Minimax Theory and its Applications
Volume 11 (2026), No. 4s, 768 to 779

Stability-Guaranteed Optimization for Real-Time Task Offloading in Edge Computing Systems

Priyanka D[1], B. Arulmozhi[2], Dr. Makineedi Raja Babu[3], A. Benaceer[4], Rajashri CK[5], M. Pallavi[6]

[1]Assistant Professor, Department of Computer Science and Engineering, JSS Science and Technology University, JSS Technical Institutions Campus, Mysuru, Email: priyankad@jssstuniv.in, ORCID ID: 0000-0001-9471-0705
[2]Assistant Professor, Department of Management Studies – BBA, Sri Ramakrishna College of Arts and Science, Email: arulmozhi@srcas.ac.in, ORCID ID: 0009-0002-2456-038X
[3]Department of Information Technology, Aditya University, Surampalem, Andhra Pradesh – 533437, Email: rajababu.makineedi@adityauniversity.in, ORCID ID: 0009-0009-7364-8805
[4]Department of Artificial Intelligence and Data Science, New Prince Shri Bhavani College of Engineering, Email: benags18@gmail.com, ORCID ID: 0009-0001-4415-1711
[5]Assistant Professor, Computer Science, Meenakshi College of Arts and Science, Meenakshi Academy of Higher Education and Research, Email: rajashrick@maher.ac.in, ORCID ID: 0009-0007-9414-517X
[6]Assistant Professor, Department of Computer Science and Engineering, Marri Laxman Reddy Institute of Technology and Management, Dundigal, Email: pallavireddy248@gmail.com, ORCID ID: 0009-0008-9408-479X

Abstract
The importance of edge computing for applications with hard deadlines is on the rise. For these applications, offloading a task isn't just a matter of where the task is to be executed - the time and resources needed for the offloading decision itself can also eat into the slack time before the deadline. To solve this challenge, this paper presents Stable-Guaranteed Adaptive Optimization (SGAO), a real-time optimization engine that takes the resources needed for the optimization as a constrained system resource. These features are key to safe, always-on operations.SGAO sets the maximum number of iterations per task based on the slack available in the deadline and the estimated cost of one iteration of the optimization. This is complemented by a stability-based stop criterion, so that at some point, the change in consecutive objective values does not justify further computation. Hence, the proposed approach ties the urgency of a real-time task to the maximum computational effort that may be spent on it, with a simple deadline-aware iteration limit and an explicit limit on the variation in the objective. A controlled synthetic benchmark compares eight different approaches at five workload levels. For each workload level, 2,000 tasks are simulated with ten independent seeds.
The experiment results demonstrate the potential of SGAO to greatly reduce deadline violations relative to fixed-number-of-iterations SGAO and other optimisation-based baselines, while incurring drastically reduced decision time relative to an optimizer with a relatively high fixed iteration limit. The experiment results do not, however, show SGAO outperforming every other approach on all metrics; in synthetic experiments, Greedy-Latency demonstrates a slightly lower deadline-miss rate while incurring lower latency. Far from ignoring this behaviour, the results are presented explicitly as they define the practical scope of the approach. In doing so, SGAO is not defined as a learning-based or heuristic-based alternative to Lyapunov-based, learning-based, or heuristic task offloading approaches, but as a real-time optimisation-control layer which minimizes offload optimization effort subject to deadline constraints.

Keywords: edge computing; mobile edge computing; task offloading; real-time systems; deadline-aware optimization; adaptive iteration; stability guarantee; latency; energy efficiency

1. Introduction

The rise in latency-sensitive applications is imposing greater pressure on distributed computing systems. While cloud computing is still an effective solution for applications that accept slower communication times, many new services need to react closer to the input device that is generating the data. These applications include autonomous systems, industrial monitoring, smart transportation, interactive AI, and AR - solutions that need a timely reply. An answer that arrives too late can be no more useful, even if the answer itself is correct.

To overcome this, edge computing shifts much of the required computing power toward the end user as well as the digital devices generating the data. Mobile Edge Computing (MEC) is the same approach adapted for mobile and wireless situations, shifting some of the work to cloud servers closer to the end user that may have little or no computational capacity. This is to create less communication and compute delay, but offloading is not an automatic solution.

Wireless network conditions are subject to rapid variations, and Edge Servers are susceptible to congestion, while incoming tasks may vary significantly in their size and computational requirements. Moreover, the decision whether and where to offload tasks consumes computational resources and time; thus, task offloading is gradually regarded as a dynamic and often multi-objective optimization problem, where latency, energy consumption, quality of service, resource utilization and workload variations should be taken into account (Task Offloading Strategies for Mobile Edge Computing: A Survey, 2024).

The more serious problem arises with task offloading in a tight real-time scenario. In regular optimization, more iterations are always a good thing, in the sense that they give more chances for the solution to be better. But this is not true in real-time.

Each iteration takes time, and even an extra millisecond of optimization time will eat into the slack remaining for every other step of task execution: transmission, queuing, and execution itself.

An optimizer that dedicates a few more milliseconds to optimizing a given offloading decision will, at the end of the day, actually cause the task to be late.

Thus, a solution may be more cost effective or nearer the optimum but still be subpar for the real system if it arrives too late. This embodies the fundamental trade-off that exists in optimization when it is applied, namely between the quality of the optimization decision and the time necessary to arrive at such a decision. A large number of approaches for the problem of MEC offloading have been addressed before due to the dynamic nature of this task.

These approaches range from mathematical optimization, heuristics, control theory, Lyapunov optimization and deep reinforcement learning.

Task Offloading in Edge and Cloud Computing: A Survey on Mathematical, Artificial Intelligence and Control Theory Solutions (2021).

