Starjob: Dataset for LLM-Driven Job Shop Scheduling

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

[1]LAMSADE, Université Paris Dauphine - PSL, CNRS, Paris, France
henrik.abgaryan@dauphine.eu, ararat.harutyunyan@dauphine.eu, tristan.cazenave@dauphine.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 challenge in combinatorial optimization that requires efficient job allocation to machines to minimize makespan. To this end, we introduce Starjob, the first supervised dataset for JSSP, comprising 120k instances specifically designed for training LLMs. Leveraging this dataset, we fine-tune the LLaMA 8B model with the LoRA method to develop an end-to-end scheduling approach. Our evaluation on standard benchmarks demonstrates that the proposed LLM-based method not only surpasses traditional Priority Dispatching Rules (PDRs) but also achieves notable improvements over state-of-the-art neural approaches like L2D, with an average improvement of 11.28% on DMU and 3.29% on Taillard benchmarks. These results highlight the untapped potential of LLMs in tackling combinatorial optimization problems, paving the way for future advancements in this area.

Introduction

Large Language Models (LLMs), despite their powerful capabilities in natural language processing, have not traditionally been associated with solving computationally intensive problems. Specifically, their applicability to NP-hard combinatorial optimization problems is often considered limited compared to other neural approaches. This perception is reinforced by the lack of examples where LLMs have successfully outperformed methods like reinforcement learning in such domains. Furthermore, LLMs are prone to "hallucinations," where they not only fail to solve problem instances but also produce infeasible solutions. Consequently, LLMs have yet to be seriously explored (including fine-tuning) for tackling hard combinatorial problems.

In this paper, we challenge this prevailing intuition by demonstrating that LLMs, when fine-tuned, can be effective for certain combinatorial optimization problems. We present the first fine-tuned LLM model for the Job Shop Scheduling Problem (JSSP)—and, to the best of our knowledge, for any NP-hard combinatorial problem. Our results show that for JSSP, LLMs not only generate feasible solutions but also surpass Priority Dispatch Rule (PDR) methods and outperform the earliest neural approaches that first exceeded PDR performance (e.g., L2D (Zhang et al. 2020)). These findings suggest that with further refinement, LLM-based approaches could rival the most advanced neural methods for combinatorial optimization. This work opens the door to exploring LLM applications in a broader range of computational problems, potentially establishing a new paradigm in the field.

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

The job shop scheduling problem (JSSP) remains a well-studied and computationally challenging problem in the field of production scheduling and optimization. It entails the efficient allocation of a set of N_J jobs, each with heterogeneous processing times, to a limited number of N_M machines. The primary objective is to optimize a performance metric, such as minimizing the total completion time (makespan, denoted by C_{max}) or reducing the flow time (average completion time) of individual jobs. JSSP finds application in diverse manufacturing and service environments, impacting factors like production throughput, resource utilization, and ultimately, customer service levels. Traditional approaches to JSSP have primarily relied on mathematical programming techniques and heuristic algorithms (Chaudhry and Khan 2015). However, these methods often exhibit limitations in scalability and effectiveness, particularly for large-scale problems, or those with complex job-machine precedence relationships. This has motivated the exploration of alternative approaches, particularly with the recent advances in artificial intelligence (AI). Techniques like reinforcement learning and graph neural networks have shown promise in addressing JSSP, offering data-driven solutions to this problem(Zhang et al. 2020)(Corsini et al. 2024).

(Huang et al. 2022) examined the graph reasoning capabilities of large language models (LLMs) on tasks like connectivity, shortest paths, maximum flow, and Hamilton paths. While LLMs show promise, their performance declines on complex problems, often relying on spurious correlations. To address this, (Huang et al. 2022) introduced improved prompting strategies. (Valmeekam et al. 2022) introduce a benchmark to test for evaluating the planning/reasoning capabilities of LLMs. Recently, (Chen et al. 2024a) investigate the application of LLMs to the task of graph node classification.

