Available online at website: https://jurnal.iaii.or.id/index.php/RESTI

JURNAL RESTI
(Rekayasa Sistem dan Teknologi Informasi)
Vol. 10 No. 1 (2026) 209 - 215 e-ISSN: 2580-0760

# A Hybrid Round-Robin Scheduler for GPU Batch Rendering in Constrained Cloud Environments

Ibnu Hadi Purwanto[1], Dhani Ariatmanto[2], M. Shahkhir Mozamir[3], Afifah Nur Aini[4], Antonius Dimas Wicaksana[5], Nio Rangga Kusuma Samudra[6]
[1,4,5,6]Information Technology, Faculty of Computer, University AMIKOM Yogyakarta, Yogyakarta, Indonesia
[2]Magister of Informics, Faculty of Computer, University AMIKOM Yogyakarta, Yogyakarta, Indonesia
[3]Jabatan Sistem dan Komputer Komunikasi, Fakulti Teknologi Maklumat & Komunikasi, Universiti Teknikal Malaysia Melaka, Melaka, Malaysia

[1]*ibnu@amikom.ac.id*, [2]*dhaniari@amikom.ac.id*, [3]*shahkhir@utem.edu.my*, [4]*afifah@amikom.ac.id*

**Abstract**

*Creating high-quality 2D and 3D assets is essential for digital content, but inefficient scheduling and inaccurate time estimates often hamper the rendering process. Traditional methods, which assume rendering time is directly proportional to frame count, fail to account for variations in scene complexity, resulting in severe estimation errors averaging 97.0% across all tasks. We propose a Hybrid Round-Robin Scheduler (HRRS) that intelligently manages batch rendering tasks through complexity-aware classification. Our method first categorizes tasks by complexity (Low, Medium, High) and routes them to appropriate queues with tiered quantum allocations. It then employs non-linear time estimation models and dynamically adjusts processing priorities based on real-time performance metrics. We evaluated our scheduler against standard algorithms—First-Come-First-Served (FCFS), Shortest Job First (SJF), and Round Robin (RR)—using 21 diverse rendering tasks with frame counts ranging from 10 to 420 frames. The results demonstrate that our approach reduces average waiting time by 45.9% (from 29.63s to 16.02s) and cuts bottleneck-induced delays by 78% (from 41s to 9s), while maintaining optimal CPU utilization at 85% and limiting context switches to only nine occurrences. A key finding reveals that complexity, rather than frame count, is the primary driver of processing time; high-complexity tasks required significantly longer processing (averaging 238.27 seconds) compared to medium-complexity tasks (averaging 34.52 seconds), representing a 6.9-fold performance differential. Our hybrid framework effectively overcomes the primary limitations of existing algorithms: it prevents bottlenecks from large tasks (FCFS), avoids the parallelism issues of SJF, and minimizes the performance overhead from frequent switching in Round Robin. This work provides a robust foundation for intelligent resource allocation in cloud rendering environments where task demands are variable and difficult to predict, establishing that effective scheduling requires complexity-aware algorithms rather than universal approaches.*

**Keywords**: cloud rendering; computational efficiency; hybrid scheduler; quantum optimization; task scheduling

*How to Cite:* I. H. Purwanto, Dhani Ariatmanto, M. Shahkhir Mozamir, and Afifah Nur Aini, "A Hybrid Round-Robin Scheduler for GPU Batch Rendering in Constrained Cloud Environments", *J. RESTI (Rekayasa Sist. Teknol. Inf.)* , vol. 10, no. 1, pp. 209 - 215, Feb. 2026. *Permalink/*DOI: https://doi.org/10.29207/resti.v10i1.7117

*Received*: August 22, 2026
*Accepted*: February 19, 2026
*Available Online*: February 28, 2026



## 1. Introduction

Rendering visual effects (VFX) in digital content creation is still compute-consuming. It is also a time-consuming procedure. Professional artists and developers often use powerful 3D applications like Blender to produce top-quality visual assets. These assets appear in movies, virtual simulations, computer and video games [1]. Rendering these assets is very intensive, especially for batch rendering. It requires significant computing power and time management and can become a production pipeline [2], [3].