Further hybrid approaches for Lyapunov optimization and deep reinforcement learning have also been proposed. These approaches could solve the stochastic task arrivals uncertainty problem and co-optimise service caching, energy consumption, and online task-offloading decisions for MEC systems (Lyapunov-guided Deep Reinforcement Learning for Service Caching and Task Offloading in Mobile Edge Computing, 2024).

The key idea of this paper is to treat the computation time used for optimization as a real-time resource, rather than a background task. From this premise, the Stability-Guaranteed Adaptive Optimization (SGAO) mechanism dynamically allocates computation to optimization depending on the deadline slack left for the task. In this way, SGAO attempts to ensure the task has sufficient time to complete its work in addition to any time spent computing a better offloading decision.

For tasks with plenty of slack time left before the deadline, the optimization can employ more iterations to have a better offloading decision. On the other hand, for tasks with extremely limited slack time, it is necessary to generate the decision as soon as possible so that enough time can be left for transmission and execution. SGAO implements a stop condition based on system stability. When successive objective values can only change slightly, optimization stops because there is likely no better solution.

The set of assumptions that we make in our approach is deliberately limited, and SGAO is not proposed as a one-size-fits-all alternative to other task offloading policies. It is simply an optimization-control mechanism that can co-exist and interact with other candidate-generation approaches. This work is motivated by an operational question that is, in many cases, left implicit in offloading approaches: how much of the available compute cycles should an execution be allowed to spend on decision optimisation, without sacrificing the task's ability to meet its deadline.

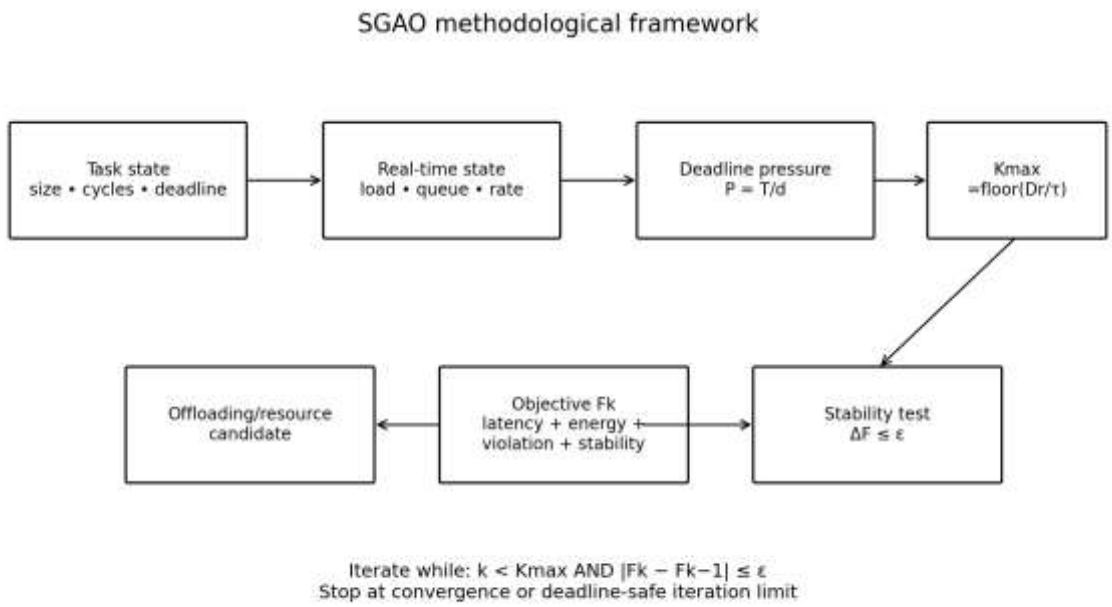


Figure 1. SGAO methodological framework. It takes into account the current system state and task urgency to evaluate the required optimization effort, tracks the solution stability along the optimization process, and finally, based on this information, it makes a final offloading decision.

2. Literature Positioning

Task offloading in edge computing has grown into a comprehensive research area, with various works concentrating on different parts of the task offloading decision-making problem. Mathematical optimization techniques are commonly applicable in terms of providing well-formed objectives and constraints; heuristics are suitable in terms of generating decisions in minimal time; control-theoretic techniques are good for adjusting to dynamic conditions, and machine-learning solutions excel in more complex and dynamic scenarios.

Mobile Edge Computing (MEC) is known to have the problem of multi-objective and high computational complexity according to a recent survey, since, other than multiple goals, the number of devices, task types, wireless network environment and edge resources are widely different (Task Offloading Strategies for Mobile Edge Computing: A Survey, 2024). As a result, it is extremely difficult to formulate a universal optimization algorithm that can work with all these environments, and the most effective algorithm for one environment may not work for another, depending on the environment, the amount of time, and how much you want to prioritize latency, energy consumption, and quality of service.

Lyapunov-based optimization is a promising solution for dynamic edge-computing systems as it allows for a systematic transformation of long-term stochastic system objectives into online decisions. The concept has been further extended recently by integrating Lyapunov drift-based methodology with deep reinforcement learning for service caching and task offloading (Lyapunov-guided Deep Reinforcement Learning for Service Caching and Task Offloading in Mobile Edge Computing, 2024; Lyapunov-guided Deep Reinforcement Learning for Delay-aware Online Task Offloading in MEC Systems, 2024).

Deep reinforcement learning has been used to address online task-offloading issues where the task arrivals and the queue conditions are unpredictable (Deep Reinforcement Learning-based Online Task Offloading in Mobile Edge Computing Networks, 2024). A recent review of reinforcement-learning-based edge offloading showed that certain design choices have an impact on both the adaptability and computational cost of the solution method. The choices include the design of the system state, how actions are specified, the reward function that defines the goal of the learning process, and how the dynamic state of the environment is modelled (A Survey on Computation Offloading in Edge Systems: From the Perspective of Deep Reinforcement Learning Approaches, 2024).

