

Adaptive Meta-Heuristic Optimization for Cloud Resource Allocation: A Comparative Study of TLBO, PSO, and GA

Monika Bishnoi^{1 a)} and Neha Dutta^{2 b)}

¹ Computer Science & Engineering, University of Institute & Engineering, Chandigarh University, Mohali-140413, Punjab, India

² Computer Science & Engineering, University of Institute & Engineering, Chandigarh University, Mohali-140413, Punjab, India

^{a)} Corresponding author: monikabishnoi3@gmail.com

^{b)} Neha.e12830@cumail.in

Abstract: Cloud computing provides scalable, on-demand infrastructure, but effective resource allocation remains challenging when workloads fluctuate and virtual machines (VMs) differ in processing capacity, memory, storage, and bandwidth. Fixed scheduling rules may not adapt adequately to these conditions, contributing to increased response latency, longer task completion time, inefficient utilization, and service-level agreement (SLA) violations. This paper presents a simulation-oriented, multi-objective framework for cloud resource allocation and compares three population-based meta-heuristic algorithms: Teaching-Learning-Based Optimization (TLBO), Particle Swarm Optimization (PSO), and Genetic Algorithm (GA). The framework considers task execution time, makespan, response latency, resource utilization, throughput, and SLA compliance. Cloud Sim Plus is selected as the simulation platform because it supports repeatable modelling of data centres, hosts, VMs, and workloads without the cost and variability of production deployment. Reported evaluation results indicate that a proposed meta-heuristic scheduler reduced average response latency from 420 ms to 275 ms (approximately 34.5%), reduced makespan from 1,180 s to 790 s, increased throughput from 68 to 92 tasks per minute, improved CPU utilization from 63% to 84%, improved memory utilization from 58% to 81%, and reduced SLA violations from 14.2% to 5.6%. In the comparative algorithm analysis supplied for this study, TLBO showed the strongest overall balance, including approximately 92% CPU utilization and lower makespan than PSO and GA. These findings are promising but should be interpreted as simulation-based results; further statistical reporting and real-cloud validation are required before generalization.

Keywords: Cloud computing, Resource allocation, Task scheduling, Meta-heuristic Optimization, TLBO, Particle swarm optimization, Genetic algorithm, Latency, SLA.

1. Introduction

Cloud computing has become a widely used model for delivering computing, storage, networking, and application services on demand. Virtualization and elastic provisioning allow providers to share infrastructure among multiple users and adjust capacity as requirements change. These properties are valuable for data analytics, Internet of Things (IoT) applications, machine learning, and enterprise services. At the same time, they make resource management a dynamic decision problem: incoming tasks vary in size and urgency, virtual machines are heterogeneous, and resource availability can change during execution.

Latency is an important quality-of-service measure because it affects application responsiveness and user experience. Delays may also contribute to missed service-level agreement (SLA) targets, inefficient resource use, and increased operating costs. Latency is influenced not only by task length but also by queueing, placement decisions, VM capacity, workload bursts, and contention for CPU, memory, storage, and network bandwidth. A scheduler that performs well for a stable workload may perform poorly when demand changes rapidly.

Conventional rule-based and static scheduling methods are attractive because they are relatively simple and inexpensive to execute. However, their fixed decision rules may not explore the large solution space associated with heterogeneous cloud infrastructure. Exact optimization can also become computationally expensive as the number of tasks, VMs, and constraints increases. Meta-heuristic methods offer an alternative by searching for good solutions through population-based exploration and iterative improvement. They do not guarantee a global optimum in every run, but they can produce useful allocations for complex optimization problems within practical computational limits.

This study focuses on three meta-heuristics: Teaching-Learning-Based Optimization (TLBO), Particle Swarm Optimization (PSO), and Genetic Algorithm (GA). TLBO updates candidate solutions through teacher and learner phases and does not require the same set of algorithm-specific control parameters as many population-based methods. PSO updates particles using personal and global best positions, which often supports rapid early convergence. GA uses selection, crossover, and mutation to evolve candidate schedules and maintain solution diversity. Comparing these methods provides a basis for selecting an algorithm according to the workload and operating priorities.