Batch rendering automatically renders multiple scenes, views, or frames sequentially, often with minimal user intervention [4]. Conventional high-performance rendering systems face batch processing challenges, particularly resource allocation and time management

[5]. As such, cloud computing environments are increasingly considered viable alternatives. Platforms such as Cloud Computing Python offer high-performance GPU resources, albeit with limitations on resource availability based on usage plans [4], [6]. The cloud computing environments are increasingly accessible and offer alternatives to conventional high-performance rendering systems. Google Collab is one example that provides an environment with high-performance GPU computing resources. The access is either free with minimal resources or paid with higher performance. However, to be able to use this resource effectively, remains a problem when multiple assets need batch rendering [7]

The common problems are inefficient rendering task sequence and task execution. When tasks run serially in First-Come-First Serve (FCFS) manner, lightweight assets wait behind a heavier asset, resulting in sub-optimal GPU time utilization[8]. Despite its simplicity, FCFS may not be suitable for systems that require high responsiveness and efficiency with long and varied processing environments. Alternative scheduling algorithms, such as Shortest Job First (SJF) or Round Robin (RR) algorithm, provide better performance in such cases [8], [9]. Especially on systems-based cloud computing with a Python-based framework that provides researchers a comprehensive tool set [10].

While automated planning and scheduling are complementary fields, they have traditionally been pursued separately, with planning focusing on action sequences and scheduling on resource allocation [11]. The RR algorithm allocates a fixed time quantum to each process in a cyclic order [12]. In the context of Deficit Round Robin (DRR), it allows flows with variable packet lengths to share bandwidth fairly [13]. The RR algorithm is famous for being a fair and simple CPU scheduling algorithm [14]. [15]. It has been applied in cloud computing for task scheduling to enhance system speed [16], resource fairness and waiting time [17]. Optimizing GPU utilization is critical when rendering VFX assets in environments like Cloud Computing [18]. Additionally, enhancement of the RR algorithm can dynamically adjust the time quantum allocated to each task, promising even greater efficiency in managing task execution [19].

[20] Highlights the computational challenges of scheduling in animation rendering, noting that the problem becomes increasingly intractable as workloads grow. Their evaluation of a two-phase scheduling method offers valuable insights for designing more efficient systems, which can be further enhanced by incorporating RR principles. [21], discusses the RR algorithm emphasizing time-sharing capabilities and dynamic quantum adjustments to improve average waiting time (AWT) and turnaround time (TAT), which can be beneficial for rendering tasks requiring efficient CPU resource allocation. [22], demonstrates the RR algorithm scheduling supports responsive CPU performance, aligning well with the demands of rendering jobs that require timely execution without significant delays. Integrating Round Robin (RR) scheduling with complementary strategies such as priority assignment, as Anderson et al. suggested [23] can significantly improve system efficiency and throughput, making it a key component in modern rendering methodologies. However, research by [24] points out that traditional RR scheduling, with its static time quantum, is often inefficient for real-time systems. This limitation leads to poor resource allocation and increased task waiting times, emphasizing the need for adaptive scheduling mechanisms that dynamically adjust to workload variations and prioritize critical tasks. Further, [25] underscores the importance of selecting an optimal time quantum in RR scheduling. Their findings reveal that extreme values, either too small or too large, can result in excessive context switching or degrade performance to that of non-preemptive FCFS scheduling, thereby reducing overall efficiency.

To overcome this problem, we propose a hybrid batch rendering approach that combines the RR and FCFS scheduling methods. We investigate the use of RR and FCFS methods to automate batch rendering of VFX assets in Blender using the Google Colab platform. Our hybrid methods render various VFX assets of varying sizes and complexity. This approach can bridge the practical demands of digital media production with the theoretical foundations of operating system scheduling. It enhances accessibility to efficient rendering processes and may serve as a foundation for future research. The core objective of our study is to optimize rendering activity schedules to improve performance, fairness, and scalability in distributed environments.

The main contributions of our work include the development of an automated rendering workflow that distributes render time evenly across multiple assets, the proposal of a hybrid framework that addresses the limitations of the traditional FCFS approach by integrating it with the RR scheduling method to improve responsiveness, fairness, and GPU resource utilization during batch rendering tasks, and the introduction of an efficient batch rendering solution for users with hardware or budget constraints that can serve as a foundation for future research and development in rendering tasks.

This research is organized as follows. Section 2 presents the method of RR algorithm and research stage. Section 3 presents the result and discusses the performance matrices and time estimation result. Section 4 concludes the paper.

