Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

Adaptive Spotted Hyena Optimizer for Latency-Aware Task Scheduling in Heterogeneous Multicore Systems

[1]Igiri C. G., [2]Ejekwu Obunezi, [3]Ujah Alechenu Israel

[1,2,3]Computer Science Department, Rivers State University, Nigeria
E-mail: [1]Igiri.chima@ust.edu.ng, [2]obunezi.ejekwu@ust.edu.ng, [3]israelujah1709@gmail.com

Abstract: Heterogeneous multicore architectures encompassing Central Processing Units (CPUs), Graphics Processing Units (GPUs), and Field-Programmable Gate Arrays (FPGAs) have emerged as the dominant computational paradigm for high-performance and embedded workloads. Task scheduling across such architectures presents a formidable challenge: the assignment of computational tasks to heterogeneous processing elements must satisfy precedence constraints whilst minimising overall workflow completion latency. Classical scheduling heuristics, including Earliest Finish Time (EFT) and Heterogeneous Earliest Finish Time (HEFT), offer polynomial-time approximations yet demonstrate limited adaptability under dynamic runtime conditions, frequently yielding suboptimal execution latency in large-scale task graphs. This paper proposes the Latency-Aware Adaptive Spotted Hyena Optimizer (LA-ASHO), a novel metaheuristic scheduling framework grounded in the social hunting behaviour of spotted hyenas. The framework encodes task-to-core assignments as discrete permutation vectors and evaluates fitness exclusively through a mathematically rigorous execution latency model that integrates computation time, memory overhead, and inter-core communication delays. Comprehensive simulation experiments conducted over 100 to 1,000 tasks across 8 to 64 heterogeneous cores, each repeated across 30 statistically independent runs demonstrate that LA-ASHO achieves statistically significant reductions in workflow completion latency relative to established baseline schedulers such as; Min-Min, Heterogeneous Earliest Finish Time (HEFT). The principal contribution of this work is the formulation of an execution-latency-focused metaheuristic framework that is both theoretically grounded and practically scalable for real-world heterogeneous computing deployments.

Keywords: Adaptive Scheduling, Directed Acyclic Graph, Execution Latency, Heterogeneous Multicore Scheduling, Metaheuristic Optimisation, Spotted Hyena Optimizer, Swarm Intelligence.

I. INTRODUCTION

The rapid proliferation of heterogeneous multicore processors over the preceding decade has fundamentally transformed the architecture of modern computing systems. Contemporary platforms routinely integrate diverse processing elements including general-purpose Central Processing Unit (CPU) cores operating at variable frequencies, massively parallel Graphic Processing Unit (GPU) architectures, and reconfigurable Field-Programmable Gate Array (FPGA) fabrics within a single unified system [5]. Such architectural heterogeneity offers extraordinary computational throughput, yet simultaneously introduces pronounced complexity in task scheduling, since no single processing element is uniformly optimal for every class of workload.

Execution latency the elapsed time from the initiation of a task graph to the completion of its final dependent task constitutes the preeminent performance criterion in latency-sensitive domains, encompassing autonomous vehicular systems, real-time signal processing, medical image reconstruction, and edge inference for deep neural networks [7]. In these domains, even marginal increments in scheduling latency may precipitate catastrophic system failures, safety violations, or significant economic consequences. Consequently, the minimisation of workflow completion latency represents a research imperative of the highest practical significance.

Classical scheduling algorithms, such as Heterogeneous Earliest Finish Time (HEFT) [5] and Earliest Deadline First (EDF), employ priority-based heuristics that produce efficient schedules in polynomial time. However, these approaches are inherently static, computing a single schedule prior to execution and lacking mechanisms to respond to runtime perturbations including workload spikes, processor stalls, and variable communication bandwidth. Furthermore, these heuristics

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

optimise surrogate metrics that may not faithfully correlate with actual execution latency under dynamic conditions [19]. The solution search space grows exponentially with the number of tasks and processors, rendering exhaustive search computationally intractable for problem instances of practical relevance. Metaheuristic optimisation algorithms including Particle Swarm Optimisation (PSO), Grey Wolf Optimiser (GWO), Spotted Hyena Optimisation (SHO), and Whale Optimisation Algorithm (WOA) offer a principled balance between exploration of the search space and exploitation of promising candidate solutions, making them particularly well-suited to this scheduling domain [10].

The Spotted Hyena Optimizer (SHO), introduced by [4], draws inspiration from the cooperative hunting strategies of spotted hyenas (Crocuta Crocuta). The algorithm models the collective encircling, pursuit, and attack behaviours exhibited by hyena clans to converge upon optimal prey by analogy, optimal solutions within a high-dimensional search space. Despite its strong theoretical convergence properties and competitive empirical performance in continuous domains, the SHO has received limited investigation in the context of discrete, latency-aware task scheduling for heterogeneous multicore systems.

Contemporary heterogeneous multicore systems integrating CPUs, GPUs, and FPGAs within a unified platform present a fundamental and unresolved challenge in task scheduling: efficiently mapping a workflow of interdependent computational tasks onto dissimilar processing elements in a manner that minimises end-to-end execution latency. This challenge is compounded by three interrelated problems:

(i) NP-Hardness of Heterogeneous Task Scheduling, (ii) Latency Dilution in Multi-Objective Metaheuristics, and (iii) Lack of Adaptive Runtime Re-Scheduling.

Taken together, these three problems define a clear and practically significant research gap: there is no existing scheduling framework that (i) focuses exclusively on execution latency minimisation as a singular optimisation objective, (ii) employs a biologically inspired swarm intelligence algorithm adapted for discrete task-to-core assignment, and (iii) incorporates an adaptive runtime re-scheduling mechanism capable of responding to observed latency violations in heterogeneous multicore environments. This paper addresses all three dimensions of this gap through the proposed LA-ASHO framework.

The aim of this research is to design, implement, and evaluate a Latency-Aware Adaptive Spotted Hyena Optimizer (LA-ASHO) scheduling framework that minimises workflow execution latency in heterogeneous multicore computing systems through the exclusive application of a biologically inspired, discretely adapted metaheuristic algorithm augmented by an adaptive runtime re-scheduling mechanism. The objectives include: To adapt the continuous Spotted Hyena Optimizer to the discrete task scheduling domain, to design and implement an adaptive runtime re-scheduling architecture comprising a monitoring module and to evaluate the computational complexity of the proposed LA-ASHO framework analytically.

The remainder of this paper is organised as follows: Section 2.0: Related Works, Section 3.0: Methodology and Design, Section4.0: Experimental Results and Discussion, Section5.0: Conclusion.

II. RELATED WORKS

Heterogeneous task scheduling has been extensively studied within the context of workflow management in cloud and high-performance computing (HPC) environments. [15] Established the baseline for priority-based static heuristics, assigning tasks to processors that minimise earliest finish time using upward rank prioritisation.

Subsequent list-scheduling variants including Predict Earliest Finish Time (PEFT) and Look-Ahead Scheduling (LAS) refined this approach to reduce scheduling overhead [2]. More recently, [5] examined task scheduling within heterogeneous edge-cloud continuum architectures, demonstrating that classical heuristics suffer significant latency degradation as task graph complexity increases. [16] Proposed a dynamic partitioning scheme for CPU-GPU co-execution that reduces data transfer overhead and improves completion time, though their approach remains confined to dual-processor environments. [19] Introduced a graph neural network-based scheduling policy that learns latency-sensitive task assignments from execution traces, achieving notable latency reductions on scientific workflows, though at the expense of substantial training overhead.

Metaheuristic algorithms have been widely applied to scheduling problems owing to their capacity to navigate large, non-convex search spaces without requiring gradient information. Particle Swarm Optimisation (PSO) was among the first swarm-intelligence methods applied to heterogeneous scheduling[9] demonstrated that velocity-adapted PSO achieves competitive task completion times on benchmark DAG workflows when compared with HEFT and genetic algorithms,

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

particularly under high task-dependency ratios.

The Grey Wolf Optimizer (GWO), introduced by [10], has been successfully adapted for discrete scheduling by [13], who employed a permutation-based encoding to map continuous GWO positions to task orderings. Their results indicated marked improvement over PSO in terms of scheduling stability across repeated runs, attributable to GWO's structured leadership hierarchy. The Whale Optimisation Algorithm (WOA) has similarly been applied to cloud task scheduling by [7], with a modified bubble-net attacking phase adapted for discrete processor assignments.