Collectively, these studies underscore the growing use of LLMs for tasks involving implicit structures, while their application to scheduling problems remains unexplored. This paper is the first to utilize LLMs for end-to-end scheduling in JSSP, leveraging their ability to process and reason over complex information to tackle this challenge. To this end, we introduce the first supervised dataset Starjob [1] designed to fine-tune LLMs specifically for the task of JSSP. Instead of traditional matrix representation format, this dataset includes natural language description of the JSSP problem and solution. On two well-known JSSP benchmarks Tai(Taillard 1993) and DMU(Demirkol, Mehta, and Uzsoy 1998), we show that minimal fine-tuning through RsLoRA (Kalajdzievski 2023) on the proposed dataset enables LLM to schedule, by finding high-quality solutions, surpassing PDRs and exceeding or equating neural approaches.

The contributions of this work to the field of JSSP are multifaceted:

- We introduce the first-ever supervised dataset Starjob containing 120,000 instances specifically designed for training LLMs in the context of JSSP
- This paper pioneers the use of LLMs for end-to-end scheduling in JSSP. By fine-tuning LLMs with our Starjob dataset using the RsLoRA method, we demonstrate that LLMs can effectively process and reason over the complexities of JSSP, offering a novel approach that leverages the inherent strengths of language models in handling complex information.
- We perform a comparative analysis of LLM-based scheduling against four traditional priority dispatching rules (PDRs) (Veronique Sels and Vanhoucke 2012): Shortest Processing Time (SPT), Most Work Remaining (MWKR), Most Operations Remaining (MOPNR), and the minimum ratio of Flow Due Date to Most Work Remaining (FDD/MWKR). Additionally, we compare our approach to the state-of-the-art neural method L2D (Zhang et al. 2020), highlighting the effectiveness of end-to-end LLM-based scheduling in comparison to existing classical and neural techniques.
- Leveraging the language capabilities of LLMs, our approach enables natural language interactions with the scheduler. Users can pose questions about specific JSSP instances directly to the LLM-based solver, gaining insights into the problem's inherent constraints. This feature enhances the transparency and usability of the scheduling system.

Related Work

JSSP with more than two machines is proven to be NP-hard (Garey, Johnson, and Sethi 1976). As a result, finding exact solutions for JSSP is generally infeasible, leading to the widespread use of heuristic and approximate methods for practical efficiency (Cebi, Atac, and Sahingoz 2020). Traditional approaches to solving JSSP have primarily relied on search and inference techniques developed by the constraint programming community (Beck, Feng, and Watson 2010). 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 (Nowicki and Smutnicki 2005). A widely used heuristic method in real-world scheduling systems is the Priority Dispatching Rule (PDR) (Zahmani et al. 2015). 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 (Bonetta et al. 2023; Zhang et al. 2020; Corsini et al. 2024). 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(Zhang et al. 2020).

Large language models (LLMs) are now being applied to a wider range of tasks beyond language processing. In areas like robotics and planning, LLMs have been employed to direct agents through structured environments (Huang et al. 2022).

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 (Chen et al. 2023; Wei et al. 2022; Ahn et al. 2024; Yang et al. 2023). Optimization using large language models (LLMs) has gained significant interest in recent years, with several works exploring their capabilities across various domains (Yang et al. 2023). 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(Yang et al. 2023). Notably, (Chen et al. 2023) have done a comprehensive evaluation of LLMs, incorporating an examination of their performance in mathematical problem-solving. (Chen et al. 2023) introduces a novel approach called "Program of Thoughts" (PoT) prompting. Unlike the Chain of Thoughts (CoT) method(Wei et al. 2022), which uses language models to generate both reasoning steps and computations, PoT separates these tasks. PoT uses language models to generate programming language statements for the reasoning steps and then delegates the actual computation to a program interpreter. In (Ahn et al. 2024) the authors conduct a comprehensive survey of mathematical problems and corresponding datasets investigated in the context of LLMs. (Ahn et al. 2024) examines the spectrum of LLM-oriented techniques for mathematical problem-solving, providing insights into their strengths and weaknesses. (Frieder et al. 2024) explores the impact of LLMs on mathematicians' workflows, envisioning changes in research and education through automated assistance and new exploration methods. It provides empirical evidence on LLMs' performance in solving problems and generating proofs, highlighting both successes and failures to give a balanced view of their current capabilities.