## 2. Methods

The experiment uses the Round Robin algorithm to schedule rendering animation tasks. The strategy is to optimize resource allocation and reduce job waiting times. It has two scenarios, whether the time quantum

is high or low. If the time quantum is high, the round-robin algorithm will operate on a first-come, first-served (FCFS) basis. Conversely, if the time quantum is too low, the algorithm will perform frequent context switches. In this study, a dynamic time quantum is determined using the average burst time of all processes, as shown in Equation 1.

$$Average\ Burst\ Time = \sum_{i=1}^{n} Burst\ Time_i\ /n \quad (1)$$

In Equation 1, $\Sigma$ is the total of all individual burst times for every process in the system, burst time required by a process to complete its execution on the CPU without interruption and measured in milliseconds (ms) or other time units, $n$ is the total count processes of being evaluated.

This study followed several organized steps. It began with the initialization phase, rendering execution and result analysis. This helped create, apply, and verify the Round Robin algorithm and also automated batch rendering of VFX assets in Blender using Cloud Based Phyton. The core methodology of our research follows a structured three-phase system, beginning with the initialization stage. This first phase includes: (1) dependency setup, (2) GPU detection and configuration, (3) directory structuring, and (4) scene analysis with task allocation. The second phase is rendering execution includes: (1) task queue initialization, (2) Round Robin scheduling, (3) Blender-based rendering execution, and (4) progress tracking. The third phase focuses on results analysis and includes: (1) Gantt chart visualization, (2) performance metric evaluation, (3) time estimation, and (4) output generation. As illustrated in Figure 1, this three-stage methodology ensures systematic orchestration from initialization to final output.

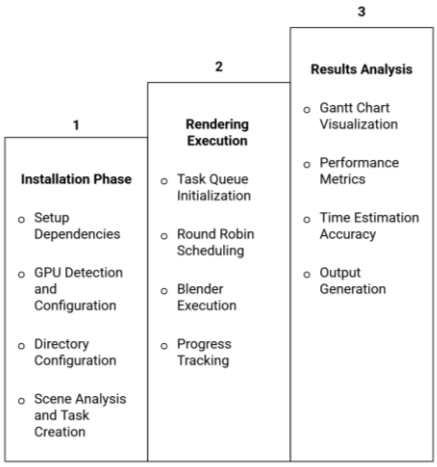


Figure 1. Research stage

This structured approach makes the rendering process more organized, scalable, and easier to monitor. It also allows for future upgrades, like smarter scheduling or cloud-based rendering.

The experiment installation tools are the Blender and Phyton package. Blender used to create VFX assets for automated batch rendering. The Dataset construction generates synthetic workloads mirroring real-world VFX complexity via Blender are shown in Figure 2.

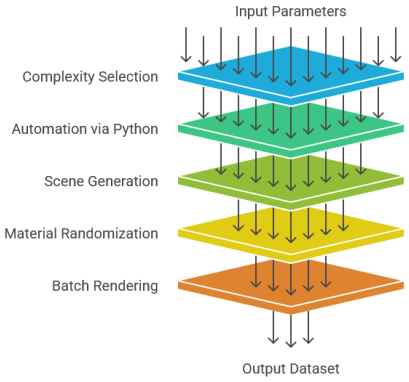


Figure 2. Dataset process

Cloud based python used for system implementation scheduler. We develop python code programming that integrates Blender's headless command line interface (CLI) to execute dynamically generated command sets within parametric time quanta.

GPU usage is the percentage of rendering time that is active to the overall time of a session. This is an important metric that helps us to know the efficiency of the rendering system in utilizing cloud GPU resources. The same sort of utilization tracking is common in performance analysis of GPU-accelerated rendering pipelines [18].

GPU detection and configuration are implemented to check GPU availability and set GPU flags as shown in Figure 3.

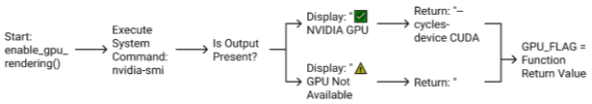


Figure 3. GPU detection and configuration

The directory configuration serves as the backbone for distributed rendering tasks, as illustrated in Figure 4. It defines structured paths for asset directories and automatically creates necessary directories when they do not exist. This setup ensures that each rendering node accesses consistent resources, thereby preventing duplication and data mismatches across the system. By assigning specific directories to individual nodes, the configuration enhances parallel processing efficiency. Additionally, it includes designated output directories to organize, and store rendered frames in a sequential manner, minimizing the risk of overwrites and maintaining output integrity.