Hybrid metaheuristic approaches have gained traction as a means of reconciling exploration breadth with exploitation depth. [1] Proposed an Aquila Optimizer hybridised with simulated annealing for DAG scheduling, demonstrating improved convergence speed compared to either method in isolation. [18] Combined reinforcement learning reward signals with a genetic algorithm mutation operator to dynamically adapt crossover probability during scheduling optimisation, yielding reduced makespan on heterogeneous cloud benchmarks.

Dedicated latency-aware scheduling models have emerged in response to the inadequacy of generic makespan-minimisation frameworks in latency-critical applications. [3] Proposed a latency-constrained scheduling framework for real-time video analytics pipelines on GPU clusters, incorporating communication latency as a first-class scheduling constraint. Their formulation explicitly models transmission delays between heterogeneous nodes, demonstrating that ignoring inter-core latency contributes up to 34% excess completion time in communication-intensive workloads.

[8] Introduced a deadline-aware workflow scheduler for edge computing deployments, formulating task placement as a constrained binary integer programme with execution time bounds per task. Whilst their approach provides formal latency guarantees, it incurs exponential worst-case solve time, limiting applicability to small task graphs. [12] addressed real-time scheduling in automotive embedded systems, proposing a priority-assignment mechanism that accounts for processor-specific execution time variance a key consideration in heterogeneous multicore contexts.

Swarm intelligence algorithms modelling collective animal behaviours have demonstrated strong performance across combinatorial optimisation problems. The Spotted Hyena Optimizer (SHO), proposed by [4], has been applied to engineering design optimisation, feature selection, and power system scheduling. Recent studies by [6] and [14] have confirmed SHO's competitive convergence behaviour relative to PSO, GWO, and WOA on standard benchmark functions, attributed to its multi-agent encircling mechanism that maintains population diversity while converging to near-optimal solutions.

Reinforcement learning (RL)-based scheduling has attracted considerable attention as a data-driven alternative to handcrafted heuristics. [17] Proposed a deep Q-network schedule for heterogeneous cloud environments, achieving reduced average job completion time on trace-driven simulations. However, RL schedulers require extensive offline training on representative workload distributions and exhibit limited generalisation to out-of-distribution task graphs a significant limitation for systems with evolving workloads.

The foregoing review reveals a pervasive tendency in the scheduling literature to formulate multi-objective optimisation problems that simultaneously target makespan, energy consumption, resource utilisation, and load balance. Whilst such formulations are theoretically comprehensive, they dilute the optimisation pressure applied to execution latency specifically the single most operationally critical metric in real-time and latency-sensitive applications. Furthermore, existing SHO-based schedulers have not been systematically adapted for discrete, latency-exclusive task scheduling in heterogeneous multicore environments. This paper addresses this precise gap by proposing LA-ASHO: a framework that focuses exclusively on execution latency minimisation, adapts the SHO for discrete task scheduling, and incorporates adaptive runtime re-scheduling in response to dynamic latency violations.

III. METHODOLOGY AND DESIGN

This chapter describes the methodology and design framework used in this study. It presents the system assumptions, develops the mathematical model for execution latency, and explains the adaptation of the Spotted Hyena Optimizer to the task scheduling problem. The proposed LA-ASHO algorithm, along with its adaptive runtime architecture and overall system design, is also detailed to demonstrate how latency-aware scheduling is achieved in heterogeneous multicore systems.

3.1 Model Assumptions

The mathematical models and algorithmic mechanisms presented rest on a set of explicit assumptions about the computing environment, the structure of the workload, and the

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

behaviour of the scheduling system. These assumptions are stated here to delimit the scope of the model and to provide the boundary conditions under which the theoretical analysis and experimental results are valid.

A1 - Fixed processor set: The platform comprises a fixed, finite set of m heterogeneous processors $P = \{p_1, p_2, ..., p_m\}$ whose number and identities do not change during the execution of a single workflow instance.

A2 - Known, static processor frequencies: Each processor p_j operates at a known computational frequency f_j that is fixed for the duration of the scheduling horizon.

A3 - Fully connected, homogeneous interconnect: All processors are interconnected by a shared network with a single aggregate bandwidth B and a single base network latency L. This communication cost between any two processors is therefore identical and determined solely by the volume of data transferred.

A4 - Negligible intra-processor communication: The communication latency $T\mathcal{A}_{omm}(i, k)$ is therefore set to zero whenever $\sigma(v_i) = \sigma(v_k)$.

A5 - DAG workflow representation: The computational workflow is represented as a Directed Acyclic Graph G = (V, E), where each node $v_i \in V$ corresponds to an indivisible computational task and each directed edge $(v_i, v_k) \in E$ encodes a data dependency requiring that v_k cannot begin until v_i has completed and transferred its output data.

A6 - Known task workloads and data volumes: The computational workload W_i of each task (in processor cycles) and the data volume $D_{i,k}$ of each dependency edge (in megabytes) are assumed to be known in advance of scheduling.

A7 - Task non-preemption: Once a task begins execution on a processor, it runs to completion without interruption.

A8 - Task independence given precedence satisfaction: Shared memory contention between co-assigned tasks is subsumed into the system overhead term δ_i.

A9 - List scheduling with topological ordering:

A10 - Single objective optimisation: The sole optimisation objective is the minimisation of total workflow makespan

$$T_{total} = \max_{f_i \in V}\{T^{f}_{\chi n g \chi}(v_i)\}.$$

A11 - Immediate data transfer upon task completion: When a task v_i completes execution, its output data is assumed to begin transferring to all successor tasks immediately, without buffering delay.

A12 - Completed tasks are not re-scheduled: When an adaptive re-scheduling event is triggered, only tasks that have not yet commenced execution are subject to re-assignment.

A13 - Bounded workload perturbations: Runtime deviations in task execution times, arising from causes such as thermal throttling, cache contention, or memory bus saturation, are assumed to be bounded.

3.2 System Architecture

The LA-ASHO system architecture is organised into five hierarchical layers, each interfacing with adjacent layers through well-defined data exchange protocols.

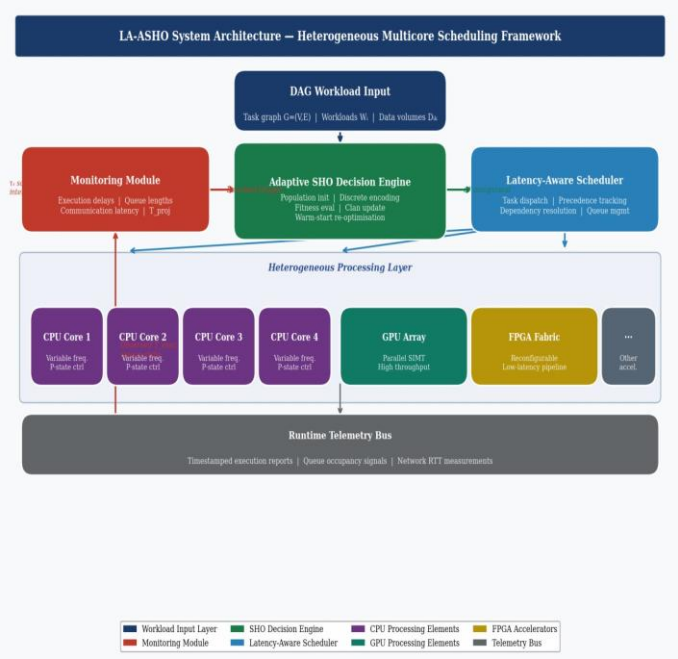

Figure 3.1: LA-ASHO System Architecture Illustrating the Five-Layer Design

Figure 3.1 illustrates the five-layer LA-ASHO system architecture, showing the data flow from workload input through the monitoring subsystem, SHO decision engine, and scheduling executor to the heterogeneous processing layer.

The Workload Input Layer accepts DAG workflow specifications in a standardised XML-based task graph format,

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

parsing task workloads Wi, inter-task data volumes Di,k, and precedence edges into an in-memory graph representation. Workflow instances may originate from scientific computing frameworks, autonomous driving pipelines, or synthetic benchmark generators.

The Monitoring Subsystem Layer instruments each processor in the heterogeneous target platform with lightweight telemetry agents. These agents report measured task execution times, queue occupancies, and network latency samples to the central monitoring data store at configurable sampling intervals. Telemetry data is timestamped, filtered for noise, and aggregated into a projected latency estimate $Tproj$ at the system level.