More recent works, such as (Yang et al. 2023) highlight

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

the potential of LLMs as optimizers, capable of iteratively refining solutions based on a trajectory of previously evaluated solutions. By leveraging the unique strengths of LLMs, such as their natural language understanding and generation capabilities. Paper demonstrates case studies on two fundamental optimization problems: linear regression and the traveling salesman problem. (Yang et al. 2023) demonstrates that in small-scale optimization scenarios, LLMs can generate high-quality solutions solely through prompting, sometimes matching or even surpassing the performance of manually crafted heuristic algorithms.

Explorations into using LLMs for graph learning tasks have yielded notable approaches. (Huang et al. 2022) noted that LLMs exhibit some initial graph reasoning capabilities, but their performance decreases with problem complexity, (Huang et al. 2022) introduced prompting strategies to improve LLMs graph reasoning. (Valmeekam et al. 2022) developed a benchmark for assessing the planning and reasoning abilities of LLMs. More recently, (Chen et al. 2024a) examined the use of LLMs for graph node classification tasks. (Chen et al. 2024b) introduces two pipelines: LLMs-as-Enhancers, where LLMs refine textual data for Graph Neural Networks (GNNs), and LLMs-as-Predictors, where LLMs generate predictions directly from graph structures in natural language. Additionally, (Zhao et al. 2024) presents GRAPHTEXT, a method that translates graphs into natural language for LLM-based reasoning. GRAPHTEXT constructs graph-syntax trees for training-free, interactive reasoning, achieving performance on par with or exceeding supervised GNNs through in-context learning, highlighting LLMs' potential in graph machine learning. Together, these studies emphasize the increasing application of LLMs for tasks related to implicit graphs and structures, while their use in scheduling problems remains largely unexamined.

Preliminary

JSSP is formally defined as a problem involving a set of jobs J and a set of machines M. The size of the JSSP problem instance is described as $N_J \times N_M$, where N_J represents the number of jobs and N_M the number of machines. For each job $J_i \in J$, it must be processed through n_i machines (where n_i is the number of operations for job J_i) in a specified order $O_{i1} \to \ldots \to O_{in_i}$, where each O_{ij} (for $1 \leq j \leq n_i$) represents an operation of J_i with a processing time $p_{ij} \in \mathbb{N}$. This sequence also includes a precedence constraint. Each machine can process only one job at a time, and switching jobs mid-operation is not allowed. The objective of solving a JSSP is to determine a schedule, that is, a start time S_{ij} for each operation O_{ij}, to minimize the makespan $C_{\max} = \max_{i,j}\{C_{ij} = S_{ij} + p_{ij}\}$ while meeting all constraints. The complexity of a JSSP instance is given by $N_J \times N_M$.

Dataset Generation

In order to try to solve the JSSP with LLM, we first need to represent the problem in natural language. To do that, we have to transform the matrix-based representation in standard JSSP format to a human-readable format. See the example in Listing 1.

Listing 1: Job Shop Scheduling Problem instance (ft06)(Fisher and Thompson 1963) with $N_J = 6$ and $N_M = 6$. The problem instance begins with the problem size on the first row, followed by the operations for each job. Odd columns list machines, and even columns list durations. The last row indicates the makespan (55.0)

```
1 6 6
2 2 1 0 3 1 6 3 7 5 3 4 6
3 1 8 2 5 4 10 5 10 0 10 3 4
4 2 5 3 4 5 8 0 9 1 1 4 7
5 1 5 0 5 2 5 3 3 4 8 5 9
6 2 9 1 3 4 5 5 4 0 3 3 1
7 1 3 3 5 9 0 10 4 4 2 1
8 55.0
```

Converting JSSP problem instance to Natural Language: Feature Generation

The approach describes the machines required for each job, providing a job-centric view of the scheduling problem.

- **Initialization:** Begins by introducing the problem, detailing the number of jobs and machines involved.
- **Problem Organization:** Enumerates jobs, specifying the sequence of the corresponding machines, and their respective durations.

