Robotics & Automation Engineering Journal
ISSN: 2577-2899

Juniper Publishers
key to the Researchers

Research Article
Volume 6 Issue 5 - July 2025
DOI: 10.19080/RAEJ.2025.06.555696

Robot Autom Eng J

Evaluating Large Language Models for Workload Mapping and Scheduling in Heterogeneous HPC Systems

Aasish Kumar Sharma[1] and Julian Kunkel[1]*

[1]*Faculty of Mathematics and Computer Science, Georg-August-Universita¨t G¨ottingen, Germany*

Submission: July 04, 2025; **Published:** August 18, 2025

***Corresponding author:** Julian Kunkel, Faculty of Mathematics and Computer Science, Georg-August- Georg-August-Universita¨t Gottingen, Germany

Abstract

The rapid advances in Large Language Models (LLMs) such as ChatGPT and Claude have opened new possibilities for AI-assisted optimization and reasoning in complex technical domains. In this work, we explore the potential and limitations of LLMs for workload mapping and scheduling in heterogeneous High-Performance Computing (HPC) landscapes, a domain traditionally dominated by mathematical programming, heuristics, and reinforcement learning. We design a benchmarking framework where LLMs are prompted with structured HPC resource and workload descriptions and are evaluated against MILP-based and heuristic solvers. Our pre- liminary findings show that LLMs can generate feasible and explainable scheduling solutions, though there are some limitations in constraint satisfaction. Beyond so- lution generation, LLMs can also empower users new to the optimization domain to produce code, discuss alternative approaches, and thereby accelerate both research and production significantly. However, the benchmark results reveal that many current LLMs struggle with strict enforcement of resource and dependency constraints-particularly those involving inter-node data transfers-leading to suboptimal or infeasible schedules in some cases. Despite these challenges, leading models such as GPT-4o and Mistral Large Instruct demonstrate near-optimal performance and high solution explainability. Overall, our study highlights both the promise and current limitations of LLMs for HPC scheduling, and suggests that further research, especially in hybrid architectures and domain-specific fine-tuning, may unlock their full potential for practical high-performance computing applications.

Index terms: Large Language Models (LLMs); HPC Scheduling, Workload Mapping; Reinforcement Learning; Mathematical Programming; Heuristics; AI-assisted Optimization; High-Performance Computing (HPC); Constraint Satisfaction; Explainable Scheduling

Introduction

Heterogeneous High-Performance Computing (HPC) systems-composed of CPUs, GPUs, and specialized accelerators-enable efficient execution of diverse scientific and industrial workloads. Optimal mapping and scheduling of jobs across such resources is crucial for minimizing makespan, maximizing throughput, and ensuring efficient utilization [1-2]. Traditionally, this optimization problem has been addressed using mathematical programming, heuristic/metaheuristic algorithms, and, more recently, reinforcement learning [3-5]. However, these methods require explicit modeling, domain expertise, and are often computationally intensive for large-scale, real-world deployments.

Large Language Models (LLMs), exemplified by ChatGPT, Claude, and others, have demonstrated remarkable capabilities in code generation, reasoning, and problem de- composition. This raises a natural question: *Can LLMs contribute meaningful, feasible, and interpretable solutions to the classic problem of workload mapping and scheduling in heterogeneous HPC environments?*

In this paper, we survey the emerging intersection of LLMs and HPC optimization, provide a preliminary benchmarking methodology, and present initial findings comparing LLM-generated schedules to MILP and heuristic baselines.

Literature Review

Classical Optimization Approaches

Workload mapping and scheduling in HPC has a long history of research, with prominent approaches including integer and mixed-integer linear programming (ILP/MILP), heuristics such as Heterogeneous Earliest Finish

Time (HEFT) and Opportunistic Load Balancing (OLB), and metaheuristics (e.g., Genetic Algorithms (GA), Simulated Annealing (SA)) [4] [6-10]. These methods are valued for their guarantees of feasibility and, in the case of mathematical programming, provable optimality for small-to-medium-scale instances.

Recent work has introduced Graph Neural Networks (GNNs) and Reinforcement Learning (RL) for dynamic, adaptive scheduling in heterogeneous and time-varying contexts [11-12] [8]. Figure 1 represents a standard Petri net workflow with process and state representation. Likewise (Figure 2) represent a Standard Task Graph [9] for robot control complex graph in scheduling with default communication overheads as some of the examples of standard workloads graphs representations.

