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-Universität Göttingen, Germany

July 2025

Abstract

Large Language Models (LLMs) are increasingly investigated for technical reasoning, yet their capacity to perform structured, constraint-based optimization purely from natural-language input remains insufficiently understood. This study evaluates 21 publicly available LLMs on a representative heterogeneous High-Performance Computing (HPC) workload mapping and scheduling problem. Each model receives the same textual description of system nodes, task requirements, and scheduling constraints, and must assign tasks to nodes, compute the total makespan, and explain its reasoning. A manually derived analytical optimum of 9 h 20 s serves as the ground-truth reference.

Three models (`o3-mini`, `GPT-4.1 Mini`, and `Gemini Pro 2.5`) exactly reproduced the analytical optimum while satisfying all constraints, whereas twelve additional models achieved near-optimal results within two minutes of the reference. The remaining models produced suboptimal schedules, often due to arithmetic inaccuracies or dependency violations. All models achieved full task coverage and feasible node mappings, though only about half maintained strict constraint adherence. 19 generated partially executable verification code, and 18 provided coherent step-by-step reasoning, demonstrating strong interpretability even when logical errors occurred.

These results delineate the current capability boundary of LLM reasoning in combinatorial optimization: leading models can reconstruct optimal schedules directly from natural-language descriptions, but most still struggle with precise timing, data-transfer arithmetic, and dependency enforcement. The findings underscore LLMs' promise as explainable co-pilots for optimization and decision-support tasks, rather than autonomous solvers.

Index terms— Large Language Models (LLMs), HPC Scheduling, Constraint Reasoning, Makespan Optimization, Explainable AI

1 Introduction

Heterogeneous High-Performance Computing (HPC) systems that integrate CPUs, GPUs, and domain-specific accelerators form the backbone of modern scientific computing and large-scale data processing [1, 2]. Efficient workload mapping and scheduling across such diverse resources are essential to minimize makespan, maximize throughput, and ensure balanced utilization. However, the scheduling problem is inherently combinatorial and NP-hard, typically addressed through mathematical programming, heuristics, or meta-heuristics [4, 6, 3]. While these classical methods can yield optimal or near-optimal solutions for small-scale instances, they require explicit system modeling, precise mathematical formulations, and domain expertise, making them less accessible and computationally intensive for dynamic real-world scenarios.

Recent advances in Large Language Models (LLMs), such as GPT-4, Claude, and Qwen, have shown that they can perform symbolic reasoning, follow complex instructions, and generate structured code directly from natural-language input. These emerging capabilities raise a compelling research question:

> *Can LLMs reason about multi-constraint optimization problems, such as HPC workload mapping and scheduling, purely from natural-language descriptions, without relying on formal equations or optimization solvers?*

To address this question, we designed a capability evaluation experiment that isolates reasoning quality rather than algorithmic performance. Each LLM is provided with a full natural-language description of a heterogeneous HPC system, including nodes, resources, task requirements, dependencies, and objectives. Without any structured or mathematical input, the model must produce a valid mapping and scheduling plan, compute the overall makespan, and explain its reasoning process. The generated solutions are compared against a *manually derived and analytically verified optimal schedule* that serves as the ground truth. This setup enables a direct assessment of whether an LLM can reconstruct the logic of constrained optimization and emulate human-style reasoning in HPC scheduling contexts.

Research Questions

- **RQ1:** Can LLMs comprehend and reason over natural-language descriptions of resource, feature, and dependency constraints in HPC scheduling problems?
- **RQ2:** Can they independently derive optimal or near-optimal schedules and compute accurate makespans?
- **RQ3:** What reasoning errors and failure patterns emerge, and what do they reveal about the current limits of LLMs' structured optimization reasoning?

Contributions. This paper bridges HPC workload mapping and scheduling with AI reasoning by: (i) introducing a manually validated framework to assess the capability of Large Language Models (LLMs) to perform constraint reasoning from natural-language descriptions of systems and workloads on a single prompt; (ii) conducting a systematic evaluation of 21 publicly available models using a common analytical benchmark; (iii) establishing a taxonomy of typical reasoning and constraint-violation patterns that characterize LLM-based optimization attempts; and (iv) defining the current capability

boundary of LLMs in structured reasoning and positioning them as explainable co-pilots for future hybrid human–solver scheduling systems.

