Evaluating the Efficacy of LLM-Based Reasoning for Multiobjective HPC Job Scheduling

Prachi Jadhav
University of Tennessee, Knoxville
Oak Ridge National Laboratory
Oak Ridge, TN, USA
jadhavpr@ornl.gov

Hongwei Jin
Argonne National Laboratory
Lemont, IL, USA
jinh@anl.gov

Ewa Deelman
University of Southern California
Los Angeles, CA, USA
deelman@isi.edu

Prasanna Balaprakash
Oak Ridge National Laboratory
Oak Ridge, TN, USA
pbalapra@ornl.gov

Abstract

High-Performance Computing (HPC) job scheduling involves balancing conflicting objectives such as minimizing makespan, reducing wait times, optimizing resource use, and ensuring fairness. Traditional methods, including heuristic-based, e.g., First-Come-First-Served(FJFS) and Shortest Job First (SJF), or intensive optimization techniques, often lack adaptability to dynamic workloads and, more importantly, cannot simultaneously optimize multiple objectives in HPC systems. To address this, we propose a novel Large Language Model (LLM)-based scheduler using a ReAct-style framework (Reason + Act), enabling iterative, interpretable decision-making. The system incorporates a scratchpad memory to track scheduling history and refine decisions via natural language feedback, while a constraint enforcement module ensures feasibility and safety. We evaluate our approach using OpenAI's O4-Mini and Anthropic's Claude 3.7 across seven real-world HPC workload scenarios, including heterogeneous mixes, bursty patterns, and adversarial cases etc. Comparisons against FCFS, SJF, and Google OR-Tools (on 10 to 100 jobs) reveal that LLM-based scheduling effectively balances multiple objectives while offering transparent reasoning through natural language traces. The method excels in constraint satisfaction and adapts to diverse workloads without domain-specific training. However, a trade-off between reasoning quality and computational overhead challenges real-time deployment. This work presents the first comprehensive study of reasoning-capable LLMs for HPC scheduling, demonstrating their potential to handle multiobjective optimization while highlighting limitations in computational efficiency. The findings provide insights into leveraging advanced language models for complex scheduling problems in dynamic HPC environments.

CCS Concepts

• **General and reference** → **Experimentation**; **Evaluation**; **Measurement**; • **Computer systems organization** → **Grid computing**; *Heterogeneous (hybrid) systems*; *Real-time system specification*; **Multicore architectures**.

This work is licensed under a Creative Commons Attribution 4.0 International License.
SC Workshops '25, St Louis, MO, USA
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1871-7/2025/11
https://doi.org/10.1145/3731599.3767584

Keywords

Large Language Models, ReAct prompting, multiobjective scheduling, high performance computing

ACM Reference Format:
Prachi Jadhav, Hongwei Jin, Ewa Deelman, and Prasanna Balaprakash. 2025. Evaluating the Efficacy of LLM-Based Reasoning for Multiobjective HPC Job Scheduling . In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3731599.3767584

1 Introduction

High-Performance Computing (HPC) systems enable scientific breakthroughs but require efficient job scheduling to be optimized. This NP-hard optimization problem [Wang et al.(2021)] demands rapid decisions balancing conflicting objectives: minimizing makespan, wait times, and fairness while maximizing utilization [Abdurahman et al.(2024), Li et al.(2022)]. Traditional approaches use heuristic algorithms (FCFS with backfilling) [Srinivasan et al.(2002)] or optimization techniques, but struggle with dynamic workloads and hardware heterogeneity [Goponenko et al.(2022)]. Deep Reinforcement Learning (DRL) shows promise but faces sample inefficiency and partial observability challenges [Wang et al.(2021)].

Modern LLMs demonstrate remarkable reasoning capabilities through Chain-of-Thought (CoT) approaches [Wang(2025), Kayabali(2025)]. We present the first LLM-based scheduler using a ReAct framework [Yao et al.(2023)] enhanced with a persistent scratchpad memory, that tracks decisions and feedback across timesteps, enabling interpretable multiobjective optimization without domain-specific training. Our hybrid architecture separates natural language reasoning from constraint enforcement, ensuring reliability while providing decision transparency critical for HPC operations.

1.1 Related works

HPC scheduling involves allocating heterogeneous jobs with resource demands (nodes, memory, walltime) to finite compute resources. The core challenge lies in optimizing makespan and utilization while handling continuous job arrivals [Li et al.(2022)]. Traditional approaches range from simple FCFS policies to multiobjective optimization strategies [Srinivasan et al.(2002), Blazewicz

et al.(2019),Zhao and Stankovic(1989),Simon et al.(2013)]. Moreover, classical metaheuristic methods like Genetic Algorithms (GA) [Mirjalili and Mirjalili(2019)], Simulated Annealing (SA) [Bertsimas and Tsitsiklis(1993)], and Particle Swarm Optimization (PSO) [Wang et al.(2018)] have been applied to HPC scheduling, primarily to optimize a single objective (e.g., makespan or throughput) through iterative search over job permutations [Blum and Roli(2003)].

LLMs are emerging in HPC ecosystems for job script generation [Yin et al.(2025)] and policy design. Recent works explore LLM-based resource allocation frameworks [Zhu et al.(2025)] and multi-agent coordination [Amayuelas et al.(2025)]. Key challenges include domain adaptation, computational costs, and constraint integration [Chen et al.(2024)]. Moreover, the concept of LLM agents capable of planning and tool use suggests future applications in complex decision-making tasks like scheduling [Wang et al.(2024),Fu et al.(2024)]. More direct applications involve LLMs in the decision-making process for resource allocation. For instance, LLMSched [Zhu et al.(2025)] proposes an uncertainty-aware scheduling framework for compound LLM applications themselves, highlighting the complexity LLMs introduce as workloads. Enhanced reasoning in LLMs employs reinforcement learning for intermediate step optimization via internal search or path exploration. Reliability techniques include self-correction [Ferraz et al.(2024)] and contrastive methods, while applications span structured planning model extraction [Tantakoun et al.(2025)] and multiagent coordination. Unlike previous focus on NLP tasks, our work directly applies LLM reasoning to scheduling decisions through constraint-aware prompting.

1.2 Reasoning LLMs

State-of-the-art LLMs employ "slow thinking" mechanism for complex problem-solving:

- **OpenAI's O4-Mini**: RL-trained for multi-step reasoning, excelling in mathematics and coding benchmarks [OpenAI(2024)]
- **Claude 3.7 Sonnet**: Hybrid architecture offering speed/quality tradeoffs with 200K token context [WandB(2025)]

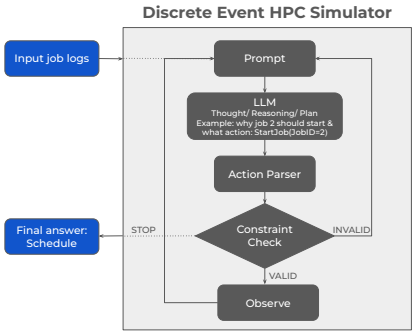

Figure 1: LLM-based HPC scheduling with ReAct-style reasoning.

While LLMs show success on static benchmarks, HPC scheduling presents dynamic challenges with noisy inputs [Naghshnejad and Singhal(2020)]. Our framework addresses this through: 1) ReAct (reasoning + action) prompting, shown in Figure 1, for constraint-aware decisions, 2) Scratchpad memory maintaining scheduling context, and 3) Natural language feedback for constraint violations. This enables zero-shot adaptation to diverse workloads while ensuring feasibility through modular constraint enforcement.

