Starjob: Dataset for LLM-Driven Job Shop Scheduling

Henrik Abgaryan[1], Tristan Cazenave[1], and Ararat Harutyunyan[1]

LAMSADE, Université Paris Dauphine - PSL, CNRS, Paris, France
{henrik.abgaryan, tristan.cazenave, ararat.harutyunyan}@dauphine.psl.eu

Abstract. Large Language Models (LLMs) have shown remarkable capabilities across various domains, but their potential for solving combinatorial optimization problems remains largely unexplored. In this paper, we investigate the applicability of LLMs to the Job Shop Scheduling Problem (JSSP), a classic NP-hard challenge. We introduce Starjob, the first large-scale supervised dataset for JSSP, comprising 130,000 instances with natural language representations designed specifically for training LLMs. Leveraging this dataset, we fine-tune a Llama-3.1 8B model using the resource-efficient RsLoRA method to create an end-to-end scheduler. Our evaluation on standard benchmarks demonstrates that this LLM-based method surpasses traditional Priority Dispatching Rules (PDRs) and achieves significant performance gains over foundational neural approaches like L2D and RASCL, with an average gap improvement of 17.36% on DMU and 7.85% on Taillard benchmarks relative to L2D. These results highlight the untapped potential of fine-tuned LLMs in combinatorial optimization, establishing a new direction for developing interactive and high-performance scheduling systems.

Keywords: Job shop scheduling · Combinatorial optimization · Large language models · Supervised dataset · Fine-tuning · Llama

1 Introduction

Despite their success in natural language processing, Large Language Models (LLMs) have not been traditionally considered strong candidates for solving computationally intensive problems. Their applicability to NP-hard combinatorial optimization problems is often viewed as limited, a perception reinforced by the scarcity of empirical evidence showing LLMs outperforming specialized methods like reinforcement learning in these domains. Furthermore, the propensity of LLMs to "hallucinate" can lead to infeasible solutions, making their direct application unreliable. Consequently, the systematic exploration of fine-tuned LLMs for hard combinatorial problems has remained limited.

In this paper, we challenge this prevailing view by demonstrating that representation is the key to unlocking the scheduling capabilities of LLMs. We present the first fine-tuned LLM for the Job Shop Scheduling Problem (JSSP). Our results show that when trained on a properly structured, text-based representation of the problem, an LLM can not only generate feasible schedules

but also outperform classic Priority Dispatching Rules (PDRs) and foundational dedicated neural methods that first surpassed them (e.g., L2D [27] and RASCL[18]). These findings suggest that with appropriate data representation and fine-tuning, LLMs can become a competitive new paradigm for combinatorial optimization, complementing existing specialized solvers.

JSSP is a fundamental optimization problem with critical applications in manufacturing and logistics, where jobs must be scheduled on machines to minimize metrics like makespan (C_{max}). While traditional methods face scalability challenges, modern AI techniques, particularly reinforcement learning and Graph Neural Networks (GNNs), have offered promising data-driven alternatives [27,9]. Concurrently, LLMs have been explored for tasks involving structured reasoning, such as graph analysis [17,7] and planning [23]. However, their application to the explicit, constraint-heavy domain of scheduling remains largely unexplored.

This work bridges that gap. We are the first to employ a fine-tuned LLM for end-to-end JSSP scheduling. To enable this, we introduce **Starjob**[1], a novel supervised dataset where JSSP instances and their solutions are framed in natural language. By fine-tuning a Llama model with the RsLoRA method [19] on this dataset, we demonstrate on the well-known Taillard [22] and DMU [11] benchmarks that our approach finds high-quality solutions, surpassing both classic PDRs and the L2D neural baseline.

Our contributions are:

- We introduce **Starjob**, the first supervised dataset with 130,000 instances designed to train LLMs for JSSP using a structured natural language format.
- We present the first end-to-end JSSP scheduler based on a fine-tuned LLM, demonstrating its ability to reason over complex constraints using the Starjob dataset and the RsLoRA method.
- We conduct a rigorous evaluation against four PDRs and neural methods L2D and RASCL, showing our model's superior performance and generalization, particularly on large-scale instances with up to 1000 operations.
- Our LLM-based approach unlocks a new modality of interaction: users can query the scheduler in natural language to understand scheduling constraints or solution characteristics, significantly enhancing transparency and usability, as presented in Listing 1.3.

It is important to note that while our approach demonstrates competitive performance against several established baselines, we do not claim to have developed the absolute best scheduler for JSSP. Rather, this work represents the first systematic application of LLMs to end-to-end large JSSP instances, establishing a foundation for future research at the intersection of natural language processing and combinatorial optimization.

