi-task learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 7561–7570.

[35] N. Gählert, N. Jourdan, M. Cordts, U. Franke, and J. Denzler, "Cityscapes 3d: Dataset and benchmark for 9 dof vehicle detection," *arXiv preprint arXiv:2006.07864*, 2020.