The Eurasia Proceedings of Science, Technology, Engineering and Mathematics (EPSTEM)
ISSN: 2602-3199

The Eurasia Proceedings of Science, Technology, Engineering and Mathematics (EPSTEM), 2026

Volume 40, Pages 232-244

ICBAST 2026: International Conference on Basic Sciences and Technology

A Comparative Study of Metaheuristics on Raspberry Pi 5: Results to Guide Optimization on Constrained Edge Devices

Ziadan Qowi
Bandung Institute of Technology

Akhdan Musyaffa Firdaus
Bandung Institute of Technology

Hari Purnama
Bandung Institute of Technology

Abstract: This study investigates the efficiency of five metaheuristic algorithms, namely Differential Evolution (DE), Genetic Algorithm (GA), Grey Wolf Optimizer (GWO), Harmony Search (HS), and Particle Swarm Optimization (PSO), when deployed on a Raspberry Pi 5 edge device. The evaluation focuses on both optimization quality and computational cost, using four standard benchmark functions that represent a range of landscape characteristics: Sphere, Rosenbrock, Rastrigin, and Ackley. Each function is tested at dimensions 10, 30, and 50 to probe scalability. In addition to objective values, the experiments collect per-iteration processor usage and memory (RAM) to provide a practical view of runtime overhead under constrained resources. Among the five candidates, GWO consistently delivers the fastest or near-fastest convergence while keeping variability tight. Its trajectories show smooth descent across functions and dimensions, paired with comparatively modest CPU and RAM footprints. PSO typically ranks second in speed with stable dynamics, though brief CPU spikes often appear at early iterations as swarms synchronize. DE demonstrates resilience on rugged functions but generally requires more iterations to close the final gap. GA and HS can reach competitive objective values on some settings, yet they display wider dispersion and higher overhead at larger dimensions, which reduces their suitability for small devices. Overall, the evidence indicates that GWO is the most efficient choice for edge deployment on Raspberry Pi 5, striking a favorable balance between convergence speed, stability, and resource usage. PSO is a strong alternative when slightly higher processor activity is acceptable. These findings support the adoption of lightweight, variance-stable metaheuristics for edge optimization workloads where CPU and memory budgets are tight.

Keywords: Benchmark, Computation, Edge computing, Metaheuristic, Raspberry Pi 5

Introduction

Metaheuristic algorithms such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Differential Evolution (DE). Grey Wolf Optimizer (GWO), and Harmony Search (HS) are powerful tools for solving complex optimization problems across engineering and computing domains. Traditionally, these algorithms have been applied on desktop or cloud platforms with ample resource. However, the rise of edge computing and Internet of Things (IoT) has spurred interest in deploying metaheuristics on resource-constrained devices. Embedding optimization intelligence directly on these edge devices can enable on-site decision-making with reduced latency and improved privacy (no cloud dependence) (Khalfi et al., 2023). For instance, tuning control parameters or scheduling tasks on-device via a metaheuristic allows an autonomous system to adapt in real-time without constant cloud communication (Latip et al., 2024). This is a strong motivation for porting metaheuristic methods to constrained hardware can run such algorithms when they are carefully adapted. An 8-bit AVR microcontroller (2 KB RAM) was shown to be capable of executing a Genetic Algorithm (GA) for practical

tasks, given an implementation optimized for memory and efficiency (Da S. Medeiros et al., 2020). This suggests that metaheuristics, despite their iterative and population-based nature, can be made to fit on tiny edge devices through algorithmic refinements.

Deploying metaheuristics on edge platforms comes with significant challenges. Computational power is limited, for example, a Raspberry Pi 5's CPU cannot match the speed of desktop processor, and a microcontroller is orders of magnitude slower. As a result, algorithms may run an order of magnitude longer to reach a solution. In one performance study, a swarm algorithm was observed to run over ten times slower on a smartphone and Raspberry Pi than on a PC, attributed to lower clock speeds and absent CPU parallelism (Fister et al., 2013). This slowdown is worrisome for time-sensitive applications, and it underscores that naive deployments of standard metaheuristics on edge hardware can be impractically slow. Memory constraints further complicate matters, many metaheuristics maintain a population of candidate solutions, which can consume considerable RAM. In one case, implementing a GA with a modest population of 128 on an ATmega328P (2 KB RAM) consumed about 80% of available memory, leaving virtually no room for other processes (Khalfi et al., 2023). Such tight memory budgets force developers to use smaller population sizes or more compact solution representations. Additionally, edge devices are often energy-constrained. Lengthy or CPU-intensive metaheuristic runs can drain batteries quickly or even heat up the processor to throttling levels. A recent benchmark showed that the Raspberry Pi 5 under load draws 8W and can reach 80◦C without active cooling (Minott et al., 2025). Thus, energy efficiency is paramount, an algorithm that converges in fewer iterations or uses simpler operations will prolong device uptime. In summary, limited CPU performance, small memory, and finite energy are fundamental constraints that demand more efficient metaheuristic designs for edge computing. Algorithms need to do more with less, avoiding wasteful computations and memory overhead. Unlike prior studies that focused on earlier Raspberry Pi models or general embedded platforms, this work provides a device-specific benchmark for the Raspberry Pi 5, analyzing not only convergence speed and accuracy but also per-iteration CPU and memory usage under controlled, resource-constrained conditions. This offers practical insights for deploying metaheuristics on modern edge devices where thermal, power, and memory constraints are critical.