2 Related Work

JSSP with more than two machines is proven to be NP-hard [13]. As a result, finding exact solutions for JSSP is generally infeasible, leading to the widespread

[1] `https://github.com/starjob42/Starjob`

use of heuristic and approximate methods for practical efficiency [5]. Traditional approaches to solving JSSP have primarily relied on search and inference techniques developed by the constraint programming community [2]. These techniques effectively leverage constraints to define the relationships and limitations between jobs and resources, enabling efficient exploration of feasible solution spaces and the identification of optimal or near-optimal schedules [20]. A widely used heuristic method in real-world scheduling systems is the Priority Dispatching Rule (PDR) [26]. PDRs are simple and effective, although designing an efficient PDR is time-consuming and requires extensive domain knowledge.

Recently, approaches utilizing Deep Learning and Neural Networks have gained attention for finding promising solutions to the JSSP [3,27,9]. These methods can be broadly categorized into supervised learning and reinforcement learning (RL). Current research in deep reinforcement learning (DRL) is actively focused on developing advanced methods to tackle JSSP. Existing DRL methods typically represent JSSP as a Markov Decision Process (MDP) and learn a policy network based on DRL techniques[27].

Large language models (LLMs) are now being applied to a wider range of tasks beyond language processing, in areas like robotics and planning [17]. While there are currently no papers that directly address the scheduling of Job Shop Scheduling Problems (JSSP) using LLMs, some notable works explore the potential of LLMs in mathematical reasoning and programming [6,24,1,25]. Optimization using LLMs has gained significant interest in recent years, with several works exploring their capabilities across various domains [25]. The ability of LLMs to understand and generate natural language has opened new possibilities for optimization tasks that were traditionally solved using derivative-based algorithms or heuristic methods [25]. [6] evaluated LLMs' performance in mathematical problem-solving and introduced "Program of Thoughts" (PoT) prompting. Unlike Chain of Thoughts (CoT) [24], which combines reasoning and computation, PoT generates reasoning as code statements and delegates computation to an interpreter. [1] surveys mathematical problems and datasets studied with LLMs, analyzing their strengths and weaknesses. [12] examines LLMs' impact on mathematicians, exploring their role in research, education, problem-solving, and proof generation, offering a balanced view of their capabilities. Recent works [25] explore LLMs as optimizers, using prompts to refine solutions iteratively. Case studies on linear regression and the traveling salesman problem show LLMs can produce high-quality solutions, sometimes matching heuristic algorithms in small-scale scenarios. Explorations into using LLMs for graph learning tasks have yielded notable approaches. [17] noted that LLMs exhibit some initial graph reasoning capabilities, but their performance decreases with problem complexity, [17] introduced prompting strategies to improve LLMs graph reasoning. [23] developed a benchmark for assessing the planning and reasoning abilities of LLMs. More recently, [7] examined the use of LLMs for graph node classification tasks. [8] presents LLMs as enhancers for GNNs and as direct predictors from graph structures. [28] proposed GRAPHTEXT, which translates graphs to

natural language for training-free reasoning, often rivaling GNNs. While LLMs show promise in graph tasks, their use in scheduling is still largely unexplored.

3 Preliminaries

We consider the classical JSSP, which is defined as follows. Given N_J jobs and N_M machines, each job J_i is comprised of an ordered sequence of operations $(O_{i1}, \ldots, O_{in_i})$. Each operation O_{ij} must be processed on a designated machine m_{ij} for a processing time p_{ij}. The scheduling variables S_{ij} denote the start time of operation O_{ij}.

The JSSP is governed by two principal constraints: (i) *Precedence constraints* require that each operation starts only after the completion of its predecessor within the same job, i.e., $S_{i,j+1} \geq S_{ij} + p_{ij}$; and (ii) *Resource constraints* ensure that no two operations assigned to the same machine overlap in time, i.e., for any pair O_{ij}, O_{kl} such that $m_{ij} = m_{kl}$,

$$[S_{ij}, S_{ij} + p_{ij}) \cap [S_{kl}, S_{kl} + p_{kl}) = \emptyset.$$

The objective is to minimize the makespan, defined as the maximum completion time across all operations:

$$C_{\max} = \max_{i,j} \{S_{ij} + p_{ij}\}.$$

4 Dataset Construction and Representation

The use of LLMs for combinatorial optimization requires translating traditional mathematical representations into natural language encodings that maintain the problem structure for language-based processing. We present a methodology for representing JSSP instances as structured natural language, mapping the conventional matrix-based form (see Listing 1.1) into explicit descriptions of all constraints and requirements. This mapping is defined as a deterministic, bijective transformation $\mathcal{T} : P \to \mathcal{L}$, where P denotes the space of standard JSSP instances and $\mathcal{L}$ the corresponding space of natural language descriptions.

```
Optimize the schedule for 3 jobs (J0, J1, J2) across 3
    machines (M0, M1, M2) to minimize the makespan. Each
    machine can process only one job at a time and jobs are
     non-preemptive.

J0: M0:105, M1:29, M2:213
J1: M0:193, M1:18, M2:213
J2: M0:78,  M1:74, M2:221
```

Listing 1.1: Example: Natural language encoding of a JSSP instance with $N_J = 3$ and $N_M = 3$.