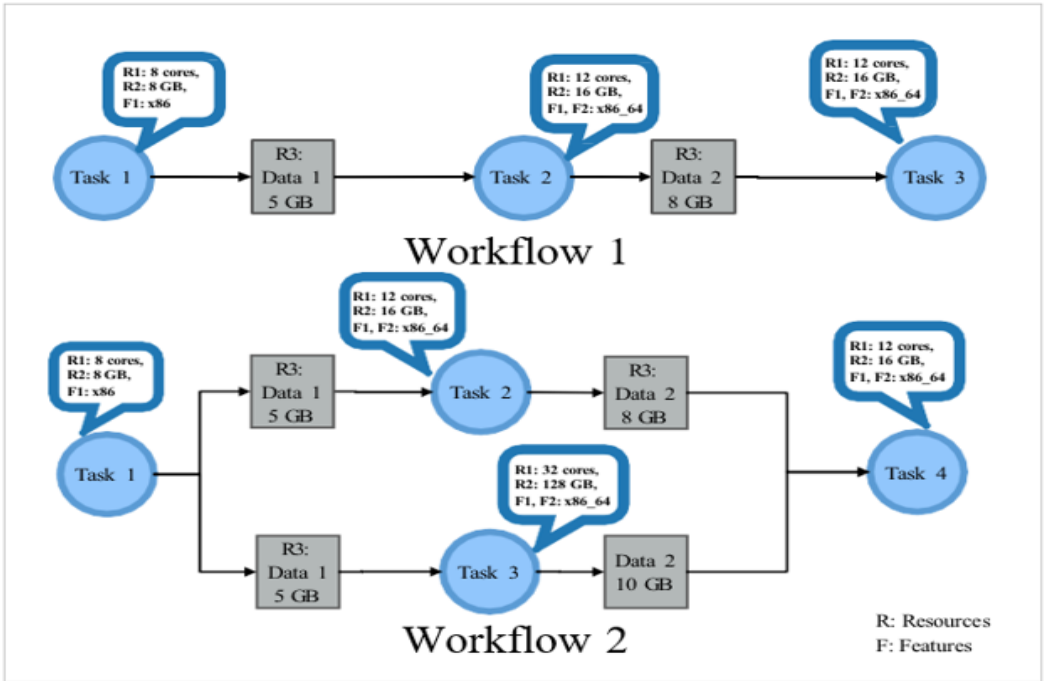

Figure 1: Simplified Petri Net representing task dependencies and resource tokens [7].

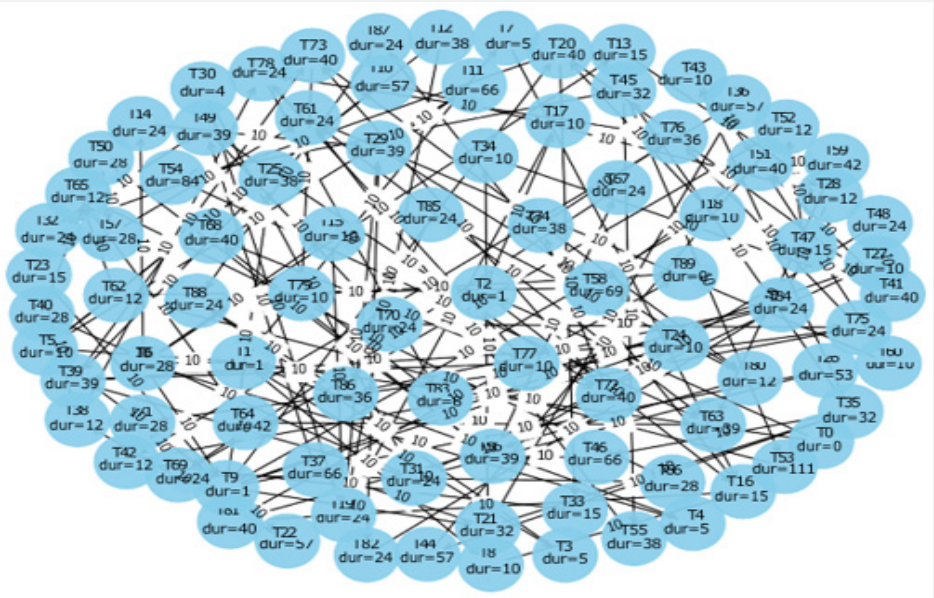

Figure 2: Workflow 4: Robot control STGS (complex) graph [8].

How to cite this article: Aasish Kumar Sharma and Julian Kunkel*. Evaluating Large Language Models for Workload Mapping and Scheduling in Heterogeneous HPC Systems. Robot Autom Eng J. 2025; 6(5): 555696. DOI: 10.19080/RAEJ.2025.06.555696

LLMs for Optimization and Scheduling

There is a growing body of work exploring LLMs for optimization tasks. Wang et al. [13] propose a benchmark suite evaluating LLMs' performance in code optimization and scheduling. Gupta et al. [14] introduce HPC-Coder, a model for automated code annotation and parallelism suggestion. LLM-based agents are increasingly proposed as planners or co-pilots for scheduling, resource allocation, and configuration tasks.

However, direct application of LLMs to multi-constraint, large-scale HPC scheduling remains under-explored, with known challenges in context length, mathematical precision, and constraint adherence. There is currently no standard benchmarking methodology for LLM-driven HPC workload mapping, motivating the present study.