2 LLM-based reasoning agent for multiobjective job scheduling

We present a language-model-driven scheduling agent designed to solve the HPC job scheduling problem through iterative, interpretable reasoning. We refer to our system as an agent because it operates autonomously in a closed-loop interaction with the HPC environment: observing the system state, reasoning over multiobjective goals, and issuing actions to schedule jobs. Unlike traditional schedulers, this agent learns from feedback and evolves its behavior over time, embodying key traits of intelligent agent-based systems. To ensure correctness and realism, the agent interacts with a discrete event simulator, which functions as the underlying HPC environment. The simulator maintains the global simulation clock, tracks system resources, enforces scheduling constraints, and executes valid agent-proposed actions. This section details the problem formulation, the architecture of the *ReAct-style agent*, and the overall decision-making loop.

2.1 Problem Formulation

We define the HPC scheduling problem as a sequential decision-making process. At each timestep t, the agent receives a snapshot of the system state S_t, a job queue $Q_t = \{j_1, \ldots, j_n\}$, where each job j_i has resource demands $r_i = (n_i, m_i)$ for nodes and memory, and a cumulative reasoning history ("scratchpad").

The goal is to construct a schedule $\pi = \{x_1, x_2, \ldots, x_n\}$ that assigns start times to jobs while optimizing multiple objectives, including makespan, resource utilization, fairness, machine throughput, subject to the system node and memory capacity, and job feasibility and eligibility constraints.

2.2 ReAct-style LLM Scheduling Agent

To apply large language models (LLMs) in a domain like HPC scheduling–characterized by numeric constraints and dynamic job arrival–we use a prompting framework called *ReAct (Reason + Act)*. ReAct [Yao et al.(2023)] was introduced to enable LLMs to solve complex problems by alternating between:

- **Thought:** a free-form reasoning step where the model explains its logic,
- **Action:** a structured command representing a concrete scheduling decision.

This design allows the model to break down complex tradeoffs (e.g., fairness vs. throughput) in human-readable terms. In our system, the ReAct agent is prompted with the combination of the current system state (e.g., available nodes, memory), the job queue with metadata, and a running scratchpad that logs all past thoughts, actions, and feedback.

The model selects from *action spaces*:

- `StartJob(job_id=X)`: Start job X immediately,
- `BackfillJob(job_id=Y)`: Opportunistically run a smaller job earlier,

- `Delay`: Wait and defer action until conditions change,
- `Stop`: End the scheduling process.

This scratchpad-based prompting acts as a form of memory, enabling continuity across steps without retraining or fine-tuning.

2.3 Reasoning Loop

The ReAct prompting framework supports a structured decision loop in Algorithm 1:

Algorithm 1 LLM Scheduling Agent Loop

```
Input: Job queue Q, initial state S_0
Initialize scratchpad P ← ∅
while not all jobs are scheduled do
    Construct prompt with S_t, Q_t, and P
    Query LLM → (Thought, Action)
    if Action is valid then
        Update system state
    else
        Generate feedback
    end if
    Append Thought, Action, and Feedback to P
end while
```

This loop enables correction through natural language feedback and requires no model retraining. The LLM adapts in a zero-shot manner using only the structured prompt context.

2.4 Constraint Enforcement

Traditional HPC schedulers hard-code feasibility checks. LLMs, by contrast, can hallucinate or misinterpret numeric limits. To ensure validity: the simulator validates each LLM-suggested action; feasible actions are executed; invalid ones are rejected; violations (e.g., memory overflow) are explained in natural language; these explanations are appended to the scratchpad to inform future decisions. This separation of reasoning from enforcement enables reliable LLM control in HPC environments.

3 Experiments

3.1 HPC jobs instance generation

To emulate diverse and realistic HPC workloads, we define seven benchmark scenarios, each reflecting a distinct operational pattern observed in real-world job traces. The workloads are synthetically generated using a scenario-driven simulator that produces job logs with varying runtime distributions, resource demands, and user metadata. This simulation approach enables precise control over workload characteristics while capturing the variability seen in production HPC environments.

Jobs arrive dynamically over time, following workload-specific interarrival patterns. To mimic realistic submission behaviors in HPC environments, we simulate job arrival times using Poisson processes, commonly used to model inter-arrival patterns in real-world queuing systems. For each workload scenario, we define a scenario-specific arrival rate λ, which governs the average time between job submissions. This dynamic queueing captures timing variability and contention, enabling a more realistic evaluation of scheduling strategies. The simulated HPC environment models a cluster partition with 256 compute nodes and 2048 GB of total memory capacity, representing typical shared-resource configurations found in scientific computing centers. Each submitted job specifies node counts, memory requirements, and estimated walltime, executing non-preemptively once resources become available and scheduling decisions are made. The simulator operates as a discrete event system, advancing simulation time only at key events such as job arrivals and job completions. At each step, the simulator injects any newly arrived jobs into the waiting queue, updates the status of running jobs (releasing resources for those that have finished), and then determines the next scheduling action. If there are jobs ready to be scheduled, the agent queries the LLM for a decision; otherwise, it advances time to the next event (either the next job arrival or the next job completion). This event-driven approach ensures that the simulation efficiently models the dynamic scheduling environment without unnecessary time increments, accurately reflecting real-world system behavior where state changes occur only at discrete points in time.

We evaluate scheduling performance across seven dynamically generated, each reflecting distinct HPC usage patterns and arrival behaviors: **Homogeneous Short** (uniform 30-120s jobs with 2 nodes, 4GB) simulates lightweight CI/test workloads; **Heterogeneous Mix** features varied runtime (Gamma distribution, shape=1.5, scale=300) and resource requirements reflecting realistic production environments; **Long-Job Dominant** combines 20% extremely long jobs (50,000s, 128 nodes) with short jobs (500s, 2 nodes) to test convoy effect handling; **High Parallelism** uses large parallel jobs (64-256 nodes, Gamma walltime distribution) simulating tightly-coupled simulations; **Resource Sparse** employs lightweight jobs (1 node, <8GB, 30-300s) testing sparse workload efficiency; **Bursty + Idle** alternates between short and long-running jobs with modest resource demands to evaluate responsiveness under uneven job durations; and **Adversarial** stress-tests schedulers with one large blocking job (128 nodes, 100,000s) followed by many small jobs (1 node, 60s) to expose convoy effects.

Each scenario was instantiated with [10, 20, 40, 60, 80, 100] jobs. We collect metrics including makespan, average wait time, turnaround time, node/memory utilization, throughput, and fairness to compare scheduler performance under these evolving conditions.

3.2 Objectives

We evaluate scheduling performance using seven standard objectives that capture both system-level efficiency and user-perceived responsiveness. **Makespan** [Englert et al.(2008)] measures the total elapsed time from earliest job submission to completion of the last job ($\max_{j \in J}(x_j + d_j)$), reflecting overall schedule efficiency. **Average wait time** [Smith et al.(1999)] captures user-perceived latency as the time jobs spend queued before execution ($w_j = x_j$ since all $s_j = 0$), while **average turnaround time** [Smith et al.(1999)] measures the total time from submission to completion ($x_j + d_j$). **Throughput** [Mekkittikul and McKeown(1998)] indicates queue processing speed as jobs completed per unit time ($n/(\text{Makespan} - \min_j(x_j))$). **Resource utilization** [Lee et al.(2014)] is measured separately for nodes ($\frac{1}{C \cdot \text{Makespan}} \sum_{j \in J} n_j \cdot d_j$) and memory ($\frac{1}{M \cdot \text{Makespan}} \sum_{j \in J} m_j \cdot d_j$), where C and M represent total system

capacity and n_j, m_j are per-job resource requirements. **Fairness** [Ajtai et al.(1998)] is evaluated using Jain's index [Sediq et al.(2013)] from both per-job ($\frac{(\sum_{j\in J} w_j)^2}{n\cdot\sum_{j\in J} w_j^2}$) and per-user ($\frac{(\sum i\in U u_i)^2}{|U|\cdot\sum_{i\in U} u_i^2}$) perspectives, where u_i represents average wait time per user, with values ranging from 0 (worst) to 1 (perfect fairness).