For a JSSP instance with N_J jobs and N_M machines, we construct a natural language encoding that systematically specifies:

1. The problem dimensions ($N_J \times N_M$)
2. The operational constraints (non-preemption, machine exclusivity)
3. The sequential processing requirements for each job
4. The corresponding processing durations

This encoding establishes a bijective mapping between mathematical and linguistic representations, preserving all information required for solution generation while rendering the problem interpretable to language models. As illustrated in Listing 1.1, the natural language encoding presents the problem parameters in a clear, structured format that delineates job requirements across machines.

4.1 Corpus Generation for Model Training

To facilitate effective learning of the mapping between problem instances and their solutions, we constructed a comprehensive corpus of JSSP instances and their corresponding optimal or near-optimal solutions. The corpus encompasses approximately 130,000 random JSSP instances spanning dimensions from 2×2 to 20×20, supplemented by $\sim$1,000 larger and asymmetric instances to enhance generalization capabilities across problem scales. Operation durations were sampled from a uniform distribution ranging from 5 to 500 time units, ensuring comprehensive coverage of the solution space and robustness to varying temporal constraints. The testing dataset is out of distribution dataset from the trainng dataset. We conduct evaluations on the TAI [22] and DMU [11] benchmark sets, which are entirely held out from the training phase.

For solution generation, each instance was processed using Google's OR-Tools optimization framework with parameters configured to balance computational efficiency and solution quality. The solver was allocated a 300-second time limit with 42 parallel workers utilizing the `AUTOMATIC_SEARCH` strategy,

providing near-optimal solutions even for larger problem instances. For problems exceeding 10×10 dimensions, we acknowledge potential suboptimality due to computational constraints while maintaining solution feasibility.

The solution encoding adopts a structured natural language format, specifically designed to guide the autoregressive nature of the LLMs. Each entry in the solution sequence (see Listing 1.2) details a job-machine assignment along with its explicit start time, duration, and resulting completion time. Notably, the use of summation notation (e.g., "J2-M0: $0+78 \rightarrow 78$") forces the model to compute the current makespan incrementally, based on the start time and duration, while taking into account the completion times of all previously scheduled operations.

This stepwise representation leverages the LLM's autoregressive generation process, requiring it to "think" about the current scheduling decision by explicitly calculating and verifying the timing constraints before proceeding to the next operation. The format ensures that each scheduling step is conditioned on the already constructed partial schedule, thus embedding temporal dependencies and constraint satisfaction directly into the generation process.

```
Solution:
J2-M0: 0+78 -> 78,
J1-M2: 0+193 -> 193,
J0-M0: 78+105 -> 183,
J0-M1: 183+29 -> 212,
J2-M2: 193+74 -> 267,
J1-M1: 212+18 -> 230,
J1-M0: 230+213 -> 443,
J2-M1: 267+221 -> 488,
J0-M2: 267+213 -> 480
```

Listing 1.2: Consistent schedule with correct job precedence and operation durations for a JSSP instance with $N_J = 3$ jobs and $N_M = 3$ machines. The values after "->" denote operation completion times. The makespan is the maximum of these, i.e., 488.

Empirical results (see Table 3) show that this explicit, computation-driven format significantly enhances the feasibility of generated solutions compared to formats omitting intermediate calculations. By prompting the model to perform and record intermediate makespan computations, the approach enables real-time constraint checking and more effective optimization, reducing the frequency of infeasible schedules.

5 Methodology

We propose a novel method for solving JSSP by fine-tuning large language models with natural language representations of scheduling problems and solutions. Our framework, based on Meta-Llama-3.1-8B-Instruct (4-bit quantized), reframes JSSP as a sequence generation task and operates in two phases: (1) fine-tuning the model on problem-solution pairs using rsLoRA, and (2) generating and selecting optimal schedules for new instances. By expressing both problems and solutions in natural language, our approach leverages pre-trained knowledge and learns scheduling-specific patterns efficiently.