The study has four objectives: [1] formulate cloud resource allocation as a multi-objective optimization problem; [2] compare TLBO, PSO, and GA in a simulated cloud environment; [3] examine reported changes in latency, makespan, throughput, utilization, and SLA violations; and [4,5] identify practical limitations and future extensions. The contribution is a structured comparison that links algorithm behaviour to operational cloud metrics, while recognizing that simulation findings need validation against reproducible experiment logs and real deployments.

2. Literature Review

2.1 Cloud simulation and resource management

Simulation is a useful method for evaluating cloud scheduling policies because it enables controlled experiments over different host, VM, and workload configurations. Calheiros et al. introduced Cloud Sim as a toolkit for modelling cloud infrastructure and evaluating provisioning algorithms, including data centres, virtual machines, and application workloads [1]. Cloud Sim Plus builds on this simulation tradition with a modular, extensible Java framework intended to support experimentation with cloud infrastructure and services [2]. Simulation makes it possible to compare schedulers under consistent assumptions, repeat scenarios with different random seeds, and explore stress conditions without risking a production environment [6,7].

Nevertheless, a simulator is an abstraction of a real platform. Actual latency and cost may be affected by network geography, noisy neighbours, hardware variation, provider-specific policies, and background traffic [8]. Consequently, simulation can support algorithmic comparison, but it cannot by itself establish that the same gains will occur in a production cloud [9].

2.2 Meta-heuristic optimization for allocation

Meta-heuristics are commonly used when the search space is large, nonlinear, or constrained. In cloud scheduling, a candidate solution can represent an assignment of tasks to VMs, a VM selection policy, or a joint placement and provisioning decision [10,11]. A fitness function evaluates the quality of a candidate schedule [27,28]. Because cloud objectives may conflict, a scheduler may need to balance fast completion against high utilization, lower cost, reduced energy consumption, and SLA compliance.

GA is inspired by biological evolution. Candidate schedules are encoded as chromosomes and improved through selection, crossover, and mutation. Its population-based search can preserve

diversity and explore alternative allocations, but its performance depends on choices such as population size and genetic operator rates [12]. In large search spaces, this configuration burden may increase experimentation and runtime [13].

PSO models collective movement in a swarm. Each particle represents a candidate solution and is influenced by its own best-known position and the best position identified by the swarm [24]. PSO is relatively straightforward and can converge quickly in early iterations. However, when diversity declines too early, the swarm may stagnate around a suboptimal region [14]. Parameter settings and workload changes can therefore affect its robustness.

TLBO models learning in a classroom and uses two phases. In the teacher phase, the best candidate guides the population toward improved solutions; in the learner phase, candidate solutions learn through pairwise interaction. The original TLBO literature describes this mechanism as a population-based optimization method with comparatively limited algorithm-specific parameter tuning [15]. This feature may be useful when a scheduler must adapt without frequent manual retuning. However, parameter-light design does not remove the need to define population size, stopping criteria, encoding, constraints, or the fitness function [25,26].

2.3 Related applications and research gap

The source material supplied for this study also summarizes TLBO applications in engineering design, power-system congestion management, wind-turbine control, fault prediction, and network configuration. These studies are not direct evidence of cloud-scheduling performance, but they illustrate the portability of TLBO-style search across complex optimization tasks [16]. The supplied literature further notes that hybrid TLBO variants and combinations with neural networks or other algorithms may improve problem-specific performance. The bibliographic details for some of these studies were incomplete in the source document; they should be checked against the original publications before being cited in a submitted manuscript [17].

For cloud resource allocation, the practical research gap is not simply to show that one optimizer can find a good schedule [18]. A useful evaluation should compare algorithms under common workloads, report several service and resource metrics, explain convergence behaviour, and acknowledge the computational cost of optimization itself [19]. It should also distinguish between a general meta-heuristic framework and the particular algorithm that performs best in a given experiment. This paper addresses that gap through a multi-metric comparison of TLBO, PSO, and GA, complemented by baseline-versus-proposed results reported for the optimized scheduler [20].

3. Methodology

3.1 Research design and simulation platform

The study adopts a comparative simulation design. Cloud Sim Plus is used as the intended testbed because its Java-based, object-oriented structure supports repeatable modelling of data centres, hosts, VMs, and cloudlets (tasks) [21]. The simulator allows scheduling logic to be changed while retaining a consistent infrastructure model. It also permits workload and configuration variation without the expense and operational risk of testing an experimental scheduler directly on a public cloud.

