|| Volume 9 || Issue 9 || September 2025 || ISSN (Online) 2456-0774
INTERNATIONAL JOURNAL OF ADVANCE SCIENTIFIC RESEARCH
AND ENGINEERING TRENDS

Self-Adaptive AI for Task Scheduling in Heterogeneous Computing

Prof. Anmol Budhewar[1], Atharva Bhole[2], Vaishnavi Barhate[3], Harshad Chaudhari[4], Abhijit Sathe[5]

Prof, Computer Engineering,Sandip Institute Of Technology and Research Center Nashik(SITRC)[1]

Student, Computer Engineering,Sandip Institute Of Technology and Research Center Nashik(SITRC) [2 3 4 5]

Anmol.budhewar@sitrc.org[1],atharvabholework@gmail.com[2],vaishnavibarhate2004@gmail.com[3], engg.harshad49@gmail.com[4], satheabhijit908@gmail.com

---

Abstract: This trend towards heterogeneous computing, whereby CPUs, GPUs and FPGAs are working in concert, is not just about power but about ensuring that valid systems can be built for these applications like AI training or data-intensive research. Yet, scheduling tasks to combine these heterogeneous resources is still a challenging problem. Static techniques such as HEFT are susceptible to unexpected and varying conditions; dynamic heuristics like Min-Min generally focus to near optimal solution, but lack true adaptability. Studies (e.g., [47, 56]) have demonstrated that these kinds of applications may incur performance inefficiency as high as over 20% when the workloads modulate itself, which brings into question the robustness of existing approaches. Reinforcement Learning (RL), despite its flaws, also provides some hope in learning on-the-fly rather than rote-acceptance of stability. The average completion times for largescale GPU clusters have reportedly dropped below 320 ms through DRL-based schedulers, as opposed to over 400 ms by the HEFT. This difference means less wasted cycles and lower cost per joule. However, RL models propose challenges: they are expensive to train and tend not to be transparent. This paper contends that self-adaptive AI scheduling is a direction full of promises but that needs to be treaded carefully and with the right degree of expectations.

Keywords: *Task Scheduling, HEFT, CPU, GPU, Field Programmable Gate Arrays (FPGA), Self-Adaptive, Reinforcement Learning, Heuristics, Deep Reinforcement Learning (DRL), Min-Min Scheduling, Round Robin, AI Scheduler, Heterogeneous Computing Architecture (HCA), Cloud Computing.*

I.INTRODUCTION:

The explosive growth of computational workloads in the past decade has pushed computer systems into uncharted territory. Tasks such as AI training, massive-scale data analytics, or complex physical simulations no longer just demand raw power they demand adaptability, and a lot of it. CPUs alone, as we've learned over and over again, can't keep up with this scale. Their general-purpose design makes them versatile, yes, but painfully inefficient when pitted against workloads with extreme parallel demands. This is where heterogeneous computing has stepped in, combining CPUs with GPUs and FPGAs in ways that attempt to balance raw throughput, configurability, and general-purpose flexibility. On paper, it sounds almost ideal: CPUs handle the "ordinary," GPUs eat through parallel heavy tasks, and FPGAs quietly tackle specialized, latency-sensitive operations.