5.1 Training Methodology

We fine-tune the model using rsLoRA [16], an approach that replaces the standard scaling factor $\frac{\alpha}{r}$ with a stabilized $\sqrt{\frac{\alpha}{r}}$, which enables the use of higher ranks without causing gradient collapse and ensures more stable training dynamics. The model is initialized with pre-trained weights θ_0, which remain frozen throughout the process, while only the low-rank adaptation matrices U and V are updated to minimize the negative log-likelihood loss on tokenized problem-solution pairs. Training is conducted over 2 epoch with a learning rate of 2×10^{-4}, LoRA rank $r = 64$, scaling factor $\alpha = 64$, and a batch size of 16, utilizing a single Nvidia RTX A6000 GPU (48GB memory), with the training process taking approximately 70 hours and utilizing around 30GB of GPU RAM, highlighting the resource requirements for this procedure.

Algorithm 1: LLM Fine-Tuning for JSSP with rsLoRA

Input: Problem instance $\mathcal{L}_p$ in natural language, Fine-tuned LLM with parameters $\theta = \theta_0 + \gamma_r UV^\top$, Number of candidate solutions S

Initialize low-rank matrices $U, V \in \mathbb{R}^{d\times r}$
Define rank-stabilized factor $\gamma_r = \frac{\alpha}{\sqrt{r}}$
for *epoch* $= 1$ **to** E **do**
  for *each batch* $\{(\mathcal{L}_p^{(i)}, s^{(i)})\}_{i=1}^{B} \subset \mathcal{D}$ **do**
    Tokenize each problem $\mathcal{L}_p^{(i)}$ and solution $s^{(i)}$
    Construct inputs with problems $\mathcal{L}_p^{(i)}$ as context
    Set targets as tokenized solutions $s^{(i)} = \{w_1^{(i)}, \ldots, w_{T_i}^{(i)}\}$
    Compute effective parameters: $\theta = \theta_0 + \gamma_r UV^\top$
    Forward pass: Compute probabilities $p(w_t | w_{<t}, \mathcal{L}_p^{(i)}; \theta)$
    Compute NLL loss: $\mathcal{L} = -\sum_{i=1}^{B}\sum_{t=1}^{T_i} \log p(w_t^{(i)} | w_{<t}^{(i)}, \mathcal{L}_p^{(i)}; \theta)$
    Compute gradients $\nabla_U \mathcal{L}$ and $\nabla_V \mathcal{L}$
    Update $U \leftarrow U - \eta \nabla_U \mathcal{L}$
    Update $V \leftarrow V - \eta \nabla_V \mathcal{L}$
  Evaluate model performance on validation set

Result: Fine-tuned model parameters $\theta = \theta_0 + \gamma_r UV^\top$

5.2 Inference and Solution Selection

At inference time (Algorithm 2), we employ a generate-and-select strategy. For each problem instance, the model produces multiple candidate solutions through temperature-controlled sampling. Each candidate undergoes rigorous feasibility checking to ensure satisfaction of all JSSP constraints, including job precedence, machine exclusivity, and non-preemption requirements. From the set of feasible solutions, we select the one with the minimum makespan.

The feasibility check validates that: (1) each job's operations are scheduled in the correct sequence, (2) no machine processes multiple jobs simultaneously, (3) each operation has the correct processing time, and (4) all operations are scheduled exactly once. This comprehensive validation ensures that all solutions adhere to the fundamental constraints of the JSSP problem domain.

Algorithm 2: LLM-Based JSSP Solution Generation and Selection

Input: Problem instance $\mathcal{L}_p$ in natural language, Fine-tuned LLM with parameters $\theta = \theta_0 + \gamma_r UV^\top$,Number of candidate solutions S, temperature τ

Initialize empty set of feasible solutions $\mathcal{S}_p^f \leftarrow \emptyset$
for $i = 1$ **to** S **do**
  Generate candidate solution $s_i \sim \text{LLM}_\theta(\mathcal{L}_p, \tau)$
  Parse solution s_i to extract job-machine assignments and timings
  $\text{valid}_{\text{precedence}} \leftarrow$ Check job operation precedence constraints
  $\text{valid}_{\text{exclusivity}} \leftarrow$ Check machine exclusivity constraints
  $\text{valid}_{\text{timing}} \leftarrow$ Check correct processing times
  $\text{valid}_{\text{completeness}} \leftarrow$ Check all operations are scheduled once
  if $valid_{precedence} \wedge valid_{exclusivity} \wedge valid_{timing} \wedge valid_{completeness}$ **then**
    Compute makespan $M(s_i)$
    Add to feasible set: $\mathcal{S}_p^f \leftarrow \mathcal{S}_p^f \cup \{s_i\}$
if $\mathcal{S}_p^f \neq \emptyset$ **then**
  return $s^* = \arg\min_{s \in \mathcal{S}_p^f} M(s)$ `// Best solution by makespan`
else
  return "No feasible solution found"