The remainder of this paper is organized as follows. Section 2 reviews related work on traditional and AI-assisted scheduling approaches. Section 3 details the proposed evaluation framework, benchmark setup, and prompt design. Section 4 presents the empirical findings and comparative analysis across models along with discusses observed reasoning behaviors and failure patterns. Finally, Section 5 concludes the study and outlines directions for advancing LLM-assisted optimization in heterogeneous HPC systems.

2 Literature Review

2.1 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 [7].

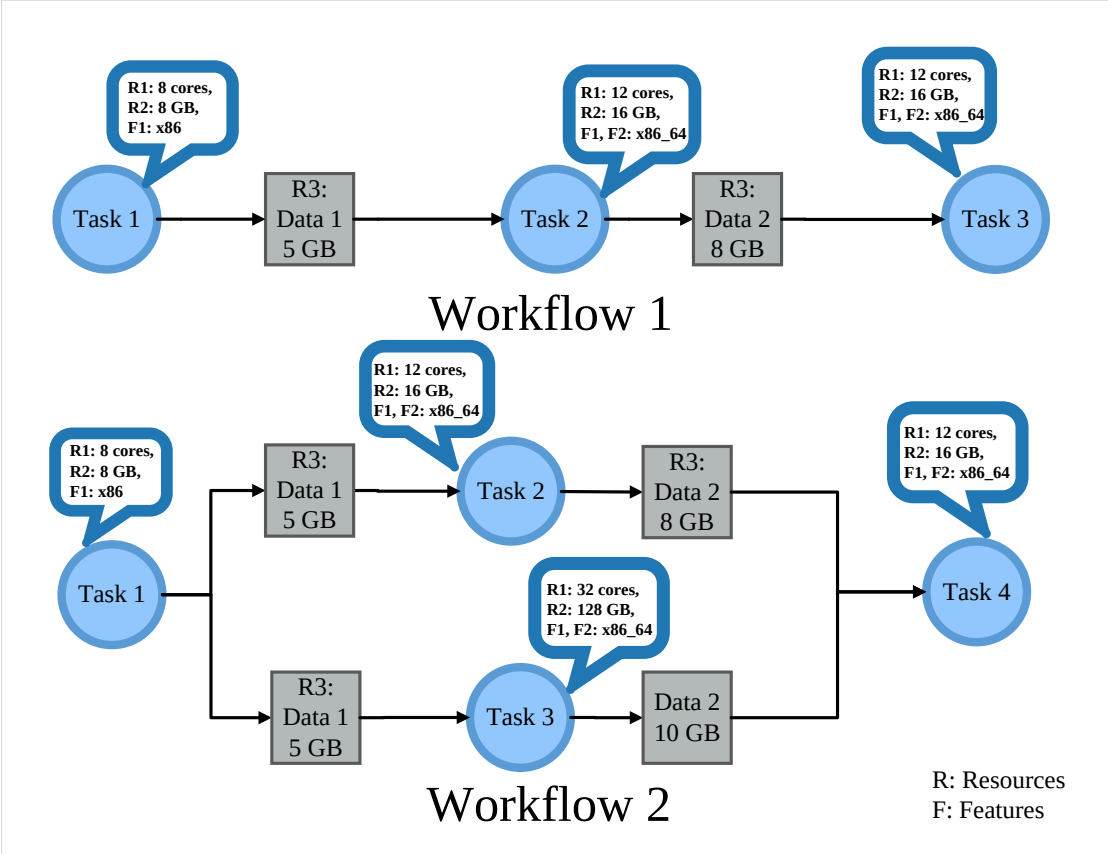

Figure 1: Simplified Petri Net representing task dependencies and resource tokens [7].

Recent work has introduced Graph Neural Networks (GNNs) and Reinforcement Learning (RL) for dynamic, adaptive scheduling in heterogeneous and time-varying con-

texts [11, 12, 8]. Figure 1 represents a standard Petri net workflow with process and state representation.

2.2 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. While classical scheduling benchmarks exist for algorithmic and solver-based approaches, there is currently no standardized methodology for evaluating LLMs' reasoning ability in HPC workload mapping from natural-language descriptions, motivating the present study.

To date, no published work has directly evaluated LLMs' intrinsic reasoning ability to perform scheduling and makespan computation from natural-language instructions alone. The present study fills this gap by providing a manually validated benchmark of reasoning correctness and constraint adherence.