Listing 2: Natural Language description of a JSSP instance of size $N_J = 3$ and $N_M = 3$

```
1
2 Optimize schedule for 3 Jobs (denoted
     as J) across 3 Machines (denoted
     as M) to minimize makespan. The
    makespan is the completion time
    of the last operation in the
    schedule. Each M can process only
     one J at a time, and once
    started, J cannot be interrupted.
3
4 J0:
5 M0:105 M1:29 M2:213
6 J1:
7 M0:193 M1:18 M2:213
8 J3:
9 M0:78 M1:74 M2:221
```

Definitions

- $\mathcal{L}_p$: Natural language representation of a problem instance p.
- s: A solution in natural language, detailing operation sequences, machine assignments, and timings.
- $\mathcal{S}_p^f$: Set of feasible solutions satisfying all JSSP constraints.
- $M(s)$: Makespan of solution s.
- Objective: Minimize $M(s)$ for feasible solutions $s \in \mathcal{S}_p^f$.

Proposed Method

1. **Fine-Tuning:** Train the LLM on problem-solution pairs $(\mathcal{L}_p, s)$ to generate valid schedules.
2. **Inference:** Generate S candidate solutions:
$$\{s_1, \ldots, s_S\} \sim \mathrm{LLM}_\theta(\mathcal{L}_p).$$
3. **Feasibility Check:** Filter feasible solutions:
$$\mathcal{S}_p^f = \{s \mid \text{All constraints are satisfied}\}.$$
4. **Optimization:** Select the solution with the minimum makespan:
$$s^* = \arg\min_{s \in \mathcal{S}_p^f} M(s).$$

Zero-shot inference and Label generation

Our choice of LLM is Meta-Llama-3.1-8B-Instruct-bnb-4bit open-source model with 128K context size. Later we will refer this model as Llama3.1 The model is one of the open-source AI models developed by Meta. Llama3.1 is an auto-regressive language model that uses an optimized transformer architecture (AI 2024a).

Initially, we considered performing zero-shot inference with the Llama3.1 to solve the JSSP. However, the model consistently produced general descriptions of how to solve the problem instead of actual solutions please refer to Figure 8 in Appendix. Occasionally, it provided partial solutions, however, during each inference time the structure of the provided solution was different, making it hard to parse the solution.

Because the zero-shot inference results were not satisfactory, we decided to finetune the large language model (LLM) using a supervised approach. This required creating a supervised dataset, which included not only the problem formulations in natural language as described in Section but also the solutions.

To generate feasible solutions, we employed Google's OR-Tools. The configuration for the Google's OR-Tools solver was set as follows:

- Maximum time allowed for the solver: `300` seconds.
- Number of search workers: `42`.
- Search branching strategy: `cp_model.AUTOMATIC_SEARCH`.

We have generated approximately 120,000 random JSSP problems of various sizes [2], ranging from 2x2 to 20x20, with the duration of each operation between 5 and 500 units. We created problems with asymmetric sizes also, such as 3x2 and 10x5, to enhance the model's generalization capability. Overall, the final dataset consists of around 120,000 natural language descriptions of JSSP problems along with their feasible solutions. Since we limited the maximum allowed time for Google's OR-Tools to 300 seconds, the optimality of solutions for problems with $N_J > 10$ and $N_M > 10$ is not guaranteed. The the generated solution is converted to LLM format as described in

[2]https://github.com/starjob42/Starjob

Listing 3: Natural Language description of the solution of JSSP problem instance of size $N_J = 3$ and $N_M = 3$

```
1 Solution:
2 J2-M0: 0+78 -> 78, J1-M2: 0+193 ->
      193, J0-M0: 78+105 -> 183,
3 J0-M1: 183+29 -> 212, J2-M2: 193+74
      -> 267, J1-M1: 212+18 -> 230,
4 J1-M0: 230+213 -> 443, J2-M1: 267+221
       -> 488, J0-M2: 267+213 -> 480
5
6 Maximum end completion time or
      Makespan: 488
```

Representation in summation format aids large language models (LLMs) in performing computations effectively, enabling them to accurately calculate the makespan and produce feasible solutions with the minimum makespan; in contrast, our comparison with solutions generated without the summation operation often resulted in infeasible outputs.

