Scheduling Generative-AI DAGs with Model Serving in Data Centers

Ying Zheng[1], Lei Jiao[2], Yuedong Xu[1], Bo An[3], Xin Wang[1], Zongpeng Li[4]
[1]Fudan University, China [2]University of Oregon, USA
[3]Nanyang Technological University, Singapore [4]Tsinghua University, China

***Abstract*—Scheduling generative-AI jobs in the edge computing environment faces multiple non-trivial challenges, including the Directed Acyclic Graph (DAG) dependency among tasks, the intrinsic intertwinement between task scheduling and model selection, and the dynamic unpredictable arrival of job DAGs. In this work, we capture all such challenges and formulate a non-linear integer program to optimize the long-term profit of the generative-AI service provider, *i.e.,* service revenue of the admitted jobs minus system costs of executing the tasks contained in such job DAGs. This problem is NP-hard even in the offline setting. To solve it, we first reformulate it into an equivalent schedule selection problem using generated schedules to tackle complex constraints. Then, we design a new online scheduling method through the online primal-dual technique. Experimental results confirm that our approach can increase the total service profit by up to 41.2% compared to existing algorithms.**

I. Introduction

Generative AI, or AI-Generated Content (AIGC), has been increasingly popular in text [1], [2], image [3], [4], and other applications [5]–[7]. A complex AIGC *job* can typically be represented as a Directed Acyclic Graph (DAG) of interdependent basic *tasks*, each of which uses a pre-trained generative-AI model to conduct inference. For instance, Hugging Face [8] provides more than 40 types of AIGC tasks with 260,000+ models. As shown in Fig. 1, a user can submit an AIGC job for generating a new image based on the pose in Image 1 and the caption in Image 2. Here, the first task selects an appropriate model from the candidate models to "extract" the pose from Image 1, and the second task selects a model to generate a text description from Image 2. Then, with the extracted pose and text, the third task ultimately creates a new image.

An AIGC *service* can receive many AIGC jobs from users at diverse geographic locations and execute such jobs upon distributed *edge clouds*, particularly with powerful processors (e.g., NVIDIA Triton Inference Server [9]) increasingly equipped at the network edge. This provides multiple benefits, including the closer proximity to the users with low service latency, better traffic and data localization, and more efficient model and resource sharing and usage across AIGC jobs.

Yet, to realize these benefits in practice, the service provider confronts fundamental and unique challenges for scheduling and executing AIGC jobs. First, edge clouds often have limited resources, and as users specify their jobs' deadlines, the service may be only capable of executing and finishing some of the submitted jobs selectively before the deadlines, which results in an admission control issue on top of resource allocation. Second, the DAG structure complicates the

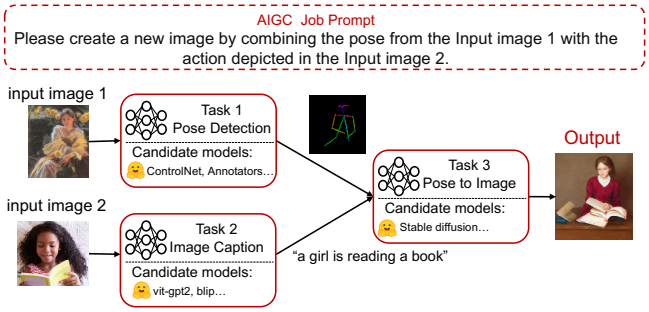


Fig. 1: An example AIGC job

scheduling of the AIGC tasks in each job, as the scheduling of each task needs to follow the dependency specified in the DAG. This impacts not only the order but also the location of executing the tasks upon the edge clouds. Third, unlike traditional jobs, each task in the AIGC job can be executed via multiple generative-AI models, making task scheduling intrinsically intertwined with model selection. For example, for a text-to-image task, both DALLE [3] and stable diffusion [10] models can be used to execute this task. Choosing different models results in varying execution times and costs, where shorter execution time does not necessarily incur lower cost due to the complexity and diversity of today's neural network model structures. Fourth, AIGC jobs arrive dynamically and unpredictably, but the schedule decision of each job needs to be made immediately and irrevocably in an online manner.

Existing literature fails to address the aforementioned challenges. Regarding AIGC task scheduling [11], [12], they fail to consider complex AIGC requests with DAG structure, and hence are not applicable to our problem. Regarding job DAG scheduling, they either focus on a single DAG [13]–[15], tackle offline static settings [16]–[18], ignore communications between components [19], or optimize the makespan or cost on a single critical path [19], [20].