Methodology

Benchmarking Framework

We propose an evaluation framework (Figure 3) where LLMs and classical solvers are presented with identical, structured descriptions of HPC systems and workloads, along with objectives and constraints. The workflow comprises:

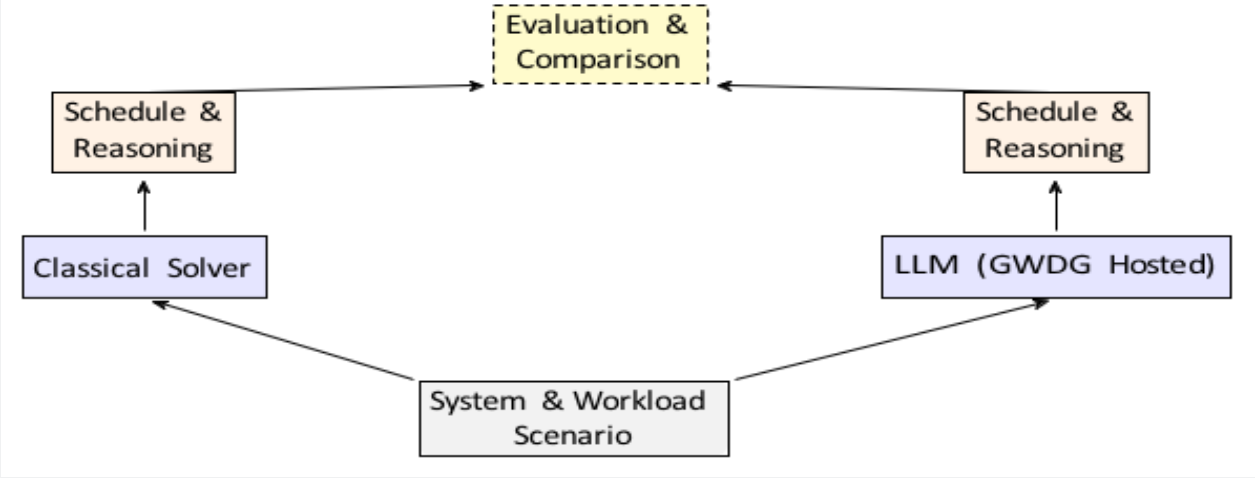

Figure 3: Methodology overview: Both classical solvers and LLMs receive the same HPC scenario, produce schedules and reasoning, which are then evaluated comparatively.

- **System Modeling**: Nodes specified with CPUs, RAM, features (e.g., GPU, SSD), data transfer rates, and power limits.
- **Workload Modeling**: Tasks defined by resource needs, feature requirements, du- rations, data sizes, and dependency graphs.
- **Objectives**: Minimizing makespan, balancing node load, and optimizing feature usage.
- **Constraints**: Task-to-node exclusivity, resource limits, dependency order, data transfer penalties, feature satisfaction, and fairness.

LLMs (ChatGPT, Claude) are prompted with the full scenario in natural language or JSON, and asked to generate job-to-node assignments, start/end times, and reasoning. Solutions are compared to those from a MILP implementation (Python/PuLP) and a heuristic baseline (e.g., OLB).

Table 3 summarizes our qualitative assessment of different LLMs in scheduling HPC workloads. Each model was prompted with the same system and workload scenario [15]. The criteria include overall makespan, task and system specification correctness, quality of reasoning, explanation clarity, code generation, and mapping validity.

Sample Scenario

A representative test scenario:

- **Nodes:**
  - NodeA: 32 CPUs, 128 GB RAM, Features: [CPU, GPU], Data Rate: 10 Gbps
  - NodeB: 64 CPUs, 256 GB RAM, Features: [CPU], Data Rate: 5 Gbps
  - NodeC: 16 CPUs, 64 GB RAM, Features: [CPU, SSD], Data Rate: 2 Gbps
- **Tasks:**
  - Task1: 8 CPUs, 32 GB RAM, [GPU], 3h, 10GB, Dependencies: []
  - Task2: 4 CPUs, 16 GB RAM, [CPU], 2h, 5GB, Dependencies: [Task1]

How to cite this article: Aasish Kumar Sharma and Julian Kunkel*. Evaluating Large Language Models for Workload Mapping and Scheduling in Heterogeneous HPC Systems. Robot Autom Eng J. 2025; 6(5): 555696. DOI: 10.19080/RAEJ.2025.06.555696

- Task3: 16 CPUs, 64 GB RAM, [CPU, SSD], 5h, 20GB, []
- Task4: 8 CPUs, 32 GB RAM, [CPU],

[1]See [15] for the current list of models hosted at GWDG.

- **Objectives**: Minimize makespan, balance load, efficient use of multi-feature nodes.
- **Constraints**: No over-allocation, respect dependencies, account for data transfer if tasks assigned to different nodes.

We use data transfer time on purpose to increase the use case to a realistic scenario when the ratio of output data increases, then it becomes an important factor. We can see if the model grasps such nuances for the experimental setup (Figure 4).