3.3 Experimental setup

We evaluated our LLM-based scheduler using two models: O4-Mini (OpenAI, hosted on Microsoft Azure) and Claude 3.7 (Anthropic, accessed via Google Cloud Vertex AI). LLMs were accessed via APIs in a single-turn ReAct-style prompting loop at each discrete step. For O4-Mini, we used the "reasoning effort: high" setting with a maximum token context of 100,000. The temperature was fixed internally and could not be controlled. For Claude 3.7, we used a maximum of 5,000 tokens with temperature = 0 to ensure deterministic outputs.

Given n jobs $J = \{1, 2, \ldots, n\}$, each job $j \in J$ arrives at submission time s_j defined by the workload's dynamic arrival process. Each job is characterized by three key parameters: d_j(job duration), n_j(nodes requested), m_j(memory requested in GB).

The LLM-based scheduler must determine start times $\{x_j\}_{j=1}^{n}$ for all jobs while optimizing system objectives (e.g., makespan minimization, resource utilization maximization) under the constraints: $\sum_{j\in A(t)} n_j \leq N_{\text{total}}$ (Node capacity), $\forall t \sum_{j\in A(t)} m_j \leq M_{\text{total}}$ (Memory capacity), $\forall t$ where $A(t) = \{j \in J \mid x_j \leq t < x_j + d_j\}$ denotes the set of active jobs at time t, and N_{total}, M_{total} represent the cluster's total node and memory resources respectively.

We compare our LLM-based scheduler to the following methods: First Come First Serve (FCFS) is the simplest scheduling algorithm that executes jobs strictly in their arrival order, subject to resource constraints.

Shortest Job First (SJF) prioritizes jobs with the shortest estimated runtime, typically reducing average turnaround time but potentially starving longer jobs, compromising fairness. Google OR-Tools [Perron and Furnon(2024)] provides an optimization-based scheduling solution, which we use as a strong baseline; it computes globally optimal or near-optimal schedules for small-to-medium workloads, offering a performance upper bound for comparison. We do not use reinforcement learning (RL) [Wang et al.(2021)] or graph neural networks (GNNs) [Adimora and Sun(2024)] in this work, as these methods typically optimize for a single scalar objective (e.g., reward or cost) and require extensive environment tuning and retraining for new objective trade-offs. In contrast, our LLM-based ReAct agent supports multiobjective reasoning in a flexible, interpretable way (without retraining) by conditioning on natural language prompts that describe scheduling goals for jobs.

Our LLM scheduler operates at the job selection and allocation level, using a first-fit strategy on a cluster (256 Nodes, 2048 GB memory). A first-fit strategy allocates each selected job to the first available set of resources that meet its requirements. While we abstract system-level details like topology and storage, these are typically handled by the underlying resource management stack. The focus is on evaluating high-level scheduling decisions, with topology-aware placement left for future work. All workload data, code, prompts, and experiment configurations are publicly available for reproducibility: https://github.com/swarm-workflows/ReasonedScheduler/tree/master/llm_scheduler.

3.4 LLM prompt and reasoning traces

A key feature of our LLM-based scheduler is that all scheduling decisions are made through text-based reasoning, providing full visibility into the model's thought process. This enhances interpretability and enables qualitative analysis of the LLM's emergent heuristics and decision strategies. Below, we highlight the prompt that we have constructed and show the excerpts from representative decision traces in Figure 2 observed during experiments.

3.5 Comparison across diverse workload scenarios

We evaluated our LLM-based scheduling approach against traditional baselines across seven workload scenarios with 60 jobs each, with performance metrics normalized relative to FCFS (baseline=1.0). The results, shown in the normalized metrics chart (Figure 3), reveal distinct performance patterns that highlight the strengths and limitations of different scheduling approaches under varying workload characteristics. Note that the figure did not include Heterogeneous Mix because it will be covered in the section 3.6.Also, the average wait time for all methods, including FCFS, was 0.0. Since the normalization is performed relative to FCFS, the resulting value becomes undefined (0/0) and is therefore omitted from the comparison.

Bursty Idle highlights the limits of traditional and optimization-based schedulers. OR-Tools exhibits a degradation in wait time fairness, while both LLM-based schedulers maintain balanced performance across all objectives, including throughput, fairness, and utilization, demonstrating adaptability under workload volatility. **Long Job Dominant** shows pronounced differences. FCFS, SJF suffer significantly from the convoy effect, with higher wait time, turnaround time. LLMs showed dramatically reduced average wait and turnaround time. O4-mini performs very well with balanced fairness and efficiency. Claude 3.7 performed slightly worse on fairness. **In High Parallelism**, OR-Tools and LLMs achieve the highest resource utilization and throughput (>1.2x), benefiting from reasoning-based job packing compared with the heuristic methods. O4-Mini performs better than Claude 3.7 in fairness. FCFS, SJF trails behind, indicating limited capacity to handle fragmentation from large job sizes. **Adversarial** conditions lead to flattened differences: all methods perform nearly identically across metrics. LLMs like Claude 3.7 and O4-Mini perform no worse than heuristics ones; a good sign of stability. OR-tools achieved optimal or near-optimal results in throughput, turnaround time, and system utilization across scenarios. However, this gain came at the cost of fairness: both wait time fairness and user-level fairness were lower compared to simpler heuristics like SJF and FCFS, indicating potential bias in scheduling priority or resource allocation. **Resource Sparse** again yields limited differentiation; all schedulers perform near the baseline, reflecting minimal contention. LLMs don't gain much advantage here, there's little room for intelligent scheduling to make an impact. O4-Mini showed notably poor fairness. Its learned policy likely optimize for system-wide efficiency, prioritizing "easy wins" (e.g., smaller jobs). This causes starvation or long delays for certain

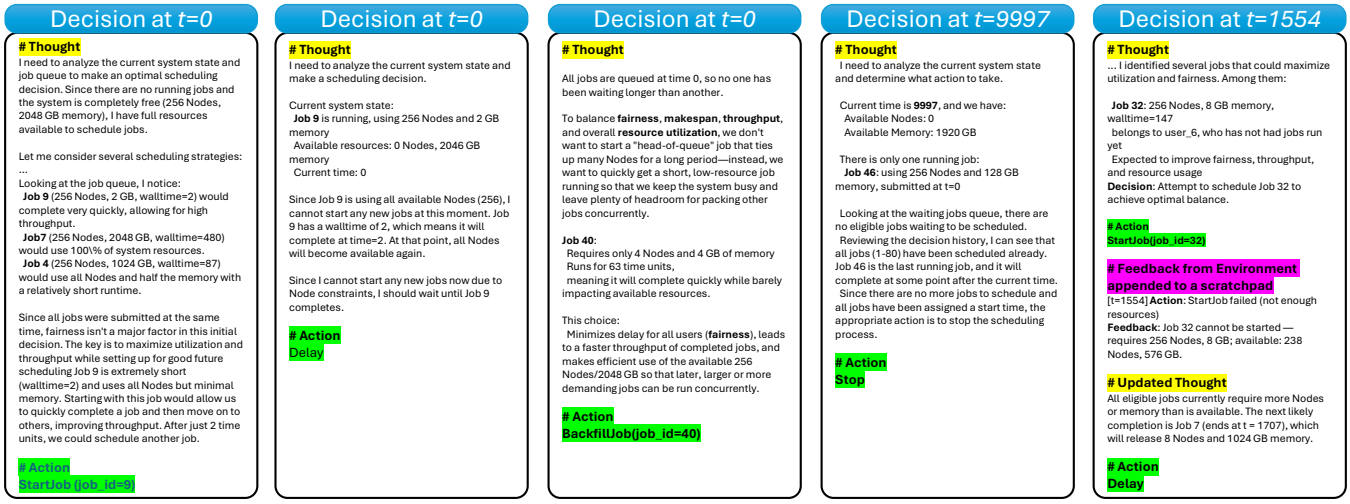

Figure 2: Representative LLM reasoning traces demonstrating interpretable scheduling decisions. The traces show multiobjective trade-off analysis (selecting Job 9 for throughput optimization), resource constraint recognition (delaying when nodes unavailable), intelligent backfilling strategies (choosing Job 40 for efficient resource utilization), and adaptive recovery from constraint violations through natural language feedback. Each decision includes explicit reasoning about fairness, utilization, and system state, providing complete transparency into the scheduling logic. Decision at t = X denotes the simulation time at which the LLM is queried to make a scheduling decision based on the current system state.

