

MAWCS: A Multi-Agent Workflow Scheduling Framework for Heterogeneous Models and Cloud Resources

Goshgar Can Ismayilov^{1,*}

¹*Independent Researcher, Istanbul, Turkey*

Abstract

Multi-agent systems powered by large language models (LLMs) have been increasingly deployed to accomplish complex tasks. As these workflows grow in scale and in heterogeneity, efficient scheduling of these agents onto the cloud resources has become a more critical challenge, which directly impacts execution time, monetary cost and resource efficiency. However, the existing workflow scheduling approaches in the literature have been primarily designed for classical computational tasks and have not considered LLM-specific execution characteristics (e.g., external tool invocations, token-based inference). Therefore, we present MAWCS (Multi-Agent Workflow Scheduling Framework), a framework specifically designed for scheduling multi-agent workflows on heterogeneous cloud resources. In this work, we first define the multi-agent workflow scheduling problem as single objective optimization problem with the minimization of makespan. The minimization of cost and workload imbalance and the maximization of resource utilization have been also monitored in scheduling solutions. MAWCS implements three algorithms as the random baseline scheduler, the HEFT heuristic scheduler and the genetic algorithm (GA) metaheuristic scheduler. For realistic experimental evaluation, we consider 75 large language models upon 30 AWS EC2 general-purpose instances across M1, M3, M4, M5 and M5D families. Our experimental evaluation across synthetically-generated workflows of varying sizes demonstrates that GA-5000 (GA with 5000 generations) consistently achieves better makespan with respect to HEFT and the random schedulers.

Keywords

multi-agent systems, workflow scheduling, cloud computing, large language models, genetic algorithms

1. Introduction

The rapid advancement of generative artificial intelligence has fundamentally transformed how complex reasoning and problem-solving tasks have been approached across scientific and industrial domains [1]. Individual large language models (LLMs) trained on large-scale datasets have exhibited several emergent capabilities in natural language understanding, reasoning and code synthesis, which are increasingly viable as autonomous scientific engines [2]. Multi-agent systems have further extended this paradigm by orchestrating multiple individual LLM agents into collaborative workflows where agents collectively decompose, process and synthesize information [3]. The design and optimization of agentic workflows has attracted considerable research attention in recent years (e.g., AFlow [4], EvoFlow [5]). However, as multi-agent workflows grow in scale and are deployed to cloud computing to support demanding scientific workloads, the need for efficient scheduling of these agents onto cloud resources emerges as a critical challenge.

Efficient scheduling of tasks onto heterogeneous cloud resources is a classical problem of distributed computing. Task scheduling concerns the assignment of interdependent computational units to a set

2nd International Workshop on Symbolic and Generative AI for Science (SymGenAI4Sci), co-located with SEMANTiCS 2026: 22nd International Conference on Semantic Systems, September 15, 2026, Ghent, Belgium

*Corresponding author.

✉ goshgarismayilov@marun.edu.tr (G. C. Ismayilov)

ORCID iD 0009-0007-9307-3637 (G. C. Ismayilov)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

of machines under certain optimization objectives and constraints [6]. Workflow scheduling in cloud computing generalizes classical task scheduling by incorporating cloud-specific execution characteristics (e.g., resource provisioning dynamics [7]). Scientific workflows have potential to benefit from efficient scheduling as multi-agent pipelines are increasingly deployed to cloud computing. A scheduling framework reducing makespan and cost might promise faster iteration cycles, broader experiments and more accessible scientific discovery with feasible budgets. However, existing scheduling frameworks are primarily designed for classical scientific workflows (e.g., Montage in astronomy [7]). LLM-based multi-agent workflows introduce a fundamentally different execution profile (e.g., token-based inference time). This structural mismatch is the core motivation of this work.

In this work, we aim to answer the following research question *how to formalize the multi-agent workflow scheduling problem with LLM-tailored execution characteristics and heterogeneous cloud resources*. Based on it, this work makes the following key contributions:

1. **Problem Formalization.** We formalize the multi-agent workflow scheduling problem on heterogeneous cloud resources as a directed acyclic graph with comprehensive agent characterization (e.g., LLM inference characteristics, tool executions and token-based data transfer). We model the scheduling as the single objective optimization with the minimization of makespan. The other objectives (cost, resource utilization and workload imbalance) are also monitored post-hoc.
2. **MAWCS Framework.** For multi-agent workflows, we adapt three scheduling algorithms: (i) random scheduler for baseline performance comparison, (ii) HEFT (Heterogeneous Earliest Finish Time) heuristic scheduler with upward rank calculation and earliest finish time selection and (iii) genetic algorithm (GA) metaheuristic scheduler by primarily optimizing workflow makespan.
3. **Realistic Modeling.** For 75 different realistic large-language models (see Table 1), we consider 30 realistic AWS EC2 (Amazon Web Services Elastic Compute Cloud) general-purpose instances ranging M1, M3, M4, M5 and M5D families with vCPU counts (1-96), network bandwidth (10-25 Gbps) and on-demand hourly pricing (\$0.044-\$5.424) (see Table 2).
4. **Experimental Evaluation.** We evaluate all three schedulers across multiple synthetically-generated workflows to analyze trade-offs between makespan, cost, utilization and imbalance. We also assess genetic algorithm convergence over the increasing evolutionary iterations and measure scheduling algorithm execution time overhead. The experimental study shows that GA-5000 achieves better makespan overall.

2. Related Work

Agentic Workflows. LLMs have demonstrated remarkable capabilities in natural language understanding, multi-step reasoning, code generation and tool invocations [2]. Early LLMs run in a single-turn input-output approach where a model generates a response based on a prompt. The introduction of the ReAct framework [8] enables LLMs to interact with external environments through tool invocations, retrieve information and iteratively refine their responses. As the complexity of tasks grows beyond the capacity of single agents, multi-agent systems emerge as a natural extension where multiple LLM agents collaborate through communication and data-sharing protocols [3]. Frameworks (e.g., MetaGPT [9]) introduce role-based multi-agents where agents with distinct personas and responsibilities collaborate to solve complex software engineering and reasoning tasks. This enables diverse reasoning patterns to

emerge (e.g., *chain-of-thought*) [10], *self-consistency*) [11], *debate* [12]). The autonomous generation and optimization of agentic workflows is a more recent research direction. AFlow [4] formulates workflow optimization as a Monte Carlo Tree Search problem. EvoFlow [5] introduces an evolutionary algorithm to search a population of heterogeneous agentic workflows for trade-offs between task performance and inference cost. Despite these advances, the problem of scheduling the resulting workflows onto the available resources remains largely unaddressed.