3 Methodology

3.1 Benchmarking Framework

We design an evaluation framework (Fig. 2) to test whether LLMs can internally reason about mapping and scheduling constraints and objectives solely from natural-language instructions. Unlike solver-based benchmarks, the reference task-to-node map and schedule in our setup is *manually derived* through analytical reasoning and validated step-by-step for feasibility, optimality, and makespan. LLMs are provided with the same textual description of the scenario-no equations, JSON input, or symbolic hints, and must independently reconstruct and solve the problem. Their responses are evaluated for constraint satisfaction, correctness of makespan computation, and clarity of explanation.

The modeling details as well as the mathematical and the heuristics solvers are based on [7]. Besides, LLMs are prompted with the full scenario in natural language, and asked to generate task-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., HEFT). [16] presents, LLMs provides better solution when refined prompts are looped with proper reasoning which could be another analysis for us, while for current benchmarking we do not loop.

Table 2 summarizes our qualitative assessment of different LLMs in scheduling HPC workloads. Each LLM model was prompted with the same system and workload scenario. The criteria include overall makespan, task and system specification correctness, quality of reasoning, explanation clarity, code generation, and mapping validity.

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

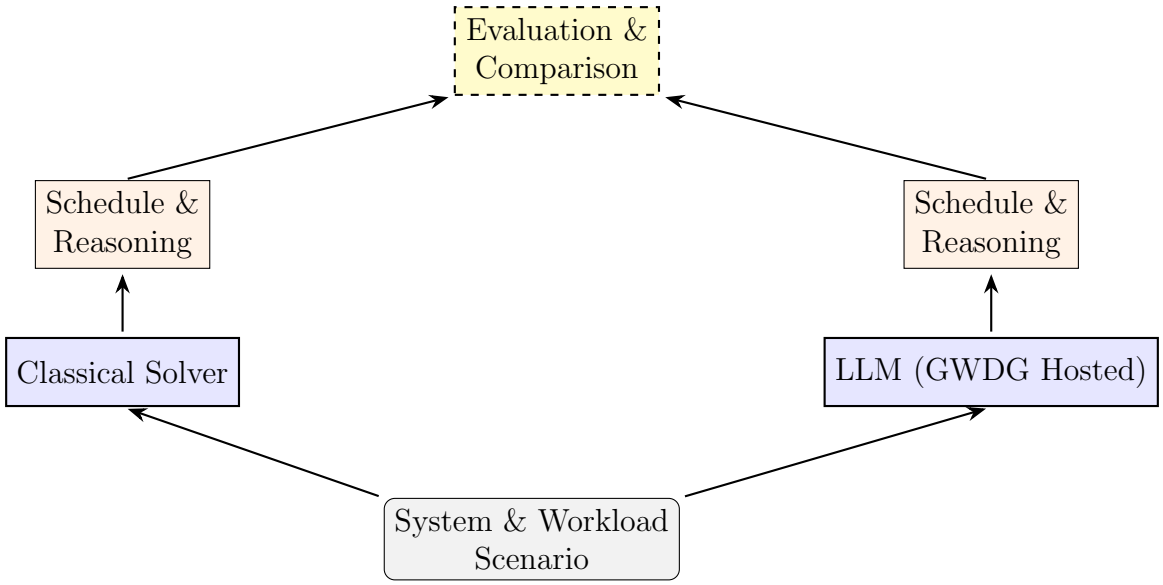

Figure 2: Evaluation methodology: Each LLM receives the same natural-language description of the HPC system and workload as used in the manually derived optimal baseline. Outputs are compared for constraint adherence, correctness of makespan, and quality of reasoning and explanation.

3.2 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]
  - Task3: 16 CPUs, 64 GB RAM, [CPU, SSD], 5h, 20GB, []
  - Task4: 8 CPUs, 32 GB RAM, [CPU], 4h, 15GB, Dependencies: [Task2, Task3]
- *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 grasp such nuances for the experimental setup.

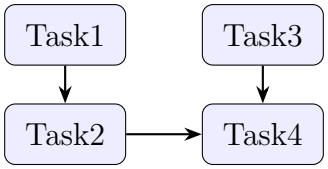

Figure 3: DAG for the sample HPC workflow

3.3 Evaluation Metrics

Following metrics are considered for the evaluation of the models 1) *Makespan*: Total completion time of all tasks, 2) *Constraint Violations*: Resource, feature, and dependency adherence, 3) *Node Utilization Balance*: Degree of load balancing, and 4) *Solution Explainability*: Clarity and logic of reasoning (qualitative).