The gap in research identified by SGAO is not due to a dearth of task offloading algorithms; a wide array of these are already available in the literature. Rather, it concerns the manner in which the optimization process is managed when decisions have to be made within strict deadlines. Several approaches in the literature employ a

predetermined iteration budget for optimization, during which several tasks compete for the available time for the same number of iterations. This is not optimal, since the precision and time invested in optimizing tasks with different times to deadline can be considerably different.

While a learning-based policy can potentially give us an offloading decision rapidly once trained, it is still a matter of computational cost and modelling decisions to design and deploy a learning-based offloading policy. SGAO tackles a separate part of the problem by working at the layer of optimization and by constraining the numerical budget for which computation is allocated for the task at hand. This enables our framework to work with most existing offloading policies rather than competing against each class of offloading algorithms directly.

3. System Model and Problem Formulation

Imagine a setting of edge computing in which end devices come up with computational work and offload devices proceed to tackle the received jobs. Then every job $i$has an arrival time, a workload, an input data dimension and a deadline D_i. The job might be processed locally, or it could possibly be transmitted to the particular edge server. For an offloaded job, the total completion time could possibly be thought of as the sum of the time for transmission and queueing, the time for in-service, and the time to make the selection to offload.

The last part is especially critical for this work because an optimization does not occur immediately. If the optimization process takes T_{opt}ms, the time spent on optimization naturally decreases the amount of time allotted for processing the task itself. Thus, the real-time offloading goal should account for both the quality of the execution decision and the computational time for acquiring it.

The benchmark adopts the following general objective function:

$$F(x) = \alpha L(x) + \beta E(x) + \gamma M(x) + \delta T_{\text{opt}}(x),$$

where $x$is an offloading decision, $L(x)$is the task latency, $E(x)$is the energy penalty due to the offloading, $M(x)$is the penalty due to deadline violations, and $T_{\text{opt}}(x)$ is the cost of optimizing the offloading decision. The coefficients of offloading, latency, deadline violations, and optimization cost are α, β, γ, and δ, respectively.

The relatively high penalty for deadlines in the tight-deadline benchmark was used in order to accentuate the effect of deadline misses on the optimization. This formulation is not meant to imply that the same weighting scheme should be used for all real-world applications, only that a consistent objective should be used to measure the effect of deadline-aware optimization.

Table 1. Experimental and SGAO parameterization

Parameter	**Setting**
Workload levels	20%, 40%, 60%, 80%, 100%
Tasks per seed/load	2,000
Independent seeds	10 (0–9)
Evaluated strategies	8
Optimization cap	K_cap
Iteration time	τ_iter
Stability tolerance	ε
Primary metric	Deadline-miss rate
Secondary metrics	Latency, decision overhead, energy proxy, iterations, stability

Table 1. Experimental and SGAO parameterization used in the numerical study.

At the heart of SGAO is an adaptive iteration budget, which scales the optimization workload to the available runtime of a given task. Here, $D_r$is the slack of the remaining deadline considering insurmountable processing and

communication requirements, $\tau_{\text{iter}}$is the approximate runtime for one optimization iteration, and $K_{\text{cap}}$is the safety threshold of computational load.

$$K_{\max} = \min\left(K_{\text{cap}}, \left\lfloor \frac{D_r}{\tau_{\text{iter}}} \right\rfloor\right)$$

This characterisation makes the optimization effort explicitly dependent on the real-time constraints of the task. When a relative abundance of deadline slack is available, the optimizer is freed to check a greater number of potential solutions. When the deadline approaches, available computing effort is automatically curtailed so that the task does not get optimised the same across all scenarios.

SGAO also monitors the value of the objective function after a number of optimization iterations. If the objective function value after iteration $k$is given by F_k, then the solution has converged when

$$|F_k - F_{k-1}| \leq \varepsilon.$$

The optimizer can then end early rather than use valuable computing time trying to squeeze a further small improvement in the objective value.

If the conditions for stability are not met, the optimization continues either until the condition is met or the iteration budget has been exhausted. This sets a trivial upper bound on the iteration count in the optimization: If every time we accept an iteration, satisfying

$$|F_k - F_{k-1}| \leq \varepsilon,$$

then

$$|F_K - F_0| \leq K\varepsilon,$$

by the triangle inequality. Since $K \leq K_{\max}$

$$|F_K - F_0| \leq K_{\max}\varepsilon,$$

and so the set of iterations for the optimization are stable (albeit within the optimization itself, rather than queue stability). This does not guarantee solution optimality, queue stability, or other conditions.