Workflow Scheduling Algorithms. Workflow scheduling in cloud computing concerns the assignment of tasks onto a heterogeneous set of cloud resources. There already exist cloud-specific scheduling techniques in the literature. For instance, the work [7] incorporates neural networks into the multi-objective evolutionary algorithm for the dynamic workflow scheduling problem. Certain frameworks (e.g., CloudSim [13]) also provide environments to evaluate scheduling algorithms under realistic cloud conditions. Despite classical workflow scheduling, agentic workflow scheduling has received little attention in the literature. The work [14] proposes a model-independent cloud-based multi-agent workflow for scientific assistance, but scheduling of these agents on cloud resources remains open. The work [15] introduces a resource-efficient system for agentic workflows that decouples workflow specification from execution configuration, achieving reductions up to $4.3\times$ in cost and $3.7\times$ in energy consumption. However, none of them formalize agentic workflow scheduling as the optimization problem on cloud resources by considering LLM-specific characteristics (e.g., inference latency).

3. Multi-Agent Workflow Scheduling Problem in Cloud Computing

In this section, we will formulate the multi-agent workflow scheduling problem in cloud computing by specifying the workflow, cloud and execution models as well as the problem objectives.

3.1. Workflow Model

A multi-agent workflow is represented as a directed acyclic graph (DAG) $\mathcal{W} = (\mathcal{A}, \mathcal{E})$ where $\mathcal{A} = \{A_0, A_1, \dots, A_{|\mathcal{A}|-1}\}$ is a finite set of agents and $\mathcal{E} \subseteq \mathcal{A} \times \mathcal{A}$ is a set of directed edges for data dependencies between agents. A directed edge $(A_i, A_j) \in \mathcal{E}$ indicates that agent A_j depends on the output of agent A_i and therefore cannot begin execution until A_i has completed and transferred its output. Agents with no incoming edges, $\mathcal{A}_{\text{entry}} = \{A_i \mid \delta_i^{\text{in}} = 0\}$, are entry agents and agents with no outgoing edges, $\mathcal{A}_{\text{exit}} = \{A_i \mid \delta_i^{\text{out}} = 0\}$, are exit agents. Each agent $A_i \in \mathcal{A}$ is characterized by the tuple $A_i = (M_i, \alpha_i, t_i^{\text{inf}}, n_i^{\text{out}}, \mathcal{T}_i)$ where $M_i \in \mathcal{M}$ is the LLM from the set of available models $\mathcal{M}$ (see Table 1). $\alpha_i \in \mathbb{R}^+$ is the network latency before inference begins. $t_i^{\text{inf}} \in \mathbb{R}^+$ is the inference time:

$$t_i^{\text{inf}} = t_i^{\text{ttft}} + n_i^{\text{out}} / \lambda_i \quad (1)$$

where $t_i^{\text{ttft}} \in \mathbb{R}^+$ is the time-to-first-token and $\lambda_i \in \mathbb{R}^+$ is the model throughput in tokens per second. The model throughput here is treated as a fixed, model-specific value (see Table 1) while it depends on several factors (e.g., context length, concurrency, compute capacity) in real-life scenarios. In this paper, the inference times t_i^{inf} are assumed to be reference values with a single-vCPU configuration. $n_i^{\text{out}} \in \mathbb{N}$ is the output token count transferred to successor agents upon completion. Finally, $\mathcal{T}_i = \{t_1, t_2, \dots, t_{|\mathcal{T}_i|}\}$ is the set of external tools invoked by agent A_i where each tool t_k has a per-invocation latency $\ell_k \in \mathbb{R}^+$ and an invocation count $r_k \in \mathbb{N}^+$, contributing a total tool execution latency of:

$$\ell_i = \sum_{t_k \in \mathcal{T}_i} \ell_k \cdot r_k \quad (2)$$

Table 1

Large-language models with reference performance characteristics of throughput and time-to-first-token [16], assumed to be measured at a single-vCPU configuration. In this work, we deliberately treat the models as open-source and self-hosted on cloud resources to simulate the scheduling algorithms.

Model	Tput (t/s)	TTFT (s)	Model	Tput (t/s)	TTFT (s)
OPENAI MODELS			OPENAI MODELS		
gpt-5.4-pro	74	151.79	o1	98	32.29
gpt-5.4	74	151.79	o3	118	5.38
gpt-5.4-mini	201	3.85	o3-mini	160	7.12
gpt-5.4-nano	191	3.64	o3-pro	27	84.93
gpt-5.3-codex	79	88.26	o4-mini	161	21.94
gpt-5.2-codex	123	87.34	ANTHROPIC CLAUDE		
gpt-5.2	73	130.34	claude-4.1-opus	29	1.66
gpt-5.1	111	57.47	claude-4.1-opus-thinking	29	15.00
gpt-5	83	36.28	claude-4-sonnet	40	1.33
gpt-5-nano	137	83.30	claude-4.6-sonnet	44	1.48
gpt-5-mini	86	65.32	claude-4.5-opus	46	1.01
gpt-4.1	108	1.02	claude-4.6-opus	40	1.78
gpt-4.1-nano	181	0.63	claude-3-haiku	138	1.16
gpt-4.1-mini	80	0.76	GOOGLE GEMINI		
gpt-4o	131	0.81	gemini-3.1-pro	109	29.71
gpt-4o-mini	33	3.16	gemini-3.1-flash-lite	205	7.50
gpt-4-turbo	30	2.84	gemini-3-pro	109	32.65
gpt-oss-120b	262	0.79	gemini-3-flash	159	1.19
gpt-oss-20b	313	0.65			
GOOGLE GEMINI			MISTRAL MODELS		
gemini-2.5-pro	117	21.19	mistral-small-4	175	0.64
gemini-2.5-flash	221	0.50	mistral-medium-3	57	1.20
META LLAMA			mistral-large-3	48	1.04
llama-4-scout	128	0.70	mistral-large-2	38	1.45
llama-4-maverick	121	0.95	xAI GROK		
llama-3.1-405b	29	2.19	grok-4.20	233	10.33
MISTRAL MODELS			grok-4.3	209	12.36
ministral-3-3b	274	0.42	grok-4.1-fast	138	0.54
ministral-3-8b	182	0.52	grok-4	54	15.60
ministral-3-14b	110	0.60	grok-3-mini	190	0.54
xAI GROK			OTHER MODELS		
grok-code-fast-1	172	2.81	glm-5	74	1.64
OTHER MODELS			glm-4.7	82	1.10
nemotron-3-super-100b	367	0.71	glm-4.7-flash	95	0.91
nemotron-3-nano-30b	152	1.90	glm-4.5	51	1.45
deepseek-v3.2	35	3.75	glm-4.5-air	106	1.18
deepseek-r1-distill-qwen-32b	60	0.84	lrm2-24b-a2b	92	0.42
mercury-2-inception	789	3.88	step-3.5-flash	87	3.03
ling-2.6-flash	210	1.07	minimax-m2.7	45	2.53
nova-pro	141	0.81	minimax-m2.5	46	2.12
mimo-v2-flash	129	2.14	kimi-k2.5	45	2.38
qwen-3.5-397b	96	2.44	kimi-k2	43	1.51
			phi-4	35	2.02
			gemma-3-27b	31	2.04

