Towards Inference Service with Edge Computing: the Framework and an Improved Firefly Algorithm

Ran Bi[1*] *Member, IEEE*, Zean Dong[1], Xiaolin Fang [2] *Member, IEEE*, Weiye Si[1], Qingxu Deng[3] *Member, IEEE*, and Zhipeng Cai[4] *Fellow, IEEE*

***Abstract*—The integration of mobile edge computing (MEC) and 5G communication has led to a new paradigm for enhancing user experiences by offloading compute-intensive inference services composed of dependent subtasks to MEC. Despite most existing research on task offloading schemes that consider subtask dependencies, the latency guarantees of these approaches has not been explored. To address this gap, we propose an inference service framework that combines Initialization Optimization (INOP) with an Improved Firefly Algorithm (IFA) to jointly optimize subtask offloading and scheduling under precedence constraints. INOP employs a greedy strategy with a constant approximation ratio, assigning each subtask to the server that minimizes the increase in average completion time across multiple inference services. IFA employs a linear position update strategy that ensures each fireflys position corresponds to a valid, precedence-compliant subtask ranking. This refined ranking is then leveraged with INOP's greedy method to finalize offloading and scheduling decisions. We prove IFA's convergence and establish sufficient conditions to prevent boundary traps. Extensive simulations demonstrate that the proposed scheme reduces system latency by 60% and convergence time by 10% compared to existing benchmark schemes across various scenarios.**

***Index Terms*—Task scheduling, firefly algorithm, approximation ratio, convergence analysis, Edge Computing.**

I. Introduction

The rapid advancement in communication and AI technologies is driving the proliferation of computation-intensive mobile applications, such as face recognition and video analysis. The computational demands of these inference services (tasks) often exceed the capabilities of mobile devices (MDs), resulting in poor user experiences. Mobile edge computing (MEC) addresses this limitation by offloading inference tasks to edge servers (ESs) located near wireless access points (APs). According to Alibaba [1], over 75% of inference services comprise computation-intensive subtasks with dependency constraints, requiring execution in a specific order [2]. A subtask is eligible for execution after receiving all outputs from its predecessors, and an inference task is complete once all dependent subtasks have finished processing [3]. As shown in Fig.1(a), an image processing service with five dependent subtasks requires the data classification subtask to wait for outputs from color, texture and shape feature extraction. The service concludes when all five subtasks have been executed.

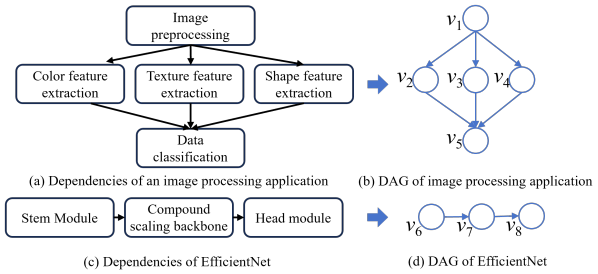

Fig. 1. DAGs for inference service with dependent subtasks

Offloading dependent subtasks is an NP-hard problem [4]. Recent studies, using directed acyclic graph (DAG) models (Fig. 1(b)), employ heuristic algorithms or deep reinforcement learning (DRL) to optimize subtask offloading [5]–[8], targeting minimal average completion time (ACT). However, these methods lack guarantees on approximation ratios, particularly in scenarios with multiple inference tasks and multi-ESs cooperation. Moreover, existing strategies rely on a predefined topological order for offloading decisions and intra-server subtask execution, overlooking subtask scheduling optimization. As shown in Fig. 1, when subtasks v_3 and v_7 are offloaded to the same ES and become executable during ES idleness, scheduling decisions are critical. Due to task dependencies, these decisions significantly impact service completion time.

The joint optimization of offloading decisions and subtask scheduling is challenging due to its nonlinear, mixed-integer property, involving both discrete (offloading) variables and continuous (scheduling) variables. Numerous studies have explored DRL for offloading optimization in MEC [3], but most struggle with hybrid discrete-continuous action spaces. Strategies, such as discretizing actions or relaxation into continuous domains, often fail to capture the interdependence between offloading and scheduling decisions. This challenge is further complicated by the dynamic nature of MEC environments. As DRL depends on stable statistical patterns, shifts in ES capacity, task arrivals, or deadlines can render a trained model ineffective, requiring retraining or sophisticated online adaptation. Moreover, DRL-based methods often neglect initial

R. Bi*, Z. Dong and W. Si are with the School of computer and Communication Engineering, Northeastern University, Qinhuangdao 066004, China (e-mail: biran@neuq.edu.cn; siweiye@stumail.neu.edu.cn; b99116610927@gmail.com).

X. Fang is with the School of Computer Science and Engineering, Southeast University, Nanjing 211189, China and also with the Key Laboratory of Embedded System and Service Computing (Tongji University), Ministry of Education, Shanghai 201804, China (e-mail: xiaolin@seu.edu.cn).

Q. Deng is with the School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China (e-mail: dengqx@mail.neu.edu.cn)

Z. Cai is with the Department of Computer Science, Georgia State University, Atlanta, GA 30302 USA (e-mail: zcai@gsu.edu).

†This work is partially supported the National Natural Science Foundation of Hebei Province

solution optimization. Leveraging an initial solution with an approximation-ratio guarantee allows DRL to start closer to an optimal strategy, reducing inefficient exploration from random initialization and accelerating convergence.

To address these challenges, we propose an inference service framework that integrates *Initialization Optimization* (INOP) with an *Improved Firefly Algorithm* (IFA) to jointly optimize subtask offloading and scheduling under precedence constraints. INOP begins by ranking subtasks to maintain topological order, then sequentially makes greedy offloading decisions, assigning each subtask to the ES that minimizes the increase in the ACT across multiple inference services. To efficiently compute this incremental value despite the exponential number of candidate solutions, we solve a relaxed version of the problem using linear programming and prove a constant approximation ratio bound for the greedy approach. IFA further refines INOP's initial ranking through a lightweight, precedence-aware probabilistic mapping operator. Unlike the standard firefly algorithm, our IFA employs a linear position update strategy that ensures each fireflys position corresponds to a valid, precedence-compliant subtask ranking. This refined ranking is then leveraged with INOP's greedy method to finalize offloading and scheduling decisions. We also analyze IFAs convergence and establish a sufficient condition to prevent boundary traps. The main contributions are as follows.

- We formulate the offloading of multiple inference services in a multi-ES system as a nonlinear mixed-integer programming problem, aiming to minimize the ACT across all inference services under precedence constraints.
- We propose a greedy based initialization optimization algorithm that determines offloading and scheduling decisions by minimizing the incremental ACT, calculated by a linear programming problem with relaxed disjunctive constraints. We prove that INOP achieves an approximation ratio (AR) of $(1 + \frac{2f_{\max}}{f_{\min}})(1 + \frac{1}{e-1})$, where $f_{\max}$ and $f_{\min}$ denote the maximum and minimum CPU frequencies among ES. INOP complements DRL based approaches for dependent task offloading and scheduling. By providing an initial solution with a proven approximation ratio, INOP accelerates DRL convergence and improves performance guarantees.
- We introduce the IFA, which utilizes a linear position update strategy to ensure each fireflys position maps to a valid subtask ranking. This refined ranking is integrated into INOP to finalize offloading and scheduling decisions, forming our inference service framework. We prove the convergence of IFA and derive a sufficient condition to prevent boundary traps.
- Experimental work shows that our inference framework soutperforms other algorithms in terms of ACT, convergence, completion ratio and rum times, under various settings, thereby validating the theoretical results.
  we conduct extensive experiments across multiple offloading scenarios using real-world applications and measurements. The results demonstrate that the proposed algorithm outperforms other algorithms in terms of application delay, local energy consumption, and learning regret, under varying task delays and learning times

The remainder of this paper is organized as follows. Section II reviews related work, and Section III introduces the system model and problem definition. Section IV details the initialization optimization method and analyzes its approximation ratio. Section V presents the proposed inference service framework and its theoretical analysis. Performance evaluation is provided in Section VI, and Section VII concludes the paper.

II. Related Work

Numerous studies have explored scheduling dependent tasks. We review literature on dependent task offloading in MEC and DAG workflows scheduling in cloud computing.

A. Dependent Tasks Offloading in MEC Systems

Task offloading in MEC systems has been extensively studied to enhance performance metrics such as latency, energy efficiency, and resource allocation. The literature can be broadly categorized into two areas: optimizing offloading for individual applications and addressing more complex scenarios involving multiple services or applications.

Some research targets latency reduction for single applications via task offloading in MEC systems. Sundar *et al*. [6] explored scheduling strategies to minimize execution costs while meeting deadlines. Chen *et al*. [5] decomposed the offloading problem into two sub-problems, using cooperative methods and a cost-aware task transfer scheme to reduce completion times. Wang *et al*. [9] proposed a DRL-based framework to learn offloading policies and identify patterns across applications. To balance makespan and energy consumption on mobile devices, Zhang *et al*. [10] developed task rescheduling algorithms. Dai *et al*. [7] addressed unknown system information by integrating Lyapunov optimization for online decisions with multi-armed bandit theory. For scenarios where multiple users request a single service, *et al*. [8] introduced a DAG-based offloading method with soft cooperation, optimizing system utility by considering latency, energy costs, and cooperation gains.

Other studies focus on optimizing offloading and resource allocation for multiple service applications. For latency minimization, Liu *et al*. [4] developed a heuristic ranking-based algorithm to minimize average makespan. Wang *et al*. [11] designed a heuristic partial offloading strategy that lowers system latency, incorporating energy considerations and theoretical bounds. For energy minimization, research such as [20] developed a heuristic algorithm that leverages task prioritization and a critical path-based approach, and [21] proposed a two-tier computation offloading framework for multi-task in heterogeneous MEC networks. He *et al*. [22] further addressed the issue by transforming a multi-layer task offloading problem into a linear programming model.

Various approaches address this dual objective. DRL techniques are prominent. Yan *et al*. [23] and Shi *et al*. [24] utilized DRL, with the latter using a Double Deep Q Network (DDQN) to jointly optimize task offloading and resource allocation for reduced delay-energy cost. Liu *et al*. [14] proposed a proximal policy optimization (PPO)- based scheme and Liu *et al*. [3] used a DDPG-based framework, combing task migration and

TABLE I
Comparison of Literature on Dependent Task Offloading in MEC and DAG Workflows Scheduling

Reference	Scenario	Single/Multiple Services	Offloading Decision	Subtask Optimaiztion	Method	Initialization Optimization	Theoretical Analysis
[5]	MEC-Cloud	Single	✓	–	Greedy	–	AR for one ES
[6]	Cloud	Single	✓	–	Heuristic	–	–
[7]	MEC	Single	✓	–	Lyapunov	–	Bound of learning regret
[8]	MEC	Single	✓	–	SAC	–	–
[4]	MEC-Cloud	Multiple	✓	–	Heuristic	–	Stability analysis
[11]	MEC	Multiple	✓	–	Heuristic	–	Bound of latency
[12]	MEC	Multiple	✓	–	Genetic	–	–
[13]	MEC-Cloud	Multiple	✓	✓	Heuristic	–	Time complexity analysis
[14]	MEC	Multiple	✓	–	PPO	–	–
[3]	MEC	Multiple	✓	✓	DDPG	–	–
[15]	HCS	Multi-work flow	–	✓	List scheduling	✓	–
[16]	Cloud	Multi-work flow	–	✓	DQN	–	–
[17]	Cloud	Multi-work flow	✓	–	DDQN	–	–
[18]	Cloud	DAG	–	✓	PPO	✓	–
[19]	Cloud	Multi-work flow	–	✓	Memetic	✓	–
This work	MEC	Multiple	✓	✓	INOP with IFA	✓	AR and convergence

merging to minimize deadline violations. Alternative methods include Zhang *et al*. [25] transformed a non-convex problem into solvable subproblems for distributed dependent applications. Guo *et al*. [12] introduced the HRRO-GA algorithm, combing response ratio scheduling with a genetic algorithm for multi-DAG applications. Lou *et al*. [13] developed a low-complexity scheduling algorithm considering future impacts. Yang *et al*. [26] proposed a search-based computation partitioning strategy for sequential task offloading.

Despite these advances, most existing algorithms overlook initial solution optimization, and DRL-based methods often struggle to adapt to dynamic environments without retraining or online adaptation.