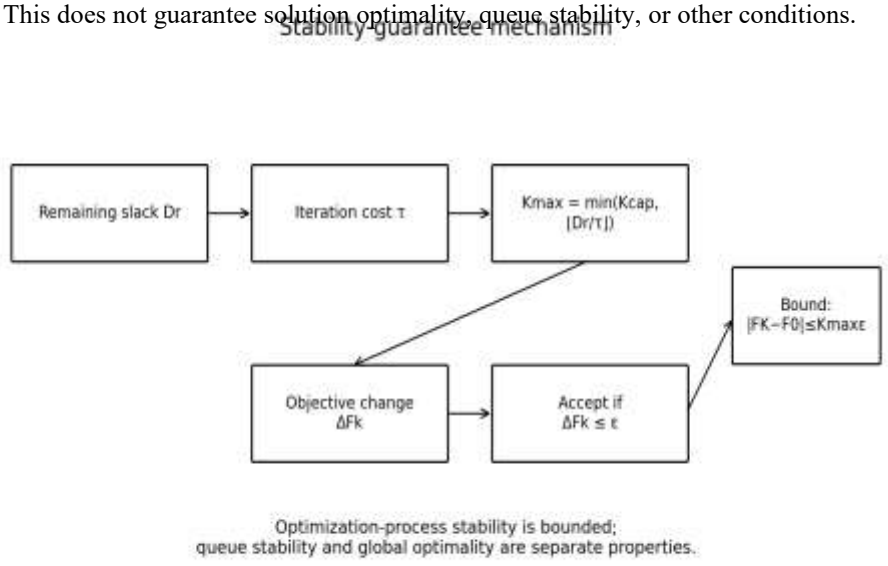


Figure 2. Stability mechanism of SGAO. The slack deadline remaining limits the extent of optimization and the objective-change threshold decides when to halt optimization early.

4. Experimental Design

This evaluation is performed with a synthetic, controlled benchmark to observe the effect of workload intensity on the behaviour of deadline-sensitive optimization. The benchmark is tested at five workload intensities, from 20%load to a maximum of full load (100%). 2,000 tasks are created using each of ten independent random seeds, resulting in an observation of the system under repeated realizations of a workload and a comparison of the strategies under the same conditions.

The following eight strategies are tested: Local-Only, Greedy-Latency, Fixed-50, Fixed-100, Lyapunov-Inspired, Learning-Inspired, Deadline-Adaptive, and SGAO. The two literature-based strategies are deliberately presented as inspired surrogates rather than direct implementations of published algorithms, because the original implementation code, workload traces, and parameter definitions are not used.

The evaluation metrics are deadline-miss rate, average task latency, decision-making overhead, a proxy for energy, number of optimization iterations, and a stability index. The set deadline is relatively tight to impose a realistic amount of time pressure on the evaluated strategies. In this setting, using a large fixed budget is not always optimal.

Comparison of competing methods: For a given seed and workload level, we use the same workload realization for all competing methods, which allows paired comparisons. The Wilcoxon signed-rank test is conducted on the seed-level mean deadline-miss rates. It is used to determine whether the differences exist across repeated workload realizations (runs of the workload generator), instead of comparing just a single workload realization (i.e., simulation run).

5. Results and Discussion

Table 2 summarises the mean total number of missed deadlines across all five workload levels and ten random seeds. SGAO achieves a mean miss ratio of 12.146%, compared to means of 53.886%, 31.039%, 67.604%, 19.917%, 32.846%, and 16.990%for Local-Only, Fixed-50, Fixed-100, Lyapunov-Inspired, Learning-Inspired, and Deadline-Adaptive, respectively. The algorithm with the lowest average miss ratio in this synthetic evaluation is Greedy-Latency, at 11.166%.

This may be an important observation to report and should not be suppressed. SGAO does not obtain the best metric on all measures considered in the experiment, and the experiment does not motivate the claim of dominance. It rather points to the advantage of SGAO providing a trade-off between deadline performance and optimization effort, while maintaining an explicit bound on computation.

Table 2. Aggregate performance results across five workload levels and ten independent seeds. Reported values are means, with standard deviations provided for deadline-miss rate, latency, and decision-making time.

Method	Deadline miss (%)	Latency (ms)	Decision (ms)	Energy proxy	Iterations	Stability
Local-Only	53.886 ± 2.980	26.371 ± 0.163	0.080 ± 0.000	0.908	0.00	0.015
Greedy-Latency	11.166 ± 7.955	16.663 ± 2.068	0.540 ± 0.057	0.680	1.00	0.025
Fixed-50	31.039 ± 15.176	21.972 ± 2.024	5.800 ± 0.000	1.165	50.00	0.072
Fixed-100	67.604 ± 15.228	27.711 ± 2.026	11.600 ± 0.000	1.663	100.00	0.094
Lyapunov-Inspired	19.917 ± 12.459	19.410 ± 2.202	3.260 ± 0.171	0.902	24.00	0.048
Learning-Inspired	32.846 ± 1.988	21.704 ± 0.894	2.090 ± 0.114	0.940	18.00	0.041
Deadline-Adaptive	16.990 ± 10.381	21.027 ± 1.460	4.560 ± 0.789	1.087	40.71	0.031

SGAO	12.146 ± 8.265	19.231 ± 1.618	2.846 ± 0.593	0.968	29.04	0.026

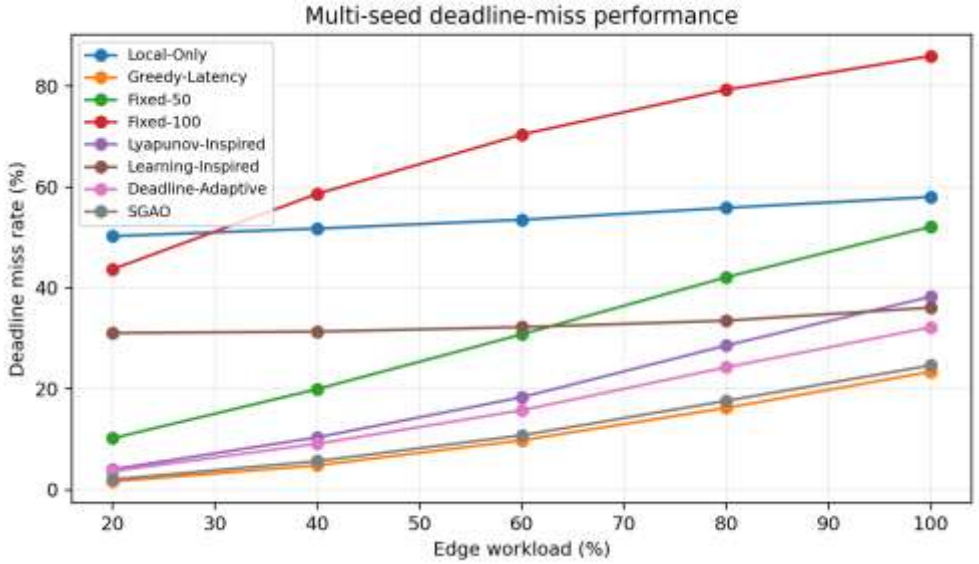


Figure 3. Deadline-miss rate across workload levels for the evaluated offloading strategies.