jobs or users when resources are scarce. **Homogeneous Short** scenarios also result in uniform performance. All methods deliver similar makespan, throughput, and fairness, as the low variability in job characteristics reduces the need for strategic scheduling. O4-Mini shows reduced fairness, likely due to job similarity making simple methods sufficient.

3.6 Scalability analysis

To evaluate how scheduling performance scales with problem complexity, we analyzed the Heterogeneous Mix workload scenario across increasing job queue sizes from 10 to 100 jobs. The results shown in Fig. 4 demonstrate a clear relationship between problem scale and the effectiveness of different scheduling approaches, with performance differentiation becoming increasingly pronounced as queue size grows. As shown in Figure 4, performance differentiation becomes more pronounced as job size increases: Small queues (10–20 jobs): All methods perform comparably across most objectives, indicating low contention and reduced room for optimization. OR-Tools slightly improves utilization early on, while LLM-based agents start to show fairness benefits. Mid-scale queues (40–60 jobs): Claude 3.7 begins to outperform others across multiple metrics, particularly in wait time, turnaround time, and fairness at the same time. This suggests emerging benefits from reasoning-based adaptation as scheduling complexity grows. OR-Tools continues to optimize resource utilization but degrades in wait time and turnaround time. Large queues (80–100 jobs): Performance differences widen further. Claude 3.7 and O4-Mini maintain multiobjective balance, with strong throughput (1.4x) and resource utilization (>1.4x vs FCFS) while still achieving fairness. In contrast, OR-Tools maximizes node and memory utilization (up to 1.8×) but shows higher wait time & turnaround time, reflecting a trade-off due to its optimization focus. FCFS and SJF remain largely static across scales, showing limited adaptability. This analysis highlights the graceful scalability of LLM-based schedulers, which maintain balanced decision-making across job sizes, contrasting with the rigidity of heuristics and the singular optimization focus of OR-Tools. Reasoning-based agents dynamically adapt to queue complexity and maintain user-centric objectives under load.

3.7 Computational Overhead analysis

A critical consideration for deploying LLM-based scheduling in production HPC environments is understanding the computational overhead and how it scales with both workload complexity and problem size. We analyze the overhead characteristics across two dimensions: workload-dependent complexity at fixed scale (60 jobs) and scalability with increasing job queue sizes for the most representative scenario.

3.7.1 Workload-dependent overhead characteristics. To ensure fair and meaningful overhead analysis, we restrict our measurements to LLM calls that resulted in feasible and accepted scheduling actions; namely, *start_job* and *backfill_job*. Calls leading to delay events are excluded, as they typically suggested either when the system is fully saturated (i.e., all resources are occupied), or when pending jobs demand more resources than currently available, making any scheduling action infeasible. Including such cases would conflate LLM reasoning latency with external system constraints. Thus, our analysis isolates the LLM's decision overhead under conditions where successful job placement is possible. For each workload model pair, we collected the total elapsed scheduling time, number of LLM calls, and the distribution of individual LLM call latencies. The plot below

Prompt Example

```
You are an expert HPC resource manager, and your task
is to schedule jobs in a high-performance computing (HPC)
environment. Use the current system state, job queue, scratchpad
(decision history), and fairness indicators to make well-balanced
decisions.
System capacity: 256 nodes, 2048 GB memory
Current time: 0
Available Nodes: 256
Available Memory: 2048 GB
Running Jobs:
None
Completed Jobs:
None
Waiting Jobs (eligible to schedule):
None
# Scratchpad (Decision History)
(nothing yet)
Your scheduling objectives are:
You must balance all of the following:
    • Fairness: Minimize variance in user wait times. Avoid
      starving any user.
    • Makespan: Minimize total time to finish all jobs.
    • Utilization: Maximize Node & memory usage over time (avoid
      idle resources).
    • Throughput: Maximize the number of jobs completed per
      unit time.
    • Feasibility: Do not exceed 256 Nodes or 2048 GB memory at
      any time.
Trade-offs are allowed. Do not over-optimize one metric at the
expense of others.
For example:
    • Prioritizing a long-waiting job improves fairness, but
      may slightly hurt makespan.
    • Choosing short jobs improves throughput, but may increase
      wait time for large jobs.
Decide:
  (1) Which job should be started now (if any)?
  (2) Justify your decision in thought.
  (3) Return only one of:
      • StartJob(job_id=X)
      • BackfillJob(job_id=Y)
      • Delay
      • Stop (when all jobs have been scheduled)
Output format:
Thought: <your reasoning>
Action: <your action>
```

in Figure 5 compares these overhead metrics between Claude 3.7 and O4-Mini. Elapsed Time (Left Plot): Claude 3.7 consistently exhibits lower total elapsed scheduling time across all workload types compared to O4-Mini, with up to 7× faster execution for Heterogeneous Mix. This highlights Claude 3.7's faster end-to-end decision efficiency. LLM Call Count (Middle Plot): All models exhibit 60 calls (equal to job count), with slight variation due to backfilling. This confirms comparable action trace lengths, indicating that runtime differences stem from per-call latency, not the number of decisions. And LLM Call Time Distribution (Right Plot): Claude 3.7's per-call latencies are tightly clustered below 10 seconds, showing low variance and stable reasoning latency. In contrast, O4-Mini exhibits high variance, with several outliers exceeding 100s, particularly in Heterogeneous Mix. This behavior indicates struggles in complex state-action mappings requiring longer reasoning chains. The elevated overhead observed for O4-Mini in the Heterogeneous Mix scenario can be attributed to the increased complexity of decision-making under resource heterogeneity and scheduling contention. In this setting, jobs vary widely in size (nodes, memory) and duration, creating a larger and more constrained scheduling search space. The observed latency patterns are consistent with increased reasoning difficulty driven by workload diversity.

3.7.2 Scalability with job queue size. The scalability analysis for Heterogeneous Mix workloads reveals how computational overhead grows with problem size, as depicted in Figure 6. This analysis is crucial for understanding deployment feasibility in large-scale HPC environments.

Total Elapsed Time (Left): Both models show a monotonic increase in total elapsed time as job size increases, but O4-Mini's growth is superlinear from 40 jobs onward. This suggests compounding latency effects, potentially due to increased reasoning complexity when the waiting queue contains more heterogeneous jobs. Claude 3.7 scales more gracefully, maintaining a near-linear growth profile. LLM Call Count (Middle): Call counts scale proportionally with job size for both models, as more scheduling decisions are needed. The similarity in call counts indicates that elapsed time differences stem from per call latency, not from a larger number of decisions. LLM Call Times Distribution (Right): Claude 3.7 maintains a tight latency distribution across job sizes, with minimal outliers. O4-Mini exhibits significant spread and heavy-tailed latency distributions, especially at 60–80 jobs, where outliers exceed 200 s. This variance indicates occasional very slow decision episodes, likely caused by increased search space and reasoning over conflicting scheduling objectives.