B. DAG Workflows Scheduling in Cloud Computing

Workflow scheduling in cloud computing is a widely explored topic [27]. Existing methods can be categorized into three types: heuristic algorithms, DRL-oriented scheduling methods, and evolutionary and meta-heuristic techniques.

Heuristic algorithms operate in two phases: task prioritization, where high-priority tasks are queued for execution, and server selection, where tasks are assigned to servers to minimize costs such as completion time or energy usage. Many studies extend foundational algorithms like Heterogeneous Earliest Finish Time (HEFT) [28] and Predict Earliest Finish Time (PEFT) [29], incorporating modified critical paths and dynamic level scheduling. Faragardi *et al*. [30] proposed GRP-HEFT, a greedy algorithm that reduces workflow completion time within budget limits. Fan *et al*. [31] introduced PACP-HEFT, which uses task dependency topology and real-time data to minimize makespan through priority and task optimization. Ma *et al*. [32] developed a three-stage approach to schedule multiple workflows, reducing costs while meeting deadlines. Cai *et al*. [15] presented urgency-based list scheduling for deadline-constrained workflows in heterogeneous computing systems (HCS). Zhang *et al*. [33] designed a heuristic method that combines an optimistic timetable and Monte Carlo simulations to optimize makespan for stochastic DAGs.

Learning-based approaches, particularly those using DRL, have recently gained traction [34]. Pan *et al*. [16] proposed an online DRL framework for scheduling multiple real-time workflows to minimize makespan and enhance resource utilization. Xie *et al*. [17] formulated scheduling as a multi-objective optimization problem and applied a DDQN for industrial IoT. Lin *et al*. [18] introduced SpotDAG, a reinforcement learning method with self-attention for auto-scaling and job scheduling, that reduces costs while meeting deadlines. Chen *et al*. [35] developed a DRL-based collaborative approach to optimize makespan, cost, fairness, and execution continuity by leveraging structural and temporal features. Studies by [36]–[38] further demonstrate DRLs effectiveness in multi-objective scheduling.

Evolutionary and meta-heuristic methods offer alternative solutions. Qin *et al*. [19] proposed RA-MOMA, a reliability-aware memetic algorithm, that employs problem-specific genetic operators for multi-objective optimization. Meta-heuristic approaches, such as genetic algorithms (GA) and firefly algorithms, are also widely used. Yin *et al*. [39] introduced a convergent firefly algorithm that combines probability-based mapping and linear updates for cloud-edge task scheduling. Qasim *et al*. [40] developed a firefly variant with transfer functions and quantization to minimize makespan. However, these mapping strategies fail to satisfy precedence constraints. Xu *et al*. [41] proposed a Multi-Tree GA integrating mobile devices, edge, and cloud servers for dynamic scheduling. Yang *et al*. [42] presented a Dual-Tree GA with adaptive mutation to optimize virtual machine and task selection.

Multi-task flow scheduling based on DAG tasks is difficult to apply directly in MEC, as traditional methods assume static

and fixed DAG structures, which conflict with the dynamic, resource-constrained, and diverse inference inferences encountered in MEC environments.

III. System Model and Problem Formulation

As shown in Fig. 2, an edge computing system comprises a wireless access point (AP) and densely deployed edge servers (ESs), such as roadside units and micro base stations, which execute user inference services. The AP, located near the ESs, serves as a gateway, enabling wireless connections between mobile devices (MDs) and ESs. The set of ESs is denoted by $\mathcal{S} = \{s_1, ..., s_M\}$. Each ES s_i has limited computational and storage resources, and can communicate with another ES s_j at a transmission rate determined by the hop distance [43]. In this model, a coordinator at the AP handles offloading and scheduling decisions for the dependent subtasks that constitute an inference service. MDs first submit inference-service requests to the coordinator. Upon acceptance, the coordinator determines the offloading and scheduling strategy for each subtask. Finally, the MD receives the inference result once all the subtasks of the requested service are completed. The main notations used in this article are listed in Table II.

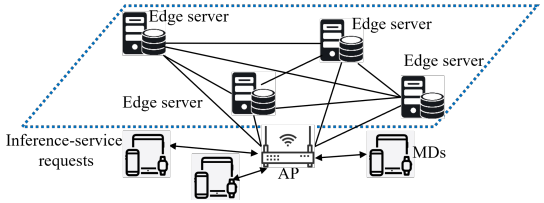

Fig. 2. System architecture

A. Inference Service Model

Edge computing systems provide deep learning inference services for MDs. Due to inherent subtask dependencies, these subtasks must be processed in a logical precedence order, where each subtask is eligible for execution after receiving all outputs from its preceding subtasks. As shown in Fig. 1(a), color feature extraction is permitted to execute only after the completion of image preprocessing. Similar to [3], [4], [7], we formulate an inference service as a directed acyclic graph (DAG). Specifically, an inference service (task) τ_k is denoted by $G_k = (V_k, E_k)$, where V_k is the set of vertices, and E_k is the set of directed edges. Each vertex $v_i \in V_k$ represents a subtask with a computation workload $c(v_i)$ (measured in CPU cycles) and an output data size $d(v_i)$. A directed edge $(v_j, v_i) \in E$ indicates that subtask v_i can not be processed until subtask v_j is completed. In this case, v_j is a predecessor of v_i, and v_i is a successor of v_j. We denote the set of predecessors and successors of subtask v_i by $\mathrm{pre}(v_i)$ and $\mathrm{succ}(v_i)$, respectively. Moreover, v_j is an ancestor of v_i, if v_j is the predecessor of v_i or is the predecessor of any ancestor of v_i. In this context, v_i is a descendant of v_j. We denote the sets of ancestors and descendants of v_i by $\mathrm{anc}(v_i)$ and $\mathrm{des}(v_i)$, respectively. For any subtask v_i, another subtask v_j is parallel with v_i, if v_j is neither an ancestor nor a descendant of v_i. The set of parallel subtasks of v_i is denoted as

$$\mathrm{par}(v_i) = \{v_j | v_j \notin v_i \cup \mathrm{anc}(v_i) \cup \mathrm{des}(v_i)\}. \quad (1)$$

TABLE II
Summary of Key Notations

Notation	Description
$\mathcal{S}$	Set of edge servers $\{s_1, ..., s_M\}$, $\lvert\mathcal{S}\rvert = M$
f_m	The CPU frequency of ES s_m
r_m	The transmission rate of ES s_m
τ_k	An inference-service task
$\delta(\tau_k)$	The deadline of inference service τ_k
G_k	The graph structure of τ_k, $G_k = (V_k, E_k)$
V_k	The set of vertices of G_k
E_k	The set of directed edges G_k
G	The DAG $G = (V, E)$ composing a dummy source (sink) vertex and T individual DAG graphs G_k
T	The number of inference tasks
v_i	A vertex of G_K, i.e., a subtask of τ_k
$c(v_i)$	The computation workload of subtask v_i
$d(v_i)$	The output data size of subtask v_i
$\mathrm{pre}(v_i)$	The set of predecessors of subtask v_i
$\mathrm{anc}(v_i)$	The set of ancestors of subtask v_i
$\mathrm{succ}(v_i)$	The set of successors of subtask v_i
$\mathrm{des}(v_i)$	The set of descendants of subtask v_i
$\mathrm{par}(v_i)$	The set of parallel vertices of v_i
$b(i,m)$	Binary offloading decision for subtask v_i executed on ES s_m
$d_c(v_i)$	The computation delay of subtask v_i
$d_{tr}(v_i)$	The transmission delay of the output of subtask v_i from v_i to its successor
$t(v_i)$	The time point of subtask v_i completion
$t_s(v_i)$	The time point of subtask v_i beginning

A subtask is defined as the *source* (*sink*) subtask v_{src} (v_{snk}) if it has no predecessors (successors). Without loss of generality, subtask and vertex will be used interchangeably.

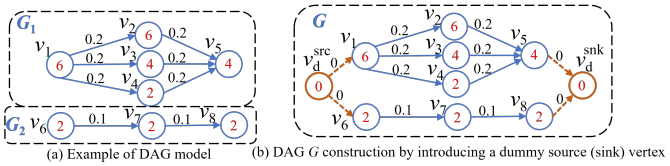

Fig. 3. DAG model and construction of multiple inference tasks

Example 1. *Fig 3(a) presents two DAG models for inference services in Fig. 1(a). The computation workload $c(v_i)$ of each subtask v_i is labeled inside the vertex and the output data size $d(v_i)$ is labeled on the edge. The source and sink subtasks of G_1 are v_1 and v_5, respectively. Subtask v_3 has one predecessor v_1, and one successor v_5. The set of ancestors of v_5 is $\mathrm{anc}(v_5) = \{v_1, v_2, v_3, v_4\}$, and the descendants of v_1 are $\mathrm{des}(v_1) = \{v_2, v_3, v_4, v_5\}$. Moreover, the set of parallel subtasks for v_3 is $\mathrm{par}(v_3) = \{v_2, v_4\}$, implying that if v_2 and v_4 are assigned to different ESs, they can execute in parallel if they receive outputs from subtask v_1.*

B. Delay Model

The delay of subtask consists of execution delay and transmission delay. For each subtask $v_i \in V$, we define a binary variable $b(i,m) \in \{0,1\}$ to indicate whether subtask v_i is offloaded to ES s_m.

$$b(i,m) = \begin{cases} 1, & \text{if subtask } v_i \text{ is processed on ES } s_m; \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

As each subtask v_i is assigned to a single ES, we have

$$\sum_{m=1}^{M} b(i,m) = 1. \tag{3}$$

We assume that at any time, a server can process at most one subtask of an inference service. Let f_m denote the average CPU frequency of ES s_m. The computation delay $d_{\mathrm{c}}(v_i)$ of subtask v_i can be expressed by

$$d_{\mathrm{c}}(v_i) = \frac{c(v_i)}{\sum_{m=1}^{M} b(i,m) f_m}. \tag{4}$$

The transmission interval represents the raw time required to transmit the output data $d(v_j)$ of subtask v_j to all potential ESs and is calculated as

$$t_{\mathrm{tr}}(v_j) = \sum_{m=1}^{M} \frac{d(v_j) b(j,m)}{r_m}. \tag{5}$$

where is $r(m)$ is transmission rate of ES s_m. When subtask v_i and its predecessor v_j are offloaded to different ESs, the transmission delay is refined as

$$d_{tr}(v_j) = \sum_{m=1}^{M} \frac{t_{\mathrm{tr}}(v_j)|b(j,m) - b(i,m)|}{2}, v_j \in \mathrm{pre}(v_i) \tag{6}$$

$|b(j,m) - b(i,m)|$ is the absolute difference between the offloading decisions for v_j and v_i at ES s_m (1 if they are offloaded to different ESs, 0 if to the same). The division by 2 normalizes the delay, since when v_j and v_i are executed on different ESs, the sum $\sum_{m=1}^{M} |b(j,m) - b(i,m)|$equals 2. The transmission interval measures the full transmission time, and the transmission delay adjusts this value to account for differences in offloading between subtasks. Delays for service registration and data transmission between the MD and AP, as well as the final result transmission from AP to the MD, are not considered in this work because they do not affect offloading or scheduling decisions.