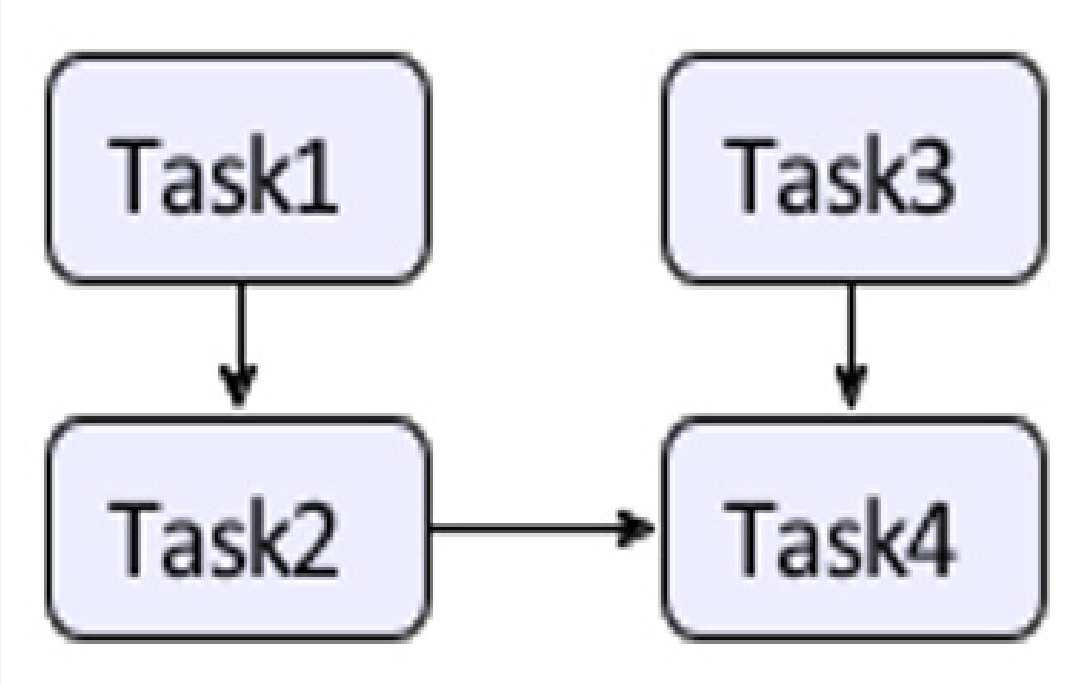

Figure 4: DAG for the sample HPC workflow.

Evaluation Metrics

- **Makespan**: Total completion time of all tasks.
- **Constraint Violations**: Resource, feature, and dependency adherence.
- **Node Utilization Balance**: Degree of load balancing.
- **Solution Explainability**: Clarity and logic of reasoning (qualitative).

Experimental Procedure

Each scenario is scheduled by:

1. A MILP optimizer (PuLP-based, extended for data transfer).
2. An OLB heuristic baseline.
3. LLMs (e.g., GPT-4, Claude) via API, with chain-of-thought prompting.

At first, we used a simple prompt, but when testing various prompts, we were assured to see already good reflection of the models and sometimes a correct result. This shows the results were often incomplete due to the ambiguity of the instructions. Therefore, we refined the prompt to this:

How to cite this article: Aasish Kumar Sharma and Julian Kunkel*. Evaluating Large Language Models for Workload Mapping and Scheduling in Heterogeneous HPC Systems. Robot Autom Eng J. 2025; 6(5): 555696. DOI: 10.19080/RAEJ.2025.06.555696

Prompt given to each LLM for benchmarking:

You are an expert high-performance computing (HPC) workload scheduler.

Below is a description of a heterogeneous HPC system and a set of computational tasks (a workflow) to be scheduled. Your objectives are:

- Each task should be assigned to a node, specifying the start and end time for each task.
- Ensure that all node resource limits (CPUs, RAM), task feature requirements, dependencies, and system specifications are satisfied.
- Consider both the computational resources and the speed of data transfer between nodes when assigning tasks, as data movement and transfer delays directly impact total makespan.
- Maximize parallel execution of independent tasks whenever node resources allow.
- Important: If a task depends on one or more other tasks that ran on different nodes, the output data transfer for each dependency may begin only after the producing (parent) task has fully completed. The dependent task may not start until all required data from its dependencies has arrived at its assigned node.
- Data transfer delay must always be added if a task's dependency ran on a different node. Use the formula:
- data transfer delay (in hours) = data size (GB) / receiving node's data transfer rate (Gbps)
- For example: if a 20GB output must be sent to a node with 10 Gbps, the transfer takes 2 hours.
- Multi-feature nodes (e.g., GPU, SSD) should be reserved for tasks that need those features; avoid assigning tasks to such nodes if not necessary.
- Provide step-by-step reasoning for each assignment, including explicit explanations for any trade-offs or data transfer considerations.
- (Optional) Generate code (Python or pseudocode) that would automate or verify the final schedule.

System Nodes

- NodeA: 32 CPUs, 128 GB RAM, Features: [CPU, GPU], Data Transfer Rate: 10 Gbps
- NodeB: 64 CPUs, 256 GB RAM, Features: [CPU], Data Transfer Rate: 5 Gbps
- NodeC: 16 CPUs, 64 GB RAM, Features: [CPU, SSD], Data Transfer Rate: 2 Gbps

Workload Tasks

- Task1: Needs 8 CPUs, 32 GB RAM, Features: [GPU], Duration: 3h, Data Output: 10GB, Dependencies: []
- Task2: Needs 4 CPUs, 16 GB RAM, Features: [CPU], Duration: 2h, Data Output: 5GB, Dependencies: [Task1]
- Task3: Needs 16 CPUs, 64 GB RAM, Features: [CPU, SSD], Duration: 5h, Data Output: 20GB, Dependencies: []
- Task4: Needs 8 CPUs, 32 GB RAM, Features: [CPU], Duration: 4h, Data Output: 15GB, Dependencies: [Task2, Task3]