3.2. Cloud Model

Let $\mathcal{R} = \{r_0, r_1, \dots, r_{|\mathcal{R}|-1}\}$ be a finite set of heterogeneous cloud resources available for agent execution. Each resource $r \in \mathcal{R}$ is characterized by the following tuple of $r = (v_r, B_r, p_r)$ where $v_r \in \mathbb{N}^+$ is the *virtual CPU count*, $B_r \in \mathbb{R}^+$ is the *network bandwidth* (Gbps) for data transfer rate between resources during inter-agent communication and $p_r \in \mathbb{R}^+$ is the *hourly on-demand price* (\$/hour) charged by the cloud provider. In this work, each LLM model is assumed to be self-hosted as an inference service on the cloud resource assigned. The virtual CPU count v_r determines the resource compute capacity to scale the agent inference time accordingly. The network bandwidth between two resources r_i and r_j is simply defined as $B(r_i, r_j) = \min(B_{r_i}, B_{r_j})$ to simulate bandwidth bottleneck. When two agents are assigned to the same resource (i.e., $r_i = r_j$), data transfer is assumed to be negligible, $\tau_{ij}(r_i, r_j) = 0$. Each resource $r \in \mathcal{R}$ maintains an availability time $\text{avail}(r) \in \mathbb{R}^+$ that tracks the earliest time at which the

resource becomes free to accept a new agent assignment. In this work, we consider 30 realistic AWS EC2 general-purpose instances spanning M1, M3, M4, M5 and M5D instance families with vCPU counts ranging from 1 to 96, network bandwidths from 10 to 25 Gbps and hourly pricing from \$0.044 to \$5.424 per hour (see Table 2). In this work, we do not consider cloud-related dynamism and uncertainties (e.g., network congestion, API rate limits).

Table 2

AWS EC2 instance specifications including virtual CPU count, network bandwidth and hourly on-demand pricing for five resource families (M1, M3, M4, M5, M5D) [17].

Instance Type	vCPUs	Network (Gbps)	Price (\$/hour)	Instance Type	vCPUs	Network (Gbps)	Price (\$/hour)
M1 FAMILY				M4 FAMILY			
m1.small	1	10	0.044	m4.large	2	10	0.1
m1.medium	1	10	0.087	m4.xlarge	4	10	0.2
m1.large	2	10	0.175	m4.2xlarge	8	10	0.4
m1.xlarge	4	10	0.35	m4.4xlarge	16	10	0.8
M3 FAMILY				m4.10xlarge	40	10	2.0
m3.medium	1	10	0.067	m4.16xlarge	64	25	3.2
m3.large	2	10	0.133	M5 FAMILY			
m3.xlarge	4	10	0.266	m5.large	2	10	0.096
m3.2xlarge	8	10	0.532	m5.xlarge	4	10	0.192
M5 FAMILY				m5.2xlarge	8	10	0.384
m5.4xlarge	16	10	0.768	m5.8xlarge	32	10	1.536
m5.12xlarge	48	12	2.304	m5.16xlarge	64	20	3.072
m5.24xlarge	96	25	4.608				
M5D FAMILY				M5D FAMILY			
m5d.large	2	10	0.113	m5d.xlarge	4	10	0.226
m5d.2xlarge	8	10	0.452	m5d.4xlarge	16	10	0.904
m5d.8xlarge	32	10	1.808	m5d.12xlarge	48	12	2.712
m5d.16xlarge	64	20	3.616	m5d.24xlarge	96	25	5.424

3.3. Execution Model

The execution model defines how agents are scheduled and executed on cloud resources over time. For each agent $A_i \in \mathcal{A}$ assigned to resource $r_i \in \mathcal{R}$, the start time $ST(A_i, r_i)$ is determined by two independent constraints that must be jointly satisfied as (i) *dependency readiness*, requiring all predecessor agents to have completed execution and transferred their output data to r_i , and (ii) *resource availability*, requiring the assigned resource r_i to be free from prior agent assignments. Formally, the start time is defined as:

$$ST(A_i, r_i) = \max \left(\text{avail}(r_i), \max_{A_j \in \text{pred}_i} (FT(A_j, r_j) + \tau_{ji}(r_j, r_i)) \right) \quad (3)$$

where $\text{avail}(r_i) \in \mathbb{R}^+$ is the earliest idle time of resource r_i after completing all previously assigned agents and $\tau_{ji}(r_j, r_i)$ is the data transfer time from the resource of predecessor A_j to r_i . The data transfer time is computed as:

$$\tau_{ji}(r_j, r_i) = \begin{cases} 0 & \text{if } r_j = r_i \\ n_j^{\text{out}} \times \beta / B(r_j, r_i) & \text{otherwise} \end{cases} \quad (4)$$

where n_j^{out} is the output data size of predecessor A_j in tokens, β is a fixed byte-per-token conversion constant and $B(r_j, r_i) = \min(B_{r_j}, B_{r_i})$ is the network bandwidth between the two resources. The execution of agent A_i passes over three phases as (i) *network latency* α_i , (ii) *tool execution* with $\ell_i = \sum_{t_k \in \mathcal{T}_i} \ell_k \cdot r_k$ and (iii) *LLM inference*. The finish time of agent A_i on resource r_i is computed as:

$$FT(A_i, r_i) = ST(A_i, r_i) + \alpha_i + \ell_i + t_i^{\text{inf}} / v_{r_i} \quad (5)$$

where the workflow terminates when all exit agents are completed $\mathcal{A}_{\text{exit}} = \{A_i \mid \delta_i^{\text{out}} = 0\}$. The execution model assumes non-preemptive scheduling where each agent executes to completion without interrupted once started.