Example 2. *Consider the DAG model in Fig. 3(a). Suppose there are two edge servers, s_1 and s_2, each with a transmission rate of 1. Assume subtask v_1 is offloaded to s_1, while v_2 and v_3 are assigned to s_1 and s_2, respectively. This assignment yields $b(1,1) = 1$, $b(2,1) = 1$, and $b(3,2) = 1$, implying $b(1,2) = 0$, $b(2,2) = 0$, and $b(3,1) = 0$ based on Eq. (3). Using formula (6), the transmission delay $d_{\mathrm{tr}}(v_1)$ of output from v_1 to v_2 is 0, and from v_1 to v_3 is 0.2.*

C. Problem Formulation

Generally, the edge computing system receives multiple inference task requests, and this paper aims to minimize the average completion time (ACT) of these tasks. For each subtask v_i of an inference-service task τ (represented by a DAG G), we define $t(v_i)$ to as its completion time. Due to precedence constraints, the completion time of v_i must satisfy

$$t(v_i) \geq t(v_j) + d_{tr}(v_j) + d_{\mathrm{c}}(v_i), \forall v_j \in \mathrm{pre}(v_i) \tag{7}$$

which means that v_i is not executable until all its predecessors have finished processing and transmitted their outputs. For any pair of parallel subtasks v_i and v_j assigned to the same edge server (i.e., $b(i,m) = b(j,m)$, for all $m = 1, 2, \ldots M$) their completion times satisfies

$$\begin{aligned} &\{t(v_i) \geq t(v_j) + d_{\mathrm{c}}(v_i)\} \vee \{t(v_j) \geq t(v_i) + d_{\mathrm{c}}(v_j)\} = 1, \\ &\forall v_j \in \mathrm{par}(v_i) \wedge b(i,m) = b(j,m), \text{for all } m = 1, 2, \ldots M. \end{aligned} \tag{8}$$

This constraint enforces sequential, non-preemptive execution of parallel subtasks on the same server. We transform the offloading and scheduling of multiple inference tasks into an equivalent single-task problem. To achieve this, we introduce a dummy source vertex $v^{\mathrm{d}}_{\mathrm{src}}$ and a dummy sink vertex $v^{\mathrm{d}}_{\mathrm{snk}}$, both with zero workload and transmission interval. For each DAG G_k representing an inference task τ_k, we add an edge from $v^{\mathrm{d}}_{\mathrm{src}}$ to the source vertex $v_{\mathrm{src}}(G_k)$ and an edge from the sink vertex $v_{\mathrm{snk}}(G_k)$ to $v^{\mathrm{d}}_{\mathrm{snk}}$. This transformation allows us to treat all tasks as components of a unified DAG. For T inference tasks, each task τ_k must complete by its deadline $\delta(\tau_k)$, enforced by the constraint:

$$t(v_i) \leq \delta(\tau_k), \forall v_i \in \mathrm{pre}(v^{\mathrm{d}}_{\mathrm{snk}}) \cap G_k, k \in \{1, \cdots, T\} \tag{9}$$

where $v_i \in \mathrm{pre}(v^{\mathrm{d}}_{\mathrm{snk}}) \cap G_k$ indicates a sink subtask of G_k. As completing all sink subtasks implies the completion of the entire inference task. This constraint for each τ_k ensures that all inference tasks finish within their respective deadlines.

Example 3. *Fig.3(b) illustrates the transformation of the offloading and scheduling problem for two inference tasks into the equivalent single-task formulation. Since $v^{\mathrm{d}}\mathrm{src}$ and $v^{\mathrm{d}}\mathrm{snk}$ have zero workload and transmission delay, they incur no additional overhead. The original sink subtask of G_1 becomes a predecessor of $v^{\mathrm{d}}_{\mathrm{snk}}$, so that $\mathrm{pre}(v^{\mathrm{d}}_{\mathrm{snk}}) \cap G_1 = \{v_5\}$. Ensuring $t(v_5) \leq \delta(\tau_1)$ confirms that τ_1 meets its deadline.*

The objective of this study is to develop an offloading and scheduling strategy that minimizes the ACT of multiple inference tasks, formulated as:

$$\begin{aligned} \mathrm{P1} \quad \min_{b(i,m), t(v_i)} \quad & \sum_{k=1}^{T} \frac{\max\{t(v_i) | v_i \in \mathrm{pre}(v^{\mathrm{d}}_{\mathrm{snk}}) \cap G_k\}}{T} \\ \text{s.t.} \quad & \text{Constratins } (2) - (9). \end{aligned} \tag{10}$$

Each $v_i \in \mathrm{pre}(v^{\mathrm{d}}_{\mathrm{snk}}) \cap G_k$ corresponds to an sink subtask $v^{\mathrm{d}}_{\mathrm{snk}}$ of G_k. Thus, $\max\{t(v_i) | v_i \in \mathrm{pre}(v^{\mathrm{d}}_{\mathrm{snk}}) \cap G_k\}$ represents the completion time of τ_k. Constraint (2) enforces that $b(i,m)$ is a binary offloading variable, while constraint (3) ensures that each subtask v_i is assigned to exactly one ES. Constraint (4) specifies that the processing latency of subtask v_i depends on its workload and the assigned server's CPU frequency. Constraints (5) and (6) capture the transmission delay for outputs from predecessor subtasks, and constraint (7) ensures that v_i cannot begin execution until all its predecessors have finished processing and transmitting their outputs. Constraint (8) enforces sequential execution of parallel subtasks on same ES, and constraint (9) guarantees that each inference service τ_k is completed within its deadline $\delta(\tau_k)$.

Without loss of generality, let $\{b^*(i,m), t^*(v_i)\}$ denote the optimal solution for P1. For each subtask v_i, the offloading decision is determined by identifying m for which $b^*(i,m) = 1$. Since $t^*(v_i)$ is the completion time of v_i, its start time is given by $t^*_{\mathrm{s}}(v_i) = t^*(v_i) - d_{\mathrm{c}}(v_i)$, yielding its optimal scheduling.

Theorem 1. *Problem* P1 *is NP-Complete.*

Proof. Given a deadline $\mathcal{D}$, the feasibility of a solution to P1 can be verified in $O(K)$ time by checking whether its objective value is less than $\mathcal{D}$, where K is the number of inference tasks and significantly smaller than the solution space. Thus, P1 belongs to NP. Next, we reduce the known NP-complete Precedence Constrained Scheduling (PCS) problem to a special case of P1. PCS is defined as follows:

PSC: A set of processors $\mathcal{W}$, a time limit l, and a set of tasks $\mathcal{T}$, each with length $l(t) = 1$, subject to a partial order $\preccurlyeq$. Is there a schedule σ for $\mathcal{T}$ such that each task completes by time l while satisfying precedence constraints (i.e. if $t_j \preccurlyeq t_i$ then $\sigma(t_i) \geq \sigma(t_j) + 1$, where $\sigma(t)$is the start time of task t)?

We construct a reduction from PCS to a specific instance P′ of P1. Consider P′ with a single inference task τ, represented by a DAG with subtasks $\mathcal{T}$ under the partial order $\preccurlyeq$. If the DAG has multiple sink vertices, we add a dummy sink vertex $v^{\mathrm{d}}_{\mathrm{snk}}$ with zero execution and transmission delay. In P′, we assume that all edge servers have uniform processing capabilities, where each subtask incurs an execution delay of 1 and a transmission delay of 0. P1 is reduced to ensuring that

$$t(v) \leq l, \ \forall v \in \mathrm{pre}(v^{\mathrm{d}}_{\mathrm{snk}}). \tag{11}$$

where each vertex $v \in \mathrm{pre}(v^{\mathrm{d}}_{\mathrm{snk}})$ represents a sink subtask of τ. Satisfying formula (11) ensures all subtasks in $\mathcal{T}$ meet the time limit l. Thus, a feasible solution to P′ solves PCS, proving that P1 is at least as hard as PCS and NP-complete. □

IV. Initialization Optimization of Inference Framework

In this section, we present INOP, a greedy-based algorithm for optimizing subtask offloading and scheduling under precedence constraints. INOP first ranks subtasks to maintain topological order, and sequentially assigns each subtask to the ES that minimizes the increase in ACT. To address the combinatorial complexity, INOP leverages a linear programming relaxation to compute this incremental value, achieving a constant approximation ratio.

A. Subtasks Ranking

For any vertex sequence $\pi = (v_1, \dots v_k)$ in DAG G, π is a path if each vertex v_i is a predecessor of v_{i+1} for $i = 1, \dots k-1$. The length of path π is defined as $\mathrm{len}(\pi)$

$$\mathrm{len}(\pi) = \sum\nolimits_{i=1}^{k} c(v_i). \tag{12}$$

A path is complete if it begins at the dummy source vertex $v^{\mathrm{d}}_{\mathrm{src}}$ and ends at the dummy sink vertex $v^{\mathrm{d}}_{\mathrm{snk}}$. Prioritizing the execution of vertices on the longest path decreases the ACT of multiple inference tasks. This is because the longest path represents the critical sequence of dependent tasks and executing these tasks earlier minimizes the overall waiting time. Alg.1 details the computation of the longest length $l(v_i)$ of complete path passing through v_i.

Algorithm 2 assigns a rank $r(v_i)$ to each vertex based on its longest complete path length $l(v_i)$. A lower rank value $r(v_i)$ indicates a higher execution priority for vertex v_i. The rank values are recursively assigned in ascending order following the topological order, ensuring that a vertex is ranked only after all its ancestors have been assigned. According to Lines 3 and 7 in Alg.2, vertex v_i with larger $l(v_i)$ is assigned with a lower rank value $r(v_i)$. As shown in Lines 3 and 7, a vertex with a larger $l(v_i)$ receives a lower rank $r(v_i)$.

Algorithm 1: Determine the Longest Path Length for Each Vertex

Input: $G = (V, E)$, $c(v_i)$ for each $v_i \in V$, vertex topological order Ω of G
Output: $l(v_i)$: the longest path passing through v_i, $l(v^{\mathrm{d}}_{\mathrm{src}}, v_i)$: the longest path length from $v^{\mathrm{d}}_{\mathrm{src}}$ to v_i, $l(v_i, v^{\mathrm{d}}_{\mathrm{snk}})$: the longest path length from v_i to $v^{\mathrm{d}}_{\mathrm{snk}}$, for each vertex $v_i \in V$.

1 $l(v^{\mathrm{d}}_{\mathrm{src}}) = 0, l(v^{\mathrm{d}}_{\mathrm{src}}, v^{\mathrm{d}}_{\mathrm{src}}) = 0, l(v^{\mathrm{d}}_{\mathrm{snk}}, v^{\mathrm{d}}_{\mathrm{snk}}) = 0$;
2 **for** $v_i \in V$ *following order* Ω *from* $v^{\mathrm{d}}_{\mathrm{src}}$ *to* $v^{\mathrm{d}}_{\mathrm{snk}}$ **do**
3 $\quad l(v^{\mathrm{d}}_{\mathrm{src}}, v_i) = c(v_i) + \max\{\ l(v^{\mathrm{d}}_{\mathrm{src}}, v_j) \mid v_j \in \mathrm{pre}(v_i)\ \}$;
4 **for** $v_i \in V$ *following reversed order* Ω *from* $v^{\mathrm{d}}_{\mathrm{snk}}$ *to* $v^{\mathrm{d}}_{\mathrm{src}}$ **do**
5 $\quad l(v_i, v^{\mathrm{d}}_{\mathrm{snk}}) = c(v_i) + \max\{\ l(v_j, v^{\mathrm{d}}_{\mathrm{snk}}) \mid v_j \in \mathrm{succ}(v_i)\ \}$;
6 $\quad l(v_i) = l(v^{\mathrm{d}}_{\mathrm{src}}, v_i) + l(v_i, v^{\mathrm{d}}_{\mathrm{snk}}) - c(v_i)$;

Algorithm 2: Rank Assignment for Each Vertex

Input: $G = (V, E)$, $l(v_i)$ for each $v_i \in V$, the next available rank value r
Output: Rank value $r(v_i)$ assigned to each vertex $v_i \in V$

1 $r = 0$;
2 **while** $V \neq \emptyset$ **do**
3 $\quad$ Find the vertex $v^* \in V$, such that its predecessor set $\mathrm{pre}(v^*) = \emptyset$ and v^* has the largest $l(v_i)$ value among all the vertices whose predecessor set is empty;
4 $\quad r(v^*) = r,\ r = r + 1,\ S = \mathrm{succ}(v^*)$;
5 $\quad$ Update G by removing v^* and its associated edge;
6 $\quad$ **while** $S \neq \emptyset$ **do**
7 $\quad\quad$ Find the vertex $\hat{v} \in S$ with the largest $l(v_i)$ value;
8 $\quad\quad$ **if** $\mathrm{pre}(\hat{v}) \neq \emptyset$ **then**
9 $\quad\quad\quad$ Construct a graph G_c comprising vertices from $\mathrm{anc}(\hat{v})$ and their conrresponding edges;
10 $\quad\quad\quad$ Call Rank Assignment Algorithm with the input G_c and r ;
11 $\quad\quad\quad$ Update G by removing $\mathrm{anc}(\hat{v})$ and their associated edges;
12 $\quad\quad$ $r(\hat{v}) = r,\ r = r + 1,\ S = \mathrm{succ}(\hat{v})$;
13 $\quad\quad$ Update G by removing $\hat{v}$ and its associated edges;

B. Marginal Delay Minimization based Greedy Algorithm

The proposed algorithm sequentially determines the offloading decision $b(i, m)$ for each subtask v_i, ordered by ascending rank values $r(v_i)$. For each ES s_m, the algorithm computes the delay increment incurred by allocating v_i to s_m by solving a relaxed linear program that minimizes the ACT of multiple inference tasks. This solution yields a feasible schedule for all subtasks ranked before v_i and including v_i. Finally, v_i is assigned to the ES that results in the least delay increment.