In this work, we model and formulate a non-linear integer program to optimize the AIGC service's long-term profit, *i.e.*, the revenue collected from the admitted AIGC jobs minus task computation cost, inter-task communication cost, and model hosting cost over time via controlling job admission, task scheduling, and model serving under DAG dependencies and resource and deadline restrictions. Unsurprisingly, this problem is NP-hard even in the offline setting. To solve it, we reformulate it into a simpler yet equivalent *schedule selection* problem, and then design an efficient online algorithm based

on primal-dual analysis. The key idea is, rather than making those multiple types of control decisions dynamically, we generate a series of static schedules containing different concrete decisions spanning future time slots as each job arrives and then just seek the best schedule for the job. Through extensive trace-driven experiments, we demonstrate that our proposed approach outperforms multiple other methods with better scalability and robustness, able to balance different cost components and achieve superior empirical competitiveness in various settings practically.

II. Modeling and Formulation

A. Edge-Cloud System

We consider an AIGC service provider which owns and operates the AIGC service upon a distributed edge infrastructure. This system consists of a set $[K] = \{1, 2, \ldots, K\}$ of distributed edge clouds, or "edges", where each edge is a server cluster or a small-scale datacenter at a specified location (*e.g.*, neighborhood, regional center) in close proximity to the AIGC service's users, as in Fig. 2. These edges are connected to one another via wired networks, and are connected to the users via wired or wireless networks. Without loss of generality, the entire system operates in slotted time $[T] = \{1, 2, \ldots, T\}$.

B. AIGC Jobs with Deadlines

Users submit AIGC jobs to the AIGC service for execution. We use $[I] = \{1, 2, \ldots, I\}$ to refer to all AIGC jobs. A job $i \in [I]$ is represented as $\{t_i, d_i, b_i, \mathcal{D}_i\}$, where t_i is the job's arrival time; d_i is its deadline, *i.e.*, the time before which this job needs to be finished; $\mathcal{D}_i$ is the job's Directed Acyclic Graph (DAG) of the tasks, which will be elaborated next; and b_i is the expense that the AIGC service charges from the user who submits this job if this job is admitted for execution.

C. AIGC Tasks and Models

Each AIGC job i contains one DAG $\mathcal{D}_i = \{\mathcal{F}_i, \mathcal{E}_i, \mathcal{M}_i\}$. Specifically, $\mathcal{F}_i$ is the set of the nodes and $\mathcal{E}_i$ is the set of the directed edges, where each node $j \in \mathcal{F}_i$ is an individual AIGC task and the directed edges specify the data flow between tasks and correspondingly the execution order of the tasks. Note that we can always add a dummy entry task and a dummy exit task with zero execution time to the DAG to make the DAG have only one entry node and only one exit node, following techniques from the literature [13], [18], as shown in Fig. 2. More than often, for a task, there could exist multiple models, any one of which can be used for executing the task. Therefore, we use $\mathcal{M}_i$ to refer to the candidate pool of the generative-AI models that are associated to the tasks in $\mathcal{D}_i$. Note that these can be just the indices or IDs of the models, as the actual models themselves are hosted in the AIGC service's edge clouds. We can write $\mathcal{M}_i = \{M_{ij}\}_{j \in \mathcal{F}_i}$, where M_{ij} is the candidate pool of the generative-AI models for executing task j of the job i. We also note that each model is often bounded with its own pre-specified configurations, *e.g.*, the number of inference steps for the stable diffusion model. In this paper, we target the situation where every single edge cloud has a full copy of all the models stored on its storage.

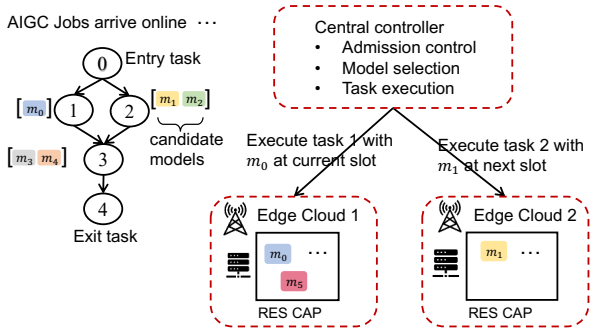


Fig. 2: The framework of AIGC job scheduling

D. Control Decisions

As a job i arrives at the system dynamically, the service collects all the information about this job in real time and makes the following control decisions in an online manner: (i) whether or not to admit the job i, denoted by $u_i \in \{1, 0\}$; (ii) whether or not to execute the task j of this job i on the edge k at the time slot t, represented by $x_{ijkt} \in \{1, 0\}$, $\forall t \geq t_i$; and (iii) whether or not to use the generative-AI model m for executing the task j of this job i, denoted by $y_{ijm} \in \{1, 0\}$.

E. Service Revenue and Costs

For the AIGC service, we consider the *revenue* received for executing the AIGC jobs, and the *costs* incurred from executing these jobs. Note that costs are not necessarily monetary, and can include performance overhead of the system.