The baseline makespan and mapping used for comparison correspond to the analytically verified optimum of 9 h 20 s, computed manually. This ensures that evaluation reflects each model's reasoning fidelity rather than differences in algorithmic implementation.

3.4 Experimental Procedure

Each scenario is evaluated manually for a baseline solution then by the solvers like a MILP optimizer (PuLP-based, extended for data transfer), an HEFT heuristic baseline and LLMs via API, with chain-of-thought prompting. For this sample scenario, we get minimum makespan of 9 hours and 20 seconds with 100% task throughput (all four tasks assigned) and with resource utilization of two out of three nodes, as we prioritized makespan over optimum resource usage.

For LLMs, at first we used a simple prompt, but during testing various prompts, we were assured to see already good reflection of the models and sometimes a correct results. This shows the results were often incomplete due to the ambiguity of the instructions, so we further refined the prompt as shown below:

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 following formula to calculate: Data Transfer Time (in hours) = ((Data Size (GB) ∗ 8 bits/Byte) / (Min(Source Node Data Rate (Gbps) , Destination Node Data Rate (Gbps)))) / 3600 seconds/hour)
- For example: if a 20GB output must be sent to a node with 10 Gbps, the transfer takes 16 seconds.
- 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]

OPTIMIZATION OBJECTIVES:
1. Minimize makespan,
2. Optimum Resource Utilization.

CONSTRAINTS:
1. One task should be assigned to only one node.
2. Task should be assigned within the node capacity.
3. Task feature request should be respected.
4. Task dependency should be respected.
5. Data transfer time should be considered in case if dependent tasks are assigned to different nodes.

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 makespan
– 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 resulting makespan for comparison.

3.4.1 Explanation of Baseline Map-Schedule and Makespan:

Table 1 enumerates all feasible task–node combinations for T_2 and T_4, since T_1 (GPU) and T_3 (SSD) have fixed node assignments. The makespan is determined by the longest dependency chain and any required inter-node data transfers. As shown, mapping T_2 to NodeA and T_4 to NodeC yields the minimum overall completion time of **9 h 20 s**, as it avoids the large 20 GB transfer from NodeC to NodeA. This configuration forms the analytical optimum and is used as the ground truth for evaluating LLM results.

Table 1: All task–node combinations for T_2 and T_4 with transfer delays and resulting makespan. $T_1 \rightarrow A$ (GPU) and $T_3 \rightarrow C$ (SSD) are fixed. Transfer times use $(\mathrm{GB} \cdot 8)/\min(\mathrm{Gbps}_{\mathrm{src}}, \mathrm{Gbps}_{\mathrm{dst}})$.

T_2 **node**	T_4 **node**	$T_1 \rightarrow T_2$ **(s)**	$T_2 \rightarrow T_4$ **(s)**	$T_3 \rightarrow T_4$ **(s)**	T_4 **start (h:m:s)**	**Makespan (h:m:s)**
A	A	0	0	80	5:01:20	9:01:20
A	B	0	8	80	5:01:20	9:01:20
A	**C**	**0**	**20**	**0**	**5:00:20**	**9:00:20**
B	A	16	8	80	5:01:20	9:01:20
B	B	16	0	80	5:01:20	9:01:20
B	C	16	20	0	5:00:36	9:00:36
C	A	40	20	80	5:01:20	9:01:20
C	B	40	20	80	5:01:20	9:01:20
C	C	40	0	0	5:00:40	9:00:40

Notes: T_2 starts at 3 h plus $T_1 \to T_2$ transfer if off-node (10 GB: A→B 16 s, A→C 40 s). T_4 begins once both T_2 and T_3 outputs arrive at its assigned node. The optimal mapping is $T_2 \to A$, $T_4 \to C$, achieving a minimal makespan of **9 h 20 s** by keeping T_3 local on NodeC and transferring only 5 GB from NodeA (20 s delay).

4 Findings and Discussion

4.1 Findings

In evaluating LLMs alongside traditional optimization approaches [7], several consistent patterns emerged. Table 2 summarizes their behavior across all 21 publicly available models tested under a 30 s response-time threshold. Every model successfully produced a feasible schedule, achieving full task coverage and balanced node utilization. However, the resulting schedules often varied across repeated runs with identical prompts, reflecting the intrinsic stochasticity of generative inference. This variability indicates that current LLM-generated schedules should be interpreted as probabilistic, near-feasible solutions rather than deterministic optima.

Table 2 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).