1) Linear Relaxation for Subtasks Scheduling with Fixed Offloading Decision: We denote $G = (V, E)$ as a DAG

constructed from T individual DAGs along with a dummy source/sink vertex, as described in Section III-C. Let $|V| = N$ and assume the vertices $v_1, v_2, \dots v_N$ are ordered in ascending order of their rank values. We denote the subset of vertices $\{v_1, \dots v_h\}$ as $V_{\text{as}}(h)$. We then present a method to determine the scheduling $t(v_i)$ of each subtask $v_i \in V_{\text{as}}(h)$, given the predetermined offloading decisions $b(i, m)$.

Even with fixed offloading decisions, Problem P1 remains challenging because constraint (8) imposes two disjunctive inequalities for each pair of parallel subtasks assigned to the same ES. Solving for the optimal solution requires considering an exponential number of candidate solutions. Notice that once the offloading decisions $b(i, m)$ are given, the computation delay $d_{\text{c}}(v_i)$ and transmission delay $d_{\text{tr}}(v_i)$ for each subtask can be easily computed. Following the approach in [44], we introduce a linear relaxation for constraint (8),

$$\sum_{i=1}^{h} d_{\text{c}}(v_i)t(v_i) \geq \frac{1}{2}\sum_{i=1}^{h} d_{\text{c}}^2(v_i) + \frac{(\sum_{i=1}^{h} d_{\text{c}}(v_i))^2}{2M}, \; V_{\text{as}}(h) \subset V. \quad (13)$$

We show that linear relaxation constraint (13) is a necessary condition for a feasible solution of problem P1.

Lemma 1. *For given offloading decisions $b(i, m)$, any feasible solution $t(v_i)$ of* P1 *satisfies linear relaxation constraint (13).*

We reorganize the cumulative delay sum over ESs and apply Jensen's inequality to establish a lower bound on the cumulative computation delay. The proof is given in App. IX-A.

With offloading decisions $b(i, m)$ fixed and constraint (8) relaxed, the opitmal scheduling $t(v_i)$ for each subtask $v_i \in V_{\text{as}}(h)$ is reduced to Problem P2,

$$\begin{aligned} \text{P2} \; \min_{t(v_i)} \; & \sum\nolimits_{k=1}^{T} \frac{\max\{t(v_i) | v_i \in G_k\}}{T}. \\ \text{s.t.} \; & \text{Constratins } (4) - (7), (9), (13), \\ & V_{\text{as}}(h) = \{v_1, \dots v_h\}, \\ & t(v_i) = 0, \text{ if } v_i \notin V_{\text{as}}(h). \end{aligned} \quad (14)$$

where G_k is the DAG for the kth inference task. Since P1 minimizes the ACT of all inference tasks, P2 specifically minimizes the ACT of the last subtask within $V_{\text{as}}(h)$ of all inference tasks. The last two constraints specify that P2 focuses on scheduling subtasks with given offloading decisions. With predetermined offloading, computation and transmission delays are known, so the delay increment is obtained by solving P2 via the simplex method.

Algorithm 3 processes subtasks in ascending order of rank, and assigns each v_i to the ES that minimizes its delay increment. Once the offloading decision for the final subtask v_N is determined, the start times for all sabtasks can be derived. Alg. 3 provides the pseudo-code for this greedy approach.

We compare the Earliest Finish Time (EFT) heuristic with Algorithm 3 for task scheduling and offloading in edge computing. EFT assigns each subtask to the edge server (ES) that can complete it the earliest. As illustrated in Fig. 3(b), we consider two edge servers: ES_1 (computing capacity: 4) and ES_2 (computing capacity: 2). Using EFT, subtasks $\{v_1, v_6, v_2, v_4\}$ are assigned to ES_1, while $\{v_3, v_5\}$ are offloaded to ES_2, resulting in an ACT of approximately 3.5. Alg. 3 optimizes task offloading and scheduling by leveraging server capabilities and parallel execution. By assigning $\{v_1, v_6, v_2\}$ to ES_1 and $\{v_3, v_4, v_5\}$ to ES_2, it enables concurrent execution of tasks like v_2 and v_4. This reduces the ACT to 2.8.

Algorithm 3: Marginal Delay Minimization based Subtask Allocation and Scheduling

Input: $v_1, v_2, \dots, v_N$ in ascending order of their rank values $r(v_i)$, compuation workload $c(v_i)$ of each subtask v_i, CPU frequency f_m of each edge server s_m
Output: Assignment decision $b(i, m)$, completion time $t(v_i)$ and start time $t_s(v_i)$ of each subtask v_i

1 $b(i, m) = 0, \; i \in \{1, \dots, N\}, \; m \in \{1, \dots, M\}$;
2 **for** $v_h \in V$ *in ascending order of rank value* $r(v_h)$ **do**
3 **for** *each edge server* $s_m \in \mathcal{S}$ **do**
4 $b(h, m) = 1$, $\{b(h, k) = 0 \mid k \neq m\}$, $V_{\text{as}}(h) = \{v_1, \dots v_h\}$;
5 Obtain objective value $C_{\text{avg}}(h, m)$ by sloving linear programming problem (14);
6 $\delta(h, m) = C_{\text{avg}}(h, m) - C_{\text{avg}}(h-1)$;
7 $\alpha = \arg\min\{\lambda(h, m) \mid m \in \{1, \dots, M\}\}$;
8 $b(h, \alpha) = 1$, $\{b(h, k) = 0 \mid k \neq \alpha\}$;
9 $C_{\text{avg}}(h) = C_{\text{avg}}(h, \alpha)$;
10 **for** $v_h \in V$ *in ascending order of rank value* $r(v_h)$ **do**
11 $t_s(v_i) = t(v_i) - d_{\text{c}}(v_i)$;

C. Performance Analysis of Initialization Optimization

Algorithm 3 addresses Problem P1 through two key procedures: (1) determining the offloading decision $b(i, m)$ for each subtask v_i in ascending order of their rank values, and (2) determining the scheduling $t_s(v_i)$ for each subtask based on assigned offloading decisions. First, we show that, under any topologically ordered execution, the subtask assignment achieves an approximation ratio of $1 + \frac{1}{e-1}$. Next, we prove the approximation ratio of the ACT achieved by Algorithm 3 with predetermined offloading decisions.

Lemma 2. *The objective function of problem* P1(2) *is submodular if the execution order of vertices $v_1, \dots, v_N$ follows a topological order.*

The proof shows that adding a subtask to a larger set results in a greater or equal increase in ACT than adding it to a smaller set, thus confirming the submodularity of the objective function. The proof is given in App. IX-B.

Theorem 2. *With an execution order that complies with the topological order, Algorithm 3 achieves an approximation ratio of $1 + \frac{1}{e-1}$ for subtask offloading.*

Proof. In each iteration for subtask v_h, the algorithm computes the objective value for each candidate offloading decision on ES s_m by solving the linear programming problem (14) (line 5), and then greedily selects the assignment that minimizes the delay increment (lines 7 and 8). Since the objective function of Problem P1(2) is submodular, this guarantees the overall approximation ratio of $1 + \frac{1}{e-1}$ [45]. □

By dividing subtasks based on their assigned ESs and applying Graham's bound to the remaining subtasks, we show

that the ACT obtained from Algorithm 3 is within a constant factor of the optimal ACT for Problem P1.

Theorem 3. *Given fixed subtask offloading decisions, Alg. 3 determines the completion time of each subtask. For each inference task τ_k represented by G_k, let $v_{k,\mathrm{snk}}$ denote the subtask with the highest ranking in G_k, i.e., $v_{k,\mathrm{snk}} = \arg\max\{r(v_i)|v_i \in G_k\}$. The completion time satisfies:*

$$t(v_{k,\mathrm{snk}}) \leq (1 + \frac{2f_{\max}}{f_{\min}})t^{\mathrm{opt}}(v_{k,\mathrm{snk}})$$

where $t^{\mathrm{opt}}(v_{k,\mathrm{snk}})$ is the optimal completion time for P1 under the same offloading decisions, and $f_{\max}$ and $f_{\min}$ are the maximum and minimum CPU frequencies among the ESs.

Proof. Assume that Algorithm 3 follows an subtask ranking $v_1, v_2, \ldots, v_N$, and without loss of generality, let $v_{k,\mathrm{snk}} = v_h$. Define $S(v_h) = \{v_1, \ldots, v_h\}$. Since the offloading decisions are fixed, the transmission delay is identical for both our algorithm and the optimal solution, and thus can be omitted. Suppose v_h is offloaded to ES s_m; then there exists a subset

$$S(v_h, m) = \{v_{m1}, \ldots, v_{ml}, v_h\} \subset V_k$$

where each suatask v_{mi} is executed on s_m in sequence. Thus, v_{mi} is an ancestor of $v_{m(i+1)}$(by the precedence constraints), and the execution order on $v_{m1}, \ldots, v_{ml}, v_h$. We divide $S(v_h)$ into two disjoint subsets: $S(v_h, m)$ and $S(v_h) \setminus S(v_h, m)$. Let $d_{\mathrm{c}}(S(v_h, m))$ and $d_{\mathrm{c}}^{\mathrm{opt}}(S(v_h, m))$ denote the computation delays for processing $S(v_h, m)$ by our algorithm and the optimal solution, respectively. As the offloading decisions are predetermined, we have

$$\begin{aligned} d_{\mathrm{c}}(S(v_h, m)) &= d_{\mathrm{c}}^{\mathrm{opt}}(S(v_h, m)) \\ &\leq d_{\mathrm{c}}^{\mathrm{opt}}(S(v_h, m)) + d_{\mathrm{c}}^{\mathrm{opt}}(S(v_h) \setminus S(v_h, m)) \\ &= t^{\mathrm{opt}}(v_h) \end{aligned} \tag{15}$$

Following the necessary condition from relaxation constraint (13) (see Lem.1), we derive

$$\begin{aligned} t^{\mathrm{opt}}(v_h) \sum\nolimits_{i=1}^{h} d_{\mathrm{c}}(v_i) &\geq \sum\nolimits_{i=1}^{h} d_{\mathrm{c}}(v_i) t^{\mathrm{opt}}(v_i) \\ &\geq \frac{1}{2} \sum\nolimits_{i=1}^{h} d_{\mathrm{c}}^2(v_i) + \frac{(\sum_{i=1}^{h} d_{\mathrm{c}}(v_i))^2}{2M}. \end{aligned} \tag{16}$$

which implies $t^{\mathrm{opt}}(v_h) \geq \frac{\sum_{i=1}^{h} d_{\mathrm{c}}(v_i)}{2M}$. By applying Graham's bound [46], the computation delay for the subtasks in $S(v_h) \setminus S(v_h, m)$ is bounded by:

$$\begin{aligned} d_{\mathrm{c}}(S(v_h) \setminus S(v_h, m)) &\leq \frac{\sum_{v_i \in S(v_h) \setminus S(v_h, m)} c(v_i)}{M f_{\min}} \\ &\leq \frac{\sum_{i=1}^{h} d(v_i)}{M} \cdot \frac{f_{\max}}{f_{\min}} \\ &\leq \frac{2f_{\max}}{f_{\min}} t^{\mathrm{opt}}(v_h) \end{aligned}$$

Thus, combining (15) and (17), the total delay is:

$$\begin{aligned} t(v_h) &= d_{\mathrm{c}}(S(v_h, m)) + d_{\mathrm{c}}(S(v_h) \setminus S(v_h, m)) \\ &\leq (1 + \frac{2f_{\max}}{f_{\min}}) t^{\mathrm{opt}}(v_h) \end{aligned} \tag{17}$$

This completes the proof. □

Corollary 1. *With fixed subtask offloading decisions, our algorithm guarantees an approximation ratio for the ACT of T inference tasks of at most $(1 + \frac{2f_{\max}}{f_{\min}})$.*

Theorem 4. *The approximation ratio bound of Alg.3 is $(1 + \frac{2f_{\max}}{f_{\min}})(1 + \frac{1}{e-1})$, where $f_{\max}$ and $f_{\min}$ denote the maximum and minimum CPU frequencies among the ESs, respectively.*

