# Task Scheduling for AI Workloads

**Deven Shah**
*Professor Mehrdad Nourai, CS575 A1, Boston University*
Date of Submission: December 2, 2025

**Abstract**

The rapid growth of AI workloads has exposed limitations in traditional CPU and GPU scheduling. CPUs provide fairness and responsiveness but struggle with parallelism and memory-intensive operations, while GPUs deliver high-throughput execution yet rely on CPU coordination for tasks such as data preprocessing and kernel management. Hybrid CPU–GPU scheduling addresses these challenges by dynamically distributing tasks to leverage both CPU flexibility and GPU parallelism. This study evaluates CPU-only, GPU-only, and hybrid approaches across benchmarks for workload scaling, composition, resource constraints, and real-world AI scenario test suites. Results from experts and experimentation show that hybrid scheduling dramatically improves throughput for compute-bound, parallel workloads while maintaining high GPU utilization, though latency-sensitive inference tasks may incur slight overhead. These findings underscore the importance of adaptive, workload-aware scheduling strategies for hybrid AI architectures.

***Keywords:*** *Task Scheduling, Parallel Computing, Thread Management, Resource Allocation*

## 1. Introduction

The rapid growth of artificial intelligence workloads has exposed limitations in traditional CPU and GPU scheduling mechanisms. While CPU schedulers excel at balancing fairness and responsiveness for general-purpose computing, they struggle to efficiently manage the massive parallelism and memory-intensive operations characteristic of modern neural network training. Conversely, GPUs provide high-throughput execution for parallelizable tasks through warp-level scheduling and specialized hardware such as Tensor Cores, yet they rely on effective coordination with CPUs for tasks like data preprocessing, kernel launching, and I/O management.

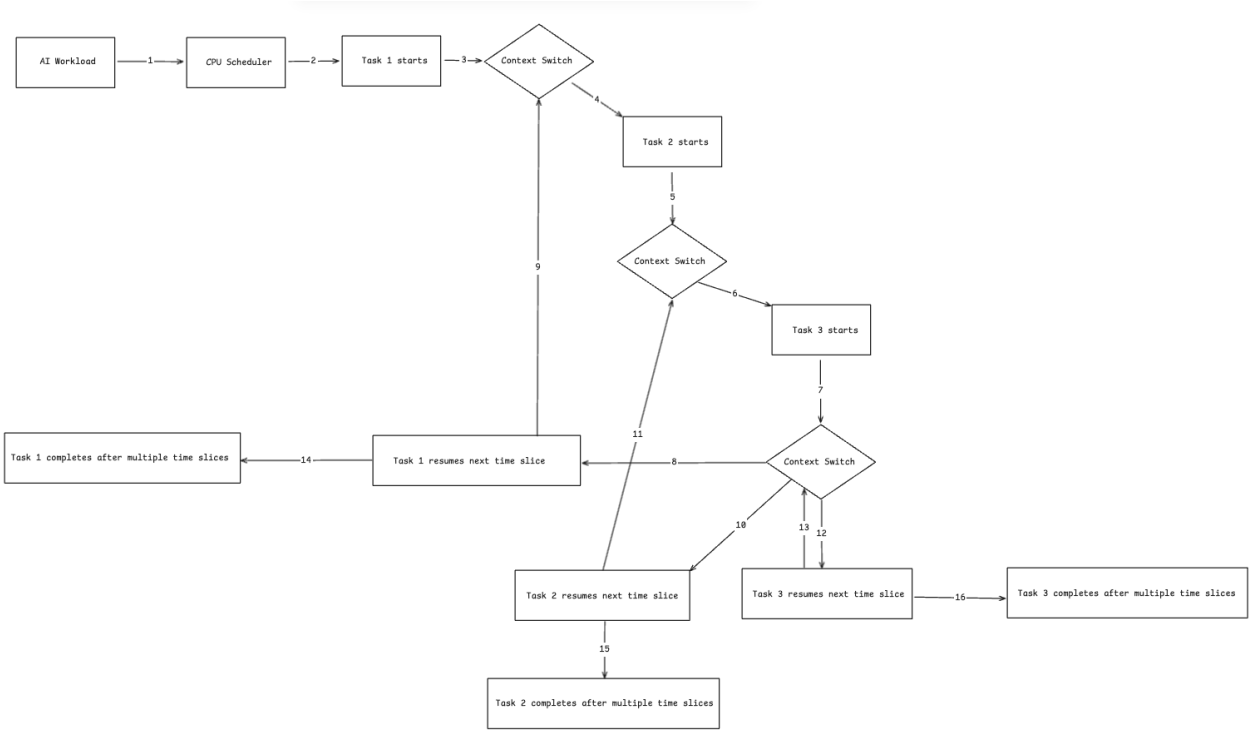
This paper also investigates the performance benefits and limitations of hybrid scheduling in AI workloads. We evaluate CPU-only, GPU-only, and hybrid approaches using comprehensive benchmark suites that test workload scaling, composition, resource constraints, and real-world AI scenarios, including training, inference, research, and production environments. By grounding the analysis in detailed experimental results, this study provides both practical insights and a foundation for designing next-generation scheduling frameworks that can efficiently leverage heterogeneous CPU–GPU architectures for AI workloads.