3.4. Problem Formulation

An assignment solution is a mapping $\sigma : \mathcal{A} \rightarrow \mathcal{R}$ that assigns each agent $A_i \in \mathcal{A}$ to a cloud resource $\sigma(A_i) \in \mathcal{R}$. The multi-agent workflow scheduling problem seeks an optimal assignment σ^* that minimizes the workflow makespan, subject to a set of scheduling constraints. Formally, the problem is:

$$\sigma^* \in \arg \min_{\sigma} W_{\text{makespan}}(\sigma) \quad (6)$$

where the workflow makespan is defined as:

$$W_{\text{makespan}}(\sigma) = \max_{A_i \in \mathcal{A}} FT(A_i, \sigma(A_i)) \quad (7)$$

where $FT(A_i, \sigma(A_i))$ denotes the finish time of agent A_i on its assigned resource $\sigma(A_i)$, subject to the following constraints:

- **Precedence Constraint.** For every directed edge $(A_i, A_j) \in \mathcal{E}$, agent A_j cannot begin execution before agent A_i has completed and its output data has been fully transferred:

$$ST(A_j, \sigma(A_j)) \geq FT(A_i, \sigma(A_i)) + \tau_{ij}(\sigma(A_i), \sigma(A_j)), \quad \forall (A_i, A_j) \in \mathcal{E} \quad (8)$$

where ST is the agent start time and τ is the data transfer time among resources.

- **Resource Serialization Constraint.** Each resource $r \in \mathcal{R}$ executes at most one agent at a time under the non-preemptive execution model where for any two agents $A_i, A_j \in \mathcal{A}$ assigned to the same resource $\sigma(A_i) = \sigma(A_j) = r$, their execution intervals must be non-overlapping:

$$ST(A_j, r) \geq FT(A_i, r) \quad \text{or} \quad ST(A_i, r) \geq FT(A_j, r), \quad \forall A_i \neq A_j : \sigma(A_i) = \sigma(A_j) \quad (9)$$

- **Assignment Completeness Constraint.** Every agent in the workflow must be assigned to exactly one resource:

$$|\{\sigma(A_i)\}| = 1, \quad \forall A_i \in \mathcal{A} \quad (10)$$

- **Non-Negativity Constraint.** All start times must be non-negative, ensuring that no agent begins execution before the workflow is initiated:

$$ST(A_i, \sigma(A_i)) \geq 0, \quad \forall A_i \in \mathcal{A} \quad (11)$$

The primary scheduling objective is the makespan minimization as defined in Equation 6, which drives all three scheduling algorithms presented in this work. The remaining metrics (cost, average resource utilization and average workload imbalance) are evaluated post-hoc on the resulting schedule σ to provide a comprehensive multi-dimensional characterization of scheduling quality.

3.5. Problem Objectives

The multi-agent workflow scheduling problem on heterogeneous cloud resources considers several objectives concerning efficient scheduling. **Makespan** is the total workflow execution time from initiation to completion which is formally defined in the previous section. **Cost** is the total monetary cost of executing the workflow on cloud resources:

$$W_{\text{cost}} = \sum_{r \in \mathcal{R}} \lceil \frac{T_r}{3600} \rceil \times p_r \quad (12)$$

where T_r is the total duration in seconds from first agent start to last agent finish on resource r , p_r is the hourly on-demand pricing of resource r , and $\lceil \cdot \rceil$ is the ceiling function for per-hour billing granularity. This cost metric focuses on EC2 infrastructure rental costs while agent-level LLM inference costs are not considered. Refer to Table 2 for the cloud resource pricing. **Average Utilization** measures the average resource utilization across all resources in the scheduling:

$$W_{\text{utilization}} = \frac{1}{|\mathcal{R}'|} \sum_{r \in \mathcal{R}'} \frac{T_r^{\text{busy}}}{T_{\text{exec}}} \times 100\% \quad (13)$$

where $\mathcal{R}' \subseteq \mathcal{R}$ is the set of resources that execute at least one agent, T_r^{busy} is the cumulative execution time of all agents on resource r and T_{exec} is the total workflow makespan. Higher utilization means more efficient resource use by successfully reducing idle times. Note that per-hour billing granularity of cloud computing can yield low utilization values for short-lived agentic workflows. Alternative billing models (e.g. serverless) might be promising from the perspective of utilization. **Average Imbalance** quantifies the load distribution balance across resources:

$$W_{\text{imbalance}} = \frac{\max_{r \in \mathcal{R}'} T_r^{\text{busy}} - \min_{r \in \mathcal{R}'} T_r^{\text{busy}}}{\frac{1}{|\mathcal{R}'|} \sum_{r \in \mathcal{R}'} T_r^{\text{busy}}} \quad (14)$$

where it shows the ratio of the range between maximum and minimum busy times to the average busy time. Lower imbalances mean more balanced workload distribution by reducing resource bottlenecks. This single objective optimization might carry certain risks by producing disproportionate scheduling solutions (e.g., with severe workload imbalance). Hence, an extension to multi-objective optimization and employ multi-objective evolutionary frameworks (e.g., NSGA-II [18]) might be promising.

4. The MAWCS Framework

In this section, we present MAWCS, a comprehensive scheduling framework that is designed for multi-agent workflows on heterogeneous cloud resources. The framework implements three scheduling algorithms with increasing sophistication: (i) Random (RAND), (ii) HEFT and (iii) GA.