The supplied methodology describes a multi-tenant data-centre model with interconnected hosts, VMs, and cloudlets. The broader project configuration previously specified by the authors consists of one data centre, 20 physical hosts, four cores per host, 16 GB RAM and 1 TB storage per host, and

1,000 Mbps bandwidth. The VM pool contains 50 VMs with 1,000–2,500 MIPS processing capacity, 2–4 GB memory, and 50–100 GB storage. The workload comprises 500 cloudlets in the reference setup, with additional scenarios described as varying from 500 to 2,000 tasks and 50 to 200 VMs. The final manuscript should state clearly which configuration produced each result and should include the Cloud Sim Plus version, host/VM scheduling policies, task-length distribution, and number of independent runs.

3.2 Problem formulation

Let $T = \{t_1, t_2, \dots, t_n\}$ be a set of tasks and $V = \{v_1, v_2, \dots, v_m\}$ a set of available VMs. The scheduler assigns each task to a feasible VM while respecting processing capacity and resource constraints. The allocation problem is multi-objective because improving one measure can worsen another. The principal objectives considered are:

Minimize makespan: $M = \max$ over all VMs of the completion time of tasks assigned to that VM.

Minimize response latency and task execution time: reduce the time between task submission and response/completion, using a consistent definition across all compared schedulers.

Maximize resource utilization and throughput: increase the proportion of available CPU and memory resources productively used and the number of completed tasks per unit time.

Minimize SLA violations: reduce the proportion of tasks or service intervals that breach the defined service-level constraints.

A general fitness function can be represented as $F(x) = w_1 \cdot L(x) + w_2 \cdot M(x) + w_3 \cdot SLA(x) - w_4 \cdot U(x) - w_5 \cdot TP(x)$, where x is a candidate allocation, L is latency, M is makespan, SLA is the violation measure, U is resource utilization, TP is throughput, and w_i are non-negative weights. Each metric must be normalized before combination because the measures have different units. Alternatively, Pareto-based selection can preserve non-dominated solutions without reducing all objectives to one weighted score. The supplied material mentions Pareto-front analysis of makespan, cost per task, and energy use, but does not provide numerical Pareto-front values.

3.3 Algorithm workflow

The common workflow begins by generating a population of feasible candidate allocations. Each candidate is evaluated using the same workload, resource constraints, and objective metrics. The optimizer then updates candidate solutions according to its search mechanism: teacher and learner phases for TLBO; velocity and position updates for PSO; and selection, crossover, and mutation for GA. Infeasible assignments are repaired or penalized consistently. The process continues until a fixed iteration limit, a time budget, or a convergence criterion is reached [22]. The best feasible candidate is then passed to the scheduling layer.

For a fair comparison, all three algorithms should use comparable population sizes, stopping budgets, objective definitions, and workload traces. Random seeds should be recorded and each scenario repeated multiple times [23]. Results should be reported using mean and standard deviation, and paired statistical tests or confidence intervals should be considered where the sample size permits. The source material indicates repeated runs with varying seeds and configurations, but does not provide the run count or inferential statistics; therefore, this paper describes the design intent without claiming formal statistical significance.

3.4 Performance metrics

The evaluation uses response latency (milliseconds), makespan (seconds), throughput (tasks per minute), CPU utilization (percentage), memory utilization (percentage), and SLA violation rate (percentage). Makespan captures the time needed to complete the workload, whereas response latency captures service responsiveness. Throughput measures completed work over time. Utilization indicates how effectively available resources are used, but high utilization alone is not sufficient if it causes queueing or SLA breaches. SLA violation rate is therefore interpreted alongside latency and utilization rather than as an isolated measure.

4. Comparative Analysis of TLBO, PSO, and GA

The algorithms differ in their search behaviour, configuration needs, and likely suitability for particular cloud conditions. Table 1 summarizes the qualitative comparison reported in the supplied material. These descriptions are comparative assessments, not universal guarantees; outcomes can change with encoding, workload, fitness function, population size, and stopping budget.