Proof. The proof combines separate approximation bounds on the offloading and scheduling procedure to show that their product yields the overall approximation ratio. Algorithm 3 solves Problem P1 through two procedures: (1) determining the offloading decision $b(i, m)$ for each subtask v_i and (2) scheduling its beginning time $t_s(v_i)$ based on those offloading decisions. Let $S^{\mathrm{opt}}(A^{\mathrm{opt}})$ denote the optimal solution (with S^{opt} representing the optimal scheduling and A^{opt} the optimal offloading). According to Thm.2, given optimal scheduling S^{opt}, the offloading decisions A^{gre} returned by Alg.3 satisfy:

$$\frac{S^{\mathrm{opt}}(A^{\mathrm{gre}})}{S^{\mathrm{opt}}(A^{\mathrm{opt}})} \leq 1 + \frac{1}{e-1}. \tag{18}$$

Furthermore, by Corollary 1, with the fixed assignment A^{gre}, the scheduling S^{gre} returned by Algorithm 3 satisfies:

$$\frac{S^{\mathrm{gre}}(A^{\mathrm{gre}})}{S^{\mathrm{opt}}(A^{\mathrm{gre}})} \leq 1 + \frac{2f_{\max}}{f_{\min}}. \tag{19}$$

Combining (18) and (19), we have

$$\begin{aligned} \frac{S^{\mathrm{gre}}(A^{\mathrm{gre}})}{S^{\mathrm{opt}}(A^{\mathrm{opt}})} &\leq \frac{S^{\mathrm{gre}}(A^{\mathrm{gre}})}{S^{\mathrm{opt}}(A^{\mathrm{gre}})} \cdot \frac{S^{\mathrm{opt}}(A^{\mathrm{gre}})}{S^{\mathrm{opt}}(A^{\mathrm{opt}})} \\ &\leq (1 + \frac{2f_{\max}}{f_{\min}})(1 + \frac{e}{e-1}). \end{aligned} \tag{20}$$

This completes the proof. □

V. Inference Service Framework Integrating INOP with IFA

In this section, we introduce the Improved Firefly Algorithm (IFA) to enhance INOP's initial subtask ranking. Inspired by fireflies' social behavior, IFA employs a linear position update strategy that ensures each firefly's position corresponds to a valid subtask ranking. This refined ranking is then integrated with INOPs greedy method to finalize offloading and scheduling decisions. In addition, we analyze IFA's convergence and provide a sufficient condition to avoid boundary traps.

A. Solution Representation

We define the firefly population as $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_{|\Phi|}\}$, where $|\Phi|$ is the population size. Each firefly $\varphi_i \in \Phi$ has a position represented by an N-dimensional vector $(p_{i1}, p_{i2}, \ldots, p_{iN})$, where N is the number of vertices in DAG G. The position of a firefly corresponds to an execution order of vertices $v_1, v_2, \ldots, v_N$. In the firefly algorithm, fireflies with lower brightness are attracted to and adjust their positions based on brighter fireflies. We introduce a low-complexity position update strategy and establish a mapping between firefly positions and subtasks rankings, enabling broader exploration of high-quality solutions.

B. Linear Position Update Strategy

During the t-th iteration, the brightest firefly in the population, denoted as $\varphi^{\min}(t)$, has a position $(p_1^{\min}(t), \ldots, p_N^{\min}(t))$ corresponding to the subtasks rankings that minimize the ACT. Each firefly φ_i update its position $p_{ij}(t+1)$ in the $(t+1)$-th iteration as follows:

$$\begin{aligned} p_{ij}(t+1) = p_{ij}(t) + \eta \left(p_j^{\min}(t) - p_{ij}(t)\right) \\ + \mu(\varepsilon(t) - \frac{1}{2}),\ j = 1, 2, \ldots, N. \end{aligned} \tag{21}$$

where η and μ are two coefficients satisfying $\eta \in (0, 1)$ and $\eta + \mu = 1$. $\varepsilon(t)$ represents the mean of t independent and identically distributed samples drawn from a uniform distribution over $(0, 1)$. It is computed recursively as: $\varepsilon(t) = \frac{(t-1)\varepsilon(t-1)+\varepsilon}{t}$, where ε is a random variable sampled from the same uniform distribution.

The traditional firefly algorithm updates each firefly's position by iterating over every brighter firefly in the population, requiring $\frac{|\Phi|(|\Phi|-1)}{2}$ iterations. Since each update involves computing the Euclidean distance, with a complexity of $O(N)$ per computation, the overall complexity for updating all firefly positions is $O(N|\Phi|^2)$. In contrast, our linear update strategy reduces this complexity to $O(|\Phi|)$.

C. Mapping Algorithm

The traditional firefly algorithm utilizes a ranked-order value (ROV) mapping method, where firefly positions are sorted in ascending or descending order to determine subtask rankings. However, this approach fails to leverage the advantages of the current optimal firefly position and does not enforce precedence constraints. Moreover, ROV-based mapping often leads to boundary trapping, causing convergence to identical subtask rankings after a few iterations. To overcome these issues, we propose a precedence-aware probabilistic mapping algorithm to explore high-quality subtask rankings. By incorporating the position of the current optimal firefly, our method inherits beneficial characteristics from the best solution while ensuring precedence constraints are maintained.

The mapping algorithm takes as input the firefly position $\varphi_i = (p_{i1}, \cdots, p_{iN})$ and the current optimal execution order $(v_1^{\min}, \cdots, v_N^{\min})$, with its corresponding firefly position $(p_1^{\min}, \ldots, p_N^{\min})$ as detailed in Section V-B. The output is a feasible subtask ranking that satisfies the precedence constraints. First, we initialize an empty subtask ranking $\tau_i = (\tau_{i1}, \ldots, \tau_{iN})$, where each τ_{ij} is initially set to $\varnothing$. We then generate a random vector $(\gamma_1, \ldots, \gamma_N)$, with each γ_i drawn from a uniform distribution over $(0, 1)$. For each p_{ij}, if $\frac{1}{1+e^{p_{ij}}} > \gamma_j$, the j-th element τ_{ij} of vector τ_i is assigned $v_j^{\min}$. This means that in the mapping subtask ranking, the one ranked at the j-th is $v_j^{\min}$. Otherwise, τ_{ij} remains $\varnothing$. Finally, any unassigned subtasks from $(v_1^{\min}, \cdots, v_N^{\min})$ are allocated at remaining positions of τ_i with equal probability when they are parallel vertices. This approach generates a feasible τ_i that balances exploration of execution orders with guidance from the current best solution.

Algorithm 4: Precedence-Aware Probabilistic Mapping Algorithm

Input: $\varphi_i = (p_{i1}, \cdots, p_{iN})$ and the current optimal execution order $(v_1^{\min}, \cdots, v_N^{\min})$
Output: A feasible subtask ranking $\tau_i = (\tau_{i1}, \ldots, \tau_{iN})$
1 $(o_1, \ldots, o_N) = (v_1^{\min}, \cdots, v_N^{\min})$;
2 Initialize an empty execution order $\tau_i = (\tau_{i1}, \ldots, \tau_{iN})$ with each $\tau_{ij} = \varnothing$;
3 Generate a random vector $(\gamma_1, \ldots, \gamma_N)$;
4 **foreach** p_{ij} **do**
5 **if** $\frac{1}{1+e^{p_{ij}}} \geq \gamma_j$ **then**
6 $\tau_{ij} = o_j$; $o_j = \varnothing$;
7 $C = 1$;
8 **foreach** $v_j^{\min}$ **do**
9 **if** $v_j^{\min}$ *is parallel with* $v_{j+1}^{\min}$ **then**
10 $C = C + 1$;
11 **else**
12 For the first C empty elements in τ_i, $\tau_{ij} = o_j$ with probabtily $\frac{1}{C}$;
13 $C = 1$;

D. Algorithm Design

Algorithm 5 details the inference service framework that integrates INOP with IFA. The initialization phase includes both population initialization and the initialization of the current optimal subtask ranking (Lines 14). Each firefly's initial position is randomly generated from a uniform distribution over $[p_{\mathrm{L}}, p_{\mathrm{U}}]$, where p_{L} and p_{U} define the position domain's lower and upper bounds. The initial optimal subtask ranking $(v_1, v_2, \ldots, v_N)$ is determined by sorting the subtasks in ascending order based on their rank values $r(v_i)$. The algorithm then iteratively refines the subtask ranking, offloading, and scheduling decisions: in each iteration, firefly positions are updated (Line 8), converted into a precedence-aware subtask ranking via Algorithm 4 (Line 13), and the objective value Δ_i for Problem P1 is evaluated using Algorithm 3 (Line 14). When a firefly with a lower objective value is found, the current optimal position, its corresponding subtask ranking, and objective value are updated (Lines 16-18). The iterative process terminates once the maximum number of iterations is reached, and the best-found ACT is returned as the final result.

E. Performance Analysis

Theorem 5. *The computation complexity of IFA algorithm is* $O(G_{\max}|\Phi|N^{3.5}M)$.

Proof. The complexity of generating the initial firefly population is $O(|\Phi|N)$ (Lines1-2). Given a feasible subtask ranking as input, Algorithm 3 involves two nested loops: the outer loop iterates over N subtasks, while the inner loop iterates over M available ESs, leading to $N \times M$ iterations. In each iteration, solving the relaxed linear programming problem (14) has an approximate complexity of $O(N^{2.5})$. Thus, the overall time complexity of Algorithm 3 is $O(N^{3.5}M)$ (Lines 4 and 14).

Algorithm 5 executes $G_{\max} \times |\Phi|$ iterations. In each iteration, the position update and mapping strategy have complexities of $O(1)$ and $O(N)$, respectively. The complexity of

Algorithm 5: Inference Service Framework Integrating INOP with IFA

Input: $v_1, v_2, \ldots, v_N$ in ascending order of their rank values $r(v_i)$, η, μ, $G_{\max}$ and $t = 0$

Output: Optimal subtask ranking $(v_1^{\min}, \cdots, v_N^{\min})$, offloading decision $b(i,m)$, scheduling decision for completion time $t(v_i)$ and start time $t_s(v_i)$ of each subtask v_i

Initialize firefly position of each firefly $\varphi_i(0)$ with $p_{ij}(0) \sim U[p_{\mathrm{L}}, p_{\mathrm{U}}]$;
Initialize firefly position $(p_1^{\min}, \cdots, p_N^{\min})$ with $p_i^{\min} \sim U[p_{\mathrm{L}}, p_{\mathrm{U}}]$;
$(v_1^{\min}, \cdots, v_N^{\min}) = (v_1, \ldots, v_N)$;
Obtain objective value $\Delta_{\min}$ of problem P1 by calling Algorithm 3;
while $t < G_{\max}$ **do**
  foreach $\varphi_i \in \Phi$ **do**
    foreach j **do**
      $p_{ij}(t+1) = p_{ij}(t) + \eta\left(p_j^{\min} - p_{ij}(t)\right) + \mu\left(\varepsilon(t) - \frac{1}{2}\right)$;
      if $p_{ij}(t+1) \le p_{\mathrm{L}}$ **then**
        $p_{ij}(t+1) = p_{\mathrm{L}}$;
      if $p_{ij}(t+1) \ge p_{\mathrm{U}}$ **then**
        $p_{ij}(t+1) = p_{\mathrm{U}}$;
    Obtain subtasks rankings $\tau_i = (\tau_{i1}, \ldots, \tau_{iN})$ by calling Algorithm 4;
    Obtain objective value Δ_i of problem P1 by calling Algorithm 3 with the subtasks rankings τ_i as input;
    if $\Delta_i < \Delta_{\min}$ **then**
      $(v_1^{\min}, \cdots, v_N^{\min}) = (\tau_{i1}, \ldots, \tau_{iN})$;
      $(p_1^{\min}, \cdots, p_N^{\min}) = (p_{i1}, \ldots, p_{iN})$;
      $\Delta_{\min} = \Delta_i$;
  $t = t + 1$;

computing the objective value is $O(N^{3.5}M)$. Therefore, the time complexity of Algorithm 5 is $O(G_{\max}|\Phi|N^{3.5}M)$. □

Theorem 6. *When $\eta \in (0,1)$, the proposed IFA converges with probability 1 to the global optimal position $\varphi^{\mathrm{opt}} = (p_1^{\mathrm{opt}}, \cdots, p_N^{\mathrm{opt}})$. Specifically, for each firefly $\varphi(i)$ in the population, we have*

$$(p_{i1}(t), \cdots, p_{iN}(t)) \xrightarrow{a.s.} (p_1^{\mathrm{opt}}, \cdots, p_N^{\mathrm{opt}}), \text{ as } t \to \infty. \quad (22)$$