## 2. CPU Scheduling

CPU scheduling is a core function of any operating system, responsible for determining which process runs on the processor at a given moment. Its main goals are to maximize CPU utilization, maintain system responsiveness, and ensure fairness among processes. The scheduler selects processes from the ready queue and assigns them to the CPU based on predefined policies. For general-purpose computing, this approach balances interactive and batch workloads effectively. However, in compute-heavy environments like AI model training, the sequential nature of CPU scheduling becomes a bottleneck. The following are CPU scheduling algorithms commonly found throughout our everyday machines.

**Round Robin** scheduling allocates each process a fixed time slice and cycles through processes in order. This ensures fairness but introduces frequent context switching, which adds overhead when dealing with compute-bound tasks such as matrix multiplications. As illustrated in Figure 1, AI tasks repeatedly enter the CPU queue, are executed for short time slices, and then preempted for the next task. This repeated context switching causes long AI tasks to take significantly more time to complete reducing overall CPU throughput.

**Figure 1**: Round Robin process under AI workload.




**Shortest Job First (SJF)** selects the process with the smallest estimated execution time next. While efficient for predictable workloads, it performs poorly when task durations vary dynamically, as is common in AI workloads where kernel execution times depend on model size and data complexity.

**Priority Scheduling** executes higher-priority processes first, which can be useful in latency-sensitive applications but often leads to starvation for lower-priority tasks. Moreover, traditional CPU schedulers are unaware of GPU-related dependencies and may delay essential data preprocessing tasks.

**Multi-Level Feedback Queue (MLFQ)** algorithms attempt to balance adaptability and fairness by promoting or demoting processes across queues based on observed behavior. While MLFQ improves responsiveness for diverse workloads, it still cannot manage the data and compute parallelism required for neural network training efficiently.

## 3. GPU Scheduling

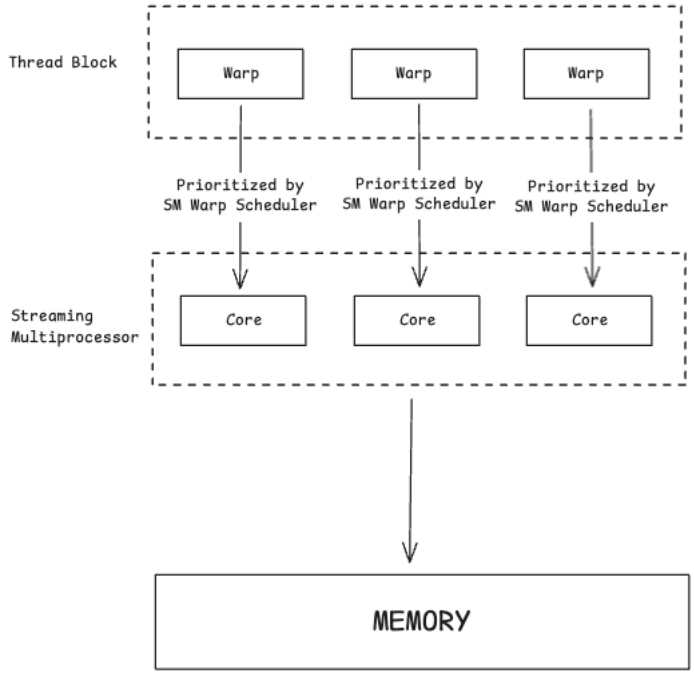
GPUs are built with thousands of lightweight cores grouped into Streaming Multiprocessors (SMs). Each SM can execute many threads concurrently, allowing GPUs to perform a large number of arithmetic operations in parallel, as shown in Figure 2. This architectural design is ideal for deep learning tasks, where similar operations must be applied to large volumes of data simultaneously. As shown by Gebhart, Johnson, et al. (2012), modern GPUs maintain a large number of active threads across SMs massively improving performance-per-watt over CPU-based systems. Unlike CPUs that minimize latency for single-threaded tasks, GPUs maximize throughput by ensuring as many threads as possible are active at once. The primary advantage of GPU scheduling lies in its ability to hide memory latency and execute a