4.1. Random Baseline Scheduling

The random baseline scheduler provides a performance lower-bound reference for evaluating more sophisticated scheduling strategies. The algorithm does not employ any intelligent task prioritization

and resource selection mechanisms. However, it obeys the problems constraints for workflow dependencies and resource constraints to produce valid and realistic schedules on cloud resources. The random scheduler operates in two main phases as (i) *dependency-aware task ordering* through topological sort where agents at the same dependency level are selected and randomly shuffled and (ii) *random resource assignment* where each agent A_i is mapped to a randomly-selected resource $r_i \sim \mathcal{U}(\mathcal{R})$. Let L_k denote agents at dependency level k whose predecessors are in $\{L_0, L_1, \dots, L_{k-1}\}$. The algorithm processes levels sequentially while randomly permuting agents within each level before scheduling.

4.2. HEFT Scheduling

In this work, we adapt HEFT [19] into our problem, which prioritizes agents based on their critical path contribution and selects resources by minimizing earliest finish time. The algorithm operates in two main phases as (i) *upward rank computation* and (ii) *agent scheduling*. The upward rank $\text{rank}_u(A_i)$ estimates the length of the critical path from agent A_i to the exit of the workflow and is defined as:

$$\text{rank}_u(A_i) = \overline{w}_i + \begin{cases} 0 & \text{if } A_i \in \mathcal{A}_{\text{exit}} \\ \max_{A_j \in \text{succ}_i} (\overline{c}_{ij} + \text{rank}_u(A_j)) & \text{otherwise} \end{cases} \quad (15)$$

where $\overline{w}_i$ is the average execution time of agent A_i across all resources $r \in \mathcal{R}$ and $\overline{c}_{ij}$ is the average data transfer time between agents A_i and A_j . The upward rank is computed in reverse topological order starting from $\mathcal{A}_{\text{exit}}$ and propagating backwards to $\mathcal{A}_{\text{entry}}$. Agents with higher upward rank values reside on longer critical paths and are hence prioritized in scheduling. Once upward ranks are computed, agents are sorted in non-increasing order of $\text{rank}_u(A_i)$ to form a priority list $\mathcal{L}$. Agents are then scheduled one-by-one from $\mathcal{L}$ where each agent A_i is assigned to the resource $r^* \in \mathcal{R}$ that minimizes its earliest finish time:

$$r^* = \arg \min_{r \in \mathcal{R}} EFT(A_i, r) \quad (16)$$

where the earliest finish time on resource r is computed as:

$$EFT(A_i, r) = EST(A_i, r) + \alpha_i + \ell_i + t_i^{\text{inf}} / v_r \quad (17)$$

with the earliest start time $EST(A_i, r)$ defined as:

$$EST(A_i, r) = \max \left(\text{avail}(r), \max_{A_j \in \text{pred}_i} (FT(A_j, r_j) + \tau_{ji}(r_j, r)) \right) \quad (18)$$

where $\text{avail}(r)$ is the earliest time resource r becomes free after completing its previously assigned agents and $\text{pred}_i \subseteq \mathcal{A}$ is the set of predecessor agents of A_i . The pseudocode of the complete HEFT scheduling procedure is given in Algorithm 1.

4.3. Genetic Algorithm Scheduling

GA scheduler relies on evolutionary optimization by iteratively evolving a population of several candidate scheduling solutions. Unlike HEFT that greedily constructs a single schedule the GA scheduler explores a broader solution space through the mechanisms of evolutionary operators and elitism mechanism. In this scheduler, each candidate scheduling solution is represented as a chromosome $\mathbf{x} = [x_0, x_1, \dots, x_{|\mathcal{A}|-1}]$ where each gene $x_i \in \{0, 1, \dots, |\mathcal{R}|-1\}$ is the resource index assigned to agent $A_{\pi(i)}$.

Algorithm 1 HEFT Scheduling

```
1: Phase 1: Upward Rank Computation
2: Compute topological ordering  $\pi$  of  $\mathcal{A}$ 
3: for  $A_i \in \pi$  in reverse order do
4:    $\bar{w}_i \leftarrow \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} (\alpha_i + \ell_i + t_i^{\text{inf}} / v_r)$ 
5:   if  $A_i \in \mathcal{A}_{\text{exit}}$  then
6:      $\text{rank}_u(A_i) \leftarrow \bar{w}_i$ 
7:   else
8:      $\text{rank}_u(A_i) \leftarrow \bar{w}_i + \max_{A_j \in \text{succ}_i} (\bar{c}_{ij} + \text{rank}_u(A_j))$ 
9:   end if
10: end for
11: Phase 2: Agent Scheduling
12:  $\mathcal{L} \leftarrow \text{sort } \mathcal{A} \text{ in non-increasing order of } \text{rank}_u(A_i)$ 
13: Initialize  $\text{avail}(r) \leftarrow 0$  for all  $r \in \mathcal{R}$ 
14: for  $A_i \in \mathcal{L}$  do
15:   for  $r \in \mathcal{R}$  do
16:      $EST(A_i, r) \leftarrow \max(\text{avail}(r), \max_{A_j \in \text{pred}_i} (FT(A_j, r_j) + \tau_{ji}(r_j, r)))$ 
17:      $EFT(A_i, r) \leftarrow EST(A_i, r) + \alpha_i + \ell_i + t_i^{\text{inf}} / v_r$ 
18:   end for
19:    $r^* \leftarrow \arg \min_{r \in \mathcal{R}} EFT(A_i, r)$ 
20:    $\sigma(A_i) \leftarrow r^*$ 
21:    $FT(A_i) \leftarrow EFT(A_i, r^*)$ 
22:    $\text{avail}(r^*) \leftarrow FT(A_i)$ 
23: end for
```

The initial population $\mathcal{P}_0 = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(N_{\text{pop}})}\}$ of size N_{pop} is generated by uniformly sampling each gene as $x_i \sim \mathcal{U}(\{0, \dots, |\mathcal{R}| - 1\})$. The fitness of each individual $\mathbf{x}$ is the workflow makespan W_{makespan} , which models the scheduling problem as the minimization problem.