3.7.3 Deployment implications. The overhead analysis reveals several critical considerations for deploying LLM-based scheduling in production HPC environments. The wall-clock times required (up to an hour for 100 jobs) indicate that, at the moment, LLM-based scheduling is not suitable for real-time job submission scenarios. However, it may be practical for batch scheduling systems, periodic resource optimization, or strategic scheduling decisions where quality is prioritized over immediate response.

For HPC installations regularly processing hundreds or thousands of concurrent jobs, the computational overhead may become prohibitive. The analysis suggests practical deployment limits around 100-200 jobs for periodic scheduling optimization, beyond which the overhead may outweigh the scheduling benefits.

The results suggest that on-premise fast reasoning models are critical to overcome the computational overhead barriers identified in this analysis. As reasoning model efficiency continues to improve and deployment costs decrease, LLM-based scheduling may become viable for increasingly latency sensitive and large-scale HPC applications.

4 Statistical Robustness and Repeatability

Given the inherent stochasticity of LLMs, we evaluate the statistical robustness of our results through multiple independent runs of each scheduling method. Specifically, we conduct 5 repetitions of the full scheduling pipeline for the Heterogeneous Mix workload with 100 dynamically arriving jobs (We selected the Heterogeneous Mix scenario with 100 jobs to evaluate scheduler robustness under

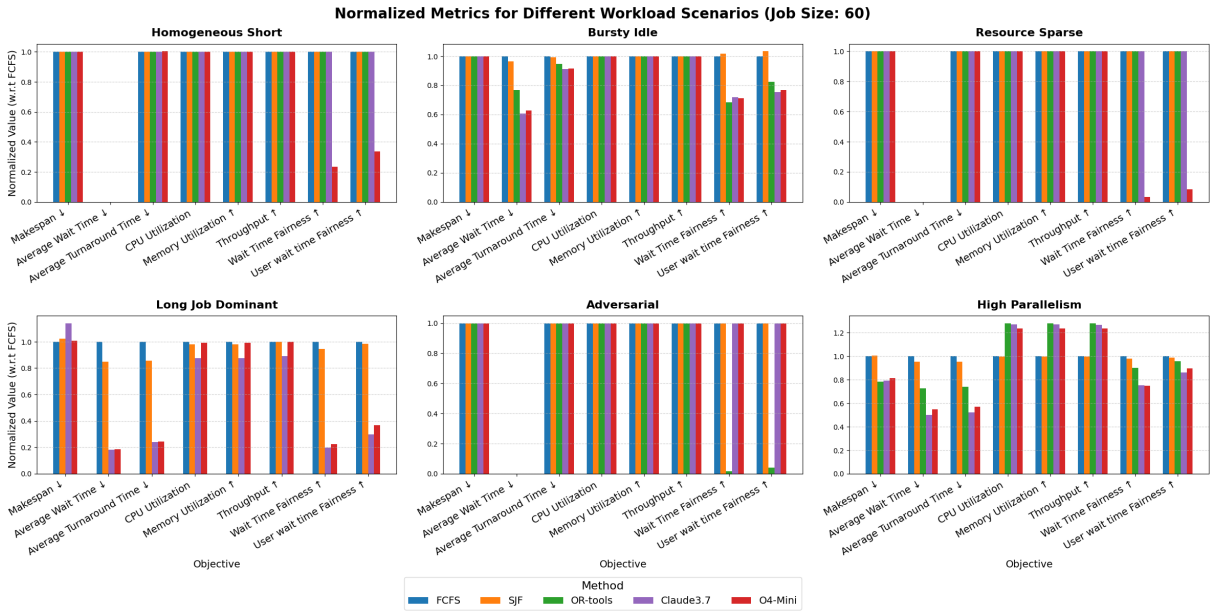

Figure 3: Normalized performance metrics across six workload scenarios with 60 jobs each. All metrics are normalized relative to FCFS performance. Lower values indicate better performance for negative metrics (makespan, wait & turnaround times), while higher values indicate better performance for positive metrics (utilization, throughput, and fairness). The comparison includes traditional heuristics (FCFS, SJF), optimization-based scheduling (OR-Tools), and LLM-based approaches. Claude 3.7 demonstrated superior adaptability and multiobjective balance in Bursty Idle and Long Job Dominant. In contrast, OR-Tools focuses on utilization but degrades in fairness, while heuristics show limited responsiveness to workload variation.

realistic, high-load conditions that involve diverse job durations and resource demands—emulating the variability and contention typical in production HPC workloads.), and repeat metric distributions normalized against the FCFS baseline. Figure 7 presents box plots for eight performance metrics across five methods. Each box represents the distribution of outcomes across five trials per method, providing insight into the stability and variance of each scheduler under repeated execution. Key observations include: LLM-based schedulers show tight variance bounds for makespan, wait time, turnaround time, and throughput, with consistent improvements over heuristics. Claude 3.7 exhibits strong user-level fairness and low wait times with minimal variance, indicating reliable performance across runs despite non-determinism. OR-Tools achieves the highest utilization but shows greater variance in fairness, revealing a trade-off between resource efficiency and user experience consistency. FCFS and SJF remain deterministic and thus flat in their metric distribution, yet underperform in multiobjective balance. No significant outliers were observed for LLM methods on negative metrics, suggesting robustness in the ReAct-style reasoning framework. These results demonstrated that, even without fine-tuning or deterministic control, LLM-based agents produce repeatable and statistically stable scheduling behavior under complex, dynamic workloads with the lowest temperature parameter. This supports the viability of using zero-shot LLMs for multiobjective HPC scheduling, especially when transparency and fairness are critical.

5 Evaluation on real workload traces

To assess the applicability of our LLM-based scheduler on real-world workloads, we evaluated it using job traces from the Polaris supercomputer at Argonne National Laboratory. Polaris consists of 560 compute nodes, each with 512 GB of memory. Polaris data was generated from resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. We preprocessed 100 jobs submitted in November 2024 from the public job history logs, selecting a contiguous segment of completed jobs for evaluation. Data Preprocessing: The raw trace was filtered to remove failed jobs (i.e., $EXIT_STATUS = -1$) and sorted by submission time. Key fields such as job name, user, group, submission, start, and end timestamps, requested nodes, walltime, and queued wait time were retained. Timestamps were normalized relative to the earliest submission to produce relative submission, start, and end times in minutes. Job identifiers, user, and group labels were factorized to anonymized IDs (e.g., User_1, Group_1). We used node count as is and derived total memory (using 512 GB per node) per job, yielding a dataset compatible with our LLM agent.