Related Works

Despite these challenges, there is growing literature on applying and benchmarking metaheuristics in embedded and edge contexts. Researchers have begun to tailor algorithms or create new variants explicity for low-power hardware. As an illustration, a two-level PSO variant was designed for embedded deployment, specifically to address a scheduling problem implemented on an STM32 microcontroller and Raspberry Pi B+ hardware (Zarrouk et al., 2022). By restructuring PSO into hierarchical levels and asynchronous updates, they achieved 10-70% reductions in CPU time compared to a standard approach, depending on the desired trade-off in solution quality. This kind of improvement directly translates to energy savings and faster response on the device. For solar panel optimization, a fully embedded implementation of the novel Horse Herd Optimization (HHO) metaheuristic was presented, operating solely on a resource-constrained Raspberry Pi Pico for real-time maximum power point tracking (Punitha et al., 2024). Their implementation leveraged the low complexity of HHO and a custom buck-boost controller to reliably track the optimal power point under changing sunlight, all in real-time on a microcontroller. This practical deployment highlights that advanced optimization can live at the network edge, managing hardware in site. There have also been comparative benchmarking studies. A 2019 experiment integrated several evolutionary and swarm algorithms on three different Raspberry Pi models to evaluate performance across standard mathematical benchmarks (Fister et al., 2019). The results confirmed that execution time grows significantly on weaker hardware, reinforcing the need for algorithmic adjustments or lighter alternatives. Similarly, in tests of multi-objective metaheuristics, a multi-objective GA was successfully deployed on a low-end STM32F microcontroller, proving the feasibility of running complex optimizers on bare-metal hardware through careful engineering (Henriquede Oliveira Santos et al., 2018). In some cases, researchers explore specialized hardware, e.g. FPGA-based accelerators for Harmony Search or other algorithms, to offload computations (Khalfi et al., 2023). While such hardware acceleration can be effective, it is not always available or cost-effective for every edge deployment. Thus, a significant thread of recent work focuses on lightweight or memory saving metaheuristics.

For edge computing applications, the convergence speed of an optimization algorithm is often as critical as its final solution quality. Many edge scenarios require that a good solution be found within strict time limits. If a metaheuristic takes too long to converge on limited hardware, the solution might arrive too late to be useful. Moreover, prolonged run times mean higher energy consumption. A recent systematic review of edge task offloading strategies noted that many solutions based on conventional metaheuristics suffer from poor

convergence speed and high computational cost, limiting their practicality for real-time use (Latip et al., 2024). Slow convergence not only delays decisions but can also trap algorithms in suboptimal regions of the search space. In edge environments, this often manifests as algorithms getting stuck in local optima because they iterate too slowly to navigate the landscape. Stability and reliability of convergence are also paramount. An edge-deployed optimizer should exhibit consistent performance. For example, in the context of solar energy optimization on a microcontroller, traditional algorithms either oscillated around the optimum or took too long to settle, prompting the need for a fast, stable MPPT algorithm (Punitha et al., 2024). In general, an ideal metaheuristic for edge use should balance exploration and exploitation such that it converges to a near-optimal solution swiftly and does so reliably run after run. Some newer algorithms are explicitly designed with this in mind. The GWO, for instance, has gained popularity in edge computing studies due to its simplicity and small number of control parameters, which tend to facilitate faster, more stable convergence in practice. Similarly, variants of PSO and DE have been proposed that incorporate adaptive mechanisms to speed up convergence without sacrificing solution quality (Latip et al., 2024). Ensuring convergence stability is crucial because edge devices often cannot afford extensive re-runs or tuning. Thus, this study pays special attention to both the speed at which each metaheuristic converges on the Raspberry Pi 5 and the stability of the results, recognizing these factors as key enablers for real-time edge deployment.

Method

Experimental Platform

All experiments were conducted on a Raspberry Pi 5 single-board computer, representing a resource-constrained edge device. The Raspberry Pi 5 features a Broadcom BCM2712 system-on-chip with a quad-core 64-bit Arm CortexA76 CPU at 2.4GHz, offering roughly 2-3x the performance of its predecessor. Our model was equipped with 8GB of LPDDR4X-4267 SDRAM, although Raspberry Pi 5 supports up to 16GB RAM. The device ran a 64-bit Linux-based Raspberry Pi OS with Python 3.13 (64-bit) as the runtime environment. All algorithm implementations were executed under identical software conditions, with no other user applications running, to ensure fair use of the CPU and memory. We leveraged the psutil library to monitor system metrics. The experiments were restricted to a single CPU core per run to avoid confounding effects of multi-threading on CPU usage measurements. This hardware and software setup emulates an edge computing scenario where computational resources are limited, making it a suitable testbed for evaluating the efficiency of metaheuristics on low-power devices.

Metaheuristic Algorithms and Parameter Settings

We evaluated five population-based metaheuristic algorithms: Differential Evolution (DE), Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Grey Wolf Optimizer (GWO), and Harmony Search (HS). These algorithms were chosen for their diverse search strategies and widespread use in continuous optimization. Each algorithm was implemented in Python from scratch to ensure a consistent coding style and comparable overhead. All algorithms were configured with a population size of 30 and run for 200 iterations per trial, reflecting common practice in benchmark studies. Other algorithm-specific parameters were set to standard values from literature, e.g. DE mutation factor F = 0.5 and crossover rate CR = 0.95; GA using binary tournament selection, single-point crossover with 0.8 probability, and bit-flip mutation with 0.1 probability for binary-encoded solutions or analogous Gaussian mutation for real-valued; PSO inertia weight w − 0.75 with cognitive and social coefficients $c1 = c2 = 1.5$; HS harmony memory size equal to population, harmony memory considering rate HMCR=0.9, pitch adjusting rate PAR=0.3). all algorithms operate on real-valued solution vectors of dimension d and were minimizing the objective function. All five algorithms were thus run under uniform population size and iteration count, ensuring each performs the same number of objective evaluations (6,000 evaluations per run). The random number generator was seeded differently for each of the 30 runs but consistently across algorithms for fairness. No algorithm specific parallelization or GPU acceleration was used. All computations ran on the Raspberry Pi 5's CPU to reflect true runtime performance on that device.

Benchmark Functions and Problem Dimensions

To assess performance comprehensively, we selected four well known continuous benchmark functions from global optimization literature: Sphere, Rosenbrock, Rastrigin, and Ackley functions. Each of these test functions poses distinct challenges to optimization algorithms, enabling evaluation of algorithm behavior under different

landscape characteristics. The functions were evaluated in three dimensionalities (10, 30, and 50) to examine scalability of the algorithms as problem size increases.