Proof. Following the convergence analysis in [39], [47], we assume the global optimal position φ^{opt} is constant and the impact of random variables within the population is negligible, treating them as constants. As each dimension of a firefly's position is independent and follows the same update strategy, we focus on the j-th demension of the firefly's position. According to equation (21), we can derive the following,

$$\begin{aligned} &p_{ij}(t+1) + (\mu\eta^{-1}(2^{-1} - \varepsilon(t)) - p_j^{\mathrm{opt}}) \\ &= (1-\eta)\left(p_{ij}(t) + \mu\eta^{-1}(2^{-1} - \varepsilon(t-1)) - p_j^{\mathrm{opt}}\right). \end{aligned}$$

By iterating this equation, we get:

$$\begin{aligned} &p_{ij}(t) + (\mu\eta^{-1}(2^{-1} - \varepsilon(t)) - p_j^{\mathrm{opt}}) \\ &= (1-\eta)^t\left(p_{ij}(0) + \mu\eta^{-1}(2^{-1} - \varepsilon(0)) - p_j^{\mathrm{opt}}\right). \end{aligned} \quad (23)$$

Since $\eta \in (0,1)$, we know that $\lim_{t\to\infty}(1-\eta)^t = 0$. We have:

$$\lim_{t\to\infty} p_{ij}(t) = p_j^{\mathrm{opt}} - \mu\eta^{-1}(2^{-1} - \varepsilon(t)). \quad (24)$$

Based on (21), $\varepsilon(t)$ is the mean of t independent and identically distributed samples drawn from a uniform distribution over $(0,1)$. According to the law of large numbers, we know that $\varepsilon(t)$ with probability 1 converges to 2^{-1}, i.e., $\varepsilon(t) \xrightarrow{a.s.} 2^{-1}$, as $t \to \infty$. We can derive the following,

$$p_{ij}(t) \xrightarrow{a.s.} p_j^{\mathrm{opt}}, \text{ as } t \to \infty. \quad (25)$$

Since this holds for every dimension j, the entire position vector converges with probability 1 to the global optimal position φ^{opt}, completing the proof. □

In the following, we show that our algorithm avoids boundary traps when the optimal firefly position $\varphi^{\mathrm{opt}} = (p_1^{\mathrm{opt}}, \cdots, p_N^{\mathrm{opt}})$ lies within $[p_{\mathrm{L}}, p_{\mathrm{U}}]^N$. Since each dimension of a firefly's position is updated independently using the same strategy, we focus on the j-th dimension.

Lemma 3. *If $p_j^{\mathrm{opt}} < p_{\mathrm{L}}$ and $\eta \in (0,1)$, then for each $\varphi_i \in \Phi$, $p_{ij}(t) = p_{\mathrm{L}}$ with probability 1, i.e., $p_{ij}(t) \xrightarrow{a.s.} p_{\mathrm{L}}$, as $t \to \infty$.*

The update rule combined with the lower (upper) bound enforcement ensures that the firefly positions converge with probability 1 to $p_{\mathrm{L}}(p_{\mathrm{U}})$. The proof is given in App. IX-D(IX-D).

Lemma 4. *If $p_j^{\mathrm{opt}} > p_{\mathrm{U}}$ and $\eta \in (0,1)$, then for each $\varphi_i \in \Phi$, $p_{ij}(t) = p_{\mathrm{U}}$ with probability 1, i.e., $p_{ij}(t) \xrightarrow{a.s.} p_{\mathrm{U}}$, as $t \to \infty$.*

Lemma 5. *If $p_j^{\mathrm{opt}} \in [p_{\mathrm{L}}, p_{\mathrm{U}}]$ and $\eta \in (0,1)$, then the proposed IFA algorithm avoids boundary traps with probability 1.*

The Lemma is proved by contradiction and given in App. IX-E.

Theorem 7. *If the optimal firefly position $\varphi^{\mathrm{opt}} = (p_1^{\mathrm{opt}}, \cdots, p_N^{\mathrm{opt}}) \in [p_{\mathrm{L}}, p_{\mathrm{U}}]^N$ and $\eta \in (0,1)$, the proposed IFA algorithm avoids boundary traps with probability 1.*

Proof. Each dimension of a firefly's position is independent and follows the same update strategy. Therefore, the proof is completed by applying Lemma 5 to each dimension. □

VI. Simulation Results

This section evaluates the performance of the proposed inference framework through simulations implemented in Python 3.7 on a PC with an Intel Core i7-2650H CPU (2.3GHz) and 16GB RAM. In the first set of experiments, we use randomly generated data to compare the performance of the INOP and IFA algorithms with other firefly algorithms. The second set employs real-world DAG structures to compare our approach with existing task scheduling algorithms for dependent tasks in MEC and multi-task flows.

A. Algorithm Comparison with Synthetic Data

We model inference tasks using randomly generated DAG structures characterized by the following parameters: 1) The total number of subtasks in a single inference service ranges from 10 to 100; 2) Since inference service DAGs typically

have narrow widths, the out-degree of each subtask is randomly selected from $[1, 10]$ using a uniform distribution; 3) The computation workload of each subtask v_i follows a uniform distribution in $[1, 5]$. The number of ESs is set to 10, and the CPU frequency f_m is uniformly distributed within $[4, 8]$.

The parameter η, representing the attractiveness between two fireflies, plays a key role in the linear position update strategy (see (21)). As shown in Section V-E, η should range from 0 to 1 to ensure IFA convergence. We evaluate IFA's performance by analyzing the median of average ACT and least-significant difference (LSD) intervals at a 95% confidence level under various η values. For each data point, 1000 random experiments are conducted. The results in Fig.4(a) indicate that IFA achieves the lowest mean ACT when $\eta = 0.5$. Thus, $\eta = 0.5$ is used in all subsequent experiments. Fig.4(b) shows the run time of our inference framework when the number of inference services (T) is 20 and the number of iterations is set to 500. The run time increases with the number of populations because our framework must first identify the optimal firefly within each population and then select the best among these. For a inference service comprising 50 subtasks per inference service (1000 vertices in total), the run time remains below 700 ms when the population number $m = 1$, confirming the computational efficiency of our framework.

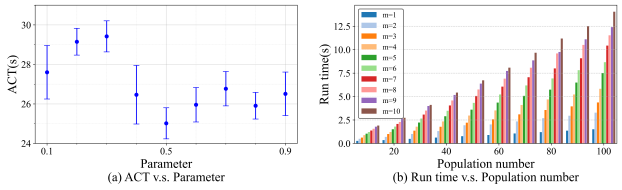

Fig. 4. The impact of parameter on the performance of our framework

We compare the proposed inference framework with the following three benchmark schemes, which utilize different FA strategies to generate subtask ranks. The ACT is then determined by inputting these subtask ranks into Alg. 3.

- DROV: Uses the traditional Euclidean distance-based position update method and the ROV mapping technique [48], [49] to generate subtask ranks.
- LROV: Employs our linear position update strategy (21) with the ROV mapping technique to create subtask ranks.
- DPM: Combines the traditional Euclidean distance-based position update method with our precedence-aware probabilistic mapping algorithm to generate subtask ranks.

All FA-based strategies in following evaluations use identical parameters: a maximum of 500 iterations and a firefly position range of $[-2, 2]$ per dimension. Each data point is derived from 1000 random experiments, with results presented as averages.

Fig. 5 shows that our INOP significantly accelerates ACT convergence. In these experiments, the number (T) of inference services is 20 and the population size is 10. As depicted in Fig.5(a), our INOP nearly achieves the optimal solution within just a few iterations, while the baseline DROV converges linearly over 0 to 1000 iterations. When a single inference service comprises 100 subtasks, our IFA framework converges to the optimal solution around 300 iterations (Fig. 5(b)). Moreover, IFA enhances computational efficiency, delivering an 11-fold speedup over DROV, which reaches comparable performance only after 3300 iterations.

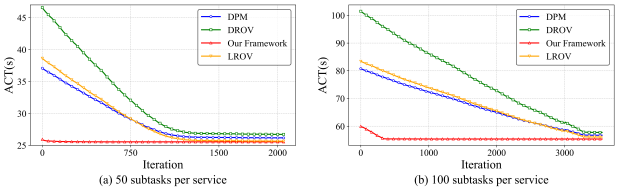

Fig. 5. Convergence comparison with the baseline FA

Fig. 6(a) shows that our precedence-aware mapping algorithm generates superior subtask rankings compared to state-of-the-art methods. In these experiments, the ACT increases with the number of subtasks per inference service due to the fixed number of ESs, which lengthens processing times as the computational load rises. Our framework achieves a 60% latency reduction compared to the baseline FA strategy DROV. Fig. 6(b) presents the approximation ratio (AR) of our framework and IFA through ablation experiments. AR of our framework (or IFA) is defined as the ratio of the ACT from Alg. 5 (or IFA) to that of the optimal solution, which is derived by an SMT solver. Due to the problem's intractability, each inference service contains between 10 and 15 subtasks, and results are based on a single run per data point. Fig. 6(b) indicates that our framework maintains a relatively stable AR, consistent with Th.4, which shows that AR depends on the ratio of the maximum to minimum CPU frequency. Since the standalone IFA lacks INOP and linear programming for scheduling parallel tasks on the same ES, it exhibits an AR that increases with the number of tasks.

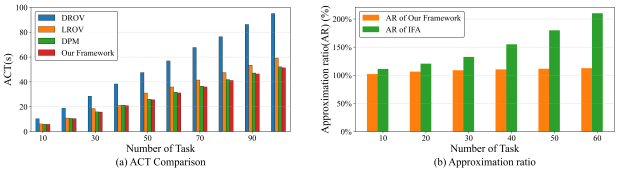

Fig. 6. Performance comparison in ACT and AR

B. Framework Evaluation with Real-World Applications

In the second set of experiments, we generate real-world DAG structures using the GE and Fast Fourier Transform (FFT) applications [28], incorporating real measurements while varying task numbers and DAG structures. Based on measurements in [50], the subtask data size $d(v_i)$ is uniformly distributed between 0.5 and 4 MB, and the required CPU cycles range from 200 to 1000 cycles/bit. The simulation uses 10 ESs with CPU frequencies uniformly distributed between 4 and 8 GHz and ES data transmission rates from 10 to 20 Mbps. We compare our framework with existing task scheduling algorithms for dependent tasks in MEC and multi-task flows, including:

- DDPG: A DDPG-based subtask offloading and scheduling scheme that optimizes deadline violations by combining task migration and merging [3].

- DQN: A DQN-based scheduling framework for multiple real-time workflows aimed at minimizing makespan [16].
- DDQN: A DDQN-based scheduling framework for multiple workflows in industrial IoT, addressing multi-objective optimization [17].

All strategies are trained for 100 epochs. Each data point is derived from 20 random experiments, and results are reported as averages.

Fig. 7(a) shows that our framework outperforms DRL-based methods by achieving a lower ACT. DRL methods struggle with hybrid discrete-continuous action spaces, where discretizing continuous scheduling actions or relaxing discrete offloading actions into continuous spaces introduces approximation errors. We evaluate algorithm performance using the completion ratio (CR), defined as the proportion of tasks completed before deadlines. Fig. 7(b)-(d) presents the CR under various deadlines as the number of inference tasks increases from 20 to 100. Here, G represents a DAG consisting of multiple server tasks, and the deadline determined using Graham bound:

$$\text{Deadline} = \frac{\text{Len}(G)}{f_{avg}} + \frac{\text{Vol}(G) - \text{Len}(G)}{M f_{avg}} \tag{26}$$

where $\text{Len}(G)$ is the length of the longest path in G, f_{avg} is the average CPU frequency of all ESs, $\text{Vol}(G)$ is the total workload of all vertices in G, and M is the number of ESs. We define three deadline scenarios: tight ($0.4\times$Deadline), medium ($0.6\times$Deadline), and loose ($0.4\times$Deadline). Fig. 7(b)-(d) illustrates the CR across these deadlines as the number of inference tasks varies from 20 to 100. Our framework achieves a higher CR than competing methods in all scenarios. CR increases as the number of tasks grows. This is because, while the total workload $\text{Vol}(G)$ rises significantly with more tasks, the longest path $\text{Len}(G)$ in practical DAGs remains relatively stable. According to Eq.(26), the deadline grows substantially, enabling more tasks to complete before deadlines.