massive number of operations concurrently. This results in significant improvements in throughput and energy efficiency for AI workloads. Compared to CPUs, GPUs achieve higher performance-per-watt ratios and drastically reduce training times for large -scale neural networks.

**Warp scheduling** determines which warps are issued on each SM at any given time. A warp is a group of threads that execute the same instruction simultaneously on different data. When one warp stalls due to a memory access delay, the scheduler switches to another ready warp to keep the pipelines utilized. This scheduling minimizes idle cycles and sustains high throughput, which is essential in AI model training. Recent work shows that warp scheduling can also improve memory behavior. For example, a warp scheduling approach WaSP issues certain warps earlier to prefetch needed data and reduce cache misses (Joseph et al., 2024). They report lower memory access latency and improved performance with minimal hardware cost. This highlights how modern schedulers do more than hide stalls. They can proactively shape memory access patterns to further increase utilization and throughput.

**Tensor operations**, such as matrix multiplication and convolution, dominate AI workloads. GPUs execute these efficiently through vectorized instructions and specialized hardware like Tensor Cores. These cores perform mixed-precision computations optimized for deep learning, allowing higher performance without significant accuracy loss. AI workloads often consist of multiple kernels launched simultaneously, each corresponding to different layers or computations within a model. Dynamic kernel prioritization allows the scheduler to assign more resources to time-critical kernels, improving execution efficiency. This flexibility enables GPUs to better manage concurrent workloads such as model training and data preprocessing.

**Figure 2**: GPU architecture showing Streaming Multiprocessor executing warps in parallel.



## 4. Hybrid Scheduling Experimentation

### 4.1. Objective

This experiment evaluates the performance of hybrid CPU–GPU scheduling compared to CPU-only and GPU-only approaches for AI workloads. The analysis examines workload scaling, composition, resource constraints, and AI-specific scenarios, including training pipelines, inference serving, research experiments, and production MLOps. Metrics such as throughput (tasks per second), GPU utilization, and scheduling overhead are measured to determine under which conditions hybrid scheduling provides measurable advantages. Lin, Wu & Bhattacharyya (2018) prove that the system throughput is positively affected by the vectorization and scheduling methods of a hybrid CPU-GPU architecture. The source code for the following experiment that is (Appendix A) is written in Rust 1.7 and includes a warp scheduler module with multiple policies.

The experimental benchmarks are organized into four test suites:

- **Workload Scaling Suite:** Evaluates 5, 10, 20, and 50 concurrent tasks using Round Robin scheduling with a 0.3-second quantum.
- **Workload Composition Suite:** Compares AI-heavy, balanced, general-heavy, and mixed-ML workloads under First-Come, First-Served (FCFS), Round Robin (0.4-second quantum), and Preemptive Priority scheduling.
- **Resource Constraint Suite:** Examines high-end (16 CPU cores, 4096 GPU cores), mid-range (8 CPU cores, 2048 GPU cores), budget (4 CPU cores, 1024 GPU cores), and CPU-only (8 CPU cores, no GPU) configurations.
- **AI-Workload Suite:** Evaluates scheduling performance in representative scenarios (training, inference, research, and production) using Preemptive Priority scheduling.