Assumptions and Limitations: Since the trace was sampled mid-stream, the exact system state at the beginning of the sequence is unknown. For simplicity, we assumed the cluster was idle at time zero, an optimistic assumption that deviates from the real system conditions, where nodes may have been partially occupied. Consequently, this setup is not intended as a fair comparison against the actual Polaris scheduler, which operates under complex

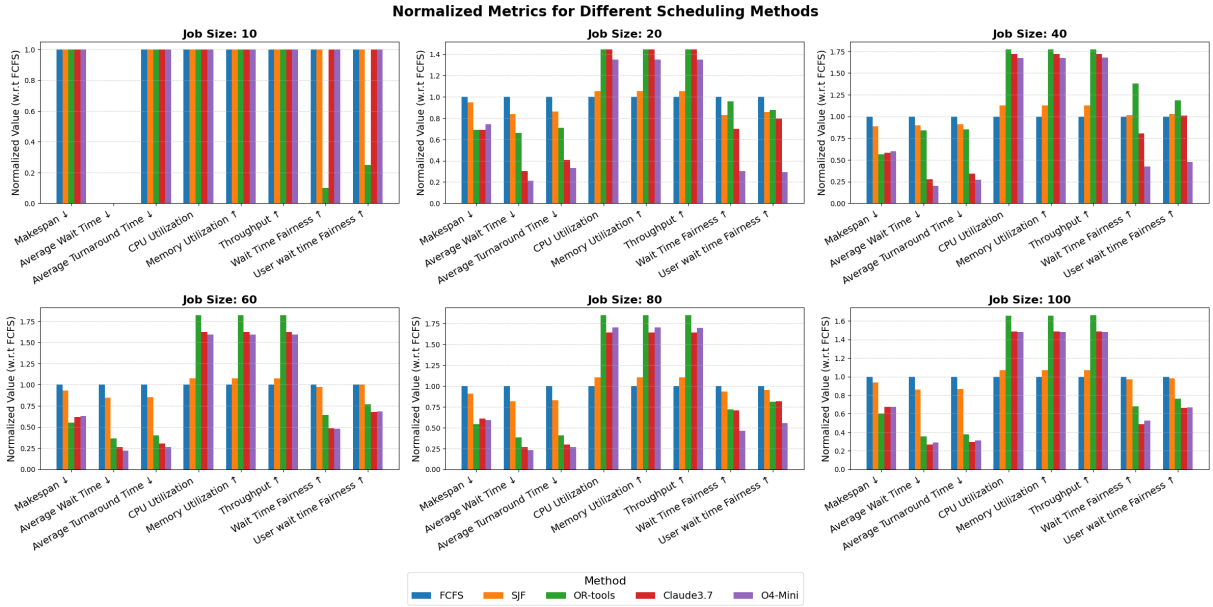

Figure 4: Scalability analysis showing normalized performance metrics across increasing job queue sizes (10 to 100 jobs) for the Heterogeneous Mix workload scenario. All metrics are normalized relative to FCFS performance (baseline = 1.0). The results demonstrate how scheduling performance differentiates as problem complexity increases. At small scales (10 jobs), all methods perform similarly, but significant performance gaps emerge with complexity. OR-Tools achieves exceptional resource utilization (up to 1.75× FCFS) at higher scales but shows more variable performance across other metrics. LLM-based schedulers maintain consistently balanced performance across all metrics and scales, demonstrating robust scalability characteristics. Traditional heuristics (FCFS, SJF) show limited improvement potential as queue size increases, highlighting the importance of advanced scheduling approaches for large-scale HPC workloads.

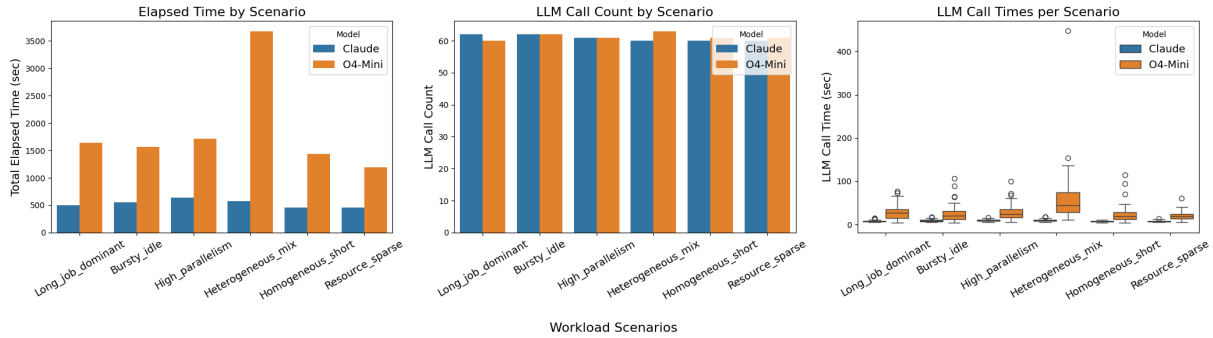

Figure 5: Computational overhead comparison across six workload scenarios. Metrics include total elapsed time (left), number of LLM calls (center), and per call latency distribution (right) for Claude 3.7 and O4-Mini. Only successful scheduling actions (start_job, backfill_job) are considered. Claude 3.7 demonstrates significantly lower and more stable inference overhead, especially in complex workloads like Heterogeneous Mix, where O4-Mini exhibits high latency variance.

queue policies and system-wide constraints. Rather, this experiment demonstrates that the LLM scheduler generalizes to real traces and can produce plausible schedules when provided with normalized historical data.

This evaluation underscores that, even under simplified assumptions, LLM-based scheduling can be grounded in real-world HPC logs, highlighting the generality of the reasoning framework. Figure 8 presents normalized performance metrics for five schedulers evaluated on 100 real jobs from the Polaris workload trace. All values are normalized for FCFS. The LLM-based schedulers achieve substantial improvements in average wait time and turnaround time, comparable to the idealized SJF policy. Importantly, both LLM agents maintain resource utilization and throughput on par with all baselines, confirming that system efficiency is preserved. In terms of wait time and turnaround time, the LLM schedulers show above and at par performance as OR-tools. Claude 3.7, Or-Tools achieves

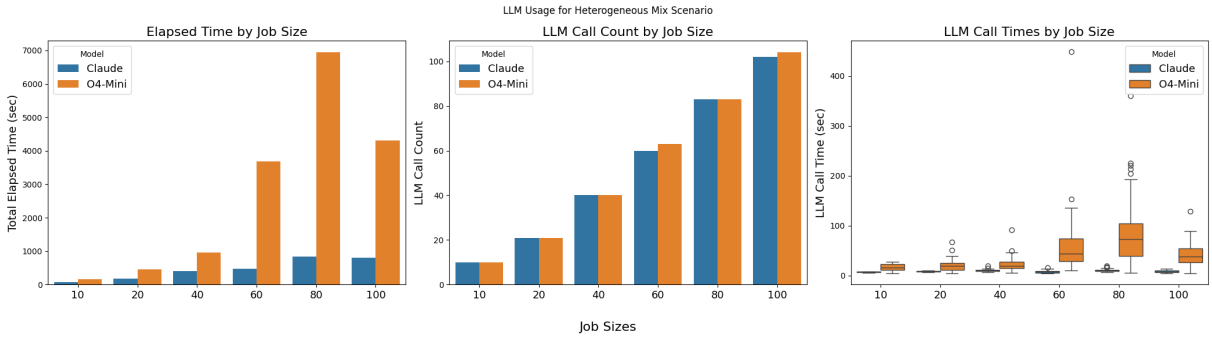

Figure 6: LLM computational Overhead scaling with job queue size for Heterogeneous Mix workloads. Scheduling time grows super-linearly, reaching 4k s (O4-Mini) and 700 s (Claude 3.7) for 100 jobs, while API calls scale linearly. O4-Mini exhibits a pronounced spike in total runtime at job size 80, exceeding 6.9k s, despite having fewer jobs than the 100-job case. It could be the case that the workload generation process or attributable to transient network/API latency during execution, rather than purely scheduling-reasoning complexity.