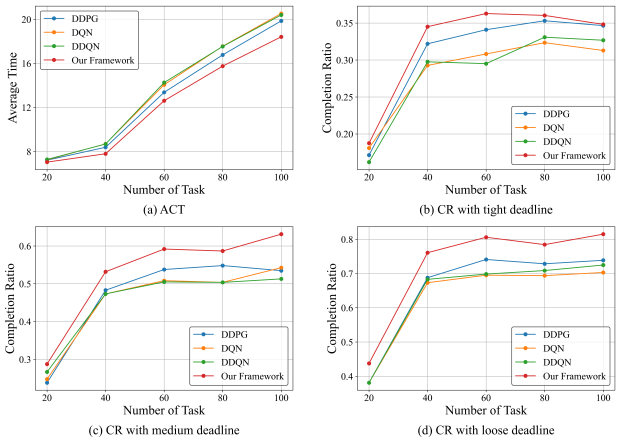

Fig. 7. Performance comparison in ACT and CR

Due to the inherent randomness in the firefly algorithm, we evaluate its performance using the relative percentage deviation (RPD) across varying numbers of inference tasks. RPD is defined as:

$$RPD = \frac{1}{K}\left(\sum_{k=1}^{K} \frac{ACT_k - ACT_{\min}}{ACT_{\min}}\right),$$

where $ACT_{\min}$ is the ACT obtained in the k-th round, and $ACT_{\min}$ is the minimum ACT across all K replications. A lower RPD indicates better algorithm effectiveness. We set $K = 10$ in all evaluation experiments. As shown in Fig. 8(a), our framework achieves lower RPD than competing methods. This improvement arises from initialization optimization and IFAs convergence, which enhance solution exploration. Fig. 8 shows that our framework has a shorter runtime compared to benchmark algorithms. This is attributed to its polynomial time complexity and independence from deep network structures, resulting in faster execution. Additionally, our framework exhibits lower runtime compared to benchmark algorithms (see Fig. 8)(b), owing to its polynomial time complexity and independence from deep network structures, resulting in faster execution.

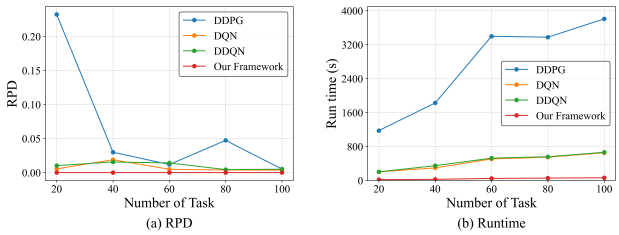

Fig. 8. Performance comparison in RPD and Runtime.

VII. Conclusion

In this paper, we study computation offloading for inference services with subtask-dependency requirements in MEC system. The proposed framework integrates a greedy algorithm for initialization optimization with an IFA to explore efficient subtask rankings. The greedy algorithm is designed to optimize subtask scheduling by minimizing the makespan, ensuring a proven constant approximation ratio. The IFA utilizes a precedence-aware probabilistic mapping operator to establish feasible subtask rankings. Extensive simulations indicate that our scheme significantly reduces system latency and convergence time compared to existing benchmark schemes across various scenarios.

VIII. Acknowledgement

The authors would like to thank the editor and anonymous reviewers for their valuable comments/suggestions that help us improve this paper.

IX. Appendix

A. *Proof of Lemma 1*

Assume $\{t(v_i)\}$ is a feasible solution of P1. By establishing an execution order for subtasks $v_1, v_2, \ldots, v_N$, we have $t(v_{i+1}) \geq t(v_i) + d_c(v_i)$ when subtasks v_i and v_{i+1} are assigned to the same ES. For any subtask v_h, we have

$$\begin{aligned}\sum_{i=1}^{h} d_c(v_i)t(v_i) &= \sum_{s=1}^{M}\sum_{b(i,s)=1} d_c(v_i)t(v_i) \\ &\geq \sum_{s=1}^{M}\sum_{b(i,s)=1} d_c(v_i)\Big(\sum_{j\leq i} d_c(v_j)\Big).\end{aligned}$$

Focusing on the right-hand side, we have

$$\begin{aligned}\text{RHS} &= \sum_{s=1}^{M}\{\sum_{b(i,s)=1} d_{\mathrm{c}}^2(v_i) + \sum_{b(i,s)=1} d_{\mathrm{c}}(v_i)\sum_{j<i} d_{\mathrm{c}}(v_j)\} \\ &= \sum_{s=1}^{M}\frac{1}{2}\{\sum_{b(i,s)=1} d_{\mathrm{c}}^2(v_i) + (\sum_{b(i,s)=1} d_{\mathrm{c}}(v_i))^2\} \\ &= \frac{1}{2}\sum_{i=1}^{h} d_{\mathrm{c}}^2(v_i) + \frac{1}{2}\sum_{s=1}^{M}(\sum_{b(i,s)=1} d_{\mathrm{c}}(v_i))^2 \\ &\geq \frac{1}{2}\sum_{i=1}^{h} d_{\mathrm{c}}^2(v_i) + \frac{(\sum_{i=1}^{h} d_{\mathrm{c}}(v_i))^2}{2M} \quad \text{(Jensen's inequality)}\end{aligned}$$

This completes the proof.

B. Proof of Lemma 2

For any two subsets $V_1 \subset V_2 \subset V$, let the induced subgraphs be $G_1^{\mathrm{in}} = (V_1, E_1)$ and $G_2^{\mathrm{in}} = (V_2, E_2)$, respectively, with $G_1^{\mathrm{in}} \subset G_2^{\mathrm{in}}$. Let $C_{\mathrm{avg}}(G)$ denote the ACT of T inference-service tasks, defined:

$$C_{\mathrm{avg}}(G_{1(2)}^{\mathrm{in}}) = \sum_{k=1}^{T}\frac{\max\{t(v_i)|v_i \in G_k\}}{T}.$$

We know that $C_{\mathrm{avg}}(G_{1(2)}^{\mathrm{in}})$ is the objective function of Problem P2 when $V_{\mathrm{as}}(h) = V_{1(2)}$. Now, consider adding a new subtask $v_n \notin V_2$. Let v_h be the last subtask in G_k contained in G_2^{in}, i.e. $t(v_h) = \max\{t(v_i)|v_i \in G_k \cap G_2^{\mathrm{in}}\}$, and denote by $t(v_h, G_{1(2)}^{\mathrm{in}})$ and $t(v_h, G_{1(2)}^{\mathrm{in}} \cup v_n)$ the completion time of v_h with subtask sets $V_{1(2)}$ and $V_{1(2)} \cup v_n$, respectively. There are two cases:

- **Case 1:** $v_h \notin V_1$. By the last constraint of Problem P2, as shown in formula (14), we have

$$t(v_h, G_1^{\mathrm{in}}) = t(v_h, G_1^{\mathrm{in}} \cup v_n) = 0 \tag{27}$$

Since $G_2^{\mathrm{in}} \subset G_2^{\mathrm{in}} \cup v_n$, v_h has more ancestors in $G_2^{\mathrm{in}} \cup v_n$, so

$$t(v_h, G_2^{\mathrm{in}} \cup v_n) - t(v_h, G_2^{\mathrm{in}}) \geq 0 \tag{28}$$

Combine formulas (27) and (28), we can derive

$$t(v_h, G_2^{\mathrm{in}} \cup v_n) - t(v_h, G_2^{\mathrm{in}}) \geq t(v_h, G_1^{\mathrm{in}} \cup v_n) - t(v_h, G_1^{\mathrm{in}}) \tag{29}$$

- **Case 2:** $v_h \in V_1$. The offloading decisions of each subtask for G_1^{in} are identical to those for G_2^{in}, resulting in two subcases.
  1) If v_n does not interfere with subtask v_h. we know that
  $$t(v_h, G_2^{\mathrm{in}} \cup v_n) - t(v_h, G_2^{\mathrm{in}}) = 0,$$
  $$t(v_h, G_1^{\mathrm{in}} \cup v_n) - t(v_h, G_1^{\mathrm{in}}) = 0.$$
  2) If v_n interferes with subtask v_h, then because $G_1^{\mathrm{in}} \subset G_2^{\mathrm{in}}$, v_n has more ancestors and parallel vertices in G_2^{in} than in G_1^{in}. Thus v_n experiences a longer wait for its execution in $G_2^{\mathrm{in}} \cup v_n$ compared to $G_1^{\mathrm{in}} \cup v_n$, and then formula (29) holds ture.

For each inference task τ_k represented by G_k, the completion time of its last subtask in G_2^{in} satisfies formula (29). Then we have

$$C_{\mathrm{avg}}(G_2 \cup v_n) - C_{\mathrm{avg}}(G_2) \geq C_{\mathrm{avg}}(G_1 \cup v_n) - C_{\mathrm{avg}}(G_1) \tag{30}$$

Since Problem P1(2) is a minimization problem, its objective function satisfies the submodular property: the marginal increase in ACT does not decrease when adding a new subtask v_n to a larger subset [51].

C. Proof of Lemma 3

From equation (23), for $t \geq \left\lceil \log_{1-\eta}^{C_j} \right\rceil$ with $C_j = \frac{p_{\mathrm{L}} - p_j^{\mathrm{opt}}}{p_{ij}(0) - p_j^{\mathrm{opt}} + \mu\eta^{-1}(2^{-1} - \varepsilon(0))}$, we derive that

$$p_{ij}(t) \leq p_{\mathrm{L}} - \mu\eta^{-1}(2^{-1} - \varepsilon(t)). \tag{31}$$

Since $\varepsilon(t) \overset{a.s.}{\to} 2^{-1}$ as $t \to \infty$, it follows that $p_{ij}(t) \overset{a.s.}{\leq} p_{\mathrm{L}}$, as $t \to \infty$. According to Algorithm 5 (Line 10), if the updated firefly position falls below p_{L}, it is reset to p_{L}. Therefore, $p_{ij}(t) = p_{\mathrm{L}}$ with probability 1, completing the proof.

D. Proof of Lemma 4

Symmetrically, for $t \geq \left\lceil \log_{1-\eta}^{C_j} \right\rceil$, where $C_j = \frac{p_j^{\mathrm{opt}} - p_{\mathrm{U}}}{p_j^{\mathrm{opt}} - p_{ij}(0) - \mu\eta^{-1}(2^{-1} - \varepsilon(0))}$, we have

$$p_{ij}(t) \geq p_{\mathrm{U}} - \mu\eta^{-1}(2^{-1} - \varepsilon(t)). \tag{32}$$

Since $\varepsilon(t) \overset{a.s.}{\to} 2^{-1}$ as $t \to \infty$, we derive $p_{ij}(t) \overset{a.s.}{\geq} p_{\mathrm{U}}$, as $t \to \infty$. Moreover, if an updated position exceeds p_{U}, it is reset to p_{U}. Thus, $p_{ij}(t) = p_{\mathrm{U}}$ with probability 1, completing the proof.