Training Details

We fine-tuned Llama 3.1, an 8 billion-parameter model from Meta, utilizing a 4-bit quantized version to minimize memory usage. We used Rank-Stabilized Low-Rank Adaptation (RSLoRA) (Kalajdzievski 2023) with a rank of $r = 64$ and $\alpha = 64$. The model was trained for one epoch, requiring roughly 70 hours and about 30GB of GPU memory. We limited the context length of the model to 10k instead of the original 128k context length, to reduce memory consumption and increase the speed of fine-tuning. "Context length" refers to the maximum number of tokens (words or subwords) the model can process at once as input. For additional details on the training process, please refer to the appendix .

Evaluation

To ensure a fair comparison, we evaluated the fine-tuned LLM on two well-known benchmarks, Tai (Taillard 1993) and DMU (Demirkol, Mehta, and Uzsoy 1998), focusing on problem instances with a maximum of $N_J = 20$ and $N_M = 20$. This limitation stems from the reduction of the model's context length to 10k tokens, which constrains its ability to handle larger instances requiring longer input sequences. Since this is the first time an LLM has been employed for end-to-end scheduling on the JSSP problem, we compared its performance to the first neural approach, L2D (Zhang et al. 2020), which was one of the first methods that demonstrated superiority over traditional priority dispatching rules (PDRs). The PDRs included in the comparison are Shortest Processing Time (SPT), Most Work Remaining (MWKR), Most Operations Remaining (MOPNR), and the minimum ratio of Flow Due Date to Most Work Remaining (FDD/MWKR).

During inference, the context length is set to 10k to align with the configuration used during the fine-tuning phase. A sampling strategy is employed, using the default hyperparameters. Additionally, a sample size of $S = 20$ is specified, meaning that at each inference step, the model generates and returns 20 different outputs for evaluation. During

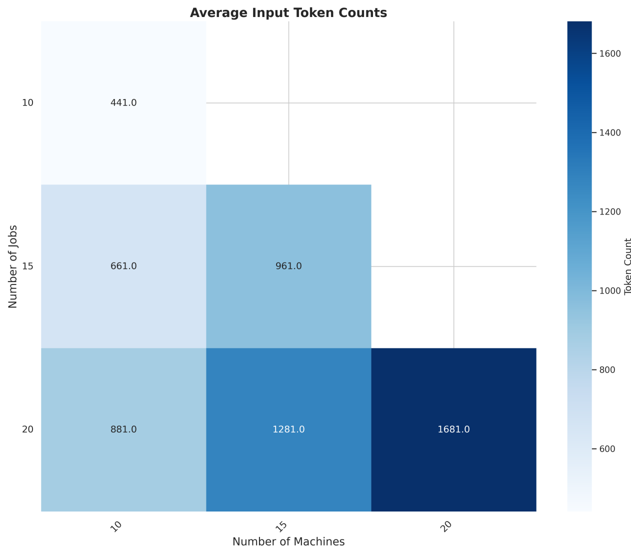

Figure 1: Starjob 120k dataset average input (problem description) token count for Llama 3.1 8B

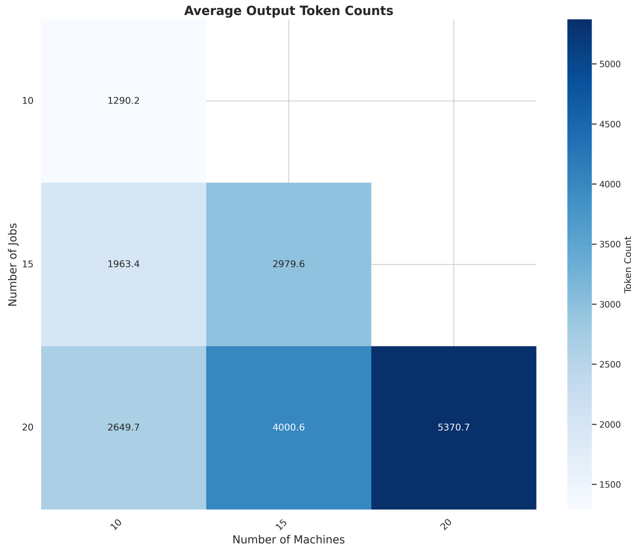

Figure 2: Starjob 120k dataset average output (solution) token count for Llama 3.1 8B