Table 1: Comparative analysis of different Meta-Heuristic algorithm

Criterion	TLBO	PSO	GA
Core mechanism	Teacher and learner phases	Particle movement guided by personal/global best	Selection, crossover, mutation
Convergence behavior	Stable, consistent improvement reported	Fast early convergence; may plateau	Often slower; occasional late improvements
Parameter burden	Relatively low algorithm-specific tuning	Requires tuning of swarm parameters	Requires tuning of multiple genetic operators
Local-optimum behavior	Reported as strong in supplied comparison	May stagnate if diversity falls	Mutation/crossover can restore diversity
Reported CPU utilization	~92%	~86%	~82%
Reported makespan	Lowest; ~8.5% below PSO and ~12.3% below GA	Higher than TLBO	Highest of the three in supplied comparison
Potential fit	Balanced, dynamic multi-objective allocation	Burst or time-sensitive decisions	Exploratory search and hybrid strategies

TLBO is the strongest balanced candidate in the reported comparison. Its low tuning burden can simplify experimentation and reduce the effort required to configure the optimizer for changing workloads. The source material reports the lowest makespan among the three algorithms and CPU utilization of approximately 92%, compared with 86% for PSO and 82% for GA. These findings suggest that TLBO distributed the evaluated workload more evenly in the tested scenario.

PSO remains useful when early convergence is important. The supplied convergence analysis reports rapid improvement during roughly the first 20 iterations, followed by a plateau. That behaviour may be suitable when a scheduler must produce a quick near-optimal decision, although premature convergence can limit later improvements. GA provides a different balance: its evolutionary operators can explore diverse assignments, but the supplied analysis reports slower convergence and greater

computational overhead. This may be acceptable when diversity and hybridization are more important than immediate decision speed.

The comparison should not be interpreted as proof that TLBO dominates PSO and GA under every cloud workload. A robust selection policy should consider the optimization budget, arrival pattern, task heterogeneity, resource constraints, and operational priorities. For example, a hybrid system could use PSO to produce a fast initial schedule and apply TLBO or GA refinement when the available decision time permits. Such a design is a future-work hypothesis and has not been established by the results reported here.

5. Result and Discussion

5.1 Baseline versus proposed framework

The reported baseline comparison evaluates a conventional heuristic scheduler against the proposed meta-heuristic optimization framework in table 2. The workload is described as dynamic and heterogeneous, with scenarios spanning 50–200 VMs and 500–2,000 tasks. The summary values below are reproduced from the supplied project results and are not independently recalculated from raw simulation logs.

Table 2: Comparison Metric

Metric	Baseline	Proposed	Reported change
Average response latency	420 ms	275 ms	34.5% lower
Makespan	1,180 s	790 s	33.1% lower
Throughput	68 tasks/min	92 tasks/min	35.3% higher
CPU utilization	63%	84%	+21 percentage points
Memory utilization	58%	81%	+23 percentage points
SLA violation rate	14.2%	5.6%	60.6% lower

Response latency decreased from 420 ms to 275 ms, a reduction of 145 ms or approximately 34.5%. This is the most direct indication that the proposed allocation strategy improved responsiveness in the tested simulation. Makespan decreased by 390 seconds, from 1,180 seconds to 790 seconds, corresponding to a reduction of approximately 33.1%. Lower makespan suggests that tasks were distributed in a way that reduced the overall completion time of the workload.

Throughput increased from 68 to 92 tasks per minute. Based on the reported values, this is an increase of approximately 35.3%. At the same time, CPU utilization rose from 63% to 84%, while memory utilization rose from 58% to 81%. The combined movement in throughput and utilization is consistent with better use of the available resources in the simulated workload. However, utilization is not inherently beneficial at every level; a production system would need to ensure that high utilization does not create queueing delays or reduce headroom for sudden demand.

The reported SLA violation rate fell from 14.2% to 5.6%, a reduction of 8.6 percentage points, or approximately 60.6% relative to the baseline. This suggests improved service reliability under the evaluation conditions. The source material also reports lower performance variance under peak loads, but numerical variance estimates are not supplied. Accordingly, the stability observation should be treated as qualitative until supported by repeated-run distributions, standard deviations, confidence intervals, and clearly defined peak-load scenarios.

5.2 Interpretation of comparative algorithm results

The separate TLBO–PSO–GA comparison reports that TLBO reduced makespan by approximately 8.5% compared with PSO and 12.3% compared with GA in a 500-task workload. It also reports CPU utilization of approximately 92% for TLBO, 86% for PSO, and 82% for GA. The convergence discussion describes TLBO as stable, PSO as fast in early iterations but prone to a plateau, and GA as slower with occasional later improvements. These algorithm-specific results complement the baseline comparison, but the two result sets should not be conflated: the 63% to 84% CPU-utilization values are baseline-versus-proposed framework figures, whereas 92%, 86%, and 82% are the reported values for the separate TLBO, PSO, and GA comparison.