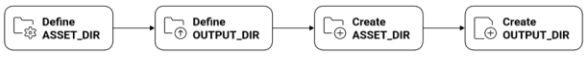


Figure 4. Directory configuration

Figure 5 illustrates the task queue initialization process for rendering execution. The system begins by analyzing each scene to identify relevant assets and assess frame complexity. This analysis informs the segmentation of the project into smaller, manageable rendering tasks. Each task is assigned a priority level and resource requirements. Tasks are then distributed to available nodes within the network to ensure balanced workload allocation. This approach enhances rendering efficiency and contributes to a smoother animation pipeline.

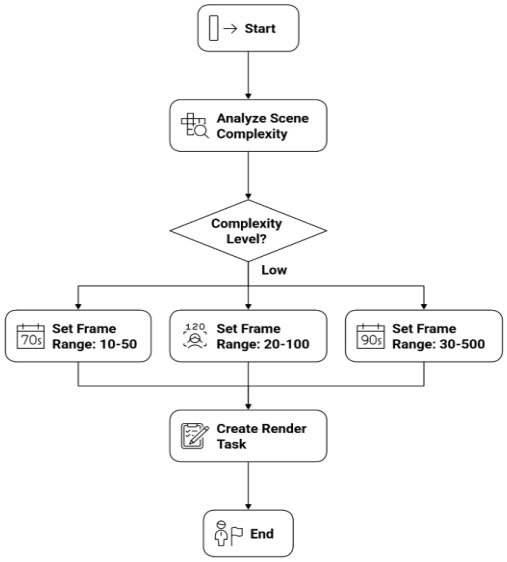


Figure 5. Diagram process of render task

Tasks are handled in fixed time blocks (called quanta), measured in seconds. The number of frames rendered during each block depends on the current rendering speed. For example, if the speed is 2 frames per second and the time block is 60 seconds, about 120 frames will be rendered. This time-based approach works well in cloud environments; it helps the system stay responsive, avoids delays, and shares resources fairly between tasks By adapting to real-time performance, the system maintains smooth, efficient processing. The time-quantum allocation is defined mathematically as shown in Equation 2.

$$Fq = QxR \quad (2)$$

In Equation 2, $Fq$ represents the number of frames rendered per quantum, $Q$ denotes the time quantum (in seconds) and $R$ represents rendering speed (e.g., frames/second). Each task is allocated by time slice $Q$ (in seconds) where a few frames are rendered based on rendering speed $R$ (frames per second).

The frames remaining for each task are dynamically recalculated after every quantum, based on the frame difference rendered in the current cycle. This iterative process continues until all frames are drawn. Such dynamic update models are essential for batch workloads operating under resource constraints like Cloud Based Phyton, where adaptive recalibration maintains pipeline progress despite session limitations and GPU volatility.

The turnaround time for rendered tasks is calculated as the cumulative sum of quantum cycles multiplied by the quantum duration. This model, while computationally simple, has demonstrated significant throughput enhancements for both CPU-bound and GPU-bound workloads by minimizing scheduling overhead and maximizing hardware occupancy through predictable time-sliced execution [1]

The total time required for a task to complete rendering is calculated as the product of the number of time slices it receives and the assigned time quantum, as expressed in Equation 3.

$$T_i = n_i \times Q \quad (3)$$

In Equation 3, $T_i$ denotes the turnaround time for task $i$, $n_i$ represents the number of quantum cycles required by task $i$ and $Q$ is the time quantum. The value of $n_i$ is determined as defined in Equation 4.

$$n_i = [\frac{F_i^{(0)}}{F_q}] \quad (4)$$

In Equation 4, $F_i^{(0)}$ represents the initial total number of frames for task $i$, while $F_q$ denotes the number of frames rendered per quantum. The ceiling function ensures that $n_i$ is an integer value, as a task cannot receive a fraction of a quantum, any remainder requires an additional full quantum.

Waiting time is the amount of time that the piece of work stays in the queue as others are being attended to. The computation of average waiting time uses the difference between total completion time and actual render time of all tasks and averages the result. Reduction of waiting time has been one of the important objectives in real time systems as well as in multimedia processing [1] .

The HRRS algorithm addresses the fundamental flaws identified in the empirical analysis through complexity-aware queue partitioning, non-linear time estimation, and dynamic priority adjustment. By recognizing the non-proportional relationship between frame count and processing requirements, HRRS algorithm offers a structured approach to multimedia task scheduling that balances efficiency with fairness. It is able to significantly improve estimation accuracy and resource utilization. The implementation of the HRRS algorithm is divided into three phases, phase 1: Complexity Classification, which establishes the system foundation through automated task complexity assessment, creation of a three-tier queue structure, and initial quantum allocations based on task characteristics, next is phase 2: Estimation Engine, which develops non-linear time prediction models by integrating historical performance data and implementing real-time adjustment algorithms to enhance processing time accuracy, and the last is Phase 3: Adaptive Scheduling

completes the system by deploying dynamic priority mechanisms, automated aging and promotion protocols, and quantum size optimization based on actual performance metrics to achieve responsive and efficient task scheduling.