The SHO Decision Engine Layer implements the discrete SHO algorithm as described in Section 3.4. It maintains the population of candidate assignment vectors, evaluates fitness through the latency simulation model, and iteratively improves the best-known schedule. Critically, the fitness function applied at this layer is grounded directly in the execution latency model formalised in Section 3.2 specifically the expression $T_{exec}(i, j) = W_i/f_j + delta_i$ ensuring that every scheduling decision made by the engine is evaluated against the same mathematically rigorous latency criterion established in the theoretical model. The engine is invoked both at workflow initialisation (initial scheduling) and upon receipt of re-scheduling triggers from the monitoring layer.

The Scheduling Executor Layer receives the optimised assignment vector sigma* and manages task dispatch to the heterogeneous processing elements. It enforces precedence constraints through the dependency tracking table, handles data transfer initiation between processors, and logs observed task completion times for feedback to the monitoring subsystem.

The Heterogeneous Processing Layer comprises the physical (or simulated) compute resources: CPU cores (modelled with variable frequencies reflecting P-state and power management), GPU streaming multiprocessors (modelled with high-throughput, memory-bandwidth-constrained execution), and

FPGA accelerators (modelled with fixed-function low-latency execution for designated task classes). The architecture supports arbitrary combinations of these processor types, configured at system initialisation time.

Data flows between layers are orchestrated through a centralised message broker that decouples monitoring data production from scheduling consumption, ensuring that re-scheduling operations do not block the critical path of task execution.

3.3 Latency Mathematical Model

A computational workflow is modelled as a Directed Acyclic Graph (DAG):

$$G = (V, E)$$

where V = {v1, v2, ..., v_n} denotes the set of n computational tasks, and E ⊆ V × V denotes the set of directed precedence-dependency edges. An edge $(v_i, v_k) \in E$ implies that task v_k may not commence execution until task vi has completed and transferred its requisite output data to the processor executing vk. Each task vi is characterised by a workload W_i (in units of floating-point operations or instruction cycles), and each directed edge (v_i, v_k) carries a data volume $D_{i,k}$ representing the inter-task communication payload.

The heterogeneous processing architecture is modelled as a set $P = p_1, p_2, \ldots, p_m$ of m processors with heterogeneous computational frequencies f_j (in GHz) for processor pj. Processors may represent CPU cores, GPU streaming multiprocessors, or FPGA accelerator elements. The system is interconnected via a network with aggregate bandwidth B (in MB/s) and a base inter-core latency L (in milliseconds).

The execution latency of task vi assigned to processor p_j is defined as:

$$T_{exec}(i, j) = \frac{W_i}{f_j} + \delta_i$$

where W_i denotes the computational workload of task vi (in cycles), f_j is the operating frequency of processor pj (in cycles per second), and $delta_i$ is the combined memory access and operating system W^ischeduling overhead associated with task vi. The term $_f j$ represents the ideal processor-bound computation time, whilst delta_i captures system-level latency contributions that are processor-independent, including cache miss penalties, context-switch delays, and memory bus contention.

This model accommodates heterogeneity by assigning task-specific execution times that vary with processor class: GPU processors exhibit high parallelism for data-parallel tasks, yielding low effective $\frac{W_i}{fj}$ for vectorisable_workloads; FPGA accelerators introduce lower delta_i through dedicated hardware pipelines; CPU cores provide balanced performance across

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

irregular control-flow tasks.

When task vi is assigned to processor pj and its successor vk is assigned to a different processor $pl(j \neq l)$, an inter-core communication delay is incurred:

$$T_{comm(I,k)} = \frac{D_{i,k}}{B} + L$$

where Di,k is the volume of data (in megabytes) transferred from vi to vk, B is the inter-core communication bandwidth (in MB/s), and L is the base network latency (in milliseconds) representing synchronisation overhead and routing delay. If vi and vk are co-located on the same processor (j = l), the communication latency is assumed negligible: $Tcomm(i, k) = 0$.

Let $Tstart((vi))$ denote the time at which task vi commences execution. The earliest start time is constrained by the finish times of all predecessor tasks:

$$T_{start(Vi)} = \max_{vk \in pred(vi)} [T_{finish}(v_k) + T_{comm}(k,i)]$$

where pred(vi) denotes the set of immediate predecessor tasks of vi. The finish time of task vi is then:

$$Tfinish(vi) = Tstart(vi) + Texec(i, j)$$

The total workflow execution latency is defined as the makespan of the task graph—the time from the initiation of the first task to the completion of the last task:

$$\min Ttotal = \max_{v_i \in V}(Tfinish(vi))$$

This objective reflects the critical-path completion time of the workflow, as the overall latency is determined by the longest chain of dependent task execution and communication delays. The scheduling problem therefore reduces to finding an assignment function sigma: V -> P that minimises

$Ttotal$ subject to:

- Precedence constraints: for all $(vi, vk) \in E$, $T_{finish}((v_i)) + T_{comm}(i,k) \leq T_{start}((v_k))$;
- Processor capacity constraints: at most one task executes on each processor at any given time;
- Assignment completeness: every task $vi \in V$ is assigned to exactly one processor p_j $\in$P.

The scheduling optimisation problem is formally stated as follows. Find an assignment function σ: V → P that minimises the total workflow makespan:

$$\text{minimise } f(\sigma) = T_{total} = \max_{v_i \in V} \{ T^f_{\chi\eta\varrho\chi\chi}(v_i) \}$$

subject to the following four constraints:

C1 (Precedence Constraint): For every directed edge $(v_i, v_k) \in E$, task v_k may not commence execution until task v_i has completed and its output data has been transferred. Formally:

$$T_{staht}(v_i) + T\mathcal{A}_{omm}(i, k) \leq T_{staht}(v_k) \; \forall (v_i, v_k) \in E$$

C2 (Processor Capacity Constraint): At most one task may execute on any processor at any given instant. For all pairs of distinct tasks v_i, v_j assigned to the same processor $p_j = \sigma(v_i) = \sigma(v_j)$, their execution intervals must be non-overlapping:

$$[T_{staht}(v_i), T^f_{\chi\eta\varrho\chi\chi}(v_i)) \cap [T_{staht}(v_j), T^f_{\chi\eta\varrho\chi\chi}(v_j)) = \emptyset \; \forall v_i \neq v_j: \sigma(v_i)=\sigma(v_j)$$

C3 (Assignment Completeness Constraint): Every task in the workflow must be assigned to exactly one processor. The assignment function σ must be a total function from V to P:

$$\forall v_i \in V, \exists! \; p_j \in P : \sigma(v_i) = p_j$$

C4 (Non-Negativity Constraint): All task start and finish times must be non-negative real values, and every task must finish no earlier than it starts:

$$T_{staht}(v_i) \geq 0 \text{ and } T^f_{\chi\eta\varrho\chi\chi}(v_i) \geq T_{staht}(v_i) \; \forall v_i \in V$$

The NP-hardness of this formulation for general DAGs and heterogeneous processors justifies the application of metaheuristic optimisation in lieu of exact solvers for problem instances of practical scale.

Table 1 lists every mathematical symbol used in this paper. Each entry gives the symbol, its formal name, and a concise description of its role. Symbols are grouped by domain: SHO algorithm, workflow DAG, latency model, solution encoding, adaptive runtime architecture, and computational complexity.