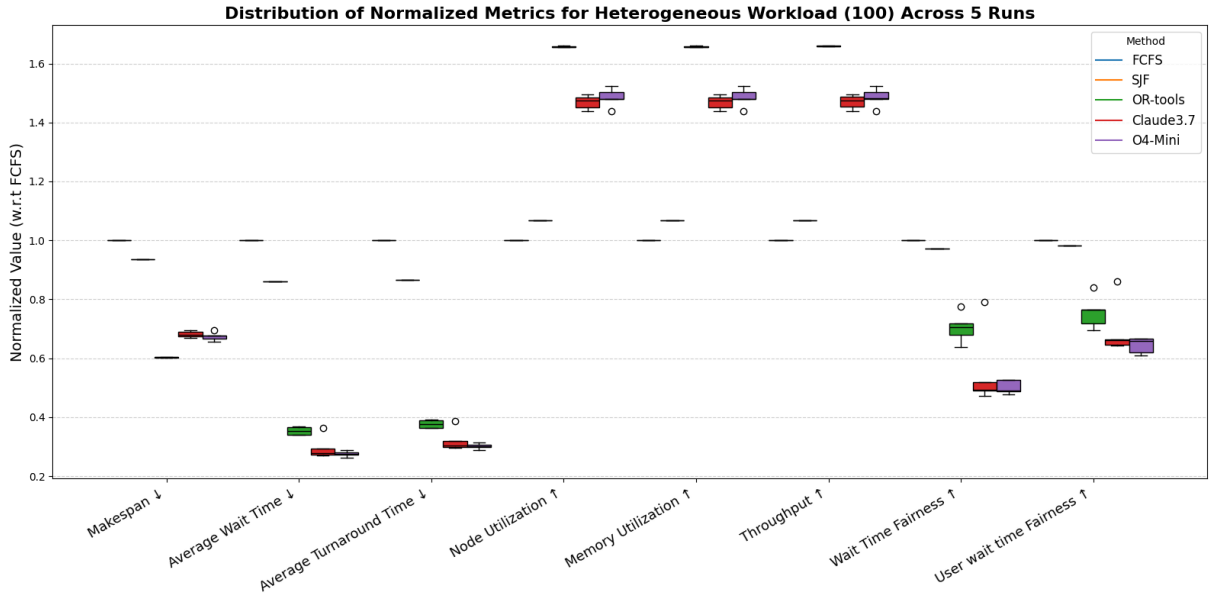

Figure 7: Distribution of normalized perofrmance metrics for the Heterogeneous Mix workload with 100 jobs over 5 independent runs, normalized to FCFS (baseline=1.0). LLM-based schedulers (Claude 3.7, O4-Mini) show low variance and consistent improvements across multiple objectives, while OR-Tools excels in utilization but exhibits fairness trade-offs. FCFS and SJF remain deterministic with static performance. Results demonstrated the robustness of LLM scheduling under stochastic reasoning.

the highest wait time fairness, and O4-Mini demonstrates superior user-level fairness. These results highlight that, even under real-world job traces and an assumed idle system state, LLM agents can generalize effectively and deliver low latency and balanced fairness without explicit optimization objectives.

6 Conclusion

We have conducted the first comprehensive evaluation of reasoning-capable Large Language Models for multiobjective HPC job scheduling, demonstrating their potential and limitations through systematic analysis across diverse workload scenarios. Our ReAct-style scheduling agent enables interpretable, zero-shot scheduling that balances makespan, utilization, throughput, and fairness without requiring domain-specific training. The separation of reasoning from

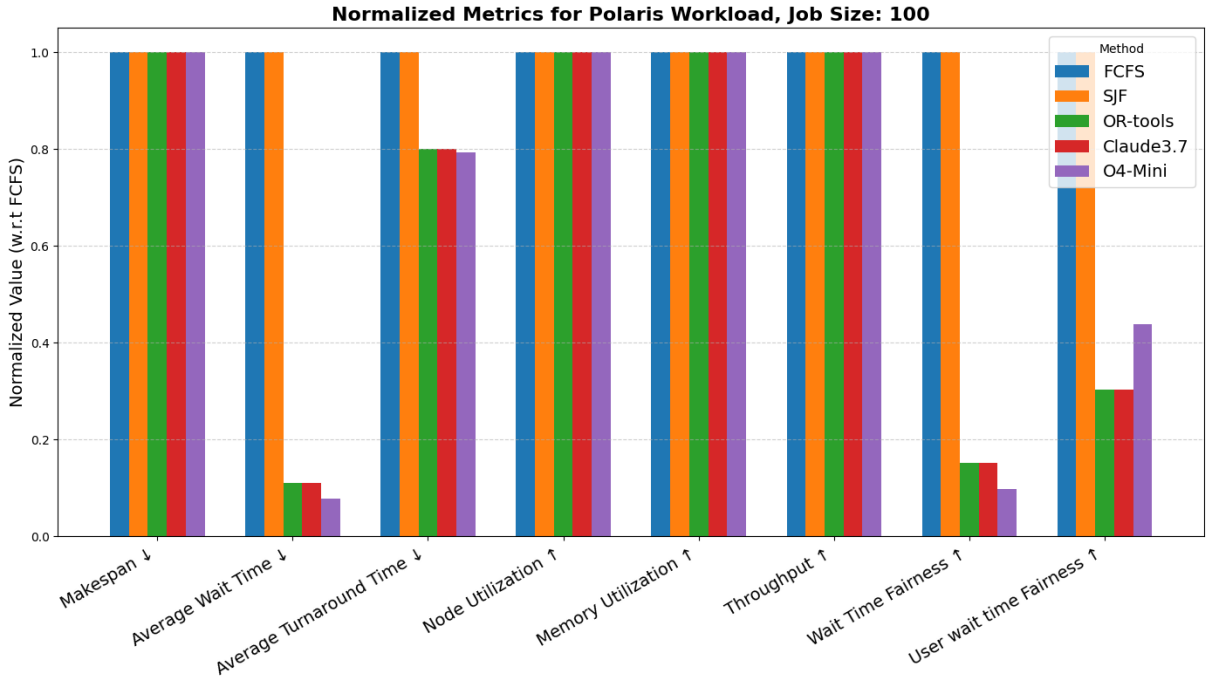

Figure 8: Normalized performance metrics for 100 Polaris job logs. LLM-based schedulers reduce latency and improve fairness while maintaining system efficiency.

constraint enforcement ensures system reliability while providing complete decision transparency essential for HPC operations.

Through evaluation across seven scenarios and varying job sizes, we demonstrate that LLM-based scheduling performs well in complex, heterogeneous environments where traditional heuristics struggle with multiobjective trade-offs. However, our computational overhead analysis reveals significant time requirements, with scheduling times ranging from 1-2 hours for 100 jobs using current cloud-based API services. Our evaluation establishes that while current cloud-based LLM services impose significant latency constraints limiting real-time deployment, the consistent multiobjective balancing and superior adaptivity demonstrated by reasoning-based approaches suggest potential for HPC scheduling applications with appropriate deployment strategies.

Our future research directions include extending our approach to use on-perm fast reasoning mode, handle dynamic job arrivals and real-time workload changes, validating performance on production traces, and exploring advanced constraint handling for job dependencies and energy-aware scheduling.

Acknowledgments

This work is supported by the U.S. Department of Energy, under grant # DE-SC0024387.

References

[Abdurahman et al.(2024)] Abubeker Abdurahman, Abrar Hossain, Kevin A Brown, Kazutomo Yoshii, and Kishwar Ahmed. 2024. Scalable HPC Job Scheduling and Resource Management in SST. In *2024 Winter Simulation Conference (WSC)*. IEEE, 2226–2237.

[Adimora and Sun(2024)] Kyrian C Adimora and Hongyang Sun. 2024. GNN-RL: An Intelligent HPC Resource Scheduler. (2024).

[Ajtai et al.(1998)] Miklos Ajtai, James Aspnes, Moni Naor, Yuval Rabani, Leonard J Schulman, and Orli Waarts. 1998. Fairness in scheduling. *Journal of Algorithms* 29, 2 (1998), 306–357.

[Amayuelas et al.(2025)] Alfonso Amayuelas, Jingbo Yang, Saaket Agashe, Ashwin Nagarajan, Antonis Antoniades, Xin Eric Wang, and William Wang. 2025. Self-Resource Allocation in Multi-Agent LLM Systems. *arXiv preprint arXiv:2504.02051* (2025).