6 Validation, Baseline Methods, and Empirical Analysis

We evaluated our end-to-end LLM-based job shop scheduler using the standard Taillard [22] and DMU [11] benchmarks, comparing it to both traditional heuristics and state-of-the-art learning-based methods. As the first application of LLMs for end-to-end JSSP solution generation, we benchmarked our model against L2D [27]—an early neural approach that outperforms classic priority dispatching rules (PDRs) such as SPT, MWKR, MOPNR, and FDD/MWKR. L2D

leverages a graph neural network and PPO [21] for generalization. Additionally, we compared our method with RASCLB [18], a state-of-the-art reinforcement learning approach designed for cross-instance generalization. Here, "B" denotes the "base" learning method in [18], which combines an RL-based method with rLSTM and set2set modules. RASCLB is trained on larger instances (30x20) with a sample size of 20. Its reverse LSTM [15] component receives static, multidimensional embeddings for all operations in a job J_i, propagating information backward from the last operation to the current one. For all experiments, inference was performed with a context length of 40,000 tokens (the maximum number of tokens the model can process in a single input sequence) using default sampling settings and $S = 20$ samples per instance, with default temperature parameter of 1. Both training and inference used `4-bit` quantization for memory efficiency, requiring about 30GB of GPU memory on an NVIDIA A6000. Our largest evaluated instance (23,000 tokens) fits comfortably within this window. For faster inference, we converted the model to the `llama.cpp` format [14], achieving 102.22 tokens/sec on an RTX A6000 (48GB), as reported by Dai et al. [10]. Notably, inference time scales with token sequence length rather than problem complexity; processing our largest instance (22,224 tokens) within the 40,000-token window took about 217 seconds per sample, regardless of task type.

6.1 Performance Metrics and Comparative Results

Performance on each benchmark was evaluated using the *Percentage Gap* (PG), defined as:

$$\mathrm{PG} \;=\; 100 \times \left(\frac{M_{\mathrm{alg}}}{M_{\mathrm{ub}}} \;-\; 1 \right),$$

where M_{alg} is the makespan produced by the algorithm, and M_{ub} is the best-known or optimal makespan. Lower PG values correspond to solutions closer to the best-known objective. Tables 1 and 2 present the gap percentages of the fine-tuned Llama model compared to L2D and other methods (SPT, MWKR, FDD/WKR, MOPNR) on Tai and DMU datasets. Benchmark solutions are available at[2] and[3]. Across instances ranging from 15 jobs and 15 machines to 50 jobs and 20 machines, the fine-tuned Llama 3.1 consistently outperforms all other methods. On larger instances (50 jobs, 20 machines, 1000 nodes), Llama achieves a 24.32% gap, surpassing L2D (26.40%). The average gap for fine-tuned Llama (21.69%) is significantly lower than SPT (60.57%), MWKR (55.29%), FDD/WKR (46.77%), MOPNR (42.99%), and L2D (29.54%). On the DMU benchmark with 50 jobs and 15 machines, fine-tuned Llama (22.14%) again outperforms all methods, including L2D (37.50%). SPT consistently has the highest gap (often above 60%), reflecting its inability to handle complex constraints. MWKR and FDD/WKR, while more sophisticated, remain outperformed by learning-based approaches due to their myopic decision processes. Notably, our method shows average improvements over the dedicated model RASCLB of 13.41% on

[2] `http://optimizizer.com/TA.php`
[3] `http://jobshop.jjvh.nl/`

the Taillard dataset (see Table 1) and comparableresult on the DMU dataset (see Table 2).

6.2 Ablation Study

To understand the impact of key components, we conducted an ablation study focusing on our **Starjob** data representation and the model size. As shown in Table 3, the our proposed summation format significantly increases the percentage of feasible solutions produced by the Llama 8B model compared to the standard matrix format, not involving summation operations, which is familiar to the LLM from pretraining [4]. Despite this familiarity, Starjob yields markedly better feasibility. The Llama 1B model, in contrast, failed on this task, highlighting the necessity of larger models for high-quality scheduling.

Example: Interactive Query with the Starjob-LLM

User:
Looking at the 3x3 problem instance, what was the main bottleneck that made it hard to solve?

Starjob-LLM:
The primary bottleneck was **Machine 2 (M2).** All three jobs required this machine for long processing times, creating a highly contended resource.
The sequential processing on M2 (J1-M2 → J2-M2 → J0-M2) formed the critical path, which ultimately determined the final makespan of 488. Any improvement to the schedule would need to resolve this contention on Machine 2.

Listing 1.3: An interactive session where the LLM explains the primary scheduling bottleneck.