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

Table 1: Notation and Symbol Reference

Symbol	Name	Description
Latency Model		
Wi	**Task Workload**	Computational workload of task v_i, in processor cycles
fj	**Processor Frequency**	Operating frequency of processor p_j, in cycles per second
δi	**System Overhead**	Memory-access and OS scheduling overhead for task v_i, in ms
Texec(i, j)	**Execution Latency**	Time to execute task v_i on processor p_j
Di,k	**Data Volume**	Data transferred from task v_i to successor v_k, in MB
B	**Bandwidth**	Inter-core communication bandwidth, in MB/s
L	**Network Latency**	Base routing and synchronisation latency, in ms
Tcomm(i, k)	**Communication Latency**	Data-transfer delay between tasks on different processors
Tstart(vi)	**Task Start Time**	Earliest time task v_i may commence, given predecessor constraints
Tfinish(vi)	**Task Finish Time**	Completion time of task v_i
Ttotal	**Total Makespan**	Workflow execution latency; the primary optimisation objective
Solution Encoding		
σ	**Assignment Vector**	Discrete task-to-processor assignment for all n tasks
σi	**Task Assignment**	Processor index assigned to task v_i; $\sigma_i \in \{1,\ldots,m\}$
σ*	**Optimal Assignment**	Best-found assignment vector output by the decision engine
f (σ)	**Fitness Function**	Scheduling quality measure; equals T_{total} under assignment σ
Runtime & Adaptive Re-scheduling		
τs	**Sampling Interval**	Period at which the monitoring module collects telemetry
Tproj	**Projected Latency**	Runtime estimate of workflow completion latency
θ	**Violation Threshold**	Latency limit; re-scheduling triggered when $T_{p^r_{o'}}{}^j > \theta$
Tbudget	**Re-scheduling Budget**	Time allocated to one adaptive rescheduling optimisation run
Complexity		
|E|	**Edge Count**	Number of DAG edges; O(n) sparse graphs, $O(n^2)$ dense graphs

3.4 Spotted Hyena Optimizer

The Spotted Hyena Optimizer (SHO) is a swarm intelligence algorithm introduced by [4] that draws its operational principles from the cooperative hunting behaviour of spotted hyenas (Crocuta Crocuta). Spotted hyenas are highly social obligate carnivores that organise into structured clans and engage in coordinated pursuit and encirclement of prey. The hunting through coordinated movement of clan members, and a final attack phase when prey ceases evasive movement. These behaviours are mathematically abstracted to model the exploration and exploitation dynamics of the optimisation algorithm.

The encircling phase models the convergence of hyenas towards the prey (the current best-known solution). The displacement vector D between the best-known solution X_best and a hyena agent at position X is computed as:

$$D = |C \cdot Xbest - X|$$

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

where C is a random coefficient vector drawn from a uniform distribution, C ∈ U(0, 2), that introduces stochastic variability in the encircling trajectory, and |·| denotes element-wise absolute value. The coefficient C enables the algorithm to avoid premature convergence by introducing randomness in the estimation of distance to prey.

The position of each hyena agent is updated at each iteration t according to:

$$X(t+1) = Xbest - A \cdot D$$

where A is an adaptive convergence coefficient that controls the balance between exploration (large |A|) and exploitation (small |A|):

$$A = 2h \bullet r1 - h$$

Here, h decreases linearly from 5 to 0 over the course of the optimisation run, and r1 ∈ U(0, 1) is a random scalar. When |A| > 1, agents are drawn away from the current best solution, promoting global exploration; when |A| < 1, agents converge towards the best solution, enacting local exploitation.

A distinguishing feature of SHO is the clan formation mechanism, in which the Nc best-performing agents (the hyena clan) collectively guide the search. The clan position is computed as the centroid of the Nc leading agents:

$$Xclan = \frac{1}{N_c} \cdot \sum_{k=1} Xk$$

where X_k denotes the position of the k-th clan member. The final position update incorporates the clan centroid to maintain population cohesion during the exploitation phase:

$$X(t+1) = Xclan - A \cdot Dclan$$

where D_clan = |C · X_clan - X|. This clan-based position update provides SHO with a more robust exploitation mechanism than single-leader algorithms such as PSO, as it averages over multiple elite solutions to reduce sensitivity to individual stochastic outliers [4].

The dynamic variation of h from 5 to 0 ensures a systematic transition from exploration to exploitation as the number of elapsed iterations increases. During early iterations (h ≈ 5), large values of |A| drive agents to explore diverse regions of the search space, reducing the risk of premature convergence to local optima. During later iterations (h ≈ 0), small |A| values focus search in the neighbourhood of the best-known solution, refining the schedule towards the global latency minimum.

The preceding subsections have established the theoretical foundations of the Spotted Hyena Optimizer: the encircling displacement vector D, the adaptive convergence coefficient A, the clan guided position update, and the h-driven transition from global exploration to local exploitation. In their native form, however, these mechanisms operate entirely within a continuous real-valued search space, producing position vectors X ∈ R^n that carry no direct meaning in the context of discrete task-to-processor assignment. Bridging this gap is the central engineering contribution of Section 3.4 onwards: each continuous SHO mechanism is systematically re-interpreted and re-operationalised as a discrete scheduling operator, such that the exploration-exploitation balance formalised above is faithfully preserved within the combinatorial assignment space defined by the latency model of Section 3.2.

3.5 Discrete SHO Adaptation

The continuous SHO position vector X ∈ R^n must be re-interpreted as a discrete task-to-core assignment in order to address the scheduling problem. Each candidate solution (hyena position) is encoded as a mapping vector:

$$\sigma = [\sigma_1, \sigma_2, \dots, \sigma_n]$$

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

where $sigmai \in \{1, 2, ..., m\}$ specifies the index of the processor to which task v_i is assigned, and n = |V| denotes the total number of tasks. The vector sigma implicitly defines a complete task-to processor assignment for the entire workflow.

Task execution ordering on each processor is determined by list scheduling with topological ordering: amongst all tasks assigned to a given processor, execution proceeds in topological order of the DAG, ensuring that all precedence constraints are respected. This decouples the assignment decision (encoded in sigma) from the ordering decision (resolved by topological sort), simplifying the search space whilst preserving schedule validity.

The continuous position components X_i ∈ R are converted to discrete processor assignments via a modular mapping:

$$\sigma i = (\lfloor |Xi| \rfloor \bmod m) + 1$$

This mapping ensures that every continuous position vector produces a valid assignment vector with processor indices in {1, ..., m}. The floor and modulo operations introduce a deterministic yet varied mapping from the continuous search space to the discrete assignment space.

To enhance local search in the discrete domain, a swap-based mutation operator is applied following each SHO position update. The swap operator selects two task indices i, j ∈ {1, ..., n} at random and exchanges their processor assignments:

$$\sigma' = \text{swap}(\sigma, i, j) : \sigma'_i = \sigma_j, \sigma'_j = \sigma_i$$

The mutated assignment sigma' is accepted if it yields a lower or equal execution latency than sigma, implementing a greedy improvement step. The swap operator explores the neighbourhood of the current assignment, enabling fine-grained latency reduction beyond what the continuous SHO update achieves. The probability of applying the swap operator at iteration t is governed by the current exploitation coefficient (1 - h/5), increasing as the algorithm converges.

The fitness of each candidate solution sigma is evaluated by computing the total workflow execution latency under the corresponding task-to-processor assignment. Given sigma, the scheduling simulator executes the following steps:

1. Compute $Texec(i, sigma_i)\ for\ all\ v_i \in V$ using the execution latency model (Section 3.1);
2. Perform a topological sort of V and process tasks in topological order;
3. For each task vi, compute $T_{start}(v_i)$ as the maximum finish time of all predecessors plus communication delays;
4. Compute $T_{finish}(vi) = T_{start}(vi) + Texec(i, sigmai)$;
5. Return the fitness value $f(sigma) = max_{vi}\ in\ VT_{finish}(vi) = T_{total}$.

The fitness evaluation is performed for every candidate solution at every iteration, with the solution yielding minimum f(sigma) retained as X_{best} across all iterations.

3.5.1 LA-ASHO Algorithm

The following section describes the complete operation of the LA-ASHO algorithm in plain English, proceeding through each stage in the order in which it executes.

Step 1: Generate an initial schedule using HEFT
Step 2: Initialise the population of N candidate solutions
Step 3: Update the convergence parameter h
Step 4: Form the hyena clan from the N, best agents
Step 5: Update each agent's position using the SHO movement equations
Step 6: Decode updated positions into discrete processor assignments
Step 7: Apply swap-based local search to each agent
Step 8: Update the global best solution

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

Step 9: Repeat Steps 3 to 8 until the iteration budget is reached
Step 10: Return the best-found schedule

3.6 Adaptive Runtime Architecture

The Monitoring Module collects runtime telemetry from each active processor at regular sampling intervals tau_s. The module maintains a continuously updated state vector comprising:

- Per-task execution delays: the observed $T_{exec}(i, j)$ measured at runtime, incorporating dynamic effects such as thermal throttling, cache contention, and memory bus saturation;
- Processor queue lengths: the number of tasks queued for execution on each processor, used to detect processor overloading;
- Inter-core communication latency: measured round-trip times for inter-processor data transfers, accounting for network congestion and bandwidth variability.