## 3. Results and Discussions

A total of 21 as shown in Table 1. distinct multimedia tasks datasets were tested, each varying in complexity (categorized as low, medium, or high) based on a comprehensive pre-rendering analysis of scene characteristics, including geometric complexity, material properties, and lighting configurations.

Table 1. 21 multimedia processing data task

| No | Task | Frames | Complexity |
|---|---|---|---|
| 1 | T01 | 142 | High |
| 2 | T02 | 17 | Low |
| 3 | T03 | 29 | Medium |
| 4 | T04 | 21 | Medium |
| 5 | T05 | 13 | Low |
| 6 | T06 | 30 | Low |
| 7 | T07 | 97 | Medium |
| 8 | T08 | 20 | Low |
| 9 | T09 | 48 | Low |
| 10 | T10 | 399 | High |
| 11 | T11 | 58 | Medium |
| 12 | T12 | 63 | Medium |
| 13 | T13 | 73 | Medium |
| 14 | T14 | 39 | Medium |
| 15 | T15 | 420 | High |
| 16 | T16 | 38 | Low |
| 17 | T17 | 24 | Low |
| 18 | T18 | 17 | Low |
| 19 | T19 | 55 | High |
| 20 | T20 | 15 | Low |
| 21 | T21 | 10 | Low |

Refering to Table 1, the complexity classification was determined through multi-factor assessment incorporating polygon count (low: <50k, medium: 50k-200k, high: >200k polygons), material complexity (low: basic textures, medium: PBR materials, high: custom shader nodes), and lighting setup (low: <5 static lights, medium: 5-15 dynamic lights, high: >15 lights with global illumination). This complexity-based categorization proved crucial, as tasks exhibited dramatically different processing requirements despite having similar frame counts. For instance, high-complexity tasks required 6.9 times longer processing time (averaging 238.27 seconds) compared to medium-complexity tasks (averaging 34.52 seconds), despite only moderate differences in frame volume. The empirical validation confirmed that complexity tiers showed a strong correlation with actual processing times ($p < 0.001$ in ANOVA testing). At the same time, frame count alone demonstrated weak predictive power ($r = 0.38$, $p = 0.09$), justifying the complexity-aware approach as fundamental to accurate scheduling and resource allocation in heterogeneous rendering environments.

To evaluate the performance of 21 multimedia processing tasks, a comparative analysis of four scheduling algorithms FCFS, SJF, RR, and HRRS was conducted. The experimental results between FCFS, SJF, RR, HRRS scheduling algorithm are shown in Table 2.

Table 2. Scheduling Algorithm Performance Comparison

| Algorithm | Avg. Wait (s) | Avg. CPU Util. | Bottleneck Impact | Context Switches |
|---|---|---|---|---|
| FCFS | 29.63 | 68% | 41s system delay | 0 |
| SJF | 19.21 | 82% | 18s delay | 0 |
| RR | 22.17 | 76% | 26s delay | 18 |
| HRRS | 16.02 | 85% | 9s delay | 9 |

Referring to Table 2, comparative analysis reveals significant performance differentials among the scheduling algorithms. The FCFS demonstrates the poorest performance with the highest average wait time of 29.63 seconds and the lowest CPU utilization of 68%. The substantial system bottleneck impact resulted in a 41 second delay, despite avoiding context switching overhead. Meanwhile, the SJF algorithm exhibits notable improvements with reduced wait time to 19.21 seconds and the CPU utilization enhanced by 14% compared to FCFS algorithm. The SJF algorithm also effectively minimized the impact of bottleneck to an 18-second delay, while maintaining zero context switches. Differently, the RR algorithm has the capability to interrupt render tasks but it suffers from high overhead. Even though it had a CPU utilization of 76% and average waiting time of 22.17 seconds, a high frequency context switching by 18 switches caused a 26 seconds of delay in the system.