1) Revenue: The total revenue earned for the service is

$$Rev = \sum_i u_i b_i. \tag{1}$$

2) Task Computation Cost: Task computation cost refers to the system overhead for executing the AIGC models' inference process, which mainly depends on the computational workload correspondingly. Let $r^f_{m,i,j}$ be the amount of computation required by executing the model m for job i's jth task per time slot, and e^f_k be the overhead per unit computation on the edge k. The task computation cost is

$$Cost^p = \sum_i \sum_j \sum_m \sum_k \sum_t x_{ijkt} y_{ijm} r^f_{m,i,j} e^f_k. \tag{2}$$

3) Inter-Task Communication Cost: Inter-task communication cost appears if any two tasks are assigned to different edge clouds for execution, where these two tasks are connected by a directed edge within an AIGC job DAG and thus the output of the predecessor task has to be transmitted as the input to the successor task. Denote a_{ij} as the edge that executes the job i's task j. If the job is admitted for execution by the AIGC service, we can have $a_{ij} = \max_{t_i \leq t \leq d_i + \tau_i} \{x_{ijkt} k\}$. This is because, although different tasks can be executed on different edge clouds, we ensure by our constraints, as shown later, that a task is always executed at consecutive time slots until it finishes, associating with the same edge cloud for execution during this process. For tasks j and j' where $(j, j') \in \mathcal{E}_i$, $\mathbb{I}(a_{ij} \neq a_{ij'})$ is an indicator function representing whether j and j' are executed on the same edge (where j' executes after j). Let $h_{jj'}$ be the communication data size between the tasks

j and j', and e_k^p be the cost of communicating unit data on the edge k. The inter-task communication cost is

$$Cost^c = \sum_i \sum_{(j,j') \in \mathcal{E}_i} \mathbb{I}(a_{ij} \neq a_{ij'}) h_{jj'} e_k^p. \quad (3)$$

4) Model Hosting Cost: Model hosting cost is the system overhead of maintaining the models in the processor memory of the edge servers, *e.g.*, memory consumption, and electricity consumption. Because different tasks, either belonging to the same or different jobs, could need the same model for execution, there could exist multiple *instances* of the same model in the memory, each model instance for a single task. Let r_m^s be the size of the model m. Denote e_k^h as the hosting cost per unit-size of any model per time slot on the edge k. The model hosting cost is

$$Cost^m = \sum_i \sum_j \sum_m \sum_k \sum_t x_{ijkt} y_{ijm} r_m^s e_k^h. \quad (4)$$

F. Problem Formulation

Our goal is to maximize the *profit* of the service provider, *i.e.*, the total revenue minus the total costs. The problem formulation is as follows:

$$P: \ \max \ Rev - Cost^p - Cost^c - Cost^m \quad (5)$$
$$\text{s.t. } (1) \sim (4), \quad (5a)$$
$$u_i \leq \sum_m y_{ijm} \leq 1, \ \forall i, j, \quad (5b)$$
$$\sum_k x_{ijkt} \leq u_i, \ \forall i, j, t, \quad (5c)$$
$$x_{ijkt} t \leq d_i, \ \forall i, j, k, t \geq t_i, \quad (5d)$$
$$\max_{t \geq t_i} \{\sum_k x_{ijkt} t\} + 1 \leq \min_{t \geq t_i} \{\sum_k x_{ij'kt} t\}, \quad \forall i, (j, j') \in \mathcal{E}_i, \quad (5e)$$
$$\sum_k \sum_{t'} \mathbb{I}(\sum_{t=t'}^{t'+f_{m,i,j}} x_{ijkt} \geq f_{m,i,j}) \geq y_{ijm}, \quad \forall i, j, m, \quad (5f)$$
$$\sum_i \sum_j x_{ijkt} y_{ijm} r_m^s \leq c_k^s, \ \forall k, t, \quad (5g)$$
$$u_i, x_{ijkt}, y_{ijm} \in \{0, 1\}, \forall i, j, k, m, t. \quad (5h)$$

Constraint (5b) ensures that if a job is accepted, then one and only one model can be selected for each task of the job. Constraint (5c) ensures that only one edge can be used for executing each task of the admitted job at each time slot. Constraint (5d) ensures that each task can only be executed no later than the job's deadline. Constraint (5e) captures the dependency relationship of task execution. Constraint (5f) ensures enough consecutive time slots for the task j if it is scheduled to execute with the model m. Constraint (5g) enforces resource capacity on each edge at each time slot. Constraint (5h) specifies the domains of the control variables.