both training and inference time, the model was loaded in the format $float4$. The inference process itself consumes approximately 30GB of memory on the NVIDIA A6000 GPU with $float4$ data type. The largest instance to be tested in total contains around 7200 tokens, please refer to Figure 1 and Figure 2. For faster inference, the fine-tuned model can be converted to the `llama.cpp` format (Gerganov 2023). This conversion achieves an inference speed of 102.22 tokens per second as reported in (Dai 2024) on an NVIDIA RTX A6000 GPU with 48 GB of memory. Consequently, for the largest instance consisting of 1,700 input tokens and an expected 5,500 output tokens, each inference takes approximately 70.4 seconds.

Overview of JSSP Solution Parsing and Validation

Given a JSSP problem instance $\mathcal{L}_p$ and a solution s in natural language, the feasibility check ensures that s satisfies all constraints and identifies feasible solutions $\mathcal{S}_p^f$. The objective is to minimize the makespan:

$$s^* = \arg\min_{s \in \mathcal{S}_p^f} M(s),$$

where $M(s)$ is the makespan of solution s.

Validation Steps

1. **Parsing Inputs:** Extract jobs J_i, machines M_k, operations O_{ij}, start times S_{ij}, processing times p_{ij}, end times C_{ij}, and declared makespan $C_{\max}$.
2. **Precedence Constraints:** For each job J_i, ensure operations O_{ij} follow their prescribed order:

$$S_{i(j+1)} \geq C_{ij}, \quad C_{ij} = S_{ij} + p_{ij}.$$

3. **Machine Constraints:** For each machine M_k, verify no overlapping operations:

$$S_{ij} \geq C_{kl} \quad \text{or} \quad S_{kl} \geq C_{ij},$$

   where O_{ij} and O_{kl} are operations assigned to M_k.
4. **Completeness and Validity:** Check that:
   - All jobs J_i and operations O_{ij} are represented.
   - Machines M_k process only one operation at a time.
   - All start and end times S_{ij}, C_{ij} are within valid bounds.
5. **Makespan Validation:** Compute:

$$C_{\max} = \max_{i,j} \{C_{ij}\},$$

   and compare it with the declared makespan. If mismatched, the solution is invalid.

If all these checks pass, the solution is deemed feasible.

Comparative Analysis with Other Neural Approaches

Since this is the first time an LLM is applied as an end-to-end scheduler for the JSSP, we compare our approach with the work presented in ”Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning” (L2D) (Zhang et al. 2020). This comparison is fair because L2D was the first approach to use neural networks to outperform traditional priority dispatching rule (PDR) methods, making it analogous to our work, which is the first to apply LLMs to JSSP. For consistency, we used the network trained on instances with $N_J = 20$ and $N_M = 20$. L2D's method utilizes a Graph Neural Network (GNN) with Proximal Policy Optimization (PPO) and employs a size-agnostic policy network for generalization. Table 1 and Table 2 presents the performance comparison of the Llama-Finetuned model on the proposed Starjob dataset against various scheduling methods (L2D, SPT, MWKR, FDD/WKR, MOPNR) on the

Tai (Taillard 1993) and DMU (Demirkol, Mehta, and Uzsoy 1998) datasets, focusing on gap percentages relative to optimal solutions makespan. On the Tai benchmark dataset instances with 15 Jobs, 15 Machines, and with 20 Jobs, 20 Machines, finetuned Llama outperforms all other methods. On instances with 20 Jobs and 20 Machines Llama (33.12%) slightly trails L2D (31.60%) but is better than other PDRs. Average Gap: Finetuned Llama (26.57%) is significantly lower than SPT (61.33%), MWKR (57.66%), FDD/WKR (48.86%), and MOPNR (45.88%).