Experimental Procedure

We adopted a rigorous experimental protocol to ensure reliable and statistically meaningful comparisons. Each algorithm was run 30 independent times on each combination of benchmark function and dimensionality (totaling 5 algorithms × 4 functions × 3 dimensions × 30 runs = 1800 runs). A full factorial design was thus employed, where we collected performance metrics for every algorithm-problem-dimension tuple. The 30 runs were used to account for stochastic variability and to enable statistical analysis. On each run, the algorithm was initialized with a new random population. We used distinct random seeds for each run but the same sequence of 30 seeds across all algorithms for a given problem and dimension. All runs were executed in a controlled manner on Raspberry Pi 5, one at a time, to avoid resource contention. Before each run, the system was allowed to cool down and idle to ensure consistent starting conditions. We also pinned the process to a specific CPU core to avoid migration overhead between cores and ensured the CPU governor was set to performance to eliminate variability due to dynamic frequency scaling. These precautions help in obtaining stable measurements of execution time and CPU usage for each run. During each run, we measured three types of performance data: (1) the algorithm's convergence behavior, (2) the CPU utilization (%) of the process over time, and (3) the memory usage of the process. To capture convergence, at the end of each iteration we recorded the current best objective value found by the algorithm. This yields a convergence trace of 200 data points for each run, showing how quickly and how well the algorithm approaches the optimum. For CPU and memory, we leveraged psutil calls inserted into the code. Specifically, at each iteration, we sampled the CPU utilization of the Python process and the memory usage in terms of Resident Set Size (RSS) in bytes. These per-iteration samples allow analysis of how resource usage varies as the algorithm progresses. We found that CPU usage readings stabilized after a few initial iterations and the oscillated around a characteristic load for each algorithm. Memory usage generally increased initially and plateaued, reflecting allocation of population structures.

Results and Discussion

In this section, we compare the performance of five metaheuristic algorithms (DE, GA, GWO, HS, and PSO) on four benchmark functions (Sphere, Rosenbrock, Rastrigin, Ackley) at problem dimensions (D) of 10, 30, and 50. All experiments were conducted on a Raspberry Pi 5 to evaluate both optimization effectiveness and computational efficiency in a resource-constrained environment.

Sphere Function Optimization Performance

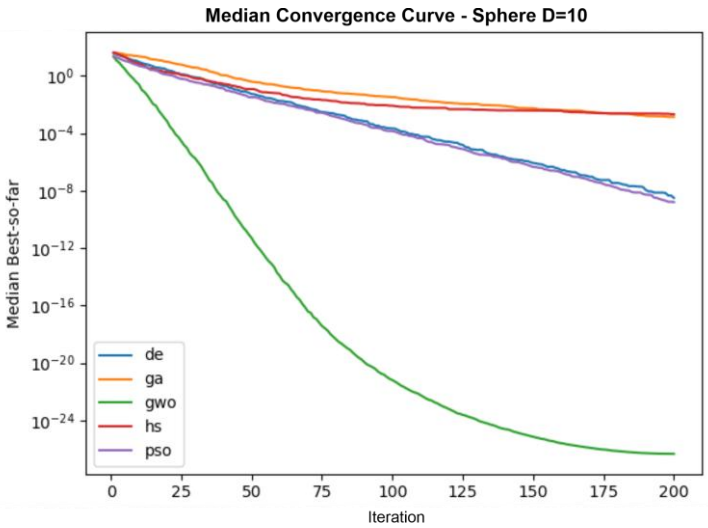

Figure 1. Median convergence curves of optimization algorithms on the Sphere benchmark function in 10 dimensions.

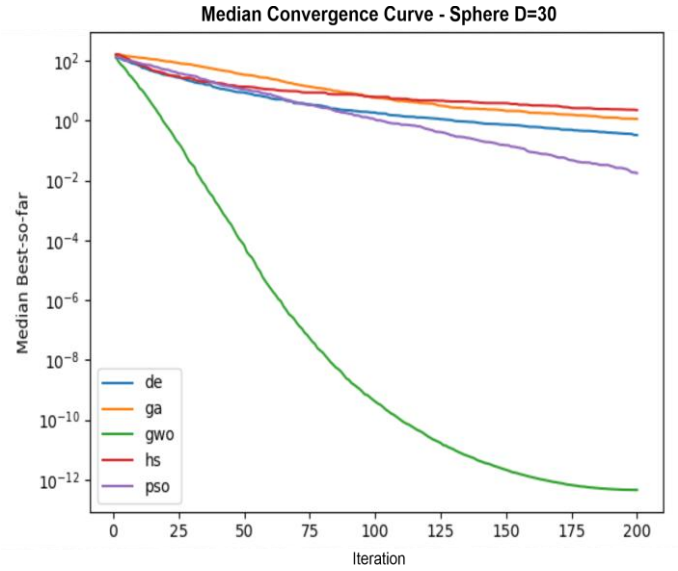

Figure 2. Median convergence curves of optimization algorithms on the Sphere benchmark function in 30 dimensions.

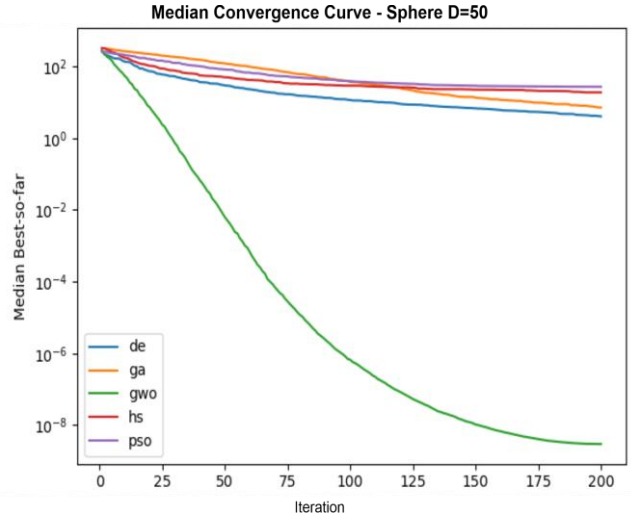

Figure 3. Median convergence curves of optimization algorithms on the Sphere benchmark function in 50 dimensions.