### 4.2. Analysis

Hybrid scheduling demonstrates substantial performance improvements over CPU-only execution across all test scenarios. Throughput increases from 2.65 tasks/sec up to 17.1 tasks/sec in workload scaling experiments with near-100% GPU utilization. In workload composition tests, AI-heavy workloads reach 16.4 tasks/sec and non-AI-heavy workloads 7.1 tasks/sec. Resource constraint testing shows consistent performance across hardware tiers: high-end systems achieve 11.62 tasks/sec, mid-range 11.51 tasks/sec, and budget configurations 10.92 tasks/sec as shown in Table 1.

**Table 1:** Throughput comparison across various resource constraints.

| Configuration | CPU (tasks/sec) | GPU (tasks/sec) | Hybrid (tasks/sec) |
|---|---|---|---|
| High-End<br>• 16 CPU cores<br>• 4096 GPU cores | 2.65 | 9.72 | 11.62 |
| Mid-Range<br>• 8 CPU cores<br>• 2048 GPU cores | 2.65 | 9.48 | 11.51 |
| Budget<br>• 4 CPU cores<br>• 1024 GPU cores | 2.65 | 8.79 | 10.92 |

AI-specific experiments demonstrate that hybrid CPU–GPU scheduling significantly improves throughput for compute-intensive workloads, demonstrated in Table 2. Training pipelines achieve a 3153.1% improvement over the CPU-only baseline, highlighting the benefit of GPU acceleration combined with coordinated CPU execution. Research experiments reach a 1392.9% improvement**,** showing that hybrid scheduling adapts effectively to workloads with moderate variability. Production MLOps workloads attain a 2337.4% improvement, demonstrating both sustained throughput and efficient resource utilization in operational scenarios. In contrast, inference serving records 10.84 tasks/sec, slightly below the CPU-only baseline of 11.54 tasks/sec, indicating that latency-sensitive tasks may suffer from hybrid scheduling overhead.

**Table 2:** Throughput comparisons and observations across various AI scenarios.

| Scenario | CPU (tasks/sec) | Hybrid (tasks/sec) | Improvement |
|---|---|---|---|
| Training Pipeline | 0.34 | 10.99 | 3153.1% |
| Inference Serving | 11.54 | 10.84 | -6.1% |
| Research Experiment | 0.77 | 11.50 | 1392.9% |
| Production MLOps | 0.46 | 11.15 | 2337.4% |

Overall, the results indicate that hybrid scheduling effectively leverages both CPUs and GPUs to maximize throughput and utilization in AI workloads. While highly effective for parallel, compute-bound tasks, its performance trade-offs highlight the need for adaptive scheduling strategies in latency-sensitive inference workloads.

## 5. Conclusion

This study analyzed CPU, GPU, and hybrid scheduling for AI workloads, highlighting their respective strengths and limitations. Hybrid CPU–GPU scheduling combines CPU flexibility with GPU parallelism, achieving substantial throughput gains in compute-bound scenarios while maintaining high GPU utilization across hardware configurations. Latency-sensitive inference tasks may experience slight overhead, indicating that hybrid scheduling is best suited for parallel, GPU-bound workloads. These results emphasize the need for adaptive, workload-aware scheduling policies and point to future

opportunities in predictive scheduling and tighter CPU–GPU coordination to optimize heterogeneous AI architectures.

## 6. References


Gebhart, M., Johnson, G., Tarjan, P., Keckler, S. W., Dally, W. J., Lindholm, E., & Skadron, K. (2012). A hierarchical thread scheduler and register file for energy-efficient throughput processors. *ACM Transactions on Computer Systems, 30*(2), 1–26. https://doi.org/10.1145/2159431.2159434

Joseph, D., Aragón, J. L., Parcerisa, J.-M., & Gonzalez, A. (2024). *WaSP: Warp scheduling to mimic prefetching in graphics workloads.* arXiv. https://arxiv.org/abs/2404.06156

Lin, S., Wu, J., & Bhattacharyya, S. S. (2018). Memory-Constrained vectorization and scheduling of dataflow graphs for hybrid CPU-GPU platforms. *ACM Transactions on Embedded Computing Systems*, *17*(2), Article 50. https://doi.org/10.1145/3157669


## 7. Appendix A: Experimentation Source Code

https://github.com/devenshah2018/task-scheduling