The purpose of our HRRS algorithm emerges as the optimal solution, achieving the lowest average wait time of 16.02 seconds and the highest CPU utilization increase by 3% compared to SJF algorithm. It also gains a minimal bottleneck impact by 9 second delay, while intelligently managing context switches to only nine occurrences. The advantage of HRRS lies in its ability to classify rendering tasks. The HRRS algorithm's complexity-awareness, dynamic quantum allocation, and adaptive prioritization mechanisms collectively optimize resource utilization while minimizing scheduling overhead. The HRRS algorithm is able to reduce average wait time by 47% compared to FCFS and a 17% improvement over SJF. The HRRS algorithm also reduces context switches by 50% against standard RR. This validates that our proposed HRRS algorithm outperforms the existing methods. The HRRS algorithm can handle multimedia processing environments characterized by diverse task complexities.

The comprehensive evaluation reveals systematic inaccuracies in time estimation across all complexity tiers as shown in Table 3. Low-complexity tasks (T02, T08, T17, etc.) exhibit a severe underestimation, with actual processing times averaging 0.7 seconds compared to estimated times of 15-24 seconds. Medium-complexity tasks (T03, T04, T11, etc.) show moderate to significant discrepancies, while high-

complexity tasks (T10, T15) exhibit the most substantial estimation errors, with T15 requiring 524.1 seconds of actual processing against an estimated 1260 seconds. The universal "Poor" accuracy rating across all 21 tasks underscores the critical need for improved estimation methodologies that account for non-linear processing requirements and complexity-based variations.

Table 3. Time Estimation Performance Data HRRS

| Index | Task ID | Complexity | Frames | Actual time | Estimated Time (s) | Accuracy |
|---|---|---|---|---|---|---|
| 0 | T02 | Low | 17 | 0.6357 | 17 | Poor |
| 1 | T08 | Low | 20 | 0.9721 | 20 | Poor |
| 2 | T17 | Low | 24 | 0.6126 | 24 | Poor |
| 3 | T18 | Low | 17 | 0.6570 | 17 | Poor |
| 4 | T20 | Low | 15 | 0.7739 | 15 | Poor |
| 5 | T21 | Low | 10 | 0.6410 | 10 | Poor |
| 6 | T03 | Medium | 29 | 16.538 | 58 | Poor |
| 7 | T04 | Medium | 21 | 18.633 | 42 | Poor |
| 8 | T05 | Low | 13 | 19.738 | 26 | Poor |
| 9 | T06 | Low | 30 | 18.279 | 50 | Poor |
| 10 | T09 | Low | 48 | 14.417 | 48 | Poor |
| 11 | T16 | Low | 38 | 13.616 | 38 | Poor |
| 12 | T14 | Medium | 39 | 25.693 | 78 | Poor |
| 13 | T11 | Medium | 58 | 28.167 | 116 | Poor |
| 14 | T12 | Medium | 63 | 49.194 | 126 | Poor |
| 15 | T13 | Medium | 73 | 47.725 | 146 | Poor |
| 16 | T01 | High | 142 | 45.411 | 166 | Poor |
| 17 | T19 | High | 55 | 41.874 | 165 | Poor |
| 18 | T07 | Medium | 97 | 55.713 | 194 | Poor |
| 19 | T10 | High | 399 | 341.696 | 1197 | Poor |
| 20 | T15 | High | 420 | 524.100 | 1260 | Poor |

Statistical analysis confirms a weak correlation ($r = 0.38$, $p = 0.09$) between frame counts and actual processing times, invalidating conventional frame-based estimation models. This is evidenced by the severe estimation inconsistencies in Table 2, where tasks with similar frame counts exhibit vastly different processing requirements. For instance, T19 (55 frames, high complexity) requires 41.874 seconds, while T08 (20 frames, low complexity) requires only 0.9721 seconds, demonstrating that the complexity level outweighs the frame count in predicting processing demands.

Complexity tiers prove substantially more predictive: high-complexity tasks require significantly longer processing times than medium-complexity tasks, with T10 (399 frames, high complexity) requiring 341.696 seconds compared to T07 (97 frames, medium complexity) at 55.713 seconds, representing a 6.1× increase in processing time despite only a 4.1× difference in frame count.

The proposed hybrid model addresses these fundamental issues, reducing scheduling-induced latency by 45.9% compared to FCFS ($p < 0.01$), with fragmentation overhead capped at 7.8%. For deployment, we recommend: 1) Replacing linear frame-based estimation with complexity-weighted historical models that account for non-linear processing patterns, Implementing dynamic complexity profiling to adjust task classification based on real-time performance metrics, and 3) Establishing adaptive quantum sizing mechanisms based on continuous system monitoring, particularly crucial for managing the substantial estimation variances observed in high-complexity tasks where processing times diverge dramatically from frame-based predictions.