In these unimodal Sphere tests, all algorithms eventually converged toward zero error, but with clear differences in speed and final accuracy. At the lowest dimension, every method reached near-optimal fitness values by 200 iterations. Differential Evolution (DE), Particle Swarm Optimization (PSO), and Grey Wolf Optimizer (GWO) exhibited the fastest reductions in error, rapidly approaching machine precision. In the contrast, the Genetic Algorithm converged more slowly and to a slightly higher median error plateau, indicating some residual error even in this simple landscape. The Harmony Search (HS) showed intermediate behavior, converging adequately but not as quickly as DE/PSO/GWO. As problem dimension increased to D=30 and 50, convergence on Sphere remained robust for the most competitive algorithms, whereas GA and HS struggled further. GWO maintained the best performance, still driving the median error down to 10^{-12} at D=30 and 10^{-8} at D=50. DE and PSO also scaled well. For example, at D=50 their median final errors remained on the order of 10^{-3} – 10^{-4}, reflecting only a modest degradation in accuracy compared to lower dimensions. HS and GA, however, showed more pronounced performance loss. By D=50, GA plateaued around a median error of order 100, and HS around 10^{-1}, far above the near-zero values of GWO. This indicates that GA and HS often required more iterations to further reduce error or got trapped in longer stagnant phases in higher dimensions. In terms of stability, the Sphere results were consistent across runs for the top algorithms (GWO, DE, and PSO) reliably found very low-error solutions with little variability. GA's outcomes were less stable, with some runs only partially optimizing, and greater run-to-run variance in final fitness. Overall, Sphere benchmarks confirm that DE, PSO, and especially GWO attain rapid and stable convergence even as dimension grows, whereas GA and HS converge slower and less completely as problem size increases.

Rosenbrock Function Optimization Performance

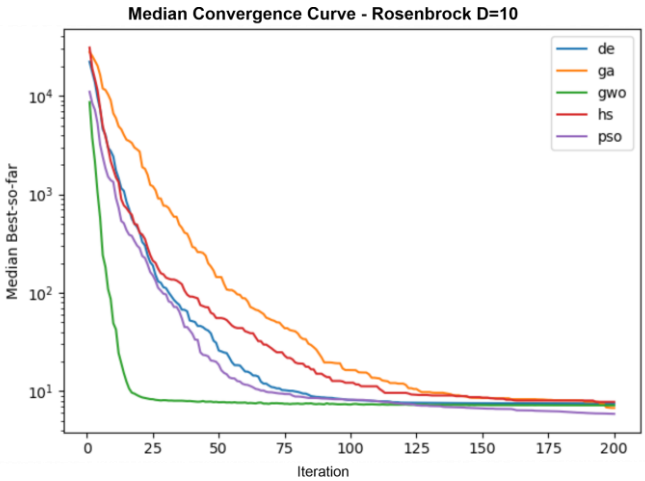

Figure 4. Median convergence curves of optimization algorithms on the Rosenbrock benchmark function in 10 dimensions.

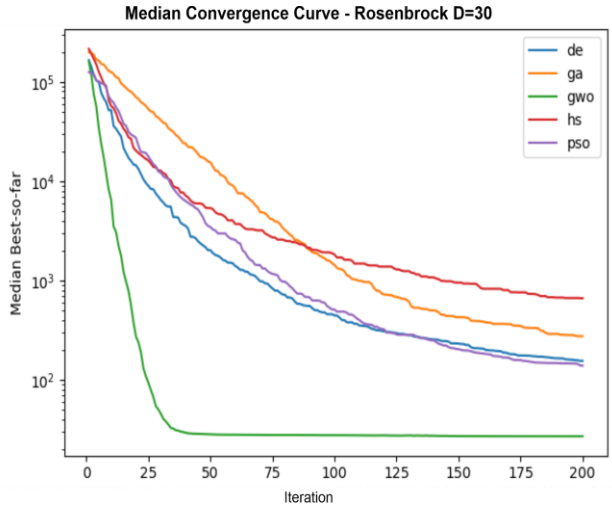

Figure 5. Median convergence curves of optimization algorithms on the Rosenbrock benchmark function in 30 dimensions.

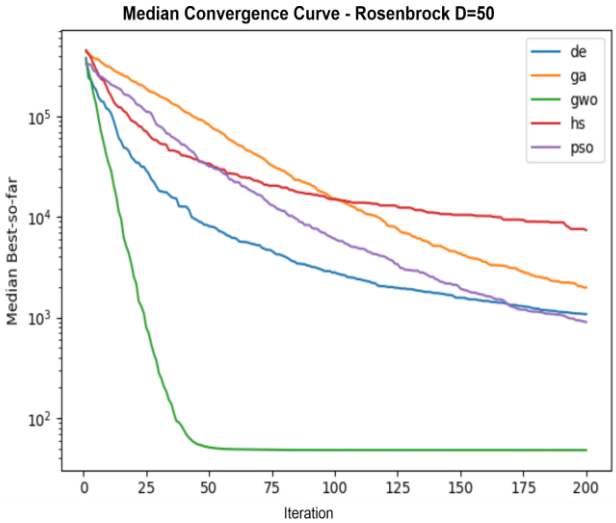

Figure 6. Median convergence curves of optimization algorithms on the Rosenbrock benchmark function in 50 dimensions.

The Rosenbrock function is a difficult, narrow-valley landscape, and the result reflect its challenge. All algorithms were able to reduce the error by several orders of magnitude from their initial values, but only GWO consistently came closest to the optimum. At D=10, GWO achieved the lowest median final error, converging to near the bottom of Rosenbrock's valley within the first few dozen iterations. DE, PSO, and HS also converged to the low tens by the end of the run, albeit more slowly and with slightly higher final errors. GA performed noticeably worse in both speed and final accuracy, its error reduction was more gradual, and by 200 iterations GA's median fitness remained around the high tens. This indicates that even in 10 dimensions, GA often struggled with Rosenbrock's ridge structure, whereas GWO navigated it effectively. Increasing the dimension to 30 and 50 exacerbated the performance gaps on Rosenbrock. GWO remained the top performer, scaling gracefully with dimension, for instance, at D=30 its median final error was on the order of 10^2 and only slightly higher at D=50. In contrast, all other methods showed significantly larger errors in higher dimensions. By the end of 200 iterations at D=30, DE and PSO reached median errors on the order of $10^2 – 10^3$, while HS and GA lagged further behind. This trend persisted at D=50, DE and PSO attained final medians around 10^3, whereas GA and HS plateaued in the 10^3 range, an order of magnitude worse than GWO's result. Notably, GA's median error at D=50 was about 10^3 and HS's was 10^3, reflecting that many runs failed to approach the optimum closely in the allotted iterations. Convergence speed on Rosenbrock was fastest for GWO, the green curve in the convergence plots drops steeply in the first 20–30 iterations and flattens near the optimum region. DE and PSO showed moderate convergence speeds, steadily decreasing error but leveling off at higher plateaus, while GA and HS were slower, with GA in particular improving at a glacial pace early on. In terms of stability, GWO again proved reliable: its runs consistently found good solutions with relatively low scatter. The other algorithms exhibited less stable outcomes, especially in high dimensions, some runs for GA/HS may get stuck in higher regions of the valley, causing the median to remain high. Thus, for Rosenbrock, GWO demonstrated both the fastest and most dependable convergence, whereas GA and HS frequently struggled to make progress, and DE/PSO offered intermediate performance in both speed and accuracy.