How to cite this article: Aasish Kumar Sharma and Julian Kunkel*. Evaluating Large Language Models for Workload Mapping and Scheduling in Heterogeneous HPC Systems. Robot Autom Eng J. 2025; 6(5): 555696. DOI: 10.19080/RAEJ.2025.06.555696

Objectives And Constraints

- Minimize total makespan (overall time to complete all tasks), considering both computation and any required data transfer delays.
- Do not exceed any node's CPU or RAM capacity at any time.
- Each task must run entirely on a single node with all required features.
- Tasks may only start after all dependencies are finished and any required data is transferred and present on the assigned node.
- If a dependent task is assigned to a different node from its parent(s), the data transfer delay must be added before it can begin.
- Provide explanations for your assignments, especially where your mapping minimizes data transfer time or resource contention.

(Optional) Provide code (Python or pseudocode) to validate or automate the schedule.

Expected Output Format

For each task, output a table with:

- Task ID
- Assigned Node
- Start Time (considering dependencies and any transfer)
- End Time
- If data transfer was needed (yes/no, specify time and data amount)
- Short explanation

After the table, provide:

- Overall schedule make span
- Any generated code (if produced)
- Any issues or assumptions you made

(Optional) If possible, also provide an alternative schedule with a less efficient mapping and its and its resulting makespan for comparison.

Analysis of Make span: Best- and Worst-Case Scenarios

The make span, i.e., the total time to complete all tasks, is determined by both the assignment of tasks to nodes and the necessary data transfers dictated by task dependencies. To guarantee the correctness of the baseline solution, we provide a step-by-step, exhaustive proof examining all plausible node assignments for each task in the benchmark scenario. This analysis takes into account every combination of resource constraints, feature requirements, task dependencies, and mandatory data transfer delays.

All Plausible Assignments:

1. Task2 on Node A (best-case):

- Task1 on Node A: 0–3h
- Task2 on Node A: 3–5h (no transfer)
- Task3 on Node C: 0–5h
- Task4 on Node A: Waits for Task2 (5h) and Task3's output from Node C:
  - 20GB / 10Gbps = 2h; `ready at 7h. Starts at 7h, ends at 11h.

2. Task2 on Node B:

- Task2 requires Task1's output:
  - 10GB / 5Gbps = 2h. Starts at 5h, ends at 7h.
- Task4 on Node B:
  - Waits for Task2 (7h) and Task3's output from Node C:
    - 20GB / 5Gbps = 4h (ready at 9h). Starts at 9h, ends at 13h.
- Task4 on NodeA:
  - Needs Task2 output from NodeB:
    - 5GB / 10Gbps = 0.5h (ready at 7.5h), Task3 output as above (ready at 7h). Starts at 7.5h, ends at 11.5h.
- Task4 on NodeC:
  - Needs Task2 output from NodeB:
    - 5GB / 2Gbps = 2.5h (ready at 9.5h), Task3 local (5h). Starts at 9.5h, ends at 13.5h.

3. Task2 on NodeC:

- Task2 requires Task1's output:

How to cite this article: Aasish Kumar Sharma and Julian Kunkel*. Evaluating Large Language Models for Workload Mapping and Scheduling in Heterogeneous HPC Systems. Robot Autom Eng J. 2025; 6(5): 555696. DOI: 10.19080/RAEJ.2025.06.555696

- 10GB /2Gbps = 5h. Starts at 8h, ends at 10h.
- Task4 on NodeC:
- Waits for Task2 (10h), Task3 local (5h). Starts at 10h, ends at 14h.
- Task4 on NodeA:
- Needs Task2 output:
  - 5GB /10Gbps = 0.5h (ready at 10.5h), Task3 output:
  - ✓ 20GB /10Gbps = 2h (ready at 7h). Starts at 10.5h, ends at 14.5h.
- Task4 on NodeB:
- Needs Task2 output:
  - 5GB /5Gbps = 1h (ready at 11h), Task3 output:
  - ✓ 20GB /5Gbps = 4h (ready at 9h). Starts at 11h, ends at 15h (Table 1).
  - ✓ **Optimal Solution:** In every scenario, the minimal possible makespan is 11 hours. This step-by-step analysis ensures the validity and reproducibility of our baseline, and sets a transparent standard for evaluating the scheduling methods.

Table 1: Summary: Earliest possible Task4 completion for all valid Task2 and Task4 node assignments.

Task2 Node	Task4 Node	Task4 Start (h)	Task4 End (h)	Makespan (h)
NodeA	NodeA	7	11	11
NodeB	NodeA	7.5	11.5	11.5
NodeB	NodeB	9	13	13
NodeC	NodeC	10	14	14
NodeC	NodeA	10.5	14.5	14.5
NodeC	NodeB	11	15	15

Explanation of Baseline Schedule and Makespan:

The scheduling graph and (Table 2) illustrate the optimal solution for our benchmark scenario. Tasks are mapped to nodes according to their feature requirements and re- source constraints. Both Task1 (GPU) and Task3 (SSD) begin in parallel on NodeA and NodeC, respectively. Task2, which depends on Task1, is co-located on NodeA to eliminate unnecessary data transfer. After both Task2 and Task3 complete, Task4 can begin only when all dependencies and required data are available on its assigned node (NodeA). Since Task3's output must be transferred from NodeC to NodeA (20GB over a 10Gbps link, resulting in a 2-hour delay), Task4's start is determined by the completion of data transfer rather than the earlier completion of Task2 (Figure 5).