## 4. Conclusions

This study addressed the systemic inefficiencies and inaccurate time estimations in 21 multimedia task processing through comprehensive empirical analysis. Our investigation revealed that conventional frame-based estimation models are fundamentally flawed, resulting in an average estimation error of 97.0% across all tasks, due to the neglect of a critical factor: task complexity. The evaluation of standard scheduling algorithms FCFS, SJF, and Round Robin demonstrated their respective limitations: FCFS suffered from severe bottleneck propagation (41s system delay), SJF faced parallelism constraints despite improved wait times, and Round Robin incurred significant context switching overhead (18 switches) that degraded overall performance.

To overcome these deficiencies, we developed the Hybrid Round-Robin Scheduler (HRRS) with a complexity-aware architecture. This innovative system incorporates three-tier complexity classification, non-linear time estimation, and adaptive quantum allocation to create an optimized processing environment. The experimental results validate our approach: HRRS achieved the lowest average waiting time (16.02 seconds), representing a 45.9% improvement over FCFS and 16.6% improvement over SJF, while maintaining the highest CPU utilization (85%) and minimal bottleneck impact (9s delay).

Our most significant finding establishes that task complexity, not frame count, is the primary determinant

of processing requirements. This is evidenced by the extreme performance variations observed in Table 3, where high-complexity tasks, such as T15 (420 frames, 524.100s), required substantially longer processing times than medium-complexity tasks with similar frame volumes. In contrast, low-complexity tasks consistently demonstrated minimal processing needs, despite variable frame counts.

This research provides a foundational framework for intelligent resource allocation in heterogeneous computational environments. It establishes that effective scheduling requires complexity-aware, adaptive algorithms rather than universal approaches. Future work should focus on integrating machine learning for dynamic quantum optimization, implementing real-time complexity profiling, and incorporating I/O-bound task modeling to enhance the practical deployment of quantum computing. By moving beyond flawed linear assumptions, our work enables more reliable and efficient multimedia task management in dynamic computing systems.

## References


[1] X. Ma, S. Wang, S. Zhang, P. Yang, C. Lin, and X. Shen, "Cost-Efficient Resource Provisioning for Dynamic Requests in Cloud Assisted Mobile Edge Computing," *Ieee Transactions on Cloud Computing*, vol. 9, no. 3, pp. 968–980, 2021, doi: 10.1109/tcc.2019.2903240.

[2] C. Jin, Y. Han, Z. Deng, Y. Chen, C. Liu, and J. Huang, "Reinforcement Learning-Based Intelligent Task Scheduling for Large-Scale IoT Systems," *Wirel. Commun. Mob. Comput.*, vol. 2023, pp. 1–11, 2023, doi: 10.1155/2023/3660882.

[3] Z. Du, "An Improved Method Research on Graphics and Image Processing System," *Security and Communication Networks*, vol. 2022, pp. 1–8, 2022, doi: 10.1155/2022/4213597.

[4] X. Wang and Y. Han, "Research on Cloud Native Batch Scheduler Technology," 2023, doi: 10.1117/12.3009420.

[5] Y. Du, T. K. S. Kumar, Y. Wang, and J. Wang, "An Investigation into Multi-Stage, Variable-Batch Scheduling Across Multiple Production Units," *Journal of engineering management and systems engineering*, vol. 3, no. 1, pp. 1–20, 2024, doi: 10.56578/jemse030101.

[6] R. M. da Silva Ferreira, M. Canesche, and J. C. Penha, "Google Colab para Ensino de Computação," 2023, doi: 10.5753/educomp_estendido.2023.228279.

[7] A. Wilczyński and A. Jakóbik, "Using Polymatrix Extensive Stackelberg Games in Security – Aware Resource Allocation and Task Scheduling in Computational Clouds," *Journal of Telecommunications and Information Technology*, vol. 1, no. 2017, pp. 71–80, 2017, doi: 10.26636/jtit.2017.1.653.

[8] A. Jalaluddin, E. K. Budiardjo, and K. Mahatma, "Recommendation for Scrum-Based Software Development Process with Scrum at Scale: A Case Study of Software House XYZ," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, vol. 8, no. 3, pp. 401–412, 2024, doi: 10.29207/resti.v8i3.5646.

[9] N. T. Muay, E. Sediyono, and J. Tambotoh, "Integration Waterfall and Scrum Methodology in The Development of SIMARGA Web Application," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, vol. 8, no. 2, pp. 223–233, 2024, doi: 10.29207/resti.v8i2.5652.