Table 2: Qualitative assessment of LLM performance on HPC workload mapping and scheduling benchmarks (inference parameters: temperature = 0.5, top_p = 50%). Columns denote: *Makespan* (total workflow completion time), *Task Throughput* (percentage of tasks successfully scheduled), *Node Utilization* (efficient resource usage), *Constraint Adherence* (compliance with resource, dependency, and transfer limits), *Reasoning* (logical step-by-step justification), *Explanation* (clarity of narrative output), *Code* (presence and functionality of generated validation code), and *Solution Explainability* (overall transparency and interpretability of the result).

Model	Optimum Makespan	Task Throughput	Node Util.	Constraint Adherence	Reasoning	Explanation	Working Code	Response time
Llama 3.1 8B	9h 32s	+	+	-	-	+	-	+
Gemma 3 27B	9h 1m 28s	+	+	-	-	+	+	+
InternVL2.5 8B MPO	20h 16s	+	+	-	-	+	+	+
Qwen 3 32B	9h 1m 20s	+	+	+	+	-	+	-
DeepSeek R1	11h	+	+	-	-	+	-	-
Llama 3.1 Sauerkraut70B	9h 16m 32s	+	+	-	+	+	+	+
Mistral Large	9h 1m 28s	+	+	-	+	+	+	-
Codestral 22B	12h 32m	+	+	-	-	+	+	-
Qwen 2.5 VL 72B	9h	+	+	-	-	+	+	-
Qwen 2.5 Coder 32B	9h 4s	+	+	-	-	+	+	+
o3	9h 60s	+	+	+	+	+	+	-
o3-mini	9h 20s	+	+	+	+	-	-	+
GPT-4o	9h 32s	+	+	-	+	+	+	+
GPT-4o Mini	9h 8s	+	+	-	-	+	+	+
GPT-4.1	9h 1m 20s	+	+	+	+	+	+	+
GPT-4.1 Mini	9h 20s	+	+	+	+	+	+	+
Gemini Flash 2.5	9h 1m 20s	+	+	+	+	-	+	-
Gemini Pro 2.5	9h 20s	+	+	+	+	+	+	+
Microsoft Copilot	9h 1m 20s	+	+	+	+	+	+	+
Grok	9h 1m 20s	+	+	+	+	+	+	+
Claue Sonnet 4	9h 1m 20s	+	+	+	+	+	+	+

Among the 21 models evaluated, three (`o3-mini`, `GPT-4.1 Mini`, and `Gemini Pro 2.5`) precisely reproduced the analytically verified optimum makespan of 9 h 20 s while

fully satisfying all resource, feature, and dependency constraints. Twelve additional models, including o3, GPT-4.1, Qwen 3 32B, Qwen 2.5 VL 72B, Qwen 2.5 Coder 32B, Mistral Large, Gemma 3 27B, GPT-4o Mini, Gemini Flash 2.5, Microsoft Copilot, Grok, and Claude Sonnet 4, achieved near-optimal makespans within one to two minutes of the reference. Models such as Llama 3.1 8B, Llama 3.1 Sauerkraut 70B, DeepSeek R1, Codestral 22B, and InternVL2.5 8B MPO produced suboptimal results or violated one or more constraints, primarily due to miscalculations in data-transfer delays or serialized execution of independent tasks. Across all models, task throughput and node utilization reached 100%, but only about half maintained strict constraint compliance. 19 models generated partially executable validation code, and 18 provided coherent step-by-step reasoning, demonstrating strong interpretability even when minor arithmetic or dependency errors occurred.

A key differentiator among models was *constraint adherence and makespan accuracy*. While most produced feasible mappings, only a few, particularly o3, o3-mini, GPT-4.1 Mini, and Gemini Pro 2.5, consistently respected dependency order and transfer timing, yielding the true optimal makespan (9 h 20 s). Others, including GPT-4.1, Qwen 3 32B, and Mistral Large, were near-optimal but occasionally underestimated transfer delays or overlooked subtle resource conflicts. These deviations highlight a persistent limitation in current LLMs' ability to precisely enforce hard temporal and resource constraints.

Regarding *mapping efficiency and node utilization*, higher-performing models effectively leveraged resource locality and hardware features, co-locating dependent tasks and assigning workloads to appropriate accelerators. In contrast, weaker models often ignored opportunities for parallelism or failed to minimize data transfer overhead, directly increasing the makespan.