Table 2: Summary of optimal schedule for the benchmark scenario (critical path method).

Task	Assigned Node	Start (h)	End (h)	Data Transfer	Explanation
Task1	NodeA	0	3	None	Only NodeA has GPU
Task2	NodeA	3	5	None	Follows Task1, no transfer needed
Task3	NodeC	0	5	None	Only NodeC has SSD
Task4	NodeA	7	11	20GB from NodeC (2h over 10Gbps)	Waits for data from Task3 via network

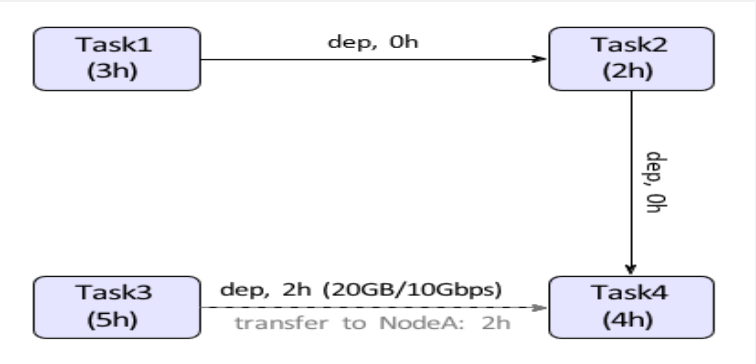

Figure 5: DAG for the sample HPC workflow.

How to cite this article: Aasish Kumar Sharma and Julian Kunkel*. Evaluating Large Language Models for Workload Mapping and Scheduling in Heterogeneous HPC Systems. Robot Autom Eng J. 2025; 6(5): 555696. DOI: 10.19080/RAEJ.2025.06.555696

This approach makes explicit the importance of both parallel task execution and minimizing data transfer delays in heterogeneous environments. The resulting critical path-Task1 → Task2 → (wait for Task3 data) → Task4-yields a minimal makespan of11 hours. This solution serves as a robust baseline for evaluating automated schedulers or LLM-based workload mapping approaches.

Results are parsed and compared for feasibility, optimality, and explanation quality.

Findings and Discussion

Our preliminary tests with scenarios of moderate size (<10 tasks, <5 nodes) show that LLMs such as GPT-4 and Claude can generate feasible job-to-node assignments and produce human-like, explainable reasoning for their choices. In many cases, LLM- generated schedules respect basic constraints (resource limits, dependencies) and approximate heuristic solutions.

However, several limitations are evident:

- **Scaling**: LLMs struggle with long, highly structured prompts as the scenario size grows, sometimes leading to incomplete or infeasible solutions.
- **Constraint Adherence**: Strict, non-negotiable constraints (e.g., data transfer delay, resource exclusivity) may be violated unless explicitly emphasized and tested.
- **Optimality**: MILP solvers consistently find lower makespan solutions, particularly for complex dependencies and high resource utilization. LLMs tend toward "greedy" or "naive" allocations unless specifically guided.
- **Explainability**: LLMs excel at providing explanations for scheduling choices, an area where classical solvers are lacking.

Notably, LLMs may be best positioned as co-pilots or advisors-generating candidate solutions, explaining heuristics, or translating human objectives into machine-usable constraints-rather than replacing solvers in mission-critical optimization.

Discussion of Model Benchmarking Results

Table 3 summarizes the comparative performance of state-of-the-art large language mod- els (LLMs) and code-oriented AI systems on the canonical HPC scheduling scenario. We assess the results based on what it produced denoted by + (correct), 0 (not given), - (wrong), NA (not able). Several key trends and insights emerge from this evaluation:

Table 3: Comprehensive qualitative assessment of LLM scheduling and reasoning on HPC mapping benchmarks. Columns: **Makespan** (total completion time), **Task Through- put** (successful task completion), **Mapping/Node Utilization** (efficient, correct use of resources), **Constraint Adherence** (resource, dependency, and transfer constraints), **Reasoning** (step-by-step logic), Explanation (clarity), **Code** (artifact produced), **Solution Explainability** (transparency and interpretability of solution).