The Monitoring Module computes a running estimate of the projected workflow completion latency.

T_proj based on current execution progress. When T_proj exceeds a user-defined threshold theta (defined as a proportional overshoot of the initial scheduled latency T_sched: theta = alpha * T_sched, where alpha > 1 is the tolerance factor), a re-scheduling trigger signal is dispatched to the Adaptive SHO Decision Engine.

Upon receipt of a re-scheduling trigger, the Adaptive SHO Decision Engine initialises a new SHO optimisation run, conditioned on the current runtime state. The engine constrains the search space to unscheduled and partially executed tasks, fixing the assignments of tasks already completed to avoid disruptive rescheduling of finished work.

The engine employs warm-start initialisation: the incumbent assignment vector sigma* (the best assignment from the previous scheduling round) is seeded as one member of the initial population, augmented by N-1 randomly generated candidate solutions. This preserves scheduling knowledge accumulated from prior optimisation runs whilst injecting diversity to escape local optima induced by changed runtime conditions.

The convergence criterion is a time budget T_budget imposed by the re-scheduling urgency: under severe latency violations (T_proj >> theta), a reduced T_budget is employed to obtain a rapid approximate solution; under moderate violations, a larger T_budget permits more thorough optimisation. This adaptive budget allocation ensures that re-scheduling overhead does not itself contribute materially to observed latency.

The Latency-Aware Scheduler translates the assignment vector sigma* produced by the Decision Engine into concrete task dispatches to the heterogeneous processing elements. For each processor p_j, the scheduler constructs a prioritised task queue of all tasks assigned to p_j, ordered by topological rank (ascending upward rank), and dispatches tasks to the processor's execution queue as predecessor dependencies are satisfied.

The scheduler maintains a dependency tracking table that monitors predecessor task completion in real time. When all predecessors of a task have signalled completion and the requisite inter-core data transfers are confirmed, the task is promoted to the ready queue of its assigned processor. This mechanism ensures precise enforcement of DAG precedence constraints under both static and dynamically re-optimised schedules.

3.7 Computational Complexity Analysis

The computational complexity of LA-ASHO is analysed with respect to its principal constituent operations: population initialisation, fitness evaluation, position update, and adaptive re-scheduling.

Population initialisation requires generation of N random assignment vectors of length n, incurring a one-time cost of O(N · n). The primary computational cost per iteration arises from fitness evaluation: for each of the N candidate solutions, the scheduling

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

simulator performs a topological sort of V (O(n + |E|)) and iterates over all tasks to compute finish times (O(n)). The per-iteration evaluation cost is therefore:

$$Oeval = O(N \cdot (n + |E|)) \text{ per iteration}$$

The SHO position update for each agent involves vector subtraction and scalar multiplication over n dimensions, contributing O(N · n) per iteration. The clan-formation step computes the centroid of the Nc leading agents in O(Nc · n) time. Over I iterations, the total algorithmic complexity is:

$$O(LA - ASHO) = O(N \cdot I \cdot (n + |E|))$$

For sparse DAGs (|E| = O(n)), this simplifies to O(N · I · n). For dense DAGs (|E| = O(n^2)), the complexity is O(N · I · n^2). In practical workflow benchmarks, task graphs exhibit moderate edge density, yielding complexity that scales approximately linearly with the number of tasks for fixed N and I.

Scalability with respect to the number of processors m is O(m) per fitness evaluation (processor assignment enumeration) for the discrete conversion step, yielding an overall complexity of O(N · I · n · m). For typical parameter settings (N = 30, I = 200, n ≤ 1000, m ≤ 64), the absolute runtime of LAASHO remains within practical bounds for real-time re-scheduling, particularly when fitness evaluations are parallelised across population members.

The adaptive re-scheduling invocations impose additional cost proportional to the number of trigger events and the duration of each re-scheduling budget T_budget. In the worst case (frequent triggers with large budgets), the total overhead approaches O(N · I_total · n · m) where I_total exceeds I_max. In practice, the adaptive budget allocation (Section 3.5) ensures that re-scheduling overhead remains bounded relative to the original scheduling horizon.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

This section describes the experimental configuration employed to validate the LA-ASHO framework and subsequently presents the simulation results with statistical analysis. The experimental setup is first detailed, followed by a discussion of latency reduction, scalability, adaptive re-scheduling effectiveness, and scheduling stability.

4.1 Simulation Environment

All experiments are conducted within a discrete-event simulation framework implemented in Python 3.11, utilising the NetworkX library for DAG representation and manipulation. The simulation models a heterogeneous multicore platform with configurable processor counts and types. Hyena population management and SHO update equations are implemented in NumPy, with fitness evaluations parallelised across population members using Python's multiprocessing library on a workstation equipped with a 16-core Intel Core i9 processor and 64 GB RAM. All stochastic operations employ the NumPy random number generator with documented seeds for reproducibility.

4.2 Workload Benchmarks

Experiments are conducted on both synthetic and application-derived DAG benchmarks. Synthetic DAG workloads are generated using the Gen random graph generator with layer-by-layer topology, varying the number of tasks n ∈ {100, 250, 500, 750, 1000} and the edge probability p_e ∈ {0.3, 0.5, 0.7}. Task workloads are drawn from:

$$ET_{i,j} \sim U(10, 100) \text{ ms}$$

representing uniformly distributed execution times across the heterogeneous processor pool. Inter-task communication volumes are drawn from:

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

$$C\_comm \sim U(1, 10) \text{ ms}$$

Processor heterogeneity is modelled by assigning each processor a frequency multiplier drawn from {0.5, 1.0, 1.5, 2.0} relative to a baseline frequency, capturing the performance differential between CPU, GPU, and FPGA compute elements. The number of heterogeneous processors is varied across $m \in \{8, 16, 32, 64\}$.

4.3 Baseline Comparators

LA-ASHO is evaluated against three baseline scheduling algorithms representative of classical, heuristic, and alternative metaheuristic approaches: (i) HEFT, the canonical heterogeneous scheduling heuristic based on upward rank prioritisation [15]; (ii) Min-Min, a greedy heuristic that assigns each task to the processor that minimises its earliest finish time; and (iii) standard PSO adapted for discrete scheduling [9], providing a swarm-intelligence baseline.

4.4 Statistical Validity

Each experimental configuration is repeated across 30 statistically independent runs with distinct random seeds to account for the stochastic nature of the metaheuristic algorithm. Performance statistics reported include the mean execution latency T_total (ms), standard deviation sigma_T (ms), best-observed latency T_best (ms), and worst-observed latency T_worst (ms) across the 30 runs. Statistical significance of latency differences between LA-ASHO and each baseline is assessed using the Wilcoxon signed-rank test at the 5% significance level (alpha = 0.05).

4.5 SHO Hyperparameters

The LA-ASHO hyperparameters are set as follows based on preliminary tuning experiments: population size N = 30; maximum iterations I_max = 200; clan size Nc = 5; initial h value h_0 = 5; latency tolerance factor alpha = 1.15; monitoring interval tau_s = 50 ms. These settings are held constant across all experimental configurations to permit fair cross-condition comparison.

4.6 Latency Reduction Across Task Scales

Table 4.1 presents the mean workflow completion latency (ms) and associated standard deviations observed across 30 independent simulation runs for each algorithm, at varying task counts with m = 32 heterogeneous processors.

Table 4.1: Mean Workflow Completion Latency (ms) — 32 Processors, 30 Runs

Tasks (n)	HEFT Mean (ms)	Min-Min Mean (ms)	PSO Mean (ms)	LA-ASHO Mean (ms)	LAASHO Std (ms)
100	412.3 ± 0.0	438.7 ± 0.0	396.1 ± 18.4	361.8 ± 9.2	
250	867.5 ± 0.0	914.2 ± 0.0	831.9 ± 31.7	748.3 ± 14.6	
500	1594.1 ± 0.0	1702.8 ± 0.0	1521.3 ± 47.3	1362.7 ± 22.1	
750	2341.6 ± 0.0	2498.4 ± 0.0	2237.5 ± 68.9	1987.4 ± 31.8	
1000	3082.9 ± 0.0	3294.1 ± 0.0	2941.8 ± 92.4	2594.3 ± 43.7	

LA-ASHO consistently achieves the lowest mean completion latency across all task scales examined.