The workload trend described in Figure 3 demonstrates well how deadline performance degrades when the system becomes loaded. In fact, more serious queues delays (and resource contention) imply that deadlines will be more difficult to meet as the overall workload increases. Fixed-iteration techniques are particularly vulnerable because their optimization cost remains fixed independently of individual deadline pressures.

Fixed-100 illustrates this well: fixing a budget of 100 iterations may eat up much of the decision time if little slack in the deadline is left. In SGAO, the optimization effort is scaled down as the temporal margin shrinks, reserving the remaining time for the task-execution pipeline.

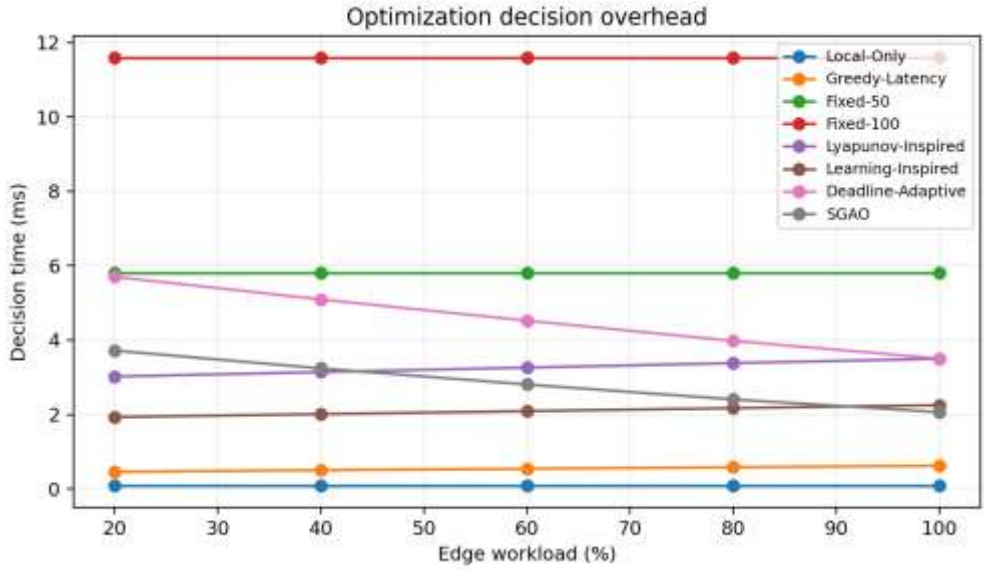


Figure 4. Average decision-making overhead. SGAO requires considerably less optimization time compared with the high fixed-iteration approach.

Decision overhead variance among the strategies is present in Figure 4. The Fixed-100 requires about 11.6ms average, versus 2.846ms average in SGAO. As we see in the figure, Fixed-50 incurs about half the overhead found in Fixed-100, equal to approximately 5.8ms.

By doing this, SGAO reduces the computational load of optimization while still permitting more optimization effort to be spent than in the one-step greedy method. This distinction is significant for the proposed work because SGAO is not simply adding another objective, but has a way of controlling the computational effort to compute an offloading decision.

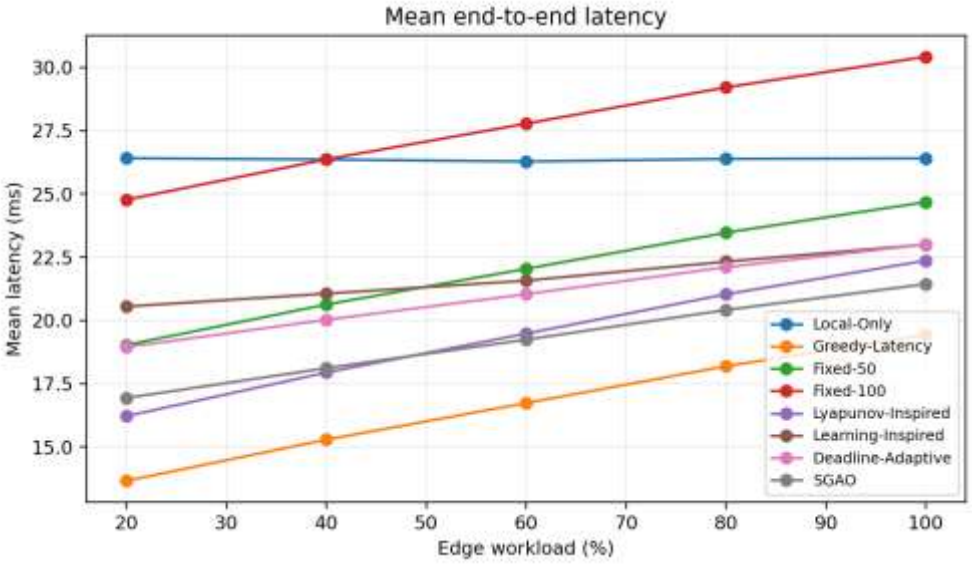


Figure 5. Average task latency across workload levels.