But the reality is not that smooth. These systems come with their own chaos figuring out how to distribute tasks across such varied resources is messy, and if scheduling decisions go wrong, the expensive hardware ends up sitting idle or burning unnecessary energy. In practice, underutilization is more common than many designers like to admit. A GPU waiting on a poorly placed task or an FPGA left underloaded is not just wasted potential; it's wasted money and energy. And the problem grows worse in environments where workloads change without warning. AI model training, for example, has highly uneven phases bursts of memory use, alternating with computation-heavy steps while real-time analytics pipelines can be perfectly calm one second and overloaded the next. The standard solutions to these issues have usually been either **static scheduling** or rule-based heuristics. Static methods, like HEFT, assume you know everything in advance: every task, every dependency, every processor's capability. The assumption is unrealistic, and once the real system shifts a task takes longer, a processor becomes unavailable the schedule collapses. Heuristic methods like Min-Min or Max-Min do add a degree of flexibility, making decisions based on the current snapshot of the system. But they still rely on rigid, pre-set rules and, more often than not, settle for "good enough" rather than truly optimal. The result is a cycle of inefficiency: schedulers react, but they don't learn. This is where Reinforcement Learning (RL) has started attracting serious attention. Unlike fixed heuristics, RL thrives in uncertainty. The scheduler becomes an agent, the system its environment, and every decision a move in a continuous trial-and-error game. Over time, with enough interactions, the agent learns which actions improve performance, which waste resources, and which balance multiple objectives like speed and energy. It is not flawless RL training is expensive, convergence can be slow, and the models are not always transparent. But its capacity to adapt in environments where traditional methods crumble is difficult to ignore. In fact, experiments in GPU-CPU clusters have already shown RL-based schedulers outperforming HEFT and Round Robin by margins significant enough to reduce both execution times and energy bills. Of course, enthusiasm should be tempered with some caution. Resource diversity complicates state representation: a scheduler must not only know how loaded a CPU is, but also predict how a streaming workload might interact with an FPGA pipeline, or whether multiple tasks competing for memory bandwidth will create bottlenecks. Energy efficiency adds another layer of complexity. Data centres are already notorious for their consumption, and inefficient scheduling makes the problem worse sometimes drastically. AI-driven schedulers promise

energy savings, but the gains are uneven, and not every deployment benefits equally. There's also the question of autonomy. RL-based scheduling has been praised for reducing the need for human intervention a major selling point in distributed and cloud-scale systems where manual oversight is impractical. Yet, this autonomy comes with a price: reduced transparency. System administrators often want to know why a particular decision was made, and RL models, with their black-box policies, don't offer simple explanations.

Still, despite the challenges, the shift toward adaptive, learning-driven scheduling seems inevitable. Static and heuristic methods have served their purpose but struggle under modern workloads. RL offers a framework that, at the very least, moves scheduling in the right direction one that acknowledges unpredictability instead of pretending it doesn't exist.

This paper explores the state of RL-driven scheduling in heterogeneous computing, examining both the promising results and the unresolved challenges. It reviews current approaches, compares them with traditional methods, and discusses their potential role in shaping the next generation of intelligent, adaptive computing systems.

II.LITERATURE SURVEY

Task scheduling has been a part of computing from the very beginning, but the stakes are very different today. With the explosion of AI and high-performance applications, the way we allocate tasks across CPUs, GPUs, and FPGAs can decide whether a system runs efficiently or falls apart under load. It's not just about getting the job done; it's about squeezing performance from hardware that is already expensive to buy and costly to power.

2.1 Traditional Scheduling Approaches Early work in distributed and parallel computing leaned heavily on **non-adaptive scheduling models**, and most of them fell into two categories: static or dynamic heuristics.

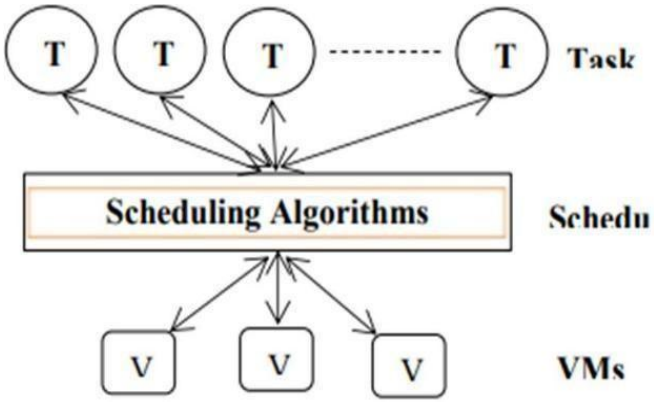

Fig 2.0.1: Task Scheduling Problem Level

Static Scheduling (Pre-calculated): Approaches like Heterogeneous Earliest Finish Time (HEFT) were designed with one big assumption that you know the system and the workload in advance. With complete knowledge of dependencies and processor speeds, you can compute a schedule that, in theory, minimizes turnaround time. It looks elegant in controlled experiments, but in practice, the world is not that predictable. One failed resource or a sudden workload spike can instantly break the neat precalculated plan, leaving processors idle and performance plunging. That fragility is the Achilles' heel of static scheduling.