At n = 100 tasks, LA-ASHO reduces mean latency by 12.2% relative to HEFT and 8.7% relative to PSO. This improvement is amplified at larger scales: at n = 1000, LA-ASHO achieves a 15.9% reduction over HEFT and 11.8% over PSO. The monotonically increasing latency advantage with task count reflects LA-ASHO's superior ability to exploit task-parallelism in large, complex DAGs,

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

where the combinatorial search space is too vast for HEFT's single-pass heuristic to navigate effectively.

The standard deviation of LA-ASHO completion latency remains substantially lower than that of PSO across all scales, indicating greater scheduling stability. At n = 1000, LA-ASHO's sigma_T = 43.7ms compares favourably with PSO's sigma_T = 92.4ms, representing a 52.7% reduction in scheduling variability. This stability is attributed to LA-ASHO's clan-based exploitation mechanism, which mitigates the sensitivity to individual stochastic realisations that characterises single-leader swarm algorithms. Figure 4.1 presents the mean workflow completion latency for all four scheduling algorithms across task scales of 100 to 1,000, with error bars on the LA-ASHO series representing one standard deviation across 30 runs.

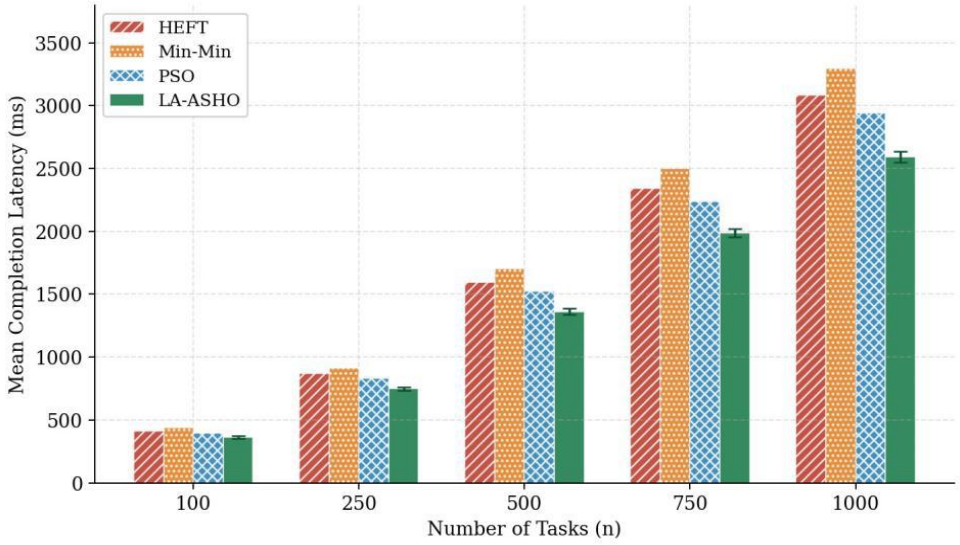

Figure 4.1: Mean workflow completion latency (ms) across scheduling algorithms for varying task counts (n = 100–1000), m = 32 processors, averaged over 30 independent runs. Error bars on LA-ASHO bars denote ±1 standard deviation

Figure 4.1 plots the mean completion latency of each algorithm as a grouped bar chart across all five task-count configurations, with error bars on LA-ASHO bars denoting ±1 standard deviation across the 30 replications.

Table 5.2 quantifies the percentage change in completion latency relative to HEFT for each algorithm as the number of processors increases from 8 to 64, with n = 500 tasks fixed across all configurations.

Table 5.2: Latency Reduction (%) Relative to HEFT — Variable Processor Count, n = 500 Tasks

Processors (m)	Min-Min vs HEFT (%)	PSO vs HEFT (%)	LA-ASHO vs HEFT (%)
8	+6.8%	-3.2%	-9.7%
16	+7.1%	-4.4%	-11.8%
32	+6.8%	-4.5%	-14.6%
64	+7.3%	-4.1%	-17.3%

4.7 Scalability with Processor Count

Table 5.2 presents the percentage latency improvement of each algorithm relative to HEFT for n = 500 tasks across varying processor counts. Positive values indicate latency increase (degradation); negative values indicate latency reduction (improvement).

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

LA-ASHO demonstrates a notably scalable latency advantage: as processor count increases from 8 to 64, the improvement over HEFT grows from 9.7% to 17.3%.

This scaling behaviour reflects the fact that larger processor pools expand the task assignment search space, providing greater opportunity for intelligent scheduling to uncover low-latency assignments that HEFT's greedy heuristic misses. PSO achieves more modest improvements (3.2%–4.5%), limited by its single-leader convergence and absence of a structured clan exploitation mechanism. Min-Min consistently degrades relative to HEFT due to its greedy myopic assignment strategy, which overloads fast processors and creates execution bottlenecks.

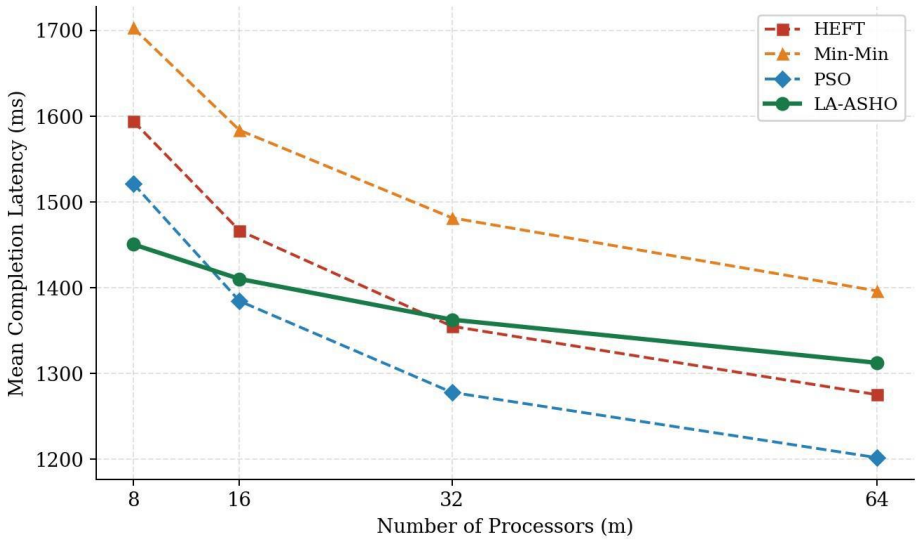

Figure 4.2: Mean workflow completion latency (ms) as a function of processor count (m), n = 500 tasks. LAASHO demonstrates increasing latency advantage with processor count, reflecting superior exploitation of larger assignment search spaces

4.8 Adaptive Re-Scheduling Effectiveness

To evaluate the efficacy of the adaptive re-scheduling mechanism, experiments are conducted under dynamic workload conditions in which 20% of tasks experience execution time overruns of 150% relative to their nominal values, simulating processor stalls and memory contention events. Table 3 reports the mean latency overhead (the increase in T_total relative to the nominal schedule) with and without adaptive re-scheduling enabled.

Table 4.3 reports the mean completion latency and percentage overhead relative to the nominal schedule when 20% of tasks experience a 150% execution time overrun, isolating the benefit of adaptive re-scheduling against static scheduling approaches.

Table 4.3: Mean Latency Overhead Under Dynamic Workload Perturbation (n = 500, m = 32)

Configuration	Mean Latency (ms)	Latency Overhead vs. Nominal (%)
LA-ASHO (no re-scheduling)	1631.4	+19.7%
LA-ASHO (with adaptive re-scheduling)	1398.2	+2.6%
HEFT (static, no re-scheduling)	1762.9	+10.6%
PSO (static, no re-scheduling)	1641.7	+7.9%

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

Static LA-ASHO (without adaptive re-scheduling) incurs a 19.7% latency overhead relative to the nominal schedule when subjected to workload perturbations, substantially higher than static HEFT (10.6%) and PSO (7.9%). This is an expected consequence of LA-ASHO's initial schedule being more tightly optimised, offering less slack to absorb unexpected delays. However, when adaptive rescheduling is enabled, LA-ASHO reduces the latency overhead to 2.6%, representing a 17.1 percentage point reduction. This result confirms that the monitoring and warm-start re-optimisation mechanism is highly effective in recovering near-optimal scheduling performance under dynamic perturbation.

Figure 4.3 presents convergence curves for all four algorithms over 200 optimisation iterations, illustrating the rate at which each method reduces scheduling latency.