7 Conclusion

This work served as a test to see whether LLMs could generate any feasible solutions for NP-hard combinatorial optimization, rather than to claim the best scheduler. We showed that, with the right data representation, LLMs can indeed act as effective schedulers. To this end, we introduced **Starjob**, a large supervised dataset for the Job Shop Scheduling Problem (JSSP) in structured natural language, and fine-tuned a Llama 8B model using resource-efficient methods. While our goal was not to surpass specialized schedulers, our LLM-based approach still outperformed traditional Priority Dispatching Rules and even some dedicated neural baselines on standard benchmarks. These findings highlight the potential of LLMs for combinatorial optimization, and suggest promising directions for more interpretable and interactive scheduling systems in the future.

8 Limitations and Future Work

While our model's performance is dependent on the Starjob dataset and constrained by the LLM's context window, it provides high-quality heuristic solutions. Future work will focus on developing hybrid solvers that combine our approach with traditional optimization methods for further refinement. We also plan to explicitly integrate Graph Neural Networks (GNNs) into the LLM latent space via cross-attention to better capture relational structure, and to investigate the impact of full fine-tuning and larger LLMs for additional performance gains.

Table 1: Comparison of different methods on the **TAI** dataset (sampling budget = 20). Lower values indicate schedules closer to the optimal solution, representing better performance. * indicates the best result according to the Percentage Gap.

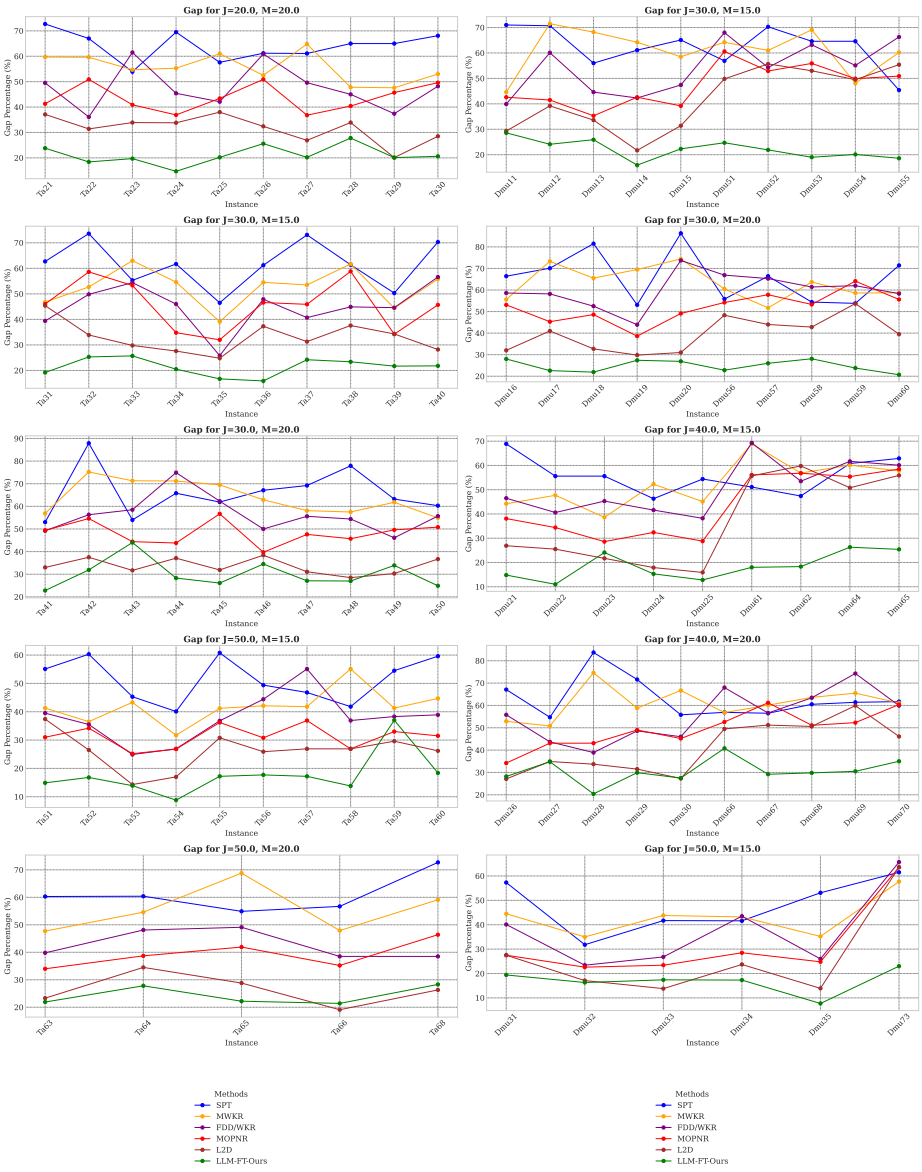
Method	15x15	20x15	20x20	30x15	30x20	50x15	50x20	Average
FDD/WKR	47.45	50.57	47.57	45.01	56.30	37.72	42.80	46.77
MOPNR	44.98	47.97	43.68	45.59	48.23	31.25	39.24	42.99
MWKR	56.74	60.65	55.60	52.61	63.93	41.90	55.62	55.29
SPT	54.64	65.24	64.11	61.61	66.03	51.37	61.00	60.57
L2D	25.95 ± 3.37	30.03 ± 3.90	31.60 ± 4.11	33.02 ± 4.29	33.62 ± 4.37	26.15 ± 3.40	26.40 ± 3.43	29.54 ± 3.84
RASCLB	20.59 ± 2.47	25.31 ± 3.04	25.47 ± 3.06	27.27 ± 3.27	30.40 ± 3.65	20.69 ± 2.48	26.40 ± 3.17	25.16 ± 3.02
LLM-FT-Ours	**19.34 ± 1.93***	**18.00 ± 1.80***	**21.11 ± 2.11***	**21.44 ± 2.14***	**30.05 ± 3.00***	**17.57 ± 1.76***	**24.32 ± 2.43***	**21.69 ± 2.17***