III. Online Algorithm Design

A. Problem Reformulation

We reformulate the problem P into an equivalent problem P_1 of *schedule selection*, in order to "absorb" the complex constraints and "simplify" the problem. We define a *schedule* of the job i as an assignment of a set of concrete values to the decision variables u_i, x_{ijkt}, and y_{ijm}, while satisfying Constraints (5a)$\sim$(5f) of the problem P. That is, a schedule of the job i is a concrete determination of whether to admit job i, at what time slots on which edge to execute each task of the job i, and what model to use for each task of the job i. Thus, a schedule is indexed by $l = \{(i, j, m, k, t) | i \in [I], j \in \mathcal{F}_i, m \in M_{ij}, k \in [K], t \in [T]\}$. Based on the concept of schedule, the problem P can be reformulated:

$$P_1: \ \max \ \sum_i \sum_{l \in \zeta_i} x_{il} b_{il} \quad (6)$$
$$\text{s.t. } \sum_{l \in \zeta_i} x_{il} \leq 1, \ \forall i, \quad (6a)$$
$$\sum_i \sum_j \sum_m \sum_{l \in \zeta_i : (i,j,m,k,t) \in l} x_{il} r_m^s \leq c_k^s, \ \forall k, t, \quad (6b)$$
$$x_{il} \in \{0, 1\}, \forall i, l \in \zeta_i, \quad (6c)$$

where the binary decision variable x_{il} indicates whether the job i is scheduled to execute following the schedule l. We use ζ_i to denote the set of all the feasible schedules for the job i, each of which satisfies Constraints (5a)$\sim$(5f). We calculate $b_{il} = Rev_i - Cost_i^p - Cost_i^c - Cost_i^m = u_i b_i - \sum_j \sum_m \sum_k \sum_t x_{ijkt} y_{ijm} r_{m,i,j}^f e_k^f - \sum_{(j,j') \in \mathcal{E}_i} \mathbb{I}(a_{ij} \neq a_{ij'}) h_{jj'} e_k^p - \sum_j \sum_m \sum_k \sum_t x_{ijkt} y_{ijm} r_m^s e_k^h$. Constraint (6a) ensures that we choose up to one schedule for each job. Constraint (6b) is equivalent to (5g). As a result, there is a one-to-one mapping between the feasible solutions for the problem P and those for the problem P_1.

B. Primal-Dual Algorithm

We design a primal-dual algorithm to solve the problem P_1 in an online manner. To that end, we relax x_{il} as $x_{il} \geq 0$, and derive the Language dual problem D_1 of the relaxed problem P_1:

$$D_1: \ \min \ \sum_i \mu_i + \sum_t \sum_k c_k^s \lambda_{kt} \quad (7)$$
$$\text{s.t. } \mu_i \geq b_{il} - \sum_k \sum_t \sum_j \sum_{m:(i,j,m,k,t) \in l} r_m^s \lambda_{kt}, \forall i, l \in \zeta_i, \quad (7a)$$
$$\mu_i \geq 0, \lambda_{kt} \geq 0, \forall i, k, t, \quad (7b)$$

where μ_i and λ_{kt} are the Language dual variables associated with Constraints (6a) and (6b), respectively.

The key idea of our online algorithm is as follows. We maintain a feasible solution for the dual problem D_1 by a carefully designed update rule of dual variables. Specifically, for each job i, we try to find an appropriate schedule denoted as l_i which maximizes the right-hand side of Constraint (7a) out of the set of all the feasible schedules, and set μ_i as

$$\mu_i \leftarrow \max_{l \in \zeta_i} \{b_{il} - \sum_k \sum_t \sum_j \sum_{m:(i,j,m,k,t) \in l} r_m^s \lambda_{kt}\}. \quad (8)$$

If $\mu_i > 0$, we admit the job and execute it according to the schedule l_i returned by (8); otherwise we deny the admission for the job i and set $\mu_i = 0$. After processing the job i with the schedule l, we update λ_{kt} as

$$\lambda_{kt} \leftarrow \lambda_{kt} (1 + \frac{r_{kt}(il)}{c_k^s}) + \alpha (\frac{\bar{b}_{il} r_{kt}(il)}{c_k^s}), \quad (9)$$

where $\bar{b}_{il} = \frac{b_{il}}{\sum_k \sum_t r_{kt}(il)}$; $\alpha = \max_i \{\frac{b_i}{r_{i,min}}\}$; $r_{i,min}$ is the minimum resource requirement of a task within the job i; and $r_{kt}(il)$ is the resource requirement for the edge k at the time