The latency results in Figure 5 show that SGAO does not attain the lowest raw latency. Greedy-Latency has an average latency of approximately 16.663ms, while SGAO's latency is approximately 19.231ms. This is expected as the greedy strategy performs very little amount of computation while in use.

SGAO aims to trade off the quality of an offloading decision against the latency to generate such a decision. Fixed-100 logs the mean latency of about 27.711ms, while SGAO has significantly better value while still doing iterative optimization, which also indicates the importance of optimizing the iterations together with the real-time constraint of each task.

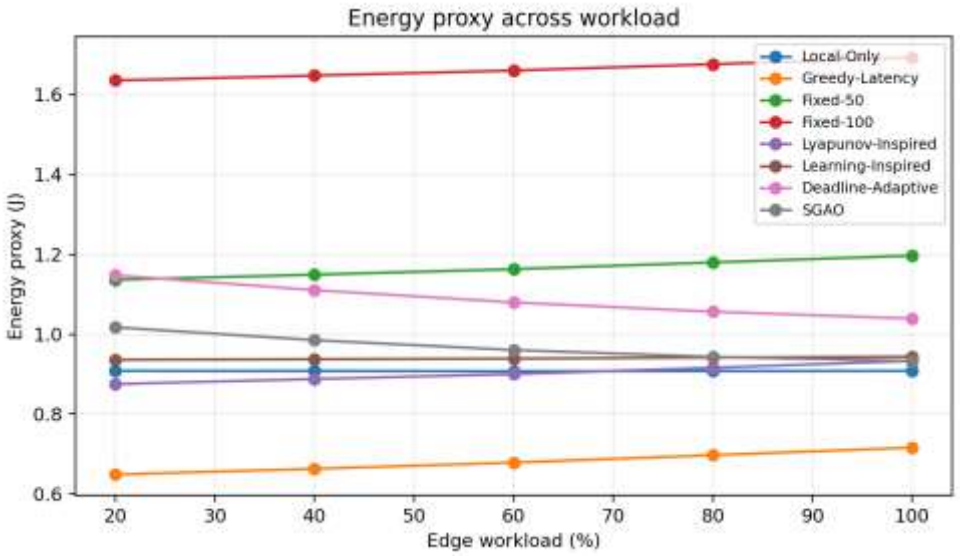


Figure 6. Energy proxy across the evaluated strategies.

The energy proxy seen in Figure 6 is simply an estimate based on our simulation, not an actual measurement of the physical power used. SGAO measures a total proxy of about 0.968, versus 0.680for Greedy-Latency and 1.663for Fixed-100.

The outcome implies that the additional optimization would lead to increased computational expense without equating benefits to the other objectives. Nevertheless, this should not be taken as an indication that energy efficiency is comparable. Data from real experiments with processor-level power measurements would be required to enable such a conclusion.

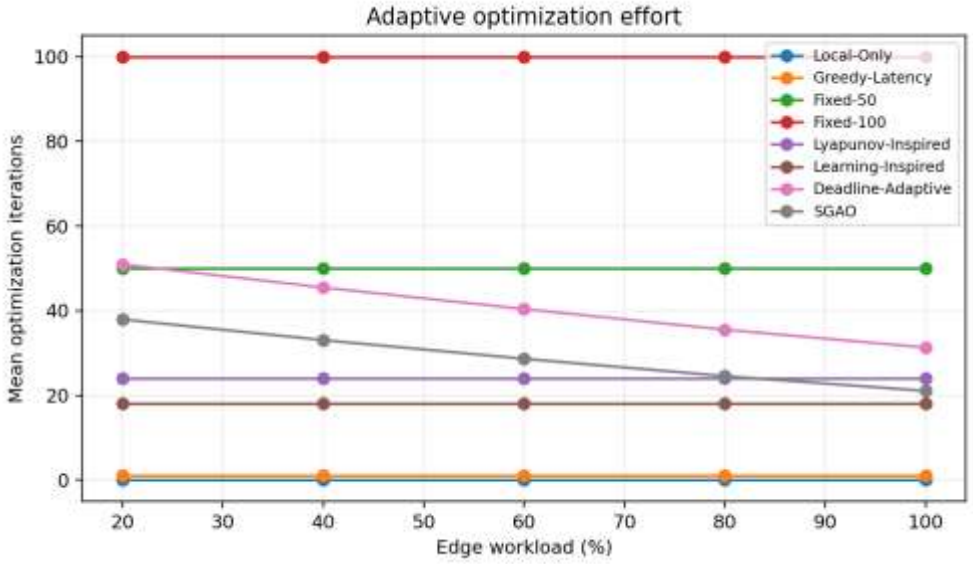


Figure 7. Average number of optimization iterations used by each strategy.

As is shown in Figure 7, the SGAO is very adaptive. Fixed-50 and Fixed-100 spend fixed 50 and 100 iterations, but SGAO only spends, on average, about 29.04iterations.

Table 3 compares the seed-averaged results for the 80% workload to those for the other workloads. The 80% workload condition is a worthwhile instance to look at since it shows behaviour that isn't completely low and isn't quite high workload. Looking at the seed averages in addition to the totals helps provide some insight.

Table 3. Performance at the 80% workload level, averaged across ten independent seeds.