For parent selections, tournament selection is used where individuals are drawn uniformly at random from the current population and the individual with the lowest fitness is selected as a parent. This is repeated independently to select two parents $\mathbf{x}^{(p_1)}$ and $\mathbf{x}^{(p_2)}$ for each offspring pair. For crossover, given two parents $\mathbf{x}^{(p_1)}$ and $\mathbf{x}^{(p_2)}$, single-point crossover is applied where a crossover point $k \sim \mathcal{U}(\{1, \dots, |\mathcal{A}| - 1\})$ is sampled uniformly and two offspring are produced as:

$$\mathbf{x}^{(c_1)} = [x_0^{(p_1)}, \dots, x_{k-1}^{(p_1)}, x_k^{(p_2)}, \dots, x_{|\mathcal{A}|-1}^{(p_2)}] \quad (19)$$

$$\mathbf{x}^{(c_2)} = [x_0^{(p_2)}, \dots, x_{k-1}^{(p_2)}, x_k^{(p_1)}, \dots, x_{|\mathcal{A}|-1}^{(p_1)}] \quad (20)$$

For mutation, each gene x_i of an offspring is independently mutated with probability $p_m \in [0, 1]$ by resampling a new resource index uniformly as $x_i \sim \mathcal{U}(\{0, \dots, |\mathcal{R}| - 1\})$. To preserve the best solutions across generations, the GA scheduler employs elitism by directly copying the top N_{elite} individuals into the next population. The GA scheduler evolves the population for a fixed number of generations N_{gen} by converging into $\mathbf{x}^* = \arg \min_{\mathbf{x}} f(\mathbf{x})$. The pseudocode of the algorithm is given in Algorithm 2.

5. Experimental Study

All experiments are conducted on a single machine of MacBook Air (Apple M2 chip, 8 GB memory and 8 CPU cores). All three schedulers are evaluated against the same set of synthetically-generated workflows and the same pool of 30 AWS EC2 resource instances from Table 2. For the genetic algorithm scheduler, we use a population size of $N_{\text{pop}} = 50$, a default number of generations of $N_{\text{gen}} = 100$, a crossover rate of $p_c = 0.8$, a mutation rate of $p_m = 0.1$ and an elite size of $N_{\text{elite}} = 5$. The execution times

Algorithm 2 Genetic Algorithm Scheduling

```

1:  $\pi \leftarrow$  topological ordering of  $\mathcal{W}$ 
2:  $\mathcal{P}_0 \leftarrow \{\mathbf{x}^{(k)} \sim \mathcal{U}(\{0, \dots, |\mathcal{A}| - 1\})\}_{k=1}^{N_{\text{pop}}}$ 
3:  $\mathbf{x}^* \leftarrow \emptyset, f^* \leftarrow \infty$ 
4: for  $g = 0, 1, \dots, N_{\text{gen}} - 1$  do
5:    $F_g \leftarrow \{f(\mathbf{x}^{(k)})\}_{k=1}^{N_{\text{pop}}}$  ▷ Evaluate fitness for all individuals
6:   if  $\min(F_g) < f^*$  then
7:      $f^* \leftarrow \min(F_g); \mathbf{x}^* \leftarrow \arg \min F_g$ 
8:   end if
9:    $\mathcal{P}_{g+1} \leftarrow$  top  $N_{\text{elite}}$  individuals from  $\mathcal{P}_g$  ▷ Elitism
10:  while  $|\mathcal{P}_{g+1}| < N_{\text{pop}}$  do
11:     $\mathbf{x}^{(p_1)}, \mathbf{x}^{(p_2)} \leftarrow \text{TournamentSelect}(\mathcal{P}_g, F_g)$ 
12:     $\mathbf{x}^{(c_1)}, \mathbf{x}^{(c_2)} \leftarrow \text{Crossover}(\mathbf{x}^{(p_1)}, \mathbf{x}^{(p_2)}, p_c)$ 
13:     $\mathbf{x}^{(c_1)}, \mathbf{x}^{(c_2)} \leftarrow \text{Mutate}(\mathbf{x}^{(c_1)}, \mathbf{x}^{(c_2)}, p_m)$ 
14:     $\mathcal{P}_{g+1} \leftarrow \mathcal{P}_{g+1} \cup \{\mathbf{x}^{(c_1)}, \mathbf{x}^{(c_2)}\}$ 
15:  end while
16: end for
17:  $\sigma \leftarrow \text{decode}(\mathbf{x}^*)$  ▷ Construct final schedule from best individual
18: Simulate  $\sigma$  in topological order  $\pi$  to compute  $FT$ 

```

are measured using wall-clock time to measure the end-to-end computational overhead. Each agent is instantiated with (i) an LLM model sampled uniformly from Table 1, (ii) an output token count sampled from $\mathcal{U}(50, 500)$ and (iii) a set of external tools (0-5 per agent) with per-tool invocation counts from $\mathcal{U}(1, 5)$ and (iv) latencies sampled from two distributions: intra-tool (local operations) from $\mathcal{U}(5, 100)$ milliseconds and inter-tool (external API calls) from $\mathcal{U}(100, 1000)$ milliseconds.

For diverse agentic workflows, we generate synthetic multi-agent workflows through a parametric generation procedure that composes six fundamental agentic reasoning topologies: *single-agent* (atomic task), *chain-of-thought* (sequential n -agent pipeline), *tree-of-thought* (one splitter agent branching into b parallel agents), *self-consistency* (tree-of-thought followed by an aggregator agent), *debate* (two agents exchanging arguments over r rounds) and *self-refine* (a chain of agents each performing r iterative refinement rounds). To maintain the acyclic property of the workflow DAG, iterative patterns such as *debate* and *self-refine* are unrolled into sequential chains of distinct agent nodes. For instance, a *debate* pattern between two agents of A_1 and A_2 with r rounds is represented as $2r$ distinct agent nodes (e.g., $A_{1,1} \rightarrow A_{2,1} \rightarrow A_{1,2} \rightarrow A_{2,2} \rightarrow \dots \rightarrow A_{1,r} \rightarrow A_{2,r}$) forming a sequential alternating chain. This procedure generates heterogeneous workflows by composing multiple pattern instances using a first-order Markovian transition mechanism. In this research, only synthetic workflows are considered for greater flexibility over parameters, allowing scheduler behavior to be more easily isolated.