E. Proof of Lemma 5

We prove this by contradiction. Suppose that Algorithm 5 gets trapped at a boundary when $p_j^{\mathrm{opt}} \in [p_{\mathrm{L}}, p_{\mathrm{U}}]$ and $\eta \in (0, 1)$. This implies that the firefly remains at either p_{L} or p_{U} indefinitely. That is, there exists an integer t' such that for all $t \geq t'$, $p_{ij}(t)$ remains at the boundary. We analyze the following two cases:

- The firefly is trapped at p_{L}. Since $p_{ij}(t) \leq p_{\mathrm{L}}$ for $t \geq t'$, applying Eq.(23) and the fact that $\varepsilon(t) \overset{a.s.}{\to} 2^{-1}$ as $t \to \infty$, we obtain

$$\begin{aligned}(1-\eta)^t\left(p_{ij}(0) + \mu\eta^{-1}(2^{-1} - \varepsilon(0)) - p_j^{\mathrm{opt}}\right) &\overset{a.s.}{\leq} p_{\mathrm{L}} - p_j^{\mathrm{opt}} \\ &\leq 0.\end{aligned} \tag{33}$$

  - If $p_{ij}(0) + \mu\eta^{-1}(2^{-1} - \varepsilon(0)) - p_j^{\mathrm{opt}} > 0$, then the right-hand side (RHS) of inequality (33) must be positive, contradicting the assumption that it is non-positive.
  - If $p_{ij}(0) + \mu\eta^{-1}(2^{-1} - \varepsilon(0)) - p_j^{\mathrm{opt}} < 0$, we can derive that $t \leq \left\lfloor \log_{1-\eta}^{C_j} \right\rfloor$, where $C_j = \frac{p_j^{\mathrm{opt}} - p_{\mathrm{L}}}{p_j^{\mathrm{opt}} - p_{ij}(0) - \mu\eta^{-1}(2^{-1} - \varepsilon(0))}$. This contradicts the assumption that $t \in (t', \infty)$, proving that $p_{ij}(t)$ cannot stay at p_{L} indefinitely.
- The firefly is trapped at p_{U}. Similarly, assuming $p_{ij}(t) \geq p_{\mathrm{U}}$ for $t \geq t'$. We obtain

$$\begin{aligned}(1-\eta)^t\left(p_{ij}(0) + \mu\eta^{-1}(2^{-1} - \varepsilon(0)) - p_j^{\mathrm{opt}}\right) &\overset{a.s.}{\geq} p_{\mathrm{U}} - p_j^{\mathrm{opt}} \\ &\geq 0.\end{aligned} \tag{34}$$

  - If $p_{ij}(0) + \mu\eta^{-1}(2^{-1} - \varepsilon(0)) - p_j^{\mathrm{opt}} > 0$, we derive that $t \leq \left\lfloor \log_{1-\eta}^{C_j} \right\rfloor$, where $C_j = \frac{p_{\mathrm{U}} - p_j^{\mathrm{opt}}}{p_{ij}(0) + \mu\eta^{-1}(2^{-1} - \varepsilon(0)) - p_j^{\mathrm{opt}}}$. This contradicts the assumption that $t \in (t', \infty)$, proving that $p_{ij}(t)$ cannot remain at p_{U} indefinitely.
  - If $p_{ij}(0) + \mu\eta^{-1}(2^{-1} - \varepsilon(0)) - p_j^{\mathrm{opt}} < 0$, then the RHS of the inequality should be negative, contradicting the fact that it must be non-negative.

Since both cases lead to contradictions, our assumption that the firefly remains trapped at the boundary is false. Thus, Algorithm 5 avoids boundary traps with probability 1.

REFERENCES

[1] Alibaba. (2019) Alibaba trace, available https://github.com/ alibaba/clusterdata.

[2] B. Xu, Z. Kuang, J. Gao, L. Zhao, and C. Wu, "Joint offloading decision and trajectory design for uav-enabled edge computing with task dependency," *IEEE Transactions on Wireless Communications*, 2022.

[3] S. Liu, Y. Yu, X. Lian, and et al., "Dependent task scheduling and offloading for minimizing deadline violation ratio in mobile edge computing networks," *IEEE Jour. on Selec. Areas in Commun.*, 2023.

[4] J. Liu, J. Ren, Y. Zhang, and et al., "Efficient dependent task offloading for multiple applications in mec-cloud system," *IEEE Trans. on Mobi. Comput.*, 2023.

[5] L. Chen, J. Wu, J. Zhang, H.-N. Dai, X. Long, and M. Yao, "Dependency-aware computation offloading for mobile edge computing with edge-cloud cooperation," *IEEE Transactions on Cloud Computing*, vol. 10, no. 4, pp. 2451–2468, 2020.

[6] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 37–45.

[7] X. Dai, Z. Xiao, H. Jiang, and et al., "Offloading dependent tasks in edge computing with unknown system-side information," *IEEE Trans. on Serv. Comput.*, 2023.

[8] X. Zhou, S. Ge, P. Liu, and T. Qiu, "Dag-based dependent tasks offloading in mec-enabled iot with soft cooperation," *IEEE Trans. on Mobile Computing*, 2023.

[9] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 242–253, 2020.

[10] Q. Zhang, Y. Yang, C. Yi, S. D. Okegbile, and J. Cai, "Energy and cost-aware offloading of dependent tasks with edge-cloud collaboration for human digital twin," *IEEE Int. of Th. Journ.*, 2024.

[11] H. Wang, W. Li, J. Sun, and et al., "Low-complexity and efficient dependent subtask offloading strategy in iot integrated with multi-access edge computing," *IEEE Trans. on Net. and Serv. Manag.*, 2023.

[12] M. Guo, X. Hu, Y. Chen, and et al., "Joint scheduling and offloading schemes for multiple interdependent computation tasks in mobile edge computing," *IEEE Int. of Th. Journ.*, vol. 11, no. 4, pp. 5718–5730, 2023.

[13] J. Lou, Z. Tang, S. Zhang, and et al., "Cost-effective scheduling for dependent tasks with tight deadline constraints in mobile edge computing," *IEEE Trans. on Mobi. Comput.*, vol. 22, no. 10, pp. 5829–5845, 2022.

[14] H. Liu, W. Huang, D. I. Kim, and et al., "Towards efficient task offloading with dependency guarantees in vehicular edge networks through distributed deep reinforcement learning," *IEEE Trans. on Veh. Technol.*, 2024.

[15] K. Cai, Q. Wu, M. Zhou, and et al., "Dynamically scheduling deadline-constrained interleaved workflows on heterogeneous computing systems," *IEEE Trans. on Serv. Comput.*, 2025.

[16] J. Pan and Y. Wei, "A deep reinforcement learning-based scheduling framework for real-time workflows in the cloud environment," *Exp. Sys. with App.*, vol. 255, p. 124845, 2024.

[17] R. Xie, D. Gu, Q. Tang, and et al., "Workflow scheduling in serverless edge computing for the industrial internet of things: A learning approach," *IEEE Trans. on Ind. Inf.*, vol. 19, no. 7, pp. 8242–8252, 2022.

[18] L. Lin, L. Pan, and S. Liu, "Spotdag: An rl-based algorithm for dag workflow scheduling in heterogeneous cloud environments," *IEEE Trans. on Ser. Comput.*, 2024.

[19] S. Qin, D. Pi, Z. Shao, and et al., "Reliability-aware multi-objective memetic algorithm for workflow scheduling problem in multi-cloud system," *IEEE Trans. on Par. & Distrib. Syst.*, 2023.

[20] Y. Geng, Y. Yang, and G. Cao, "Energy-efficient computation offloading for multicore-based mobile devices," in *IEEE INFOCOM*. IEEE, 2018.

[21] Y. Dai et al., "Joint computation offloading and user association in multi-task mobile edge computing," *IEEE Trans. Veh. Technol.*, 2018.

[22] W. He, L. Gao, and J. Luo, "A multi-layer offloading framework for dependency-aware tasks in mec," in *ICC*. IEEE, 2021.

[23] J. Yan, S. Bi, and Y. J. A. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. on Wirel. Commun.*, 2020.

[24] T. Shi, T. Zhang, R. Zhong, and et al., "Cross-user dependent task offloading and resource allocation in spatial-temporal dynamic mec networks," *IEEE Trans. on Veh. Tech.*, 2024.

[25] J. Zhang, G. Zhang, X. Bao, and et al., "Dependent application offloading in edge computing," *IEEE Trans. on Cloud Comput.*, 2023.

[26] L. Yang, B. Liu, J. Cao, Y. Sahni, and Z. Wang, "Joint computation partitioning and resource allocation for latency sensitive applications in mobile edge clouds," *IEEE Transactions on Services Computing*, vol. 14, no. 5, pp. 1439–1452, 2021.

[27] N. Khaledian, M. Voelp, S. Azizi, and et al., "Ai-based & heuristic workflow scheduling in cloud and fog computing: a systematic review," *Cluster Comput.*, 2024.

[28] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. on Par. & Distrib. Syst.*, 2002.

[29] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Trans. on Par. & Distrib. Syst.*

[30] H. Faragardi, M. Sedghpour, S. Fazliahmadi, and et al., "Grp-heft: A budget-constrained resource provisioning scheme for workflow scheduling in iaas clouds," *IEEE Trans. on Par. & Distrib. Syst.*, 2020.

[31] M. Fan, X. Zhao, X. Zuo, and et al., "A budget-constrained workflow scheduling approach with priority adjustment and critical task optimizing in clouds," *IEEE Trans. on Auto. Sci. & Eng.*, 2024.

[32] X. Ma, H. Xu, H. Gao, and et al., "Real-time multiple-workflow scheduling in cloud environments," *IEEE Trans. Netw. Serv. Manag.*, vol. 18, no. 4, pp. 4002–4018, 2021.

[33] J. Zhang and Z. Han, "Optimization of uncertain dependent task mapping on heterogeneous computing platforms," *The Jour. of Supercomp.*, vol. 80, no. 11, pp. 15 868–15 893, 2024.

[34] G. Zhou, W. Tian, R. Buyya, and et al., "Deep reinforcement learning-based methods for resource scheduling in cloud computing: A review and future directions," *Artif. Intell. Rev.*, vol. 57, no. 5, p. 124, 2024.

[35] G. Chen, J. Qi, Y. Sun, and et al., "A collaborative scheduling method for cloud computing heterogeneous workflows based on deep reinforcement learning," *Fut. Gen. Comput. Sys.*, vol. 141, pp. 284–297, 2023.

[36] A. Jayanetti, S. Halgamuge, and R. Buyya, "Deep reinforcement learning for energy and time optimized scheduling of precedence-constrained tasks in edge–cloud computing environments," *Fut. Gen. Comput. Sys.*, vol. 137, pp. 14–30, 2022.

[37] T. Li, Y. Meng, and L. Tang, "Scheduling of continuous annealing with a multi-objective differential evolution algorithm based on deep reinforcement learning," *IEEE Trans. on Auto. Sci. & Eng.*, vol. 21, no. 2, pp. 1767–1780, 2023.

[38] P. V. Reddy and K. G. Reddy, "An energy efficient rl based workflow scheduling in cloud computing," *Exp. Sys. with App.*, vol. 234, p. 121038, 2023.

[39] L. Yin, J. Sun, J. Zhou, Z. Gu, and K. Li, "Ecfa: an efficient convergent firefly algorithm for solving task scheduling problems in cloud-edge computing," *IEEE Trans. on Serv. Comput.*, 2023.

[40] M. Qasim and M. Sajid, "An efficient iot task scheduling algorithm in cloud environment using modified firefly algorithm," *International Journal of Information Technology*, pp. 1–10, 2024.

[41] M. Xu, Y. Mei, S. Zhu, and et al., "Genetic programming for dynamic workflow scheduling in fog computing," *IEEE Trans. on Ser. Comput.*, vol. 16, no. 4, pp. 2657–2671, 2023.

[42] Y. Yang, G. Chen, H. Ma, S. Hartmann, and M. Zhang, "Dual-tree genetic programming with adaptive mutation for dynamic workflow scheduling in cloud computing," *IEEE Trans. on Evol. Comput.*, 2024.

[43] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive user-managed service placement for mobile edge computing: An online learning approach," in *IEEE INFOCOM 2019-IEEE conference on computer communications*. IEEE, 2019, pp. 1468–1476.

[44] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein, "Scheduling to minimize average completion time: Offline and online approximation algorithms," *Mathematics of operations research*, vol. 22, no. 3, pp. 513–544, 1997.

[45] A. Krause and D. Golovin, "Submodular function maximization." *Tractability*, vol. 3, no. 71-104, p. 3, 2014.

[46] R. L. Graham, "Bounds for certain multiprocessing anomalies," *Bell system technical journal*, vol. 45, no. 9, pp. 1563–1581, 1966.

[47] F. Van den Bergh and A. P. Engelbrecht, "A study of particle swarm optimization particle trajectories," *Information sciences*, vol. 176, no. 8, pp. 937–971, 2006.

[48] M. K. Marichelvam, T. Prabaharan, and X. S. Yang, "A discrete firefly algorithm for the multi-objective hybrid flowshop scheduling problems," *IEEE transactions on evolutionary computation*, vol. 18, no. 2, pp. 301–305, 2013.

[49] X. Zuo, G. Zhang, and W. Tan, "Self-adaptive learning pso-based deadline constrained task scheduling for hybrid iaas cloud," *IEEE Trans. on Auto. Sci. & Eng.*, vol. 11, no. 2, pp. 564–573, 2013.

[50] J. Kwak, Y. Kim, J. Lee, and S. Chong, "Dream: Dynamic resource and task allocation for energy minimization in mobile cloud systems," *IEEE Journal on Selec. Are. in Commun.*, vol. 33, no. 12, pp. 2510–2523, 2015.

[51] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functionsi," *Mathematical programming*, vol. 14, pp. 265–294, 1978.