Algorithm 1: Online Job Scheduling Algorithm

Input: $\{(t_i, d_i, b_i, \mathcal{D}_i)\}, c_k^s, e_k^f, e_k^p, e_k^h$

1 Initialize $x_{il} = 0, \mu_i = 0, \lambda_{kt} = 0, \forall m, k, t, i, l$;
2 **for** *job* i **do**
3 Invoke **Algorithm 2** to produce μ_i and the schedule l_i;
4 **if** $\mu_i > 0$ **then**
5 Update λ_{kt} via (9);
6 **if** $\sum_{i'=1}^{i} \sum_l x_{i'l} r_{kt}(i'l) \le c_k^s, \forall k, t$ **then**
  // check resource sufficiency
7 Admit the job i and execute it using l_i;
8 **else** Decline the job i;
9 **else** Decline the job i;

slot t when executing the job i using the schedule l. Intuitively, $\bar{b}_{il}$ can be understood as the profit incurred per unit resource. λ_{kt} can be interpreted as a marginal price function of resources, and has the following properties: (i) it is initialized to zero and then increases as the resource consumption increases; (ii) if the schedule l_i causes the cumulative usage of resources to exceed the capacity (*i.e.*, to violate Constraint (6b)), then no more tasks will be scheduled on the edge k at the time slot t; and (iii) it is carefully designed so that we always maintain a dual feasible solution, equipping our online approach with a provably-good performance guarantee. The overall online AIGC job scheduling algorithm is Algorithm 1.

Algorithm 1 works as follows. Upon the arrival of a job i, Line 3 obtains μ_i and the schedule l_i via dynamic programming according to Equation (8), which will be elaborated next. Lines 4∼5 indicate that if $\mu_i > 0$, then we update the dual variable λ_{kt}. Lines 6∼7 indicate if there are sufficient resources to execute the job i using the schedule l_i, we admit this job and execute it following l_i (*i.e.*, $x_{il_i} = 1$, and $x_{il} = 0$, $\forall l \neq l_i$); otherwise, we deny admission for this job (*i.e.*, $x_{il} = 0$, $\forall l$). Note that since λ_{kt} has been updated, no more tasks can be scheduled on the edge k in the future.

Dynamic Programming: Algorithm 2 is invoked by Algorithm 1, and works as follows. This algorithm is the key to handling each job DAG. As shown in Line 2, we first determine an execution order of the tasks in the job DAG i. We do so based on the "rank" of the job i's task j, $\forall j$, calculated as

$$rank(ij) = \frac{1}{|M_{ij}|} \sum_{m \in M_{ij}} f_{m,i,j} + \max_{j':(j,j') \in \mathcal{E}_i} \{rank(ij')\}, \tag{10}$$

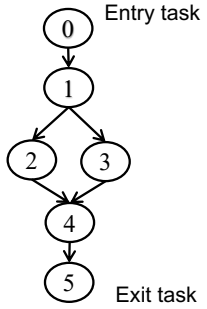
where M_{ij} is the set of candidate models for executing the job i's task j, and $f_{m,i,j}$ is the runtime of using model m to execute job i's jth task. We use (j, j') to represent an edge from the task j to the task j'. The execution order for the tasks in job i is then generated as the descending order of the ranks of the tasks. This ensures that when executing a task j, all its predecessor tasks have been completed. Line 3 in Algorithm 2 generates the set of "schedule options" H for each task that has multiple children tasks. This is because, to obtain the optimal scheduling strategy using dynamic programming, a parent task that has multiple children tasks is difficult to handle in the sense that the optimal schedule of different children tasks may

Algorithm 2: Per-Job Schedule Selection Algorithm

Input: $(t_i, d_i, b_i, \mathcal{D}_i), c_k^s, \gamma_{kt}, \lambda_{kt}, \{\widetilde{\delta}_{mkt}\}$
Output: μ_i, l_i

1 Initialize
  $F_{jkmt} = inf, V_{jkmt} = \emptyset, P_{jkmt} = -inf, \forall j, k, m, t$;
2 Generate the execution order S for the tasks as the descending order of the ranks defined in (10);
3 Generate schedule options H for each task with multiple child tasks;
4 **for** *option* $h \in H$ **do**
5 **for** *task* $j \in S$ **do**
6 **for** *edge* $k \in h_j^K$ **do**
7 **for** *model* $m \in h_j^M$ **do**
8 **for** *time* $t \in [t_i, d_i - f(m, i, j)]$ **do**
9 $F_{jkmt} \leftarrow \sum_{j':(j',j) \in \mathcal{E}_i} \min_{k',m'} \{F_{j'k'm'(t-f_{m,i,j})} \mathbb{I}(j == R_{j'}) + Cost_{ij}^p + Cost_{ij}^c + Cost_{ij}^h\}$;
10 **if** $F_{jkmt} < F_{jkm(t-1)}$ **then**
11 Add decision (i, j, m, k, t) to V_{jkmt} ;
12 **else**
13 $F_{jkmt}, V_{jkmt} \leftarrow F_{jkm(t-1)}, V_{jkm(t-1)}$;
14 **if** $j == exit\_task$ **then**
15 $P_{jkmt} \leftarrow Rev_i - F_{jkmt}$;
16 **return** *schedule* $l_i = V_{jk^*m^*t^*}$ *that achieves* $\max\{P_{jkmt}\}$, *and the maximum value is* μ_i.