Dynamic Heuristic Scheduling (Rule-based): Methods like Min-Min and Max-Min were supposed to "fix" those issues by making decisions at runtime. They do adapt better than static algorithms, but only within the narrow rules baked into them. These algorithms often chase local optima, making choices that look fine in the moment but fail globally. And because they never actually learn from experience, they repeat the same mistakes over and over. In heterogeneous settings, where CPUs, GPUs, and FPGAs behave so differently, this rigidity becomes a real bottleneck. Studies have even shown cases where these heuristics consumed more time juggling rules than actually improving efficiency, which is telling. So, while static methods are fast but fragile, heuristics are flexible but often too slow and shallow. The gap between the two is still obvious.

2.2 The Reinforcement Learning Paradigm

This is the point where **Reinforcement Learning (RL)** started to attract interest. Unlike static or heuristic models, RL doesn't assume stability, nor does it settle for hardcoded rules. It treats the scheduler like an agent in a game: take an action, get a reward (or a penalty), and try again. Over time, patterns emerge and the agent learns what works. It's not magic, but the trial-and-error approach means it can thrive in messy, unpredictable environments where traditional methods collapse.

Of course, RL is not without its downsides. Training an RL agent can be expensive, and tuning reward functions is more art than science a poorly designed reward can push the scheduler into odd, sometimes harmful behaviours. But the ability to adapt as conditions change gives RL a big edge.

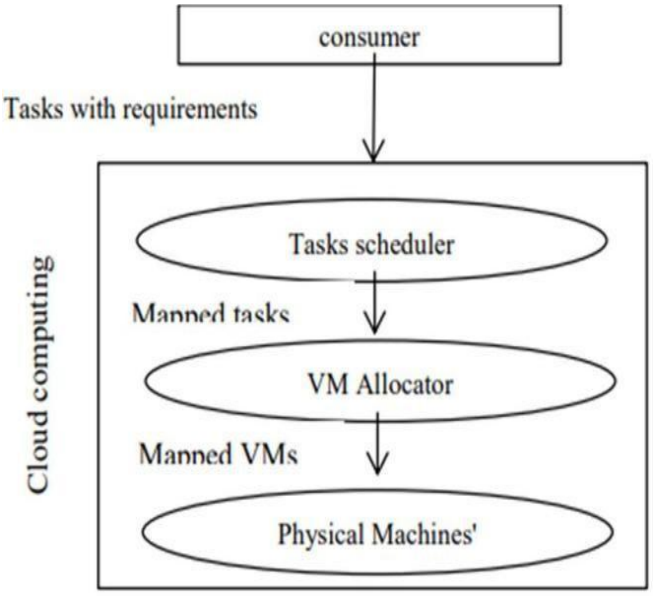

Fig 2.0.2: Resource Allocation

Levels in Cloud Computing

The extension into **Deep Reinforcement Learning (DRL)** has been even more interesting. By plugging in neural networks, DRL can handle state spaces that are simply too complex for classic RL. Real-world numbers back this up: one DRL-based scheduler (PPO-AI) cut average completion time to about 318 ms with 91% resource utilization. HEFT, under the same setup, clocked in at 421 ms with only 74% utilization. Round Robin fared even worse at 582 ms and 68%. Numbers like that don't just suggest potential they show it in action.

2.3 Current Research Trends

Recent studies have pushed RL/DRL scheduling in new directions:

- **Multi-Objective Optimization:** Instead of focusing only on execution time, new designs experiment with reward functions that juggle multiple goals like latency, energy use, and even operational cost. Sometimes the balancing act works, sometimes it leads to trade-offs nobody expected.
- **Scalability:** Large-scale systems are a nightmare for flat RL models, so researchers are trying hierarchical DRL frameworks that break big problems into smaller, easier-to train subproblems. The results are promising but far from perfect.
- **Heterogeneity and Fault Tolerance:** Some work is blending RL with older techniques to build hybrid models. The idea is to get the best of both worlds: the adaptability of RL plus the stability of rule-based systems when hardware fails or contention skyrockets.

The trend is clear: scheduling research is shifting away from fixed-rule thinking toward adaptive, learning-driven methods. Whether RL and DRL will become the *final answer* is still debatable their costs, interpretability issues, and training requirements are real hurdles. But they've already proven capable of solving problems that traditional scheduling cannot, and that alone makes them impossible to ignore.