A notable common strength across nearly all advanced models was their *reasoning transparency and code generation*. Most provided interpretable, step-by-step rationales and readable verification code. Models such as o3, GPT-4o, and Mistral Large demonstrated clear, auditable reasoning traces even when minor logical inconsistencies occurred. Interestingly, no consistent correlation was observed between model size or response latency and scheduling accuracy, larger, slower models often performed comparably to smaller, faster ones, suggesting that prompt comprehension and reasoning structure, rather than scale, govern LLM performance in this task.

Figure 4 summarizes the number of models (out of 21) that succeeded on each qualitative metric. All models achieved perfect scores in *Task Throughput* and *Node Utilization*, confirming their consistent ability to complete all tasks and allocate resources efficiently. Only ten models demonstrated full *Constraint Adherence*, indicating that nearly half occasionally violated dependency or resource limits. Performance on the *Reasoning* metric was moderate (thirteen models), reflecting variation in the depth and accuracy of step-by-step logic. In contrast, high success rates were observed for *Explanation* (eighteen models) and *Code Generation* (nineteen models), showing strong capability in producing clear rationales and functional validation scripts. *Response Time* success was achieved by fourteen models, suggesting that while most responded within acceptable latency, a notable fraction remains slower or less efficient in inference. Overall, the distribution indicates that although LLMs excel in generating feasible mappings and interpretable reasoning artifacts, their constraint-enforcement accuracy and logical consistency still require improvement.

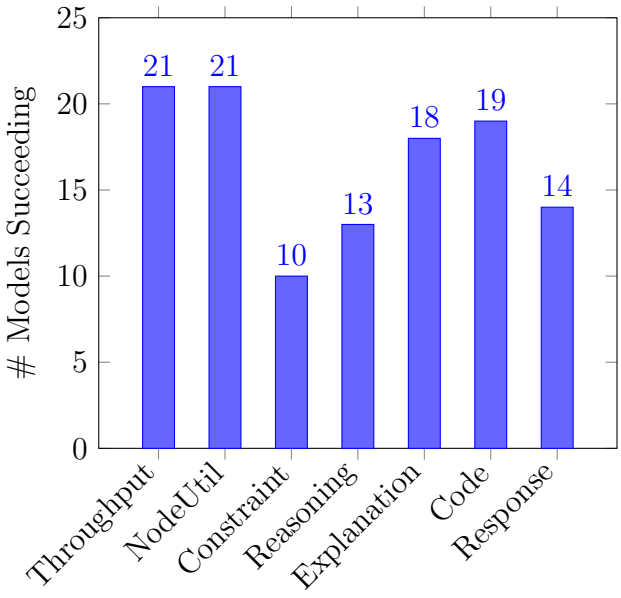

Figure 4: Number of models (out of 21) that succeeded on each qualitative metric.

4.2 Illustrative Reasoning Examples

To qualitatively demonstrate model interpretability and reasoning diversity, this section presents two representative examples of how LLMs reasoned about mapping and scheduling constraints from natural-language input. While most models produced feasible schedules, their internal reasoning strategies varied significantly, some structured the problem symbolically, others expressed it procedurally in text.

4.2.1 Llama 3.1 8B , Symbolic Graph Representation

`Llama 3.1 8B` modeled the HPC system as a directed graph using `networkx`, assigning resource attributes to nodes and defining task dependencies as edges. This demonstrates an emerging capability to abstract the scheduling problem into a formal structure without explicit instructions or templates.

Listing 1: Excerpt from Llama 3.1 8B: Graph-based abstraction of system resources and tasks.

```
G = nx.DiGraph()
G.add_node("NodeA", cpus=32, ram=128, features=["GPU"])
G.add_edge("NodeA","NodeA", task="Task1", features=["GPU"], duration=3)
```

This behavior suggests that some models attempt to reconstruct implicit optimization structures, transforming textual inputs into symbolic, machine-readable representations.

4.2.2 Qwen 3 32B , Natural-Language Reasoning

In contrast, `Qwen 3 32B` provided explicit natural-language reasoning to explain each decision step. It tracked remaining resources after every assignment and justified task placements based on feature requirements and dependencies:

First, assign Task1 to NodeA. That uses 8 CPUs and 32 GB RAM, leaving 24 CPUs and 96 GB free. Task3 requires SSD, so assign it to NodeC, which fully utilizes its 16 CPUs and 64 GB RAM.