5.1. On Increasing Workflow Size

In this experiment, we measure the effects of the increasing workflow size on the objectives of makespan, cost, average utilization, average imbalance and used resource count; and report the results in Figure 1. For all workflow sizes, both HEFT and GA-5000 successfully outplay the random scheduler in terms of makespan (i.e., the main objective). At $W = 5$, HEFT achieves a stronger makespan reduction than GA as HEFT exploits the limited task graph structure efficiently with minimal search overhead. However, from $W = 10$ onward, GA-5000 achieves better makespan at every scale. At $W = 5$, HEFT achieves a dramatically lower cost than both GA-5000 and Random. From $W = 10$ onward, GA progressively narrows and then briefly exceeds HEFT where at $W = 25$, GA-5000 gets better cost against HEFT. Average resource utilization declines with workflow size across all schedulers, falling from the range of

1–3% at $W = 5$ to approximately 0.7–0.8% at $W = 250$ because of the cloud hourly-billing model. As workflow size grows, more resource instances are provisioned, but tasks occupy only a small fraction of the available slots. At $W = 250$, all three schedulers show similarly low utilization (Random: 0.76%, HEFT: 0.76%, GA: 0.83%). Overall, GA-5000 excels in the main objective from $W = 10$ onward.

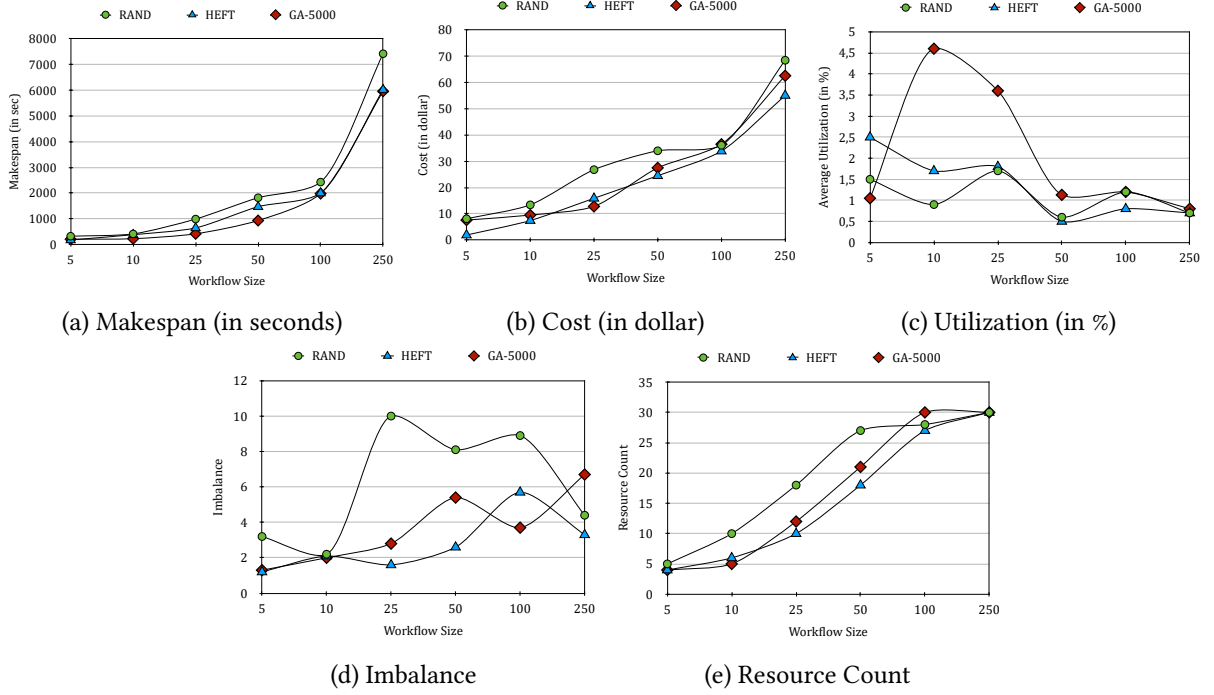


Figure 1: Effects of increasing workflow size across five objectives of (a) makespan, (b) cost, (c) average utilization, (d) average imbalance and (e) resource count for three adapted schedulers.

5.2. On Increasing Evolutionary Iteration

In this experiment, we measure performance of genetic algorithm schedulers on makespan and report the results with relative percentage-based improvements in Table 3. For all workflow sizes, increasing the number of generations improves makespan, but with diminishing marginal returns. The largest single improvement step is observed between GA-100 and GA-250: at $W = 5$, this step alone accounts for 31.6 percentage where the total 37.9% gain eventually achieved by GA-5000. Beyond GA-1000, marginal gains turn small at most workflow sizes. For instance, at $W = 100$, GA-1000 achieves 11.7% improvement while GA-2500 and GA-5000 reach only 13.2% and 24.2% respectively. This is consistent with the convergence behavior of genetic algorithms where early generations for exploration considerably improve the fitness values while the later generations for exploitation reach a plateau. For deployment scenarios with a fixed time budget, GA-1000 is a cost-effective configuration, capturing the majority of achievable gains.

5.3. On Scheduler Execution Latency

In this experiment, we measure the execution latencies of schedulers over increasing workflow sizes and report the results in Figure 2. In the figure, note that we include the log-scaled latencies for better visibility and provide six genetic algorithm variants with increasing number of generations (e.g., GA-100 with 100 generations). Firstly, both the random and HEFT schedulers exhibit negligible latency

Table 3

Performance of GA schedulers on makespan (in seconds) across varying generations for six workflow sizes.

Scheduler	W=5	W=10	W=25	W=50	W=100	W=250
GA-100	335 (0%)	349 (0%)	632 (0%)	1401 (0%)	2614 (0%)	6980 (0%)
GA-250	229 (+31.6%)	300 (+14.0%)	611 (+3.3%)	1266 (+9.6%)	2433 (+6.9%)	6830 (+2.1%)
GA-500	220 (+34.3%)	232 (+33.5%)	619 (+2.1%)	1122 (+19.9%)	2301 (+11.9%)	6410 (+8.2%)
GA-1000	210 (+37.3%)	234 (+32.9%)	533 (+15.7%)	978 (+30.2%)	2307 (+11.7%)	6427 (+8.0%)
GA-2500	222 (+33.7%)	235 (+32.7%)	479 (+24.2%)	1019 (+27.3%)	2270 (+13.2%)	6202 (+11.1%)
GA-5000	208 (+37.9%)	228 (+34.7%)	419 (+33.7%)	935 (+33.3%)	1981 (+24.2%)	5961 (+14.6%)

across all workflow sizes where the random scheduler only grows from 0.0001 s at size 5 to 0.0026 s at size 250. HEFT is relatively slower because of its upward-rank computation and insertion-based scheduling phases, reaching 0.0423 s at the largest workflow. Secondly, latencies of the GA scheduler grow consistently with workflow size for all generation configurations because each fitness evaluation involves scheduling solution with n tasks and larger workflows impose denser dependency graphs by requiring more dependency-constraint check. Thirdly, execution latency of GA scales linearly with the number of generations. For time-sensitive workflows, HEFT is a more practical choice.