Model	Makespan	Task Throughput	Mapping / Node Util.	Constraint Adherence	Reasoning	Explanation	Code	Solution Explainability
Llama 3.1 8B Instruct	14h	+	0	0	0	+	0	+
Gemma 3 27B Instruct	9h	+	0	0	0	+	+	+
InternVL2.5 8B MPO	19h	+	0	0	0	+	+	+
Qwen 3 32B	11.5h	+	+	+	+	+	0	+
DeepSeek R1	11h	+	+	+	+	+	0	+
Mistral Large Instruct	11h	+	+	+	+	+	+	+
Codestral 22B	9h	+	0	0	0	+	+	+
o3	11h	+	+	+	+	+	+	+
o3-mini	11h	+	+	+	+	+	+	+
GPT-4o	12.5h	+	+	+	+	+	+	+
GPT-4.1	11.5h	+	+	+	+	+	+	+
GPT-4.1 Mini	9h	+	+	0	0	+	+	+

- **Task Throughput and Feasibility:** All models successfully generated schedules that mapped every task to an available node, demonstrating a strong capacity for producing valid, feasible solutions in moderately sized, feature-rich HPC environments.
- **Constraint Adherence and Make span Accuracy :** The most prominent area of differentiation between models was their ability to enforce strict resource, dependency, and especially data transfer constraints. Only a subset of models-such as o3, o3-mini, DeepSeek R1, Qwen 3 32B,

How to cite this article: Aasish Kumar Sharma and Julian Kunkel*. Evaluating Large Language Models for Workload Mapping and Scheduling in Heterogeneous HPC Systems. Robot Autom Eng J. 2025; 6(5): 555696. DOI: 10.19080/RAEJ.2025.06.555696

and Mistral Large Instruct-consistently calculated correct transfer delays and critical path timings, producing the minimal or near-minimal makespan (11–11.5 hours). Many others, including Llama 3.1 8B, Gemma 3 27B, Codestral 22B, and GPT-4.1 Mini, underestimated makespan by overlooking inter-node data movement, while others (e.g., InternVL2.5 8B MPO) overestimated by serializing independent tasks.

- **Mapping and Node Utilization:** High-performing models leveraged resource locality and feature matching, co-locating dependent tasks where possible and as- signing tasks requiring special hardware (GPU, SSD) to the appropriate nodes. Suboptimal models sometimes failed to minimize data transfers or maximize par- allelism, which directly impacted solution quality.
- **Reasoning, Explanation, and Code Generation:** Almost all advanced models offered step-by-step rationales and transparent scheduling logic, and many (including o3, GPT-4o, Codestral 22B, and Mistral Large Instruct) generated readable code for schedule validation. This growing explainability makes LLMs increasingly valuable for human-in-the-loop and educational use cases.
- **Explainability of Solutions:** Even where logical errors occurred, most models presented their reasoning in a transparent manner, aiding diagnosis and benchmarking. This suggests that LLMs are already strong as "AI co-pilots" and explainers, even if their autonomous optimization skills are not yet universally reliable.
- **Summary:** These findings demonstrate that while LLMs are not yet a replacement for classical optimization methods in strict, mission-critical scheduling, the leading models can now approach or match the best-known solutions for moderately complex scenar- ios. Their outputs are interpretable, often auditable via code, and can form a powerful basis for hybrid scheduling pipelines and operator support tools. Nonetheless, rigorous prompt engineering, constraint validation, and continuous benchmarking remain essential to ensure the correctness and reliability of LLM-generated schedules.

Example Outputs

Select snippets of interesting results.

Llmama 3.1 8B Capturing Features of System and the Tasks

```
# Define the graph G = nx.DiGraph()

# Add nodes and edges
G.add node (" NodeA", cpus=32, ram=128, features= [" CPU"," GPU"], data rate=10) G.add node (" NodeB", cpus=64, ram=256, features= [" CPU"], data rate=5)
G.add node (" NodeC", cpus=16, ram=64, features= [" CPU"," SSD"], data rate=2)

G.add edge (" NodeA"," Node A", task=" Task1", cpus=8, ram=32, features= [" GPU"], duration=3, data=10)
G.add edge (" Node A"," Node A", task=" Task2", cpus=4, ram=16, features= [" CPU"], duration=2, data=5)
G.add edge (" Node"," NodeB", task=" Task3", cpus=16, ram=64, features= [" CPU"," SSD"], duration=5, data=20)
```

First, assign Task1 to NodeA. That uses 8 CPUs and 32 GB RAM. NodeA has 32 CPUs and 128 GB, so after Task1, remaining is 24 CPUs and 96 GB. That's fine.

Task3 needs SSD, so NodeC. NodeC has 16 CPUs and 64 GB. Task3 uses 16 CPUs and 64 GB, so it fills up NodeC completely. So Task3 is on NodeC.

Conclusion and Future Work

Conclusion

This benchmark evaluation demonstrates both the promise and the current limitations of Large Language Models (LLMs) in high-performance computing (HPC) workload map- ping and scheduling. Many advanced LLMs, including GPT-4o, o3, DeepSeek R1, Qwen 3 32B, and Mistral Large Instruct, are now able to generate feasible, interpretable, and near-optimal scheduling solutions for moderately sized, feature-rich scenarios. Nonetheless, significant challenges remain. In particular, strict constraint satisfaction is not always achieved, as many models underestimated or misapplied data transfer delays and critical path logic, which resulted in overly optimistic makespans or suboptimal parallelism.

Furthermore, as the scale and complexity of dependency graphs increase, some models either resorted to excessive serialization-overestimating makespan-or failed to fully exploit available parallelism. While most leading models now provide stepwise rationales and readable code, systematic and automated assessment of solution explainability and error detection remains an area for further development. These findings suggest that, while LLMs show considerable promise, additional advances are needed to fully realize their potential for robust and efficient HPC scheduling.