be based on the conflicting schedules of the same parent task. To address this issue, we first assign a specific edge cloud and a specific model to the parent task, and collect all such possible assignments in a set H. Then, for each option $h \in H$, we perform the dynamic programming process. The h_j^K in Line 6 indicates a specified edge cloud if task the j has multiple children tasks, otherwise $h_j^K = [K]$. Similarly, h_j^M in Line 7 refers to a specified model if the task j has multiple children tasks, otherwise it just takes $h_j^M = M_{ij}$. For each job i, let F_{jkmt} be the minimum cumulative cost up to executing the task j following the execution order, where the task j starts to execute with the model m no later than t on the edge k. Likewise, denote P_{jkmt} as the profit up to executing the task j with the model m no later than t on the edge k. Since we calculate $Rev_i = b_i$ only at the finishing time slot of the last task of the job i, P_{jkmt} has values only for $j = |\mathcal{F}_i|$. We use V_{jkmt} to record the scheduling decisions for each task that can achieve F_{jkmt}. Line 9 defines the updating rule of F_{jkmt}, where $R_{j'}$ represents the task to which the parent task j's cost is accumulated. Line 10 states that if executing the task j at t results in a smaller cost compared to executing it at $t-1$ or earlier, then we save this schedule for the task j in Line 11; otherwise we just keep the previous result as in Lines 12∼13. Lines 14∼15 indicate that when executing the last task, we start to calculate the revenue, and subtract the cumulative cost to obtain the final profit. Line 16 returns the schedule $l_i = V_{jk^*m^*t^*}$ that achieves $\max\{P_{jk^*m^*t^*}\}$, and the corresponding maximum value is assigned to μ_i. Fig. 3



① is the task with multiple children tasks

Schedule options	
Option 1	① → Edge 0, m_0
Option 2	① → Edge 0, m_1
Option 3	① → Edge 1, m_0
Option 4	① → Edge 1, m_1

Task	Candidate models	Estimated average runtime	Rank	Schedule order
0	--	--	--	--
1	m_0, m_1	$\overline{f_1}=\frac{1}{2}(f(m_0)+f(m_1))=5$	$rank(1)=\overline{f_1}+\max\{rank(2),rank(3)\}=16$	1
2	m_2	$\overline{f_2}=3$	$rank(2)=\overline{f_2}+\overline{f_4}=7$	3
3	m_3, m_4, m_5	$\overline{f_3}=7$	$rank(3)=\overline{f_3}+\overline{f_4}=11$	2
4	m_6, m_7	$\overline{f_4}=4$	$rank(4)=\overline{f_4}=4$	4
5	--	--	--	--

Fig. 3: An example of generating the schedule options and computing the execution order

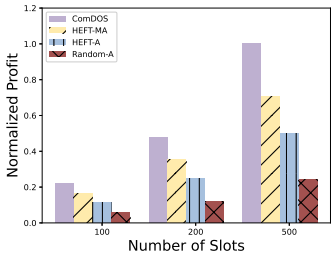


Fig. 4: Impact of number of slots

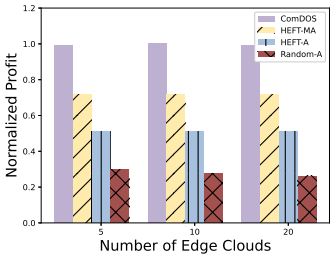


Fig. 5: Impact of number of edges

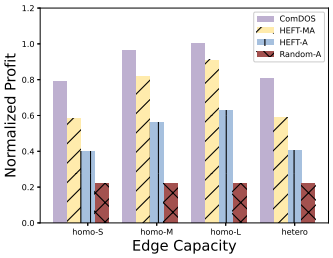


Fig. 6: Impact of edge capacities

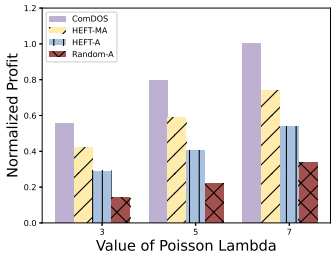


Fig. 7: Impact of workload

exhibits an example of generating the schedule options and computing the execution order when there are two edge clouds in the system.

IV. Experiments

A. Evaluation Settings

AIGC Jobs and System Environments: We generate a DAG dataset based on huggingGPT [21], a system that transforms complex AIGC requests into task DAGs. The revenue for each request is determined in proportion to its overall resource demand multiplied by a reference average completion time. The deadline for each request is generated randomly within the range between its arrival time and the experiment end time [22]. The memory and computation capacity of an edge cloud in different settings is set based on the capacity of a server with 4 NVIDIA GeForce RTX 3060, 4090, and Tesla V100, respectively. For each time slot, the number of requests arriving online follows a Poisson process with an average of $3, 5$, and 7. Each time slot lasts 5 seconds and the total number of time slots in different settings are set as $100, 200$, and 500. Note that although we only measure finite horizon in our experiments, our algorithm can easily be extended to infinite time horizon.