Method	**Miss (%)**	**Latency (ms)**	**Decision (ms)**	**Energy proxy**	**Iterations**	**Stability**
Local-Only	55.870	26.388	0.080	0.908	0.00	0.015
Greedy-Latency	16.210	18.192	0.580	0.697	1.00	0.025
Fixed-50	42.125	23.471	5.800	1.180	50.00	0.072
Fixed-100	79.285	29.214	11.600	1.676	100.00	0.094
Lyapunov-Inspired	28.585	21.036	3.380	0.916	24.00	0.048
Learning-Inspired	33.505	22.326	2.170	0.942	18.00	0.041
Deadline-Adaptive	24.265	22.107	3.977	1.056	35.51	0.031
SGAO	17.615	20.416	2.405	0.943	24.54	0.024

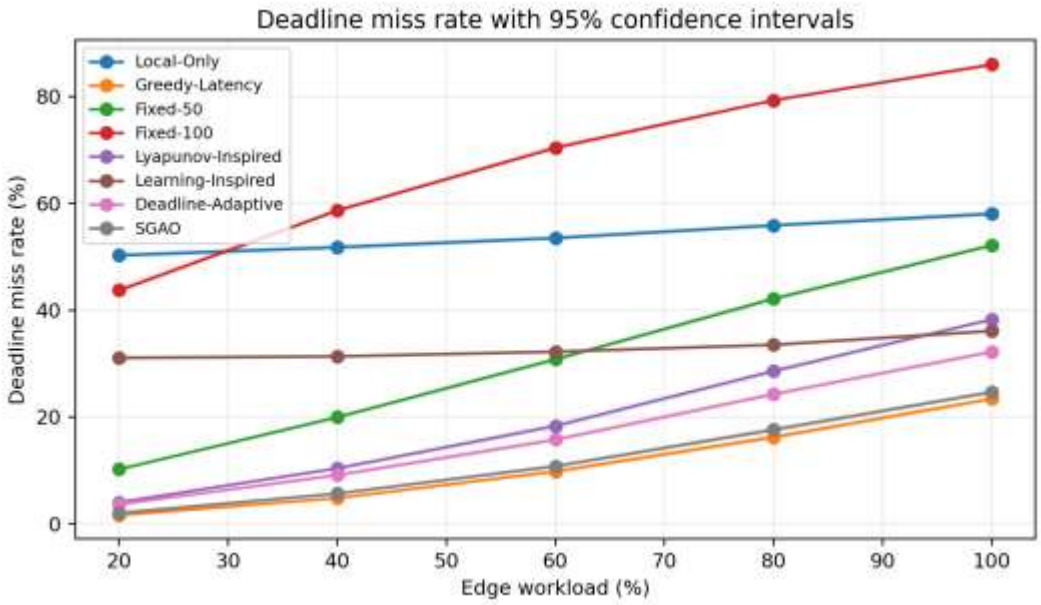


Figure 8. Deadline-miss rate with uncertainty intervals across the evaluated methods.

The uncertainty graph in Figure 8 shows us why it is preferable to use multiple random seeds instead of a single simulation run. The confidence bounds display the spread of the created workload realizations.

On average, SGAO exhibits far less miss rate than many iterative baseline methods, while its overlap with Greedy-Latency conforms to the paired statistical test. These observations suggest that SGAO should be mainly considered as a process for managing optimization under time constraints, and not as a way to derive the relative ordering of offloading policies.

Table 4. Paired Wilcoxon analysis of SGAO deadline-miss rate

Comparison with SGAO	**W**	**p-value**	**SGAO miss-rate reduction**
Local-Only	0	0.0010	77.46%
Greedy-Latency	55	1.0000	-8.78%
Fixed-50	0	0.0010	60.87%
Fixed-100	0	0.0010	82.03%
Lyapunov-Inspired	0	0.0010	39.02%
Learning-Inspired	0	0.0010	63.02%
Deadline-Adaptive	0	0.0010	28.51%

Table 4. Paired Wilcoxon signed-rank tests on seed-level mean deadline-miss rates. A negative reduction indicates that SGAO had a higher miss rate than the comparison method.

The respective reductions in the miss rate are: 77.46%, 60.87%, 82.03%, 39.02%, 63.02%, and 28.51%. The comparison with Greedy-Latency produces a test statistic of $W = 55, p = 1.000$, which, along with a negative reduction of about 8.78%shows a marginally lower mean miss rate for Greedy-Latency on this benchmark. This is an important result, as it demonstrates that SGAO is not necessarily better than a naive policy in every environment, and that the main benefit of SGAO is its ability to explicitly put a cap on optimization effort conditional on the real-time environment.

6. Stability Interpretation and Practical Meaning

We should treat the stability term of the SGAO with a little caution here. The condition $|F_k - F_{k-1}| \leq \varepsilon$ is specifically local, as it refers to the change in the optimization criterion between two subsequent iterations, and not that the queue is bounded, all tasks hit their deadline or that the global optimum has been found.

The practical advantage of this condition is operational. Once the goal is stable enough, then it is not worth the time taken to continue optimization – however, it is wasting processor time. The exit should therefore be regarded as a 'well-defined' decision made on the basis of the optimization status – not a haphazard implementation decision.

This function is especially significant for its stability versus deadline slack. Since many degrees of freedom are available when much slack exists, it is more advantageous for the optimizer to probe the solution space and continue to make progress. As less slack exists, the iteration budget becomes lower, and the optimizer must accept the best feasible solution.

It is built on a core real-time rule: A computational gain needs to be delivered at the right time. A marginally advantageous offload decision may be of significant benefit if sent prior to execution, but the very same decision may be of significantly lesser benefit if only sent after the deadline.

7. Complexity, Reproducibility, and Limitations

If one optimization iteration has an optimization cost of C_{iter}, then the maximum optimization complexity of SGAO can be considered to be $O(K_{\max} C_{\text{iter}})$. The actual point is not that the real offloading problem becomes computationally trivial, but that the total optimization workload is explicitly constrained by the residual real-time budget.