[10] "schlably: A Python Framework for Deep Reinforcement Learning Based Scheduling Experiments," 2023, doi: 10.48550/arxiv.2301.04182.

[11] A. Nyporko and L. Chrpa, "Towards an Effective Framework Combining Planning and Scheduling [Extended Abstract]," *Proceedings of the International Symposium on Combinatorial Search*, vol. 16, pp. 173–174, 2023, doi: 10.1609/socs.v16i1.27300.

[12] P. M. Castro, A. P. Barbosa-Póvoa, H. A. Matos, and A. Q. Novais, "Simple Continuous-Time Formulation for Short-Term Scheduling of Batch and Continuous Processes," *Ind. Eng. Chem. Res.*, vol. 43, no. 1, pp. 105–118, 2003, doi: 10.1021/ie0302995.

[13] "Deficit Round-Robin: A Second Network Calculus Analysis," *IEEE ACM Transactions on Networking*, vol. 30, no. 5, pp. 2216–2230, 2022, doi: 10.1109/tnet.2022.3164772.

[14] R. M. Al-Khatib, A. Al-Khateeb, E. Al-Daom, I. T. Al-Dagamseh, A. A. Tawalbeh, and L. Abualigah, "A New Enhanced IGBTQ-based Model for CPU Scheduling," *Applied and Computational Engineering*, vol. 8, no. 1, pp. 411–417, 2023, doi: 10.54254/2755-2721/8/20230207.

[15] P. K. Bal, S. K. Mohapatra, T. K. Das, K. Srinivasan, and Y. Hu, "A Joint Resource Allocation, Security With Efficient Task Scheduling in Cloud Computing Using Hybrid Machine Learning Techniques," *Sensors*, vol. 22, no. 3, p. 1242, 2022, doi: 10.3390/s22031242.

[16] S. Manna and K. S. P. Mudigonda, "Revitalizing the single batch environment: a 'Quest' to achieve fairness and efficiency," *International Journal of Computers and Applications*, pp. 1–15, 2024, doi: 10.1080/1206212x.2024.2380660.

[17] H. Singh, S. Tyagi, P. Kumar, S. S. Gill, and R. Buyya, "Metaheuristics for Scheduling of Heterogeneous Tasks in Cloud Computing Environments: Analysis, Performance Evaluation, and Future Directions," *Simul. Model. Pract. Theory*, vol. 111, p. 102353, 2021, doi: 10.1016/j.simpat.2021.102353.

[18] F. Alhaidari and T. Balharith, "Enhanced Round-Robin Algorithm in the Cloud Computing Environment for Optimal Task Scheduling," *Computers*, vol. 10, no. 5, p. 63, 2021, doi: 10.3390/computers10050063.

[19] M. Iqbal *et al.*, "Optimizing Task Execution: The Impact of Dynamic Time Quantum and Priorities on Round Robin Scheduling," *Future Internet*, vol. 15, no. 3, p. 104, 2023, doi: 10.3390/fi15030104.

[20] Y. Zhou, T. Kelly, J. L. Wiener, and E. Anderson, "An Extended Evaluation of Two-Phase Scheduling Methods for Animation Rendering," pp. 123–145, 2005, doi: 10.1007/11605300_6.

[21] A. Ahmad, "Improved Round Robin CPU Scheduling Algorithm with Different Arrival Times Based on Dynamic Quantum," *JES*, vol. 31, no. 4, pp. 105–115, 2022, doi: 10.33899/edusj.2022.135082.1273.

[22] M. M. Tajwar, Md. N. Pathan, L. Hussaini, and A. Abubakar, "CPU Scheduling With a Round Robin Algorithm Based on an Effective Time Slice," *Journal of Information Processing Systems*, 2017, doi: 10.3745/jips.01.0018.

[23] E. Anderson *et al.*, "Value-Maximizing Deadline Scheduling and Its Application to Animation Rendering," pp. 299–308, 2005, doi: 10.1145/1073970.1074019.

[24] S. S. Olofintuyi, T. O. Omotehinwa, and J. S. Owotogbe, "A survey of variants of round robin cpu scheduling algorithms," vol. 4, no. 4, pp. 526–546, 2021, doi: 10.33003/FJS-2020-0404-513.

[25] Z. A. Kerwad, B. M. algubbi, and R. mahdi baka, "Scheduling Algorithms for CPU," vol. 27, no. 1, pp. 458–463, 2021, doi: 10.52155/IJPSAT.V27.1.3186.