Comparison Algorithms: We compare our proposed approach against (i) HEFT-A, (ii) HEFT-MA, and (iii) Random-A. We call our approach ComDOS (**Com**pact-exponential-based AIGC **D**AG **O**nline **S**cheduling) in the evaluations.

- HEFT-A: HEFT [23], [24] is widely used for job DAG scheduling. It calculates the scheduling priority of each node in the DAG based on the rank, and assigns tasks to edges with the shortest completion time to minimize the overall job completion time. We introduce admission control to HEFT, *i.e.*, when the profit for executing a task is negative, the task is directly declined.
- HEFT-MA: We also extend HEFT-A to HEFT-MA with an additional model selection algorithm, where each task is executed by the model with the shortest execution time.
- Random-A: Random-A calculates the scheduling priority of each node in the DAG based on the rank, while randomly assigning tasks to edges for processing. Similarly, we also apply admission control.

B. Evaluation Results

Scalability: Fig. 4 shows the impact of the total number of time slots or the length of the time horizon on the profit. We observe that ComDOS consistently achieves the highest profit, and improves the profit by 41.2%, 100.78%, and 315.51% in the setting of 500 time slots when compared to HEFT-MA, HEFT-A, and Random-A , respectively.

Fig. 5 varies the number of the edge clouds in the system and investigates how this impacts the profit. ComDOS still outperforms others. Specifically, when compared to the HEFT-MA, ComDOS improves the profit by 38.95%, 39.76%, and 38.94% for 5, 10, and 20 edges, respectively. ComDOS also achieves an improvement of 94.95%, 95.60%, and 94.44% compared to HEFT-A.

Fig. 6 visualizes the profit with different edge capacities. In this experiment, we conduct tests based on 10 edge clouds. In the first three sets of experiments, we test homogeneous edge clouds with the same capacity for each server. The labels

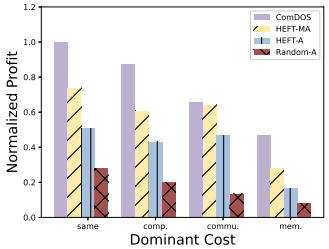


Fig. 8: Impact of cost weights

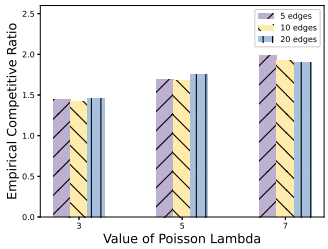


Fig. 9: Competitive ratio

'homo-S', 'homo-M', and 'homo-L' correspond to the capacities of 4 NVIDIA 3060, 4090, and Tesla V100, respectively. The last set of experiments is for heterogeneous edge clouds. ComDOS achieves the best performance in all tests.

Robustness: Fig. 7 illustrates the profit under different job arrival patterns. In the light-workload scenario, i.e., the parameter lambda of Poisson distribution is 3, ComDOS achieves improvements of 32.67%, 92.38%, and 290.28% compared to the HEFT-MA. HEFT-A, Random-A, respectively. While the improvement is 35.44%, 85.87%, and 198.16% in the heavy-workload scenario. ComDOS is robust to the workload variations.

Cost Dissection: Fig. 8 depicts the profit under different cost weights. As we consider multiple types of costs, we can associate a weight to each cost term to control the overall optimization. We set the weight to 0.1 for all costs in the first test. The remaining four tests are each dominated by one type of cost, where the dominant cost weight is set to 0.5 and the others are set to 0.1. ComDOS achieves the best performance across various weight settings.

Competitiveness: Fig. 9 evaluates the empirical competitive ratio. Results demonstrate that ComDOS achieves the empirical competitive ratios of no more than 2 in various settings.

V. Conclusion

Unlike other types of job DAGs, it is particularly difficult to dynamically schedule complex generative-AI job DAGs that arrive online. This paper makes our first step toward a more comprehensive and solid study about this problem. We model this problem, handle the complex constraints and reduce the decision dimensions through a careful and equivalent reformulation, and manage to design an online algorithm rooted in rigorous optimization theory. We also conduct preliminary experiments to validate the superior practical performance of our proposed approach over multiple existing methods. Due to the space limit, there still exist factors that have not been considered, which we intend to postpone to our future work.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China under Grant 62072117, in part by the Shanghai Natural Science Foundation under Grant 22ZR1407000, and in part by the U.S. National Science Foundation under Grants CNS-2047719 and CNS-2225949. The corresponding authors are Lei Jiao (jiao@cs.uoregon.edu) and Yuedong Xu (ydxu@fudan.edu.cn).