Table 2: Comparison of different methods on the **DMU** dataset (sampling budget = 60). Lower values indicate schedules closer to the optimal solution, representing better performance. * indicates the best result according to the Percentage Gap.

Method	20x15	20x20	30x15	30x20	40x15	40x20	50x15	Average
FDD/WKR	53.58	52.51	54.12	60.08	50.76	55.52	37.58	52.02
MOPNR	49.17	45.18	47.14	51.97	43.23	49.22	31.73	45.38
MWKR	62.14	58.16	60.96	63.15	52.40	61.09	43.23	57.30
SPT	64.12	64.55	62.57	65.92	55.89	62.99	47.83	60.55
L2D	38.95 ± 5.06	37.74 ± 4.91	41.86 ± 5.44	39.48 ± 5.13	36.68 ± 4.77	41.18 ± 5.35	26.60 ± 3.46	37.50 ± 4.88
RASCLB	19.66 ± 2.36	**15.98 ± 1.92**	**16.35 ± 1.96**	23.00 ± 2.76	17.89 ± 2.15	26.42 ± 3.17	21.84 ± 2.62	20.16 ± 2.42
LLM-FT-Ours	**19.20 ± 1.92***	20.16 ± 2.02	22.11 ± 2.21	**21.82 ± 2.18***	**17.24 ± 1.72***	**23.61 ± 2.36***	**16.85 ± 1.69***	**20.14 ± 2.01***

Table 3: Ablation study comparing our **Starjob** representation against the standard **Matrix** format, and the Llama 8B model against Llama 1B model. Average Feasibility (%) indicates the percentage of valid solutions generated, where **higher values are better**. Our proposed format dramatically increases feasibility. The complete failure of the Llama 1B model highlights the task's complexity, while the Llama 8B model using Starjob consistently produces high-quality schedules. Feasibility and time for Llama 1B model are marked as N/A where not available.

Problem Size	**Without** Summation (%)		**With** Summation (%)		**Avg. Time (s) (8B)**
	1B	8B	1B	8B	
5×5	N/A	∼8.0	N/A	∼9.5	6.1
8×8	N/A	∼8.0	N/A	∼10.5	7.2
10×10	N/A	∼1.0	N/A	∼12.1	2.6
12×12	N/A	∼4.0	N/A	∼39.6	14.3
15×15	N/A	∼1.0	N/A	∼14.4	22.5
20×20	N/A	∼1.5	N/A	∼17.4	15.1
30×30	N/A	N/A	N/A	∼30.5	18.3
50×20	N/A	∼1.0	N/A	∼14.6	22.0



(a) TAI benchmark (b) DMU benchmark

Fig. 1: Comparison of different methods on TAI [22] and DMU [11].