How to cite this article: Aasish Kumar Sharma and Julian Kunkel*. Evaluating Large Language Models for Workload Mapping and Scheduling in Heterogeneous HPC Systems. Robot Autom Eng J. 2025; 6(5): 555696. DOI: 10.19080/RAEJ.2025.06.555696

Future Work

This initial benchmarking study identifies several promising directions for advancing the integration of large language models (LLMs) into high-performance computing (HPC) scheduling workflows. First, future research should explore the fine-tuning of LLMs using realistic scheduling traces and explicit constraint rules to improve their reliability and domain alignment. The development of hybrid architectures, in which LLMs generate initial scheduling blueprints subsequently refined by exact solvers such as mixed integer linear programming (MILP) or by learning-based agents (e.g., reinforcement learning), also represents a promising avenue for achieving both efficiency and optimality. In addition, there is a clear need for systematic, large-scale benchmarking on both real-world and synthetic HPC workloads, supported by standardized prompts and reference solutions, to facilitate objective comparison and progress tracking across the community.

Finally, the creation of automated tools for scoring, validating, and providing feed- back on the explainability and correctness of LLM-generated schedules will be critical for practical adoption. Ultimately, these findings support a vision in which LLMs are not standalone replacements for established scheduling solvers, but rather serve as explainers, translators, and heuristic generators-working in concert with classical optimization techniques and human experts to advance the state of HPC workflow optimization.

Acknowledgement

This work was supported by the Faculty of Mathematics and Computer Science, University of Gottingen. The authors thank the HPC and AI research community for inspiration. The authors sincerely thank the Georg-August-Universität Göttingen (GWDG, Germany) for their valuable support. We also thank for the constructive feedback from our peers. This research is funded by the EU KISSKI Project under under grant number 01-S22093A (Forderkennzeichen).

References

1. Deelman E, Dongarra J, Hendrickson B, Randles A, Reed D, et al. (2025) High-performance computing at a crossroads. Science 387(6736): 829-831.
2. Brodtkorb AR, Dykan C, Hagen TR, Hjelmervik JM, Storaasli OO (2010) State-of-the-art in heterogeneous computing. Scientific Programming 18(1): 1-33.
3. Sharma AK, Kunkel J (2025) A Review of Tools and Techniques for Optimization of Workload Mapping and Scheduling in Heterogeneous HPC System. arXiv preprint.
4. Topcuoglu H, Hariri H, Wu M-Y (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE transactions on parallel and distributed systems 13(3): 260-274.
5. Adhikari M, Amgoth T, Srirama SN (2019) A survey on scheduling strategies for workflows in cloud environment and emerging trends. ACM Computing Surveys 52(4): 1-36.
6. Kwok YK, Ahmad I (1999) Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computing Surveys (CSUR) 31(4): 406-471.
7. Sharma AK, Boehme C, Gelß P, Yahyapour R, Kunkel J (2025) Workflow-Driven Modeling for the Compute Continuum: An Optimization Approach to Automated System and Workload Scheduling. arXiv preprint.
8. Sharma AK, Kunkel J (2025) GrapheonRL: A Graph Neural Network and Rein- forcement Learning Framework for Constraint and Data-Aware Workflow Mapping and Scheduling in Heterogeneous HPC Systems. arXiv preprint.
9. Tobita T, Kasahara H (2002) A standard task graph set for fair evaluation of multi- processor scheduling algorithms. Journal of Scheduling 5(5): 379-394.
10. Hussain H, Malik SUR, Hameed A, Khan SU, Bickler G, et al. (2013) A survey on resource allocation in high performance distributed computing systems. Parallel Computing 39(11): 709-736.
11. Grinsztajn N, Beaumont O, Jeannot E, Preux P (2021) Readys: A reinforcement learning based strategy for heterogeneous dynamic scheduling. IEEE International Conference on Cluster Computing (CLUSTER) pp.70-81.
12. Cui B, Ramesh T, Hernandez O, Zhou K (2025) Do large language models understand performance optimization? arXiv preprint.
13. Ye Z, Geo W, Hu Q, Sun P, Wang X, et al. (2024) Deep learning workload scheduling in gpu datacenters: A survey. ACM Computing Surveys 56(6): 1-38.
14. Nichols D, Marathe A, Menon H, Gamblin T, Bhatele A (2024) Hpc-coder: Modeling parallel programs using large language models. ISC High Performance 2024 Research Paper Proceedings (39th International Conference) pp.1-12.
15. (2025) GWDG Chat AI Services Available Models. Documentation for HPC.

How to cite this article: Aasish Kumar Sharma and Julian Kunkel*. Evaluating Large Language Models for Workload Mapping and Scheduling in Heterogeneous HPC Systems. Robot Autom Eng J. 2025; 6(5): 555696. DOI: 10.19080/RAEJ.2025.06.555696

DOI: 10.19080/RAEJ.2025.06.555696

How to cite this article: Aasish Kumar Sharma and Julian Kunkel*. Evaluating Large Language Models for Workload Mapping and Scheduling in Heterogeneous HPC Systems. Robot Autom Eng J. 2025; 6(5): 555696. DOI: 10.19080/RAEJ.2025.06.555696