References

[1] OpenAI, "GPT-4 Technical Report," https://cdn.openai.com/papers/gpt-4.pdf.
[2] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "LLAMA 2: Open Foundation and Fine-Tuned Chat Models," *arXiv preprint arXiv:2307.09288*, 2023.
[3] G. Daras and A. G. Dimakis, "Discovering the Hidden Vocabulary of DALLE-2," *arXiv preprint arXiv:2206.00169*, 2022.
[4] T. Karras, S. Laine, and T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," in *IEEE/CVF CVPR*, 2019.
[5] Z. Borsos, R. Marinier, D. Vincent, E. Kharitonov, O. Pietquin, M. Sharifi, D. Roblek, O. Teboul, D. Grangier, M. Tagliasacchi *et al.*, "Audiolm: A Language Modeling Approach to Audio Generation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2023.
[6] C. Chen, Y. Hu, W. Weng, and E. S. Chng, "Metric-Oriented Speech Enhancement Using Diffusion Probabilistic Model," in *IEEE ICASSP*, 2023.
[7] Z. Luo, D. Chen, Y. Zhang, Y. Huang, L. Wang, Y. Shen, D. Zhao, J. Zhou, and T. Tan, "VideoFusion: Decomposed Diffusion Models for High-Quality Video Generation," in *IEEE CVPR*, 2023.
[8] Hugging Face, https://huggingface.co/.
[9] NVIDIA Triton Inference Server, https://developer.nvidia.com/nvidia-triton-inference-server.
[10] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-Resolution Image Synthesis with Latent Diffusion Models," in *IEEE CVPR*, 2022.
[11] M. Xu, D. Niyato, H. Zhang, J. Kang, Z. Xiong, S. Mao, and Z. Han, "Joint Foundation Model Caching and Inference of Generative AI Services for Edge Intelligence," *arXiv preprint arXiv:2305.12130*, 2023.
[12] H. Du, R. Zhang, D. Niyato, J. Kang, Z. Xiong, D. I. Kim, X. S. Shen, and H. V. Poor, "Exploring Collaborative Distributed Diffusion-based AI-Generated Content (AIGC) in Wireless Networks," *IEEE Network*, no. 99, pp. 1–8, 2023.
[13] B. Hu and Z. Cao, "Minimizing Resource Consumption Cost of DAG Applications with Reliability Requirement on Heterogeneous Processor Systems," *IEEE TIT*, vol. 16, no. 12, pp. 7437–7447, 2019.
[14] W. Chen, G. Xie, R. Li, and K. Li, "Execution Cost Minimization Scheduling Algorithms for Deadline-Constrained Parallel Applications on Heterogeneous Clouds," *Cluster Computing*, vol. 24, pp. 701–715, 2021.
[15] H. Arabnejad and J. G. Barbosa, "List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table," *IEEE TPDS*, vol. 25, no. 3, pp. 682–694, 2013.
[16] N. Rizvi, R. Dharavath, and D. R. Edla, "Cost and Makespan Aware Workflow Scheduling in IaaS Clouds using Hybrid Spider Monkey Optimization," *Simulation Modelling Practice and Theory*, vol. 110, p. 102328, 2021.
[17] G. Xie, G. Zeng, J. Jiang, C. Fan, R. Li, and K. Li, "Energy Management for Multiple Real-Time Workflows on Cyber-Physical Cloud Systems," *Future Generation Computer Systems*, vol. 105, pp. 916–931, 2020.
[18] Q. Wu, F. Ishikawa, Q. Zhu, Y. Xia, and J. Wen, "Deadline-constrained cost optimization approaches for workflow scheduling in clouds," *IEEE TPDS*, vol. 28, no. 12, pp. 3401–3412, 2017.
[19] K. Agrawal, J. Li, K. Lu, and B. Moseley, "Scheduling Parallel DAG Jobs Online to Minimize Average Flow Time," in *ACM SODA*, 2016.
[20] S. Zhao, X. Dai, and I. Bate, "DAG Scheduling and Analysis on Multi-Core Systems by Modelling Parallelism and Dependency," *IEEE TPDS*, vol. 33, no. 12, pp. 4019–4038, 2022.
[21] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, "HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face," *arXiv preprint arXiv:2303.17580*, 2023.
[22] R. Zhou, Z. Li, C. Wu, and Z. Huang, "An Efficient Cloud Market Mechanism for Computing Jobs with Soft Deadlines," *IEEE/ACM ToN*, vol. 25, no. 2, pp. 793–805, 2016.
[23] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE TPDS*, vol. 13, no. 3, pp. 260–274, 2002.
[24] L.-C. Canon, E. Jeannot, R. Sakellariou, and W. Zheng, "Comparative Evaluation of the Robustness of DAG Scheduling Heuristics," *Grid Computing: Achievements and Prospects*, pp. 73–84, 2008.