Taken together, the reported outcomes support the potential of meta-heuristic scheduling to improve multiple performance indicators at once. The results also show why multi-objective evaluation is important: a scheduler should not be selected solely because it minimizes makespan or maximizes CPU utilization. Latency, throughput, SLA compliance, memory utilization, and the runtime overhead of the optimization algorithm all influence practical usefulness.

5.3 Limitations and validity considerations

Several limitations must be addressed before journal submission. First, the manuscript is based on summarized simulation outcomes rather than raw result files, so exact reproducibility cannot yet be verified. Second, details such as Cloud Sim Plus version, workload distributions, task lengths, objective weights, population sizes, iteration budgets, and number of repetitions need to be documented. Third, energy and cost are discussed as optimization objectives, but numerical values are not available in the supplied results; no quantitative conclusion about those metrics is therefore drawn here. Fourth, a simulation cannot fully reproduce network variability, shared tenancy, provider billing, or production scheduling constraints. Finally, the supplied literature review contains incomplete bibliographic information for some cited TLBO applications, and these references should be verified before publication.

Future experiments should publish the configuration files, workload traces where licensing permits, random seeds, per-run results, and analysis scripts. Reporting mean, standard deviation, confidence intervals, and suitable significance tests would strengthen the evidence. Tests on a real cloud or edge-cloud environment would help establish whether the simulated improvements persist under operational conditions.

6. Conclusion

This paper presented a multi-objective meta-heuristic framework for cloud resource allocation and compared TLBO, PSO, and GA using simulation-oriented evidence. The framework targets response latency, makespan, throughput, resource utilization, and SLA compliance under dynamic workloads and heterogeneous virtual machines. The reported baseline comparison shows response latency decreasing from 420 ms to 275 ms, makespan decreasing from 1,180 seconds to 790 seconds, throughput increasing from 68 to 92 tasks per minute, CPU utilization increasing from 63% to 84%, memory utilization increasing from 58% to 81%, and SLA violations decreasing from 14.2% to 5.6%.

In the separate algorithm comparison supplied for this study, TLBO exhibited the most balanced results, including lower makespan and higher CPU utilization than PSO and GA. PSO showed an advantage in early convergence, while GA offered flexible evolutionary exploration. These observations indicate that TLBO is a promising candidate for adaptive cloud scheduling, but they do

not establish universal superiority. The findings remain simulation-based and should be validated using complete experimental logs, consistent statistical analysis, and real-world deployment.

Future work should explore hybrid TLBO–PSO or TLBO–GA models, workload prediction using machine learning, self-tuning feedback loops, and explicit optimization of energy consumption, cost, and sustainability. GPU-accelerated implementations and integration with platforms such as Kubernetes or OpenStack could also reduce decision overhead and improve practical applicability. Such extensions may contribute to more autonomous, efficient, and SLA-aware cloud infrastructure management.