Metric	Round Robin	HEFT	PPO-AI Scheduler
Max Queue Length	43	31	**17**
Avg Node Temp (°C)	71	68	**64**
Dropped Tasks	6	2	**0**

Fig 2.0.4: Comparison between general algorithms and AI Scheduler

III.METHODOLOGY

This project tries to tackle task scheduling in heterogeneous computing, but let's be honest it's messy. Designing a system that not only assigns tasks to CPUs, GPUs, and FPGAs but also adapts on the fly feels ambitious. The plan isn't just to throw AI at it and hope it works; there's a method, although whether it will behave nicely in the wild is another question.

3.1 Project Scope and Approach

The idea is to build a self-learning agent Reinforcement Learning (RL) seems suitable, though convergence is notoriously unpredictable. The goal: maximize system performance without guzzling too much energy. Sounds straightforward, but in practice, tasks arrive unpredictably, resources fluctuate, and sometimes RL just stalls or learns weird policies. The core elements:

- **Adaptive RL Agent:** This is the brain. It should learn an optimal way to schedule tasks across heterogeneous resources. But honestly, RL might overfit to the simulation; the real-world behaviour could be quite different.
- **Simulation Environment:** The environment tries to mimic reality, with task dependencies and workload bursts. How accurate it is? Well, there's always a gap between simulation and real machines. Still, it's essential to have something concrete to test the agent.
- **Monitoring & Feedback:** Collecting data on latency, energy, utilization. Continuous learning depends on this. But in real systems, monitoring itself can become a bottleneck a nuance often ignored in papers.

Some metrics are clear-cut: the agent should keep latency under 100 ms for most tasks.

Optimizing multiple objectives performance versus energy is tricky. Sometimes, reducing energy comes at a heavy performance cost, and RL has to juggle that trade-off.

3.2 System Architecture

The system is divided into three layers, though the separation isn't always neat in practice.

- **Environment Layer:** Simulates the hardware and executes tasks. Think of it as a sandbox, but the sandbox doesn't always reflect every corner case.
- **Agent Layer:** DRL policy, observation modules, action executor. This is where the "intelligence" supposedly lives. Whether it will actually learn anything useful is uncertain.
- **Analysis Layer:** Rewards and penalties are calculated here and fed back to the agent. Choosing reward functions feels deceptively simple, but the slightest misalignment can lead to bizarre behaviour like the agent learning to idle tasks just to minimize energy.

RL Feedback Loop: Tasks arrive. Agent looks around. Makes a decision. Simulation executes. Reward comes back. Rinse and repeat. Easy in theory, chaotic in practice.

3.3 Analysis Models: SDLC models to be applied Given the research nature of developing an optimal AI policy, the project with adopt the Agile Model combined with iterative focus of Prototype Model.

Chosen Model: Agile Prototyping

This model is ideal because the project's primary risk lies in convergence of the RL algorithm, not in defining static features.

Iterative focus: The work is broken down into short, time boxed sprints

Prototype Driven: Each sprint focuses on delivering a functional prototype, starting with a basic scheduler and incrementally integrating complexity.

Prototype 1: Static Scheduler (HEFT) and basic simulation environment.

Prototype 2: Heuristic Scheduler (Min-Min) integrated with basic state feedback.

Prototype 3: Full DRL agent integrated and trained on simple workloads.

Prototype 4: Multi objective rewards and complex, dynamic heterogeneous workloads.

Risk Mitigation: The model allows for continuous evaluation of the RL agent's performance if an RL algorithm fails to converge, the team can move quickly to a new algorithm without wasting months of development on a fixed plan.

3.4 Implementation Plan

Because RL training is unpredictable, we chose Agile Prototyping. Each sprint produces something that sort-of works, or at least exposes where the model fails.