This linguistic reasoning trace highlights an interpretable, procedural approach, making the decision process auditable by humans.
Together, these examples reveal two complementary reasoning modalities, symbolic graph construction and narrative explanation, illustrating how current LLMs conceptualize scheduling constraints. Such interpretability reinforces their potential as explainable co-pilots in hybrid optimization systems rather than opaque black-box solvers.

4.3 Discussion

This study provides a clear snapshot of current LLM reasoning capability in HPC workload mapping and scheduling in heterogeneous landscape. All 21 models generated feasible task–node mappings and coherent explanations, but their probabilistic reasoning led to output variability, identical prompts sometimes produced different schedules or makespans. Unlike deterministic optimizers, LLMs approximate constraint reasoning rather than enforcing it, making their results interpretable yet non-guaranteed.

Most models achieved perfect task throughput and balanced node utilization, but only **10 of 21** satisfied all constraints without violation. Frequent errors included incorrect transfer-time arithmetic, ignored dependency ordering, or minor resource over-allocations. Only **3 models** (`o3`, `o3-mini`, `GPT-4.1 Mini`) exactly reproduced the analytical optimum of 9 h 20 s, while **12 others** were near-optimal (within one to two minutes). These deviations show that LLMs reason plausibly about parallelism and critical paths but lack consistent enforcement of temporal and resource constraints.

Scalability remains limited: as prompts grow more complex, some models truncate reasoning or omit constraints, yielding incomplete schedules. Still, their transparency stands out, LLMs articulate decision logic, trade-offs, and assumptions in natural language, an ability absent from mathematical solvers. This interpretability positions them as effective *co-pilots* that can draft candidate mappings, explain heuristic choices, and translate human objectives into structured optimization inputs.

Research Question Analysis

RQ1: Understanding constraints: LLMs generally comprehend resource and feature requirements but partially fail on inter-task dependencies and data-transfer rules. They rely on explicit prompt phrasing to maintain constraint consistency.
RQ2: Optimality and makespan accuracy: A minority matched the analytical optimum, and most remained near-optimal. Variability stems from approximate arithmetic and non-deterministic reasoning, not from lack of understanding.
RQ3: Failure patterns and implications: Typical issues include unit conversion errors, premature task starts, conservative serialization, and context loss in long prompts. These patterns suggest using LLMs as explainable front-ends, complemented by symbolic solvers or rule-based validators, to achieve reliable hybrid scheduling.
Overall, LLMs already approximate optimal solutions with strong interpretability for small, well-defined workloads. Their role is best envisioned not as autonomous optimizers but as reasoning assistants embedded within human-in-the-loop or hybrid optimization pipelines.

5 Conclusion and Future Work

This study systematically evaluated the intrinsic reasoning capabilities of 21 Large Language Models (LLMs) for workload mapping and scheduling in heterogeneous HPC systems using only natural-language descriptions. A manually derived analytical optimum of 9 h 20 s served as the ground-truth reference for correctness and constraint validation. Out of the 21 evaluated models, three (`o3-mini`, `GPT-4.1 Mini`, and `Gemini Pro 2.5`) exactly reproduced the analytical optimum, twelve achieved near-optimal makespans within two minutes, and six produced suboptimal or inconsistent schedules. Overall, approximately three-quarters of the models demonstrated the ability to reconstruct near-optimal reasoning paths for this small-scale HPC scheduling task.

These results confirm that LLMs can comprehend and reason over complex resource, feature, and dependency relationships described purely in natural language, addressing **RQ1**. In relation to **RQ2**, most models derived feasible and interpretable schedules, though only a small subset achieved exact optimality due to probabilistic reasoning and arithmetic inaccuracies. Regarding **RQ3**, observed failure patterns included unit-conversion errors, overlooked transfer delays, and premature dependency resolution, emphasizing the need for explicit constraint reinforcement and validation mechanisms.

Despite these limitations, the findings establish that LLMs can function as *explainable reasoning co-pilots* capable of generating feasible schedules, human-readable justifications, and verification code. Their interpretability and flexibility make them promising components for hybrid scheduling workflows that combine language-based reasoning with deterministic optimization engines.

Future work will expand this benchmark toward larger and more complex workflow graphs, integrate automated constraint-verification pipelines, and explore hybrid LLM architectures. In such systems, LLMs could provide natural-language reasoning and constraint translation, while formal solvers ensure quantitative precision. This hybrid paradigm represents a key step toward scalable, explainable, and reliable AI-assisted workload scheduling in heterogeneous HPC environments.