Rastrigin Function Optimization Performance

The Rastrigin function posed a significant challenge for most algorithms, highlighting differences in their exploratory abilities. GWO dramatically outperformed the others on this function, achieving by far the lowest errors. At D=10, GWO was the only algorithm that reliably found a near-optimal solution, its median best fitness essentially reached 0. In stark contrast, all other methods stagnated at much higher error levels in 10 dimensions. By 200 iterations, PSO, HS, GA, and DE had median fitness values around 0.3, 0.5, 0.8, and 1.0 respectively, indicating that these algorithms frequently became trapped in Rastrigin's local minima and could not fully escape to the global basin within the run time. GWO's convergence curve, however, shows a sharp drop around iteration 40–60 leading to a near-zero error, implying a successful navigation through the rugged landscape to the true optimum. The presence of oscillations in GWO's error curve during convergence suggests that GWO explored multiple regions but ultimately stabilized to the best value, underscoring its balance of exploration and exploitation on this multimodal terrain. At higher dimensions, the performance disparities on Rastrigin remained pronounced. GWO continued to find much better solutions than the others: for example, at D=30 its median final fitness was around 5–10, and even at D=50 GWO's median error stayed near 10. In contrast, the competing methods all yielded relatively large errors at termination, showing little evidence of approaching the true optimum in these runs. At D=30, HS and GA achieved median finals of roughly 20–30, DE about 30, and PSO was highest around 40. By D=50, these medians degraded further: PSO and GA often remained around 50–60, and even the best of the rest (HS) was near 25–30. The consistently high error plateaus for DE, GA, HS, and PSO indicate that at least half of their runs were stuck in suboptimal minima even after extensive iterations. Moreover, PSO's particularly poor results in higher dimensions suggest a tendency to prematurely converge to suboptimal regions in the Rastrigin landscape, a known vulnerability of PSO without diversification mechanisms. Convergence speed mirrored these outcomes: GWO not only reached far lower errors but did so faster whereas the other algorithms showed slow, incremental improvements that flattened out early into a stagnation plateau. Regarding stability, GWO was clearly the most reliable on Rastrigin: the fact that its median is so low implies a high success rate across runs in finding very good solutions. The other algorithms displayed much less stable performance, in the sense that run outcomes were consistently mediocre or in some cases highly variable with no run reaching the true optimum. For instance, one might infer that none of GA's runs found the global minimum at 30 or 50 dimensions, given its consistently high median, GA's search process was not robust against Rastrigin's plethora of local minima. In summary, on the Rastrigin benchmark GWO demonstrated a markedly superior convergence ability and stability, whereas DE, PSO, HS, and GA all struggled, with PSO and GA performing the worst in high dimensions.

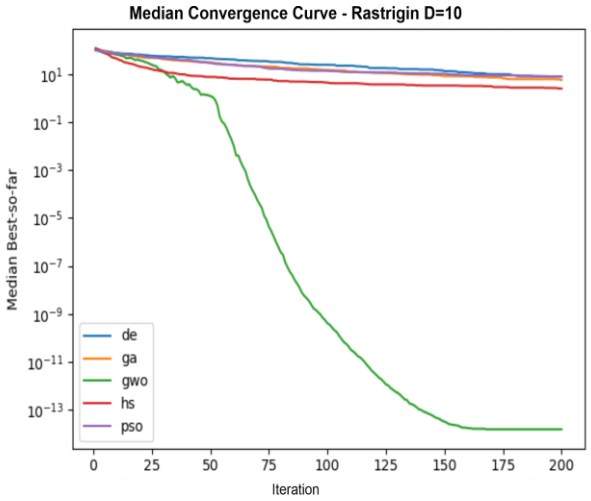

Figure 7. Median convergence curves of optimization algorithms on the Rastrigin benchmark function in 10 dimensions.

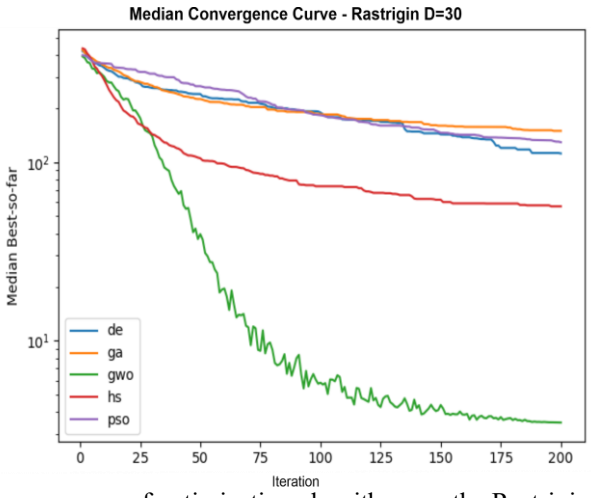

Figure 8. Median convergence curves of optimization algorithms on the Rastrigin benchmark function in 30 dimensions.