References

1. Ahn, J., Verma, R., Lou, R., Liu, D., Zhang, R., Yin, W.: Large language models for mathematical reasoning: Progresses and challenges. In: Falk, N., Papi, S., Zhang, M. (eds.) Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop. pp. 225–237. Association for Computational Linguistics, St. Julian's, Malta (Mar 2024), https://aclanthology.org/2024.eacl-srw.17
2. Beck, J.C., Feng, T.K., Watson, J.P.: Combining constraint programming and local search for job-shop scheduling. INFORMS Journal on Computing **23**(1), 1–14 (2010)
3. Bonetta, G., Zago, D., Cancelliere, R., Grosso, A.: Job shop scheduling via deep reinforcement learning: a sequence to sequence approach. Not Specified (Aug 2023)
4. Bordt, S., Nori, H., Rodrigues, V., Nushi, B., Caruana, R.: Elephants never forget: Memorization and learning of tabular data in large language models (2024), https://arxiv.org/abs/2404.06209
5. Cebi, C., Atac, E., Sahingoz, O.K.: Job shop scheduling problem and solution algorithms: A review. In: 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT). pp. 1–7 (2020). https://doi.org/10.1109/ICCCNT49239.2020.9225581
6. Chen, W., Ma, X., Wang, X., Cohen, W.W.: Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. Transactions on Machine Learning Research (2023), https://openreview.net/forum?id=YfZ4ZPt8zd
7. Chen, Z., Mao, H., Li, H., Jin, W., Wen, H., Wei, X., Wang, S., Yin, D., Fan, W., Liu, H., Tang, J.: Exploring the potential of large language models (llms) in learning on graphs (2024), https://arxiv.org/abs/2307.03393
8. Chen, Z., Mao, H., Li, H., Jin, W., Wen, H., Wei, X., Wang, S., Yin, D., Fan, W., Liu, H., Tang, J.: Exploring the potential of large language models (llms) in learning on graphs (2024)
9. Corsini, A., Porrello, A., Calderara, S., Dell'Amico, M.: Self-labeling the job shop scheduling problem. In: Self-Labeling the Job Shop Scheduling Problem. Arxiv (2024)
10. Dai, X.: Gpu-benchmarks-on-llm-inference. https://github.com/XiongjieDai/GPU-Benchmarks-on-LLM-Inference (2024), accessed: 2024-11-26
11. Demirkol, E., Mehta, S., Uzsoy, R.: Benchmarks for shop scheduling problems. European Journal of Operational Research **109**(1), 137–141 (1998)
12. Frieder, S., Berner, J., Petersen, P., Lukasiewicz, T.: Large language models for mathematicians (2024)
13. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research **1**(2), 117–129 (1976)
14. Gerganov, G.: llama.cpp: Llm inference in c/c++. https://github.com/ggerganov/llama.cpp (2023), accessed: 2024-11-26
15. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**(8), 1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735
16. Hu, E.J., yelong shen, Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W.: LoRA: Low-rank adaptation of large language models. In: International Conference on Learning Representations (2022), https://openreview.net/forum?id=nZeVKeeFYf9

17. Huang, W., Abbeel, P., Pathak, D., Mordatch, I.: Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In: Proceedings of the International Conference on Machine Learning. PMLR (2022), *equal advising
18. Iklassov, Z., Medvedev, D., Solozabal Ochoa de Retana, R., Takáč, M.: On the study of curriculum learning for inferring dispatching policies on the job shop scheduling. In: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI). pp. 5350–5358 (2023). https://doi.org/10.24963/ijcai.2023/594
19. Kalajdzievski, D.: A rank stabilization scaling factor for fine-tuning with lora (2023), https://arxiv.org/abs/2312.03732
20. Nowicki, E., Smutnicki, C.: An advanced tabu search algorithm for the job shop problem. Journal of Scheduling **8**(2), 145–159 (2005). https://doi.org/10.1007/s10951-005-6364-5
21. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
22. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research **64**(2), 278–285 (1993)
23. Valmeekam, K., Olmo, A., Sreedharan, S., Kambhampati, S.: Large language models still can't plan: A benchmark for llms on planning and reasoning about change. In: NeurIPS 2022 Foundation Models for Decision Making Workshop (2022), https://openreview.net/forum?id=wUU-7XTL5XO
24. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E.H., Le, Q.V., Zhou, D.: Chain-of-thought prompting elicits reasoning in large language models. Google Research, Brain Team (2022)
25. Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q.V., Zhou, D., Chen, X.: Large language models as optimizers. arXiv preprint arXiv:2309.03409 (2023)
26. Zahmani, M.H., Atmani, B., Bekrar, A., Aissani, N.: Multiple priority dispatching rules for the job shop scheduling problem. In: 3rd International Conference on Control, Engineering Information Technology (CEIT'2015). Tlemcen, Algeria (2015). https://doi.org/10.1109/CEIT.2015.7232991
27. Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P.S., Xu, C.: Learning to dispatch for job shop scheduling via deep reinforcement learning. In: 34th Conference on Neural Information Processing Systems (NeurIPS) (2020)
28. Zhao, J., Zhuo, L., Shen, Y., Qu, M., Liu, K., Bronstein, M.M., Zhu, Z., Tang, J.: Graphtext: Graph learning in text space (2024), https://openreview.net/forum?id=dbcWzalk6G