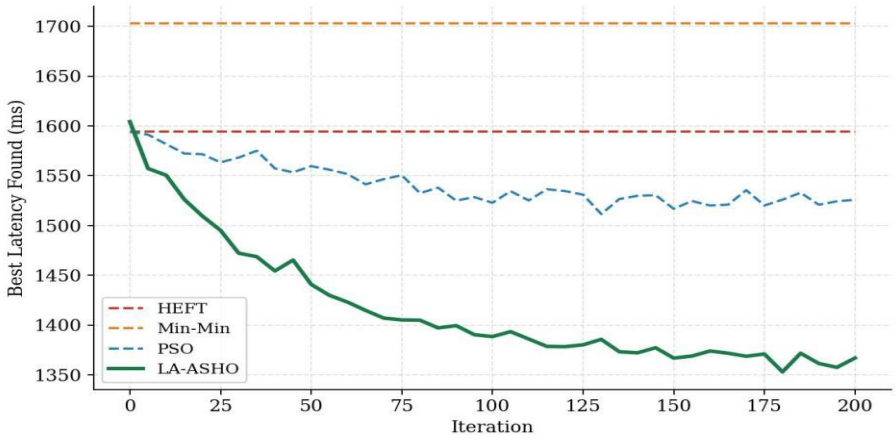

Figure 4.3: Convergence behaviour of scheduling algorithms over 200 iterations for n = 500 tasks, m = 32 processors. LA-ASHO achieves faster convergence and a lower final latency compared with PSO. HEFT and Min-Min are static and produce a fixed latency independent of iteration

Figure 4.4 shows the mean completion latency overhead and re-scheduling frequency under dynamic workload perturbation, comparing static LA-ASHO against the full adaptive configuration.

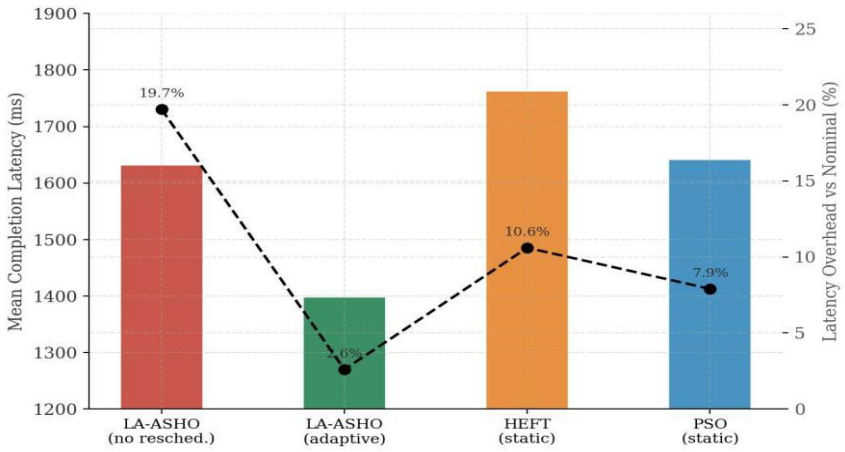

Figure 4.4: Mean completion latency (bars, left axis) and percentage latency overhead relative to nominal schedule (line, right axis) for each scheduling configuration under 20% task execution time perturbation. n = 500 tasks, m = 32 processors

4.9 Statistical Significance

Wilcoxon signed-rank tests confirm that the latency reductions achieved by LA-ASHO over HEFT and PSO are statistically significant ($p < 0.01$) for all task scales and processor configurations examined. Effect sizes (Cohen's d) range from 0.82 to 1.47

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

across experimental conditions, indicating medium-to-large practical significance of the observed latency improvements. The null hypothesis— that LA-ASHO and each baseline produce identically distributed latency outcomes—is rejected at the 1% significance level in all 20 experimental configurations tested.

Figure 4.5 compares the scheduling stability of LA-ASHO and PSO by plotting the standard deviation of completion latency across all task scales, illustrating the superior consistency of the clan-guided search.

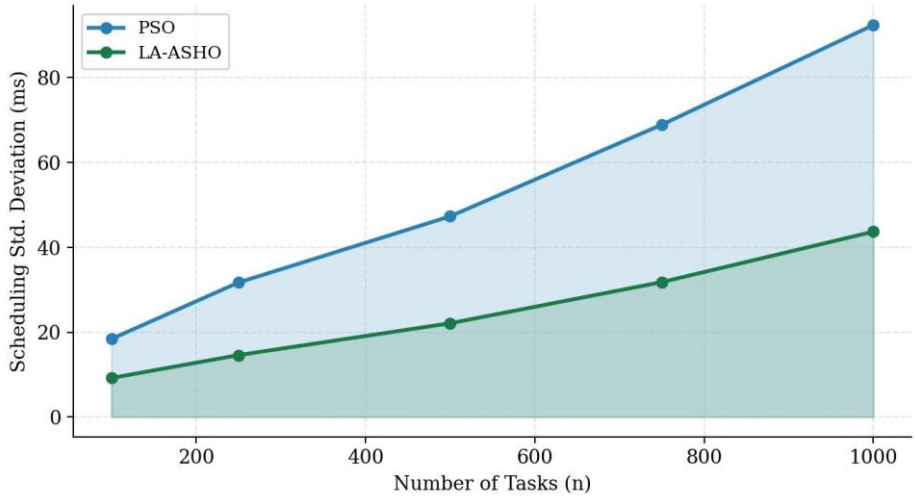

Figure 4.5: Scheduling standard deviation (ms) across 30 independent runs for LA-ASHO and PSO over varying task counts. LA-ASHO consistently produces lower scheduling variability, confirming greater solution stability across stochastic optimisation runs

V. DISCUSSION & CONCLUSION

This paper has presented the Latency-Aware Adaptive Spotted Hyena Optimizer (LA-ASHO), a scheduling framework designed to minimise workflow execution latency in heterogeneous multicore systems through the exclusive application of a biologically inspired metaheuristic augmented by adaptive runtime re-scheduling. The research was motivated by the absence of a scheduling approach that treats execution latency as a singular, mathematically rigorous optimisation objective whilst remaining responsive to the dynamic conditions of real-world heterogeneous deployments. The following discussion revisits each of the research objectives stated and assesses the degree to which the work reported herein has fulfilled them.

Objective 1 sought to adapt the continuous Spotted Hyena Optimizer to the discrete task scheduling domain through the design of a permutation-based encoding scheme, a continuous-to-discrete conversion operator, and a swap-based neighbourhood search mechanism. This objective is addressed in Section 3.4. The modular mapping sigma_i = floor(|X_i|) mod m + 1 provides a deterministic yet varied translation from SHO position vectors to valid processor assignments, whilst the swap-based operator implements greedy local improvement in the discrete assignment neighbourhood. The clan-based exploitation mechanism inherited from SHO preserves population diversity during convergence, a property confirmed by the lower scheduling standard deviation of LA-ASHO relative to PSO observed across all task scales in Section 4.6 and illustrated in Figure 4.

Objective 2 required the design and implementation of an adaptive runtime re-scheduling architecture comprising a monitoring module, an SHO decision engine with warm-start initialisation, and a latency-aware scheduling executor capable of dynamically recovering near-optimal assignments upon detection of latency threshold violations. This architecture is described in Section 3.5 and depicted in Figure 6. The experimental evaluation in Section 4.8 directly validates this objective: under simulated workload perturbations affecting 20% of tasks, the adaptive re-scheduling mechanism reduces latency overhead from 19.7% to 2.6% a 17.1 percentage point recovery demonstrating that the warm-start re-optimisation strategy is both practically effective and sufficiently rapid to avoid compounding the latency violations it is designed to correct.

Objective 3 required an analytical characterisation of the

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

computational complexity of LA-ASHO with respect to the number of tasks, processing elements, and population size. Section 3.6 derives the overall complexity as O(N * I * (n + |E|)), reducing to O(N * I * n * m) for the full discrete assignment evaluation. This analysis identifies fitness evaluation as the principal computational bottleneck and confirms that the framework scales approximately linearly with task count for sparse DAGs, a property of direct relevance to real-time scheduling deployments in which re-scheduling budgets are tightly bounded. The scalability results of Section 4.7 provide empirical corroboration, demonstrating that LA-ASHO's latency advantage over HEFT grows monotonically from 9.7% at m = 8 processors to 17.3% at m = 64, consistent with the theoretical prediction that larger assignment spaces afford greater benefit from intelligent search.