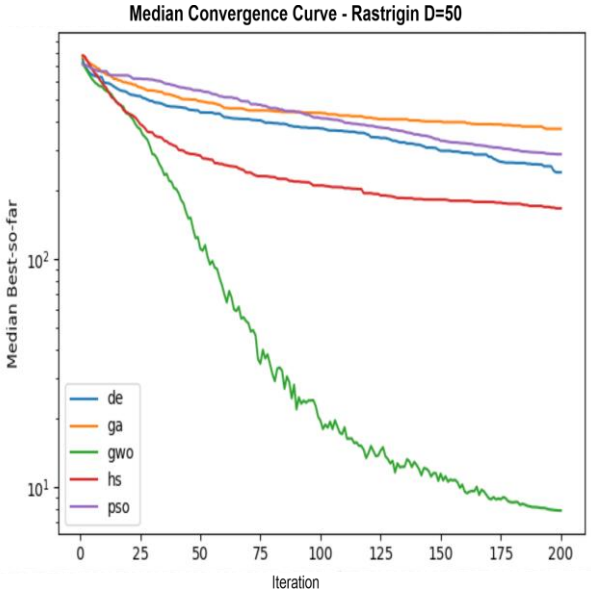

Figure 9. Median convergence curves of optimization algorithms on the Rastrigin benchmark function in 50 dimensions.

Ackley Function Optimization Performance

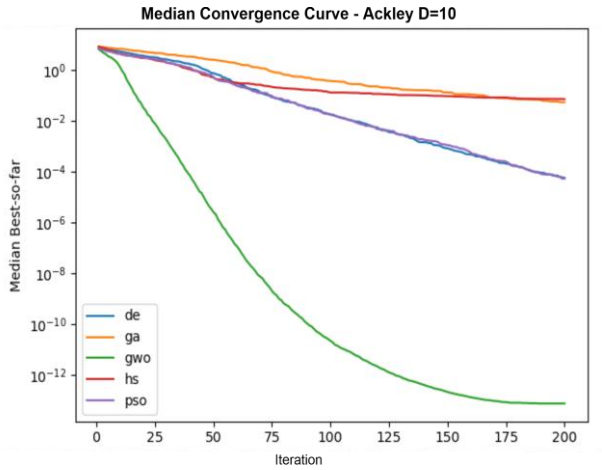

Figure 10. Median convergence curves of optimization algorithms on the Ackley benchmark function in 10 dimensions.

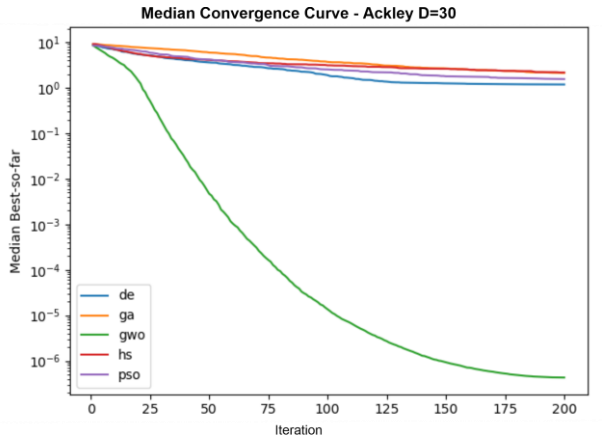

Figure 11. Median convergence curves of optimization algorithms on the Ackley benchmark function in 30 dimensions.

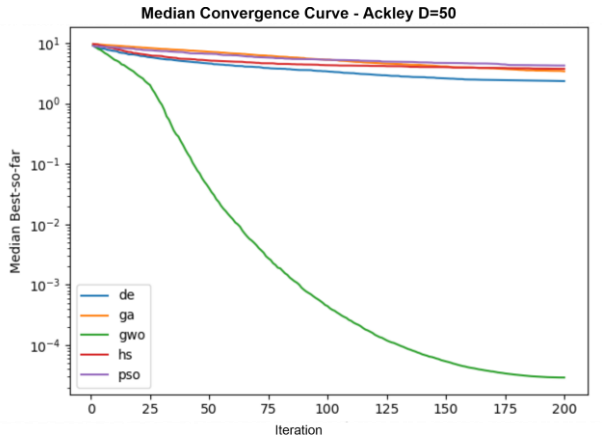

Figure 12. Median convergence curves of optimization algorithms on the Ackley benchmark function in 50 dimensions.

The Ackley function also differentiates the algorithms' performance, though its landscape features a broad basin which can facilitate convergence once an algorithm escapes the peripheral local optima. The results show a similar ranking to Rastrigin: GWO achieved the best performance by a wide margin, while GA consistently performed the worst, with DE, PSO, and HS in between. At D=10, GWO nearly reached the global optimum, effectively solving the function in most runs. DE and PSO also managed to reduce the error substantially for 10 dimensions though these were still several orders of magnitude higher than GWO's result. HS showed a slower

convergence, ending around 10^{-3}. GA, in contrast, barely improved after an initial drop: its median stagnated around 10^{-1}, indicating that GA struggled significantly with Ackley's multimodal landscape even in a low-dimensional case. This poor performance of GA suggests difficulty in maintaining diverse solutions, leading to premature convergence on one of the many local optima. As dimensionality increased, the gap between GWO and the other algorithms remained large. In 30 dimensions, GWO's median final fitness was on the order of 10^{-6}, whereas the next best algorithm (DE) stopped around 10^{-2}. PSO and HS finished with medians in the 10^{-2} range, and GA remained an outlier with an order-of magnitude higher error. By D=50, GWO still obtained a remarkably low median error, showing its scalability and effective exploration of Ackley's search space. All other methods, however, stagnated at much higher error levels by iteration 200: DE, PSO, HS were all clustered with final medians roughly (30–50% of the initial range). GA again performed worst, with a similar 10^{-1} plateau as in lower dimensions, suggesting it failed to make meaningful progress in most runs. The convergence trajectories underscore these differences: GWO's error curve drops rapidly and continuously, whereas GA's curve flattens very early, and DE/PSO/HS show moderate declines that level off well above zero. In terms of run-to-run consistency, GWO provided not only the lowest errors but also reliably so – most GWO runs found near-optimal regions (reflected in the very low median). The other algorithms had more consistent failure to fully converge (especially GA, which consistently got stuck at a relatively high error, yielding a narrow but high-error distribution of final results). One might say GA was "consistently bad" on Ackley, whereas PSO and DE were somewhat inconsistent, occasionally finding better minima but often not, leading to higher median error and presumably larger variance. HS was moderately reliable but its best efforts still fell short of the optima. Overall, Ackley results reinforce the earlier observations: GWO's balance of exploration and exploitation enabled superior and stable convergence even in complex landscapes, while GA's lack of diversity maintenance led to premature convergence and poor outcomes. DE, PSO, and HS achieved intermediate performance, with DE generally a bit better on Ackley (owing to its inherent mutation-based search diversity), and PSO/HS struggling when confronted with many local optima.