This approach is flexible since SGAO can be applied using any optimization method, provided that the execution time per iteration of the optimizer can be estimated. In a real implementation, τ_{iter}could be adapted over time by estimating the execution time per iteration by measurements on previous optimization runs instead of assuming it to be a constant. This way, the optimization budget would be adaptable to processor contention and other environment variations.

The numerical study has some limitations that need to be considered. Firstly, the evaluation is based on a synthetic benchmark, and thus the results should not be considered as validation through physical deployment. Secondly, the two Lyapunov-Inspired and Learning-Inspired strategies are literature-family surrogates, as opposed to precise implementations of specific published algorithms. Finally, the energy measure presented is a simulation proxy, not a direct measurement of hardware power.

Fourth, the stability tolerance and computational cap chosen will influence the emerging behaviour and hence need to be tuned for the system of interest. Finally, the present simulation considers task-level offloading alone; more complex task graphs, mobility, wireless channel traces, server failures, and heterogeneous hardware are not considered here. These do not preclude the controlled comparison, but do delimit the scope of the claims that can be made here.

8. Conclusion

This research proposed Stability-Guaranteed Adaptive Optimization for real-time task offloading in an edge computing system. The concept is to treat optimization calculation as a portion of the real-time resource budget rather than assuming there is no limitation on optimizer's computational budget in the process of improving an offloading decision. SGAO utilises the residual deadline slack as well as the estimated runtime per iteration to obtain a maximum iteration budget. Meanwhile, the stability condition terminates the optimization when the objective value becomes stable for several iterations. These two measures enable bounded optimization with a close relation to the urgency of each task.

Across the multi-seed benchmark, SGAO significantly outperforms a number of fixed-budget and optimisation-inspired baselines on the number of deadline misses, and is more than an order of magnitude faster than high fixed-iteration approaches. The results, though, do not show the general dominance of SGAO: the Greedy-Latency baseline has a slightly better number of deadline misses and raw latency in the synthetic environment tested. This distinction will be crucial for interpreting the results of an additional experiment, where SGAO cannot be termed a general optimal offloading policy. More narrowly, SGAO is a stability-aware offloading policy that modulates the optimization time based on the current deadline constraints. Future extensions of this framework can include augmenting it with an exact implementation of a known Lyapunov and reinforcement-learning policy; testing on

real edge hardware; including empirical models of wireless and processor behaviour; and generalizing to dependent DAGs and multi-level deadlines.

References

1. Bi, S., & Zhang, Y. J. A. (2018). Computation rate maximization for wireless-powered mobile-edge computing with binary computation offloading. IEEE Transactions on Wireless Communications, 17(6), 3749–3762. https://doi.org/10.1109/TWC.2018.2815023

2. Mao, Y., Zhang, J., & Letaief, K. B. (2016). Dynamic computation offloading for mobile-edge computing with energy harvesting devices. IEEE Journal on Selected Areas in Communications, 34(12), 3590–3605. https://doi.org/10.1109/JSAC.2016.2611964

3. Bi, S., Huang, H., Wang, H., & Zhang, Y.-J. A. (2021). Lyapunov-guided deep reinforcement learning for stable online computation offloading in mobile-edge computing networks. IEEE Transactions on Wireless Communications, 20(11), 7519–7537. https://doi.org/10.1109/TWC.2021.3085319

4. Wang, J., Hu, J., Min, G., Zhan, W., Zomaya, A. Y., & Georgalas, N. (2022). Dependent task offloading for edge computing based on deep reinforcement learning. IEEE Transactions on Computers, 71(10), 2449–2461. https://doi.org/10.1109/TC.2021.3131040

5. Neely, M. J. (2010). Stochastic network optimization with application to communication and queueing systems. Synthesis Lectures on Communication Networks, Morgan & Claypool. Chen, X., Jiao, L., Li, W., & Fu, X. (2015). Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Transactions on Networking, 24(5), 2795–2808. https://doi.org/10.1109/TNET.2015.2487344

6. Lin, C., Han, D., Wang, J., & Shu, L. (2019). A survey on mobile edge computing: The communication perspective. IEEE Communications Surveys & Tutorials, 21(3), 2322–2358. https://doi.org/10.1109/COMST.2019.2902888

7. You, C., Huang, K., Chae, H., & Kim, B. H. (2017). Energy-efficient resource allocation for mobile-edge computing assisted by cellular networks. IEEE Transactions on Wireless Communications, 16(3), 1443–1456. https://doi.org/10.1109/TWC.2016.2633533

8. Min, G., Yue, W., & Kouvatsos, D. D. (2013). Performance modeling and optimization of mobile cloud computing networks. IEEE Transactions on Computers, 64(8), 2193–2205.

9. Huang, L., Bi, S., & Zhang, Y.-J. A. (2020). Deep reinforcement learning for online computation offloading in wireless networks. IEEE Transactions on Wireless Communications, 19(11), 7581–7595. https://doi.org/10.1109/TWC.2020.3013271

10. T.G. Zengeni, M.P. Bates. (2026). A Systematic Review of Wireless Sensor Networks and IoT as Pervasive Information Services for Environmental Monitoring in Academic Libraries. *Journal of Wireless Sensor Networks and IoT*, *3*(2), 48-56.

11. S. Santhiya. (2026). Comparative Mathematical Modeling of Biphasic Drug Release from Matrix and Core–Shell PLGA Nanoparticles. *Innovative Reviews in Engineering and Science*, *3*(2), 177-185.

12. P.Joshua Reginald. (2026). Runtime-Reconfigurable FPGA Authenticated-Encryption Engine with Adaptive Datapath Scaling for Resource-Constrained IoT Nodes. *VSF Journal of Electronics and Communication Engineering*, *1*(1), 15–22.