- **Phase 1 - Foundation & Baseline (6 weeks):** Set up the simulation and implement traditional heuristics like HEFT. These act as a baseline. Sometimes these old-school methods outperform RL early on, which is humbling.
- **Phase 2 - RL Agent Development (8 weeks):** Build the DRL architecture. Decide on state, reward, action space. Integration happens here. Expect headaches; RL doesn't like clean interfaces.
- **Phase 3 - Training & Optimization (12 weeks):**

  This is the long slog. Hyperparameters, reward shaping, trial and error. Occasionally, the agent learns something brilliant. Other times… it

  just cycles uselessly. Multi-objective optimization complicates things further.
- **Phase 4 - Evaluation & Reporting (4 weeks):** Compare RL to traditional benchmarks. Stress test the system. Document what actually improved and what's still problematic. Important to not overstate gains RL shines in some scenarios but fails spectacularly in others.

In sum, the methodology is iterative, messy, and experimental. It doesn't promise perfect scheduling; it promises a self-adaptive agent that *tries* to make sense of heterogeneity and variable workloads. Whether it succeeds entirely is up for debate, but the approach gives structure to explore and fail intelligently.

IV.CONCLUSION

Heterogeneous computing is no longer a niche it's practically unavoidable if you want performance, especially with AI workloads and massive data crunching. Traditional schedulers static heuristics, the usual suspects work okay on paper, but they stumble when workloads fluctuate or when multiple objectives clash. Honestly, they feel rigid, almost naive sometimes.

This study throws a bit of chaos at that rigidity. By introducing a self-adaptive, Reinforcement Learning-driven scheduler, the system doesn't just follow rules it *learns*. It observes, reacts, tweaks itself over time. Deep RL makes the agent slightly smarter with every task, though let's admit it: convergence isn't guaranteed, and hyperparameter tuning can be a nightmare. Yet, compared to classic heuristics, the results are promising. Latency drops. Throughput improves. Resource utilization looks healthier. But don't get too optimistic simulation is forgiving. Real hardware is messier.

One thing that's worth noting: while the agent adapts, it's still a black box. You know it works, but why exactly? That's where explainable AI would come in, though this study only scratches the surface. And energy efficiency? Yes, it's better, but again, the trade-offs are nuanced. Multi-objective balancing isn't a silver bullet.

In short, self-adaptive AI-based scheduling shows potential. It's not perfect. It won't magically solve every heterogeneity headache. But it gives a flexible framework a sandbox for testing smarter, learning driven approaches. The methodology, the iterative Agile prototyping, the reward-feedback loops all these make it possible to experiment, fail, learn, and improve.

Heterogeneous systems are complicated, workloads are unpredictable, and conventional schedulers have limits. Adaptive RL scheduling isn't a cure-all, but it's a meaningful step forward. And that's exciting.

V. REFERENCES

[1]A Survey of Real-time Scheduling on Accelerator-based Heterogeneous Architecture for Time Critical Applications

[2]Design of a Self-Adaptive AI Scheduler for Dynamic Task Allocation in Heterogeneous Clusters

[3]Digital Twin-Driven Collaborative

Scheduling for Heterogeneous Task and EdgeEnd Resource via Multi-Agent Deep Reinforcement Learning

[4]Efficient deep reinforcement learning based task scheduler

[5]Literacy Deep Reinforcement LearningBased Federated Digital Twin Scheduling for the Software-Defined Factory

[6]Continual Reinforcement Learning for Digital Twin Synchronization Optimization over

Dynamic Wireless Networks

[7]A reinforcement learning based job scheduling algorithm heterogeneous computing environments

[8]Research on computing task scheduling method minimize system energy consumption

[9]A digital twin-driven flexible scheduling method in a human

[10]A survey on resource scheduling approaches in multi-access computing

[11]Digital Twin-Assisted Efficient Reinforcement Learning for Edge Task

Scheduling

[12]Agile Reinforcement Learning for Real-Time Task Scheduling in Edge Computing

[13]RL-Scheduler: An Automated HPC Batch Job Scheduler Using Reinforcement Learning

[14]A Reinforcement Learning-Driven Task Scheduling Algorithm for Multi-Tenant Distributed Systems

[15]Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors

[16]HEFT: Performance-effective and Low Complexity Task Scheduling for

Heterogeneous Computing

[17]Improved version of Round Robin scheduling algorithm based on analytic model

[18]On benchmarking task scheduling algorithms for heterogeneous computing systems

[19]**Complexity versus quality**: a trade-off for scheduling workflows in heterogeneous computing environments