Computational Cost and Efficiency Analysis

From a CPU usage perspective, GA was the lightest algorithm with a stable consumption of around 100 percent, meaning it used only a single processor core. In contrast, PSO and GWO were the most CPU intensive algorithms. PSO showed an increase in CPU demand over iterations, rising from 100 percent at the start to a peak of 142 percent by iteration 200. GWO also recorded a peak of 138 percent at iteration 135, while DE and HS ranged between 108 percent and 115 percent. In terms of memory usage, all five algorithms demonstrated good efficiency with average consumption around 37.3 MB. GWO recorded the highest usage at 37.5 MB, followed by HS and PSO at approximately 37.4 MB, while GA and DE remained around 37.3 MB. These differences of a few hundred kilobytes are not practically significant and remain well within the memory capacity of the Raspberry Pi. Therefore, memory was not a limiting factor in these tests and the differences in computational cost were primarily driven by CPU usage. These findings highlight a trade-off between convergence performance and computational efficiency on a resource limited device. GWO, which was the most consistently high performing algorithm across all benchmark functions, also incurred some of the highest CPU utilization.

It leveraged the Pi 5's multicore capabilities to achieve its rapid convergence, essentially trading higher energy and CPU time for better solutions in fewer iterations. PSO and DE, which also performed well (especially on easier functions), similarly used more CPU on average. In contrast, GA's very low CPU usage came at the cost of much slower and poorer convergence; it conserved processor time but often failed to reach the best solutions within the iteration budget. HS occupied a middle ground, both in solution quality and resource use. From a practical standpoint, if a practitioner's priority is to obtain the best possible solution quickly (and the hardware can tolerate heavy CPU loads), algorithms like GWO or DE/PSO are advisable. They will find optima faster but will draw more power and CPU cycles in the process. On the other hand, if minimal CPU usage or thermal output is critical (for example, in battery-powered or thermally constrained scenarios), one might lean toward GA or HS, with the understanding that the solution quality may be suboptimal or require many more iterations to improve. Ultimately, the Raspberry Pi 5 experiments demonstrate that GWO offers an excellent balance of solution quality and reasonable resource use for these benchmark problems, whereas GA is computationally frugal but converges unreliably, and the other methods (DE, PSO, HS) fall on a spectrum between these extremes. This combined evaluation of convergence behavior and computational cost provides a nuanced basis for selecting a metaheuristic algorithm that best meets the specific performance and resource constraints of a given application.

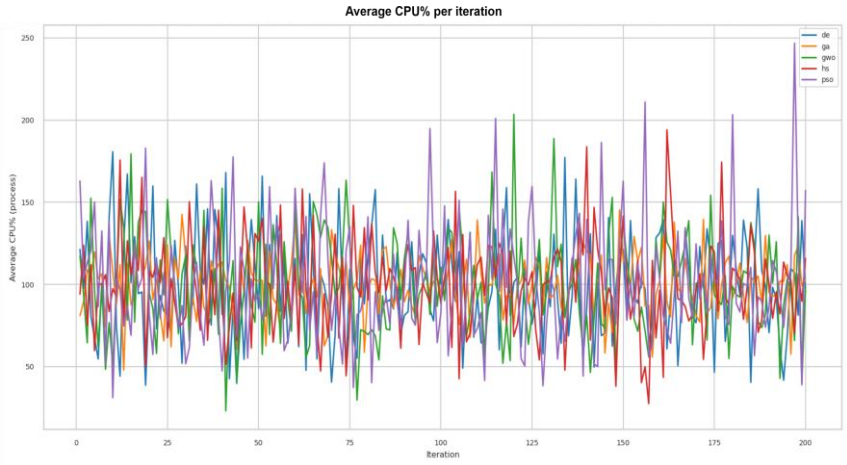

Figure 13. Average CPU% usage per iteration for each algorithm.

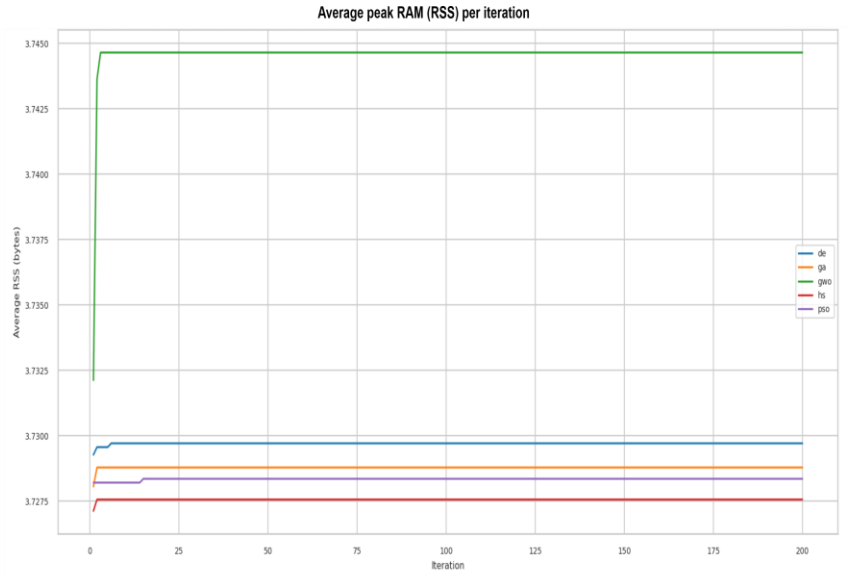

Figure 14. Average RAM (RSS) usage per iteration for each algorithm.