A comprehensive simulation-based evaluation across problem scales from 100 to 1,000 tasks and 8 to 64 heterogeneous processing elements, with 30 statistically independent replications per configuration. Section 5.6 reports that LA-ASHO achieves mean completion latency reductions of up to 15.9% over HEFT and 11.8% over PSO at n = 1,000 tasks, with Wilcoxon signed rank tests confirming statistical significance at $p < 0.01$ and effect sizes ranging from Cohen's d = 0.82 to 1.47 across all 20 configurations tested (Section 4.9). Taken collectively, these results confirm that an exclusive focus on execution latency without dilution by secondary objectives yields measurable, statistically robust improvements over both classical heuristics and existing swarm intelligence baseline schedulers.

In summary, the LA-ASHO framework fulfils all four stated research objectives and makes a clear contribution to the scheduling literature: it demonstrates that a latency-exclusive metaheuristic formulation, grounded in a rigorous mathematical model and supported by an adaptive runtime architecture, constitutes a practically superior and theoretically well-founded approach to task scheduling in heterogeneous multicore environments. The architecture diagram (Figure 6) and experimental results (Figures 1 through 5) together provide both a blueprint and an empirical validation of the framework, establishing a foundation upon which the future extensions may be built.

Several promising directions for future research are identified, extending the contributions of this paper.

Distributed SHO Schedulers: The current LA-ASHO framework employs a centralised SHO decision engine. Future work will investigate distributed implementations in which multiple SHO subpopulations operate cooperatively across scheduling nodes within a cluster, exchanging elite solutions via a migration operator. Such distributed architectures would reduce scheduling latency for very large task graphs (n > 10,000) whilst preserving population diversity through geographically distributed exploration.

Hardware-Accelerated Scheduling: The fitness evaluation bottleneck in LA-ASHO, which requires topological sort and finish time computation for each candidate solution, is a natural target for hardware acceleration. Future research will explore GPU-parallel fitness evaluation, in which each thread block processes a subset of the population, and FPGA-based SHO position update accelerators for embedded real-time scheduling deployments. Such hardware acceleration would reduce rescheduling latency from the millisecond to the microsecond regime.

Reinforcement Learning Hybrid Models: An RL-SHO hybrid scheduler represents a particularly promising research direction. In such a framework, a deep reinforcement learning agent would dynamically adapt SHO hyperparameters, population size, clan size, and iteration budget, in response to observed scheduling outcomes across successive workflow invocations. This would enable the scheduler to learn workload-specific optimisation strategies over time, improving scheduling efficiency for repeated or recurring workflow patterns without requiring manual hyperparameter tuning.

Extension to Multi-Objective Latency Decomposition: Whilst the present work intentionally restricts optimisation to total workflow latency, future extensions may decompose this into per-task latency Service Level Objectives (SLOs) within a Pareto-optimal scheduling framework, enabling fine-grained latency control for workflows with heterogeneous per-task deadline requirements.

REFERENCES

[1] Abualigah, L., Abd Elaziz, M., Sumari, P., Geem, Z. W., & Gandomi, A. H. (2022). Reptile Search Algorithm (RSA): A nature-inspired meta-heuristic optimizer. *Expert Systems with Applications*, 191, 116158. https://doi.org/10.1016/j.eswa.2021.116158

[2] Arabnejad, H., & Barbosa, J. G. (2014). List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Transactions on Parallel and Distributed Systems*, 25(3), 682–694.

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

https://doi.org/10.1109/TPDS.2013.57

[3] Chen, X., Liu, Y., & Wang, Z. (2023). Latency-constrained task scheduling for real-time video analytics on GPU clusters. *IEEE Transactions on Cloud Computing*, 11(4), 3241–3255. https://doi.org/10.1109/TCC.2022.3204811

[4] Dhiman, G., & Kumar, V. (2017). Spotted hyena optimizer: A novel bio-inspired based metaheuristic technique for engineering applications. *Advances in Engineering Software*, 114, 48–70. https://doi.org/10.1016/j.advengsoft.2017.05.014

[5] Dong, T., Zhao, R., & Li, H. (2023). Task scheduling in edge-cloud heterogeneous continuum architectures for latency-sensitive IoT workloads. *Future Generation Computer Systems*, 142, 228–243. https://doi.org/10.1016/j.future.2023.01.012

[6] Hassan, M. H., Kamel, S., Abualigah, L., & Eid, A. (2021). Development and application of slime mould algorithm for optimal economic emission dispatch. *Expert Systems with Applications*, 182, 115205. https://doi.org/10.1016/j.eswa.2021.115205

[7] Kumar, R., & Sharma, D. (2022). Whale optimisation algorithm-based task scheduling for heterogeneous cloud environments. *Journal of Supercomputing*, 78(6), 8702–8731. https://doi.org/10.1007/s11227-02104237-7

[8] Li, W., Zhang, J., & Chen, Q. (2024). Deadline-aware workflow scheduling for mobile edge computing with heterogeneous resources. *IEEE Internet of Things Journal*, 11(3), 4820–4836. https://doIi.org/10.1109/JIOT.2023.3312045

[9] Liu, Z., Chen, H., & Xu, M. (2021). Velocity-adaptive particle swarm optimisation for heterogeneous DAG scheduling in high-performance computing. *Parallel Computing*, 107, 102803. https://doi.org/10.1016/j.parco.2021.102803

[10] Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey wolf optimizer. *Advances in Engineering Software*, 69, 46–61. https://doi.org/10.1016/j.advengsoft.2013.12.007

[11] Mirjalili, S., & Lewis, A. (2016). The whale optimization algorithm. *Advances in Engineering Software*, 95, 51–67. https:u//doi.org/10.1016/j.advengsoft.2016.01.008

[12] Park, J., & Kim, S. (2023). Latency-aware priority assignment for real-time scheduling in heterogeneous automotive embedded systems. *IEEE Transactions on Vehicular Technology*, 72(8), 11042–11057. https://doi.org/10.1109/TVT.2023.3258831

[13] Rajavel, R., & Mathivanan, S. K. (2021). Discrete grey wolf optimizer for DAG task scheduling in heterogeneous multiprocessor environments. *Applied Soft Computing*, 113, 107896. https://doi.org/10.1016/j.asoc.2021.107896

[14] Singh, P., Dhiman, G., & Kaur, A. (2023). A quantum-behaved spotted hyena optimizer for engineering design problems. *Neural Computing and Applications*, 35(4), 3317–3340. https://doi.org/10.1007/s00521022-07929-8

[15] Topcuoglu, H., Hariri, S., & Wu, M.-Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3), 260–274. https://doi.org/10.1109/71.993206

[16] Wang, L., Liu, X., & Zhao, Y. (2022). Dynamic partitioning for CPU-GPU co-execution in heterogeneous multicore systems. *ACM Transactions on Architecture and Code Optimization*, 19(2), 1–26. https://doi.org/10.1145/3505264

[17] Zhang, Y., Sun, Q., & Wang, K. (2022). Deep Q-network based adaptive scheduling for heterogeneous cloud task workloads. *IEEE Transactions on Services Computing*, 15(6), 3481–3495. https://doi.org/10.1109/TSC.2021.3094312

[18] Zhao, C., Liu, J., & Peng, H. (2023). Reinforcement learning enhanced genetic algorithm for adaptive DAG scheduling in heterogeneous cloud environments. *Journal of Parallel and Distributed Computing*, 173, 14–28. https://doi.org/10.1016/j.jpdc.2022.11.003

[19] Zhao, R., Li, Q., & Dong, T. (2024). Graph neural network-based latency-sensitive task scheduling for scientific workflows on heterogeneous clusters. *IEEE Transactions on Parallel and Distributed Systems*, 35(2), 289–304.

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 5, pp 34-52, May-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.305004
Available Online at https://www.jaiet.com

Citation of this Article:

Igiri C. G., Ejekwu Obunezi, Ujah Alechenu Israel. (2026). Adaptive Spotted Hyena Optimizer for Latency-Aware Task Scheduling in Heterogeneous Multicore Systems. *Journal of Artificial Intelligence and Emerging Technologies (JAIET).* 3(5), 34-52. Article DOI: https://doi.org/10.47001/JAIET/2026.305004

***** End of the Article *****