[Bertsimas and Tsitsiklis(1993)] Dimitris Bertsimas and John Tsitsiklis. 1993. Simulated annealing. *Statistical science* 8, 1 (1993), 10–15.

[Blazewicz et al.(2019)] Jacek Blazewicz, Klaus Ecker, Erwin Pesch, Günter Schmidt, and J Weglarz. 2019. *Handbook on scheduling*. Springer.

[Blum and Roli(2003)] Christian Blum and Andrea Roli. 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)* 35, 3 (2003), 268–308.

[Chen et al.(2024)] Le Chen, Nesreen K Ahmed, Akash Dutta, Arijit Bhattacharjee, Sixing Yu, Quazi Ishtiaque Mahmud, Waqwoya Abebe, Hung Phan, Aishwarya Sarkar, Branden Butler, et al. 2024. The landscape and challenges of HPC research and LLMs. *arXiv preprint arXiv:2402.02018* (2024).

[Englert et al.(2008)] Matthias Englert, Deniz Özmen, and Matthias Westermann. 2008. The power of reordering for online minimum makespan scheduling. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 603–612.

[Ferraz et al.(2024)] Thomas Palmeira Ferraz, Kartik Mehta, Yu-Hsiang Lin, Haw-Shiuan Chang, Shereen Oraby, Sijia Liu, Vivek Subramanian, Tagyoung Chung, Mohit Bansal, and Nanyun Peng. 2024. LLM self-correction with DeCRIM: Decompose, critique, and refine for enhanced following of instructions with multiple constraints. *arXiv preprint arXiv:2410.06458* (2024).

[Fu et al.(2024)] Yichao Fu, Siqi Zhu, Runlong Su, Aurick Qiao, Ion Stoica, and Hao Zhang. 2024. Efficient LLM Scheduling by Learning to Rank. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

[Goponenko et al.(2022)] Alexander V Goponenko, Kenneth Lamar, Christina Peterson, Benjamin A Allan, Jim M Brandt, and Damian Dechev. 2022. Metrics for packing efficiency and fairness of HPC cluster batch job scheduling. In *2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 241–252.

[Kayabali(2025)] Esra Kayabali. 2025. Anthropic's Claude 3.7 Sonnet hybrid reasoning model is now available in Amazon Bedrock. https:

//aws.amazon.com/blogs/aws/anthropics-claude-3-7-sonnet-the-first-hybrid-reasoning-model-is-now-available-in-amazon-bedrock/

[Lee et al.(2014)] Minseok Lee, Seokwoo Song, Joosik Moon, John Kim, Woong Seo, Yeongon Cho, and Soojung Ryu. 2014. Improving GPGPU resource utilization through alternative thread block scheduling. In *2014 IEEE 20th international symposium on high performance computer architecture (HPCA)*. IEEE, 260–271.

[Li et al.(2022)] Boyang Li, Yuping Fan, Matthew Dearing, Zhiling Lan, Paul Rich, William Allcock, and Michael Papka. 2022. Mrsch: Multi-resource scheduling for hpc. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 47–57.

[Mekkittikul and McKeown(1998)] Adisak Mekkittikul and Nick McKeown. 1998. A practical scheduling algorithm to achieve 100% throughput in input-queued switches. In *Proceedings. IEEE INFOCOM'98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No. 98*, Vol. 2. IEEE, 792–799.

[Mirjalili and Mirjalili(2019)] Seyedali Mirjalili and Seyedali Mirjalili. 2019. Genetic algorithm. *Evolutionary algorithms and neural networks: Theory and applications* (2019), 43–55.

[Naghshnejad and Singhal(2020)] Mina Naghshnejad and Mukesh Singhal. 2020. A hybrid scheduling platform: a runtime prediction reliability aware scheduling platform to improve HPC scheduling performance. *The Journal of Supercomputing* 76 (2020), 122–149.

[OpenAI(2024)] OpenAI. 2024. Learning to reason with LLMs. https://openai.com/index/learning-to-reason-with-llms/

[Perron and Furnon(2024)] Laurent Perron and Vincent Furnon. 2024. *OR-Tools*. Google. https://developers.google.com/optimization/

[Sediq et al.(2013)] Akram Bin Sediq, Ramy H Gohary, Rainer Schoenen, and Halim Yanikomeroglu. 2013. Optimal tradeoff between sum-rate efficiency and Jain's fairness index in resource allocation. *IEEE Transactions on Wireless Communications* 12, 7 (2013), 3496–3509.

[Simon et al.(2013)] Tyler A Simon, Phuong Nguyen, and Milton Halem. 2013. Multiple objective scheduling of hpc workloads through dynamic prioritization. *UMBC Student Collection* (2013).

[Smith et al.(1999)] Warren Smith, Valerie Taylor, and Ian Foster. 1999. Using run-time predictions to estimate queue wait times and improve scheduler performance. In *Job Scheduling Strategies for Parallel Processing: IPPS/SPDP'99Workshop, JSSPP'99 San Juan, Puerto Rico, April 16, 1999 Proceedings 5*. Springer, 202–219.

[Srinivasan et al.(2002)] Srividya Srinivasan, Rajkumar Kettimuthu, Vijay Subramani, and Ponnuswamy Sadayappan. 2002. Characterization of backfilling strategies for parallel job scheduling. In *Proceedings. International Conference on Parallel Processing Workshop*. IEEE, 514–519.

[Tantakoun et al.(2025)] Marcus Tantakoun, Xiaodan Zhu, and Christian Muise. 2025. LLMs as Planning Modelers: A Survey for Leveraging Large Language Models to Construct Automated Planning Models. *arXiv preprint arXiv:2503.18971* (2025).

[WandB(2025)] WandB. 2025. Evaluating Claude 3.7 Sonnet: Performance, reasoning, and cost optimization. https://wandb.ai/byyoung3/Generative-AI/reports/Evaluating-Claude-3-7-Sonnet-Performance-reasoning-and-cost-optimization--VmlldzoxMTYzNDEzNQ

[Wang et al.(2018)] Dongshu Wang, Dapei Tan, and Lei Liu. 2018. Particle swarm optimization algorithm: an overview. *Soft computing* 22, 2 (2018), 387–408.

[Wang(2025)] Jun Wang. 2025. A Tutorial on LLM Reasoning: Relevant Methods behind ChatGPT o1. *arXiv preprint arXiv:2502.10867* (2025).

[Wang et al.(2024)] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.

[Wang et al.(2021)] Qiqi Wang, Hongjie Zhang, Cheng Qu, Yu Shen, Xiaohui Liu, and Jing Li. 2021. Rlschert: An hpc job scheduler using deep reinforcement learning and remaining time prediction. *Applied Sciences* 11, 20 (2021), 9448.

[Yao et al.(2023)] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

[Yin et al.(2025)] Junqi Yin, Jesse Hines, Emily Herron, Tirthankar Ghosal, Hong Liu, Suzanne Prentice, Vanessa Lama, and Feiyi Wang. 2025. chatHPC: Empowering HPC users with large language models. *The Journal of Supercomputing* 81, 1 (2025), 194.

[Zhao and Stankovic(1989)] Wei Zhao and John A Stankovic. 1989. Performance analysis of FCFS and improved FCFS scheduling algorithms for dynamic real-time computer systems. In *1989 Real-Time Systems Symposium*. 156–157.

[Zhu et al.(2025)] Botao Zhu, Chen Chen, Xiaoyi Fan, and Yifei Zhu. 2025. LLMSched: Uncertainty-Aware Workload Scheduling for Compound LLM Applications. *arXiv preprint arXiv:2504.03444* (2025).