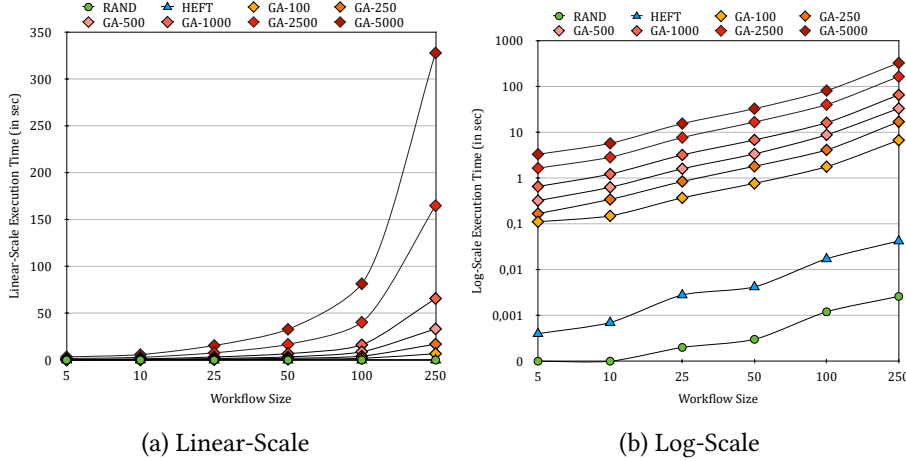


Figure 2: Execution latencies of schedulers with varying workflow sizes in (a) linear-scale and (b) log-scale.

6. Conclusion

In this work, we formalize the multi-agent workflow scheduling problem in cloud computing as the directed acyclic graph optimization by specifically considering the LLM-specific execution characteristics. For this problem, we present MAWCS which is a comprehensive framework for scheduling multi-agent workflows on heterogeneous cloud resources. The framework implements three scheduling algorithms as baseline random scheduling, HEFT scheduling and genetic algorithm scheduling by considering the minimization of makespan. During the experimental evaluation, we consider 75 LLM models and 30 AWS EC2 resource instances to enable practical deployment planning for multi-agent systems. In the future, we will extend our work into: (i) formalization of the agentic workflow scheduling problem as multi-objective optimization, (ii) consideration of dynamic scenarios and uncertainties and (iii) validation of MAWCS on realistic workflows.

Declaration on Generative AI

During the preparation of this work, the author(s) used Anthropic Claude in order to: Improve writing style, Grammar and spelling check. After using this tool, the author reviewed and edited the content as needed and takes full responsibility for the publication's content.

References

- [1] W. X. Zhao, K. Zhou, J. Li, T. Tang, Z. Dong, Y. Hou, B. Zhang, Y. Min, J. Zhang, P. Liu, et al., A survey of large language models, *Frontiers of Computer Science* 20 (2026) 2012627.
- [2] J. Huang, K. C.-C. Chang, Towards reasoning in large language models: A survey, in: *Findings of the association for computational linguistics: ACL 2023*, 2023, pp. 1049–1065.
- [3] T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, X. Zhang, Large language model based multi-agents: A survey of progress and challenges, *arXiv preprint arXiv:2402.01680* (2024).
- [4] J. Zhang, J. Xiang, Z. Yu, F. Teng, X. Chen, J. Chen, M. Zhuge, X. Cheng, S. Hong, J. Wang, et al., Aflow: Automating agentic workflow generation, in: *International Conference on Learning Representations*, volume 2025, 2025, pp. 34040–34077.
- [5] G. Zhang, K. Chen, G. Wan, H. Chang, H. Cheng, K. Wang, S. Hu, L. Bai, Evoflow: Evolving diverse agentic workflows on the fly, *arXiv preprint arXiv:2502.07373* (2025).
- [6] A. Arunarani, D. Manjula, V. Sugumaran, Task scheduling techniques in cloud computing: A literature survey, *Future Generation Computer Systems* 91 (2019) 407–415.
- [7] G. Ismayilov, H. R. Topcuoglu, Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing, *Future Generation computer systems* 102 (2020) 307–322.
- [8] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao, React: Synergizing reasoning and acting in language models, *arXiv preprint arXiv:2210.03629* (2022).
- [9] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, J. Wang, C. Zhang, S. Yau, Z. Lin, L. Zhou, et al., Metagpt: Meta programming for a multi-agent collaborative framework, in: *International Conference on Learning Representations*, volume 2024, 2024, pp. 23247–23275.
- [10] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al., Chain-of-thought prompting elicits reasoning in large language models, *Advances in neural information processing systems* 35 (2022) 24824–24837.
- [11] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, D. Zhou, Self-consistency improves chain of thought reasoning in language models, *arXiv preprint arXiv:2203.11171* (2022).
- [12] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, I. Mordatch, Improving factuality and reasoning in language models through multiagent debate, in: *Forty-first international conference on machine learning*, 2024.
- [13] T. Goyal, A. Singh, A. Agrawal, Cloudsim: simulator for cloud computing infrastructure and modeling, *Procedia engineering* 38 (2012) 3566–3572.
- [14] A. Acharya, T. Vega, R. A. Ashraf, A. Sharma, D. Parker, R. Rallo, A cloud-based multi-agentic workflow for science, *arXiv preprint arXiv:2601.12607* (2026).
- [15] G. I. Chaudhry, E. Choukse, H. Qiu, Í. Goiri, R. Fonseca, A. Belay, R. Bianchini, Murakkab: Resource-efficient agentic workflow orchestration in cloud platforms, *arXiv preprint arXiv:2508.18298* (2025).

- [16] BenchLM.AI, LLM speed benchmarks, <https://benchlm.ai/llm-speed>, 2026.
- [17] Vantage, Amazon EC2 instance comparison, <https://instances.vantage.sh>, 2026.
- [18] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: Nsga-ii, *IEEE transactions on evolutionary computation* 6 (2002) 182–197.
- [19] H. Topcuoglu, S. Hariri, M.-Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, *IEEE transactions on parallel and distributed systems* 13 (2002) 260–274.