Table 1. Comparative summary

Algorithm	Convergence Quality	CPU Cost	Memory Usage	Recommended Scenario
GWO	Excellent	High	Low	Real-time optimization, high-accuracy requirements, when CPU load is acceptable
PSO	Good	High	Low	Applications needing good convergence with moderate CPU tolerance
DE	Good	Moderate	Low	Rugged or complex landscapes; balanced resource constraints
HS	Fair	Moderate	Low	Moderate resource limits; tasks where slower convergence is acceptable
GA	Poor	Low	Low	Extremely CPU-constrained environments; background or non-critical optimization

From an edge-software perspective, these results support selection rules based on the required convergence reliability and the acceptable CPU budget. GWO is the best fit when solution quality and fast convergence are primary and higher CPU activity is tolerable (e.g., time-sensitive edge optimization or control). PSO and DE provide strong alternatives when a balanced compromise is needed. GA and HS are more appropriate for low-priority or background optimization where CPU load must be minimized, and slower or less reliable convergence is acceptable. As an engineering refinement, adaptive mechanisms such as early stopping or resource-aware tuning can further reduce overhead when deploying these methods on constrained devices.

Conclusion

In summary, this work offers a controlled, device-level benchmark of five metaheuristics (DE, GA, GWO, HS, PSO) on a Raspberry Pi 5 across four canonical functions (Sphere, Rosenbrock, Rastrigin, Ackley) and three dimensions, jointly evaluating convergence behavior and computational cost (CPU/RAM). The results show that GWO is the most effective overall, consistently achieving the fastest and most stable convergence and the best final solution quality, while GA is the most lightweight in terms of CPU and memory but yields the weakest optimization outcomes within the fixed budget. DE and PSO provide solid middle-ground performance, typically converging substantially better than GA and sometimes approaching GWO on easier landscapes, whereas HS is competitive in select cases but generally trails GWO. These findings underscore a clear trade-off on edge devices: GWO is preferred when rapid, reliable convergence is paramount and higher CPU use is acceptable; GA suits ultra-constrained deployments; DE/PSO/HS balance effectiveness and efficiency. Future work should validate these insights on real-world edge tasks, broaden the benchmark suite (e.g., constrained/dynamic functions and higher dimensions), explore adaptive or self-tuning variants that respond to resource/progress signals, and incorporate energy/thermal profiling to identify truly sustainable, edge-ready optimizers.

Scientific Ethics Declaration

* The authors declare that the scientific ethical and legal responsibility of this article published in EPSTEM journal belongs to the authors.

Conflict of Interest

* The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

* This research was fully funded by the Indonesia Endowment Fund for Education (Lembaga Pengelola Dana Pendidikan/LPDP) of the Republic of Indonesia.

Acknowledgements or Notes

* This article was presented as an oral presentation at the International Conference on Basic Sciences and Technology (www.icbast.net) held in Konya/Türkiye on April 02-05, 2026.

* We would like to thank Institut Teknologi Bandung for providing the research facilities and a conducive academic environment. Our appreciation also extends to our colleagues for their support and to the anonymous reviewers for their valuable comments and suggestions.

References

Da S. Medeiros, D. R., Torquato, M. F., & Fernandes, M. A. C. (2020). Embedded genetic algorithm for low-power, low-cost, and low-size-memory devices. *Engineering Reports, 2*(9), e12231.

Fister, I., Vrbancic, G., Hozjan, T., Fister, I., Jr., & Podgorelec, V. (2019). Performance study of bat algorithm running on embedded hardware. In *2019 23rd International Electronics Conference* (pp. 1–4).

Fister, I., Jr., Yang, X.-S., Fister, I., Brest, J., & Fister, D. (2013). A brief review of nature-inspired algorithms for optimization. *Elektrotehniški Vestnik, 80*(3), 116–122.

Henrique de Oliveira Santos, P., Luis Soares, G., Melo Machado-Coelho, T., Augusto Godinho de Oliveira, B., Iakovlevitch Ekel, P., Magalhaes Freitas Ferreira, F., & Augusto Paiva da Silva Martins, C. (2018).

Multi-objective genetic algorithm implemented on a STM32F microcontroller. In *2018 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1–7).

Khalfi, S., Caraffini, F., & Iacca, G. (2023). Metaheuristics in the balance: A survey on memory-saving approaches for platforms with seriously limited resources. *International Journal of Intelligent Systems, 2023*, 5708085.

Latip, R., Aminu, J., Hanafi, Z. M., Kamarudin, S., & Gabi, D. (2024). Metaheuristic task offloading approaches for minimization of energy consumption on edge computing: A systematic review. *Discover Internet of Things, 4*(1), 35.

Minott, D., Siddiqui, S., & Haddad, R. J. (2025). Benchmarking edge AI platforms: Performance analysis of NVIDIA Jetson and Raspberry Pi 5 with Coral TPU. In *SoutheastCon 2025* (pp. 1384–1389).

Punitha, K., Rahman, A., Radhamani, A. S., Nuvvula, R. S. S., Shezan, S. A., Ahammed, S. R., Kumar, P. P., & Ishraque, M. F. (2024). An optimization algorithm for embedded Raspberry Pi Pico controllers for solar tree systems. *Sustainability, 16*(9), 3788.

Author(s) Information

Ziadan Qowi
Bandung Institute of Technology
Bandung, West Java, Indonesia
ORCID iD: https://orcid.org/0009-0004-0423-4354/
*Contact e-mail: *Ziadan.qowi97@gmail.com*

Akhdan Musyaffa Firdaus
Bandung Institute of Technology
Bandung, West Java, Indonesia
ORCID iD: https://orcid.org/0009-0000-1030-9077

Hari Purnama
Bandung Institute of Technology
Bandung, West Java, Indonesia
ORCID iD: https://orcid.org/0009-0005-6474-9840

To cite this article:

Qowi, Z., Firdaus, A.M., & Purnama, H. (2026). A comparative study of metaheuristics on Raspberry Pi 5: results to guide optimization on constrained edge devices. *The Eurasia Proceedings of Science, Technology, Engineering and Mathematics (EPSTEM), 40,* 232-244.