References

- [1] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011. doi: 10.1002/spe.995.
- [2] M. C. Silva Filho, R. L. Oliveira, C. C. Monteiro, P. R. M. Inácio, and M. M. Freire, "CloudSim Plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness," in *Proceedings of the IFIP/IEEE Symposium on Integrated Network Management*, 2017. doi: 10.23919/INM.2017.7987304.
- [3] R. V. Rao, V. J. Savsani, and D. P. Vakharia, "Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems," *Computer-Aided Design*, vol. 43, no. 3, pp. 303–315, 2011. doi: 10.1016/j.cad.2010.12.015.
- [4] R. V. Rao, V. J. Savsani, and J. Balic, "Teaching–learning-based optimization algorithm for unconstrained and constrained real-parameter optimization problems," *Engineering Optimization*, vol. 44, no. 12, pp. 1447–1462, 2012. doi: 10.1080/0305215X.2011.652103.
- [5] S. Bhattacharya, B. R. Kuanr, A. Routray, and A. Dash, "Transmission congestion management in restructured power system by rescheduling of generators using TLBO," in *2017 IEEE International Conference on Electrical, Instrumentation and Communication Engineering*, 2017, pp. 1–7. doi: 10.1109/ICEICE.2017.8191937.
- [6] E. Roshandel, S. M. Gheasaryan, and M. Mohamadi, "A control strategy for DFIG wind turbine using SLS-TLBO and fuzzy logic," in *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation*, 2017, pp. 108–113. doi: 10.1109/KBEI.2017.8324953.
- [7] M. Bisi, "An ANN-TLBO model to predict cumulative number of failures in software," in *2020 IEEE-HYDCON*, 2020, pp. 1–4. doi: 10.1109/HYDCON48903.2020.9242777.
- [8] M. Hamzeh, B. Vahidi, and A. Foroughi Nematollahi, "Optimizing configuration of cyber network considering graph theory structure and teaching–learning-based optimization (GT-TLBO)," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2083–2090, 2019. doi: 10.1109/TII.2018.2860984.
- [9] M. Armbrust et al., "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010. doi: 10.1145/1721654.1721672.
- [10] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009. doi: 10.1016/j.future.2008.12.001.
- [11] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011. doi: 10.1002/spe.995.
- [12] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: State-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, pp. 7–18, 2010. doi: 10.1007/s13174-010-0007-6.
- [13] M. Kalra and S. Singh, "A review of metaheuristic scheduling techniques in cloud computing," *Egyptian Informatics Journal*, vol. 16, no. 3, pp. 275–295, 2015. doi: 10.1016/j.eij.2015.07.001.

- [14] E. H. Houssein, A. G. Gad, Y. M. Wazery, and P. N. Suganthan, "Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends," *Swarm and Evolutionary Computation*, vol. 62, Art. no. 100841, 2021. doi: 10.1016/j.swevo.2021.100841.
- [15] M. N. Aktan and H. Bulut, "Metaheuristic task scheduling algorithms for cloud computing environments," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 9, Art. no. e6513, 2022. doi: 10.1002/cpe.6513.
- [16] "A survey on PSO based meta-heuristic scheduling mechanism in cloud computing environment," *Journal of King Saud University – Computer and Information Sciences*, 2021. doi: 10.1016/j.jksuci.2021.01.003.
- [17] "Heuristic initialization of PSO task scheduling algorithm in cloud computing," *Journal of King Saud University – Computer and Information Sciences*, vol. 34, no. 6, Part A, pp. 2370–2382, 2022. doi: 10.1016/j.jksuci.2020.11.002.
- [18] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications*, pp. 400–407, 2010. doi: 10.1109/AINA.2010.31.
- [19] R. V. Rao, V. J. Savsani, and D. P. Vakharia, "Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems," *Computer-Aided Design*, vol. 43, no. 3, pp. 303–315, 2011. doi: 10.1016/j.cad.2010.12.015.
- [20] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, vol. 4, pp. 1942–1948, 1995. doi: 10.1109/ICNN.1995.488968.
- [21] M. Dorigo, V. Maniezzo, and A. Coloni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 26, no. 1, pp. 29–41, 1996. doi: 10.1109/3477.484436.
- [22] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in Engineering Software*, vol. 69, pp. 46–61, 2014. doi: 10.1016/j.advengsoft.2013.12.007.
- [23] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012. doi: 10.1016/j.future.2011.04.017.
- [24] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002. doi: 10.1109/71.993206.
- [25] M. A. A. Al-Arasi and A. Saif, "Task scheduling in cloud computing based on metaheuristic techniques: A review paper," *EAI Endorsed Transactions on Cloud Systems*, vol. 6, no. 17, 2020. doi: 10.4108/eai.13-7-2018.162829.
- [26] A. Jain and A. Upadhyay, "Cloud scheduling using meta heuristic algorithms," *International Journal of Computer Sciences and Engineering*, vol. 5, no. 10, 2017. doi: 10.26438/ijcse/v5i10.132139.
- [27] "Towards metaheuristic scheduling techniques in cloud and fog: An extensive taxonomic review," *ACM Computing Surveys*, vol. 55, no. 3, 2023. doi: 10.1145/3494520.
- [28] "Multiobjective task scheduling in cloud computing using hybrid algorithm (HRLM)," *Simulation Modelling Practice and Theory*, vol. 150, Art. no. 103305, 2026. doi: 10.1016/j.simpat.2026.103305.