Acknowledgments

This work was supported by the University of Göttingen. The authors gratefully acknowledge the Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG) for providing computational resources and technical support. We also thank our colleagues and peers in the HPC and AI research community for their constructive feedback and valuable discussions. This research was funded by the BMBF KISSKI Project under grant number 01 IS 22 093 A-E.

References

[1] E. Deelman, J. Dongarra, B. Hendrickson, A. Randles, D. Reed, E. Seidel, and K. Yelick, "High-performance computing at a crossroads," *Science*, vol. 387, no. 6736, pp. 829–831, 2025. doi: 10.1126/science.adu0801.

[2] A. R. Brodtkorb, E. C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli, "State-of-the-art in heterogeneous computing," *Scientific Programming*, vol. 18, no. 1, pp. 1–33, 2010. doi: 10.3233/SPR-2009-0296.

[3] A. K. Sharma and J. Kunkel, "A Review of Tools and Techniques for Optimization of Workload Mapping and Scheduling in Heterogeneous HPC System," *arXiv preprint arXiv:2505.11244*, 2025. doi: 10.48550/arXiv.2505.11244.

[4] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002. doi: 10.1109/71.993206.

[5] M. Adhikari, T. Amgoth, and S. N. Srirama, "A survey on scheduling strategies for workflows in cloud environment and emerging trends," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–36, 2019. doi: 10.1145/3325097.

[6] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys (CSUR)*, vol. 31, no. 4, pp. 406–471, 1999. doi: 10.1145/344588.344618.

[7] A. K. Sharma, C. Boehme, P. Gelß, R. Yahyapour, and J. Kunkel, "Workflow-Driven Modeling for the Compute Continuum: An Optimization Approach to Automated System and Workload Scheduling," in *Proc. 2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 2170–2177, 2025, doi: 10.1109/COMPSAC65507.2025.00343.

[8] A. K. Sharma and J. Kunkel, "Grapheon RL: A Graph Neural Network and Reinforcement Learning Framework for Constraint and Data-Aware Workflow Mapping and Scheduling in Heterogeneous HPC Systems," in *Proc. 2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 489–494, 2025, doi: 10.1109/COMPSAC65507.2025.00341.

[9] T. Tobita and H. Kasahara, "A standard task graph set for fair evaluation of multiprocessor scheduling algorithms," *Journal of Scheduling*, vol. 5, no. 5, pp. 379–394, 2002. doi: 10.1002/jos.116.

[10] H. Hussain, S. U. Khan, S. A. Madani, J. Min-Allah, I. Ahmad, C.-Z. Xu, S. S. Wang, B. B. Khan, N. Klodawski, K. Hayat, and others, "A survey on resource allocation in high performance distributed computing systems," *Parallel Computing*, vol. 39, no. 11, pp. 709–736, 2013. doi: 10.1016/j.parco.2013.09.009.

[11] Z. Ye, Y. Lv, Y. Zhang, J. Liu, Z. Zhou, K. Li, and C. Wang, "Deep learning workload scheduling in GPU datacenters: A survey," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–38, 2024. doi: 10.1145/3638757.

[12] N. Grinsztajn, R. Ben Mansour, I. Golan, N. Gavish, and S. Zohar, "Readys: A reinforcement learning based strategy for heterogeneous dynamic scheduling," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 70–81, 2021. doi: 10.1109/Cluster48925.2021.00031.

[13] B. Cui, J. Tang, X. Chen, and Z. Yang, "Do large language models understand performance optimization?" *arXiv preprint* arXiv:2503.13772, 2025. doi: 10.48550/arXiv.2503.13772

[14] D. Nichols, A. Marathe, H. Menon, T. Gamblin, and A. Bhatele, "HPC-Coder: Modeling parallel programs using large language models," in *ISC High Performance 2024 Research Paper Proceedings*, 2024, pp. 1–12. doi: 10.23919/ISC.2024.10528929.

[15] GWDG Chat AI Services, "Available Models," documentation page (accessed July 3, 2025). URL: https://docs.hpc.gwdg.de/services/chat-ai/models/index.html.

[16] P. Jadhav, H. Jin, E. Deelman, and P. Balaprakash, "Evaluating the Efficacy of LLM-Based Reasoning for Multiobjective HPC Job Scheduling," *arXiv preprint* arXiv:2506.02025, 2025. URL: https://arxiv.org/abs/2506.02025.