On the DMU benchmark dataset with 20 Jobs and 15 Machines finetuned Llama (25.64%) again demonstrates superior performance against all methods including L2D(38.95 %) (Zhang et al. 2020). Finetuned Llama (28.50%) is also notably lower average gap on DMU benchmark dataset instances having 20 Jobs and 20 Machines. Interestingly, the solutions generated by our model—the first end-to-end LLM-based scheduling approach—demonstrate competitive performance, achieving results within 20% of the *current* state-of-the-art neural method (Corsini et al. 2024), which utilizes pseudo labels generated by the policy for self-supervised training.

Table 1: Comparison of PDRs on the **TAI** dataset. Lower values indicate schedules closer to the optimal solution, representing better performance.

Method	**15x15**	**20x15**	**20x20**	**Average**
L2D	25.95	30.03	31.60	**29.86**
SPT	54.64	65.24	64.11	**61.33**
MWKR	56.74	60.65	55.60	**57.66**
FDD/WKR	47.45	50.57	47.57	**48.86**
MOPNR	44.98	47.97	43.68	**45.88**
Llama-Finetuned-Ours	19.68	26.91	33.12	**26.57**

Table 2: Comparison of PDRs on the **DMU** dataset. Lower values indicate schedules closer to the optimal solution, representing better performance.

Method	**20x15**	**20x20**	**Average**
L2D	38.95	37.74	**38.35**
SPT	64.12	64.55	**64.34**
MWKR	62.14	58.16	**60.15**
FDD/WKR	53.58	52.51	**53.05**
MOPNR	49.17	45.18	**47.18**
Llama-Finetuned-Ours	25.64	28.50	**27.07**

Empirical Performance Analysis

In this section, we provide an in-depth comparison of various job scheduling approaches in terms of the gap percentage, which measures the deviation from the optimal solution. The comparison includes several Priority Dispatching Rules (PDRs), a neural approach (L2D), and a fine-tuned Llama model on proposed Starjob dataset. Figure 3 and Figure 4 presents the performance on both Tai(Taillard 1993) and DMU (Demirkol, Mehta, and Uzsoy 1998) datasets across various configurations of jobs (J) and machines (M). The lower the gap percentage, the closer the schedule is to the optimal solution, thus representing better performance.

The five configurations analyzed are:

- $J = 20$, $M = 20$ (Tai dataset)
- $J = 20$, $M = 20$ (DMU dataset)
- $J = 20$, $M = 15$ (Tai dataset)
- $J = 20$, $M = 15$ (DMU dataset)
- $J = 15$, $M = 15$ (Tai dataset)

The *SPT* consistently exhibits the highest gap percentages, exceeding 60% for most problem instances. This is expected since *SPT*, while simple, often fails to account for job-shop constraints in complex problem settings. The *MWKR* and *FDD/WKR* heuristics, which are more sophisticated than *SPT*, perform moderately better, with gap percentages ranging between 50% and 70%. However, these heuristics are still outclassed by the machine learning-based approaches, likely due to their myopic decision-making, which does not factor in longer-term scheduling impacts.

The L2D (Zhang et al. 2020) model, which leverages neural networks for decision-making, offers significant improvements, reducing the gap to the 30%-40% range. This highlights the benefits of learning-based approaches over traditional PDRs, as L2D can implicitly model complex job-shop interactions and adapt to different problem instances. Surprisingly fine-tuned Llama model on Starjob outperforms all pdr methods, consistently achieving gap percentages below 45%. This demonstrates the ability of LLMs to generalize across problem instances, effectively and sometimes even outperforming the specialized neural L2D model.

The results for the DMU dataset with $J = 20$, $M = 20$ mirror those of the Tai dataset (top-middle plot of Figure 3 and Figure 4). Here, we observe that traditional PDRs (*SPT*, *MWKR*, *FDD/WKR*) consistently exhibit high gap percentages, with little to no improvement across problem instances . The L2D model once again shows significant improvements over the PDRs, with gap percentages reduced to the 20%-50% range. Please refer to Table 3 and Table 4 in the appendix for more detailed comparison.

Overall, the results highlight that with minimal fine-tuning on the proposed Starjob dataset, not only Llama was able to provide feasible solutions, but also surpass other traditional approaches.

Conclusion

This paper demonstrates the potential of Large Language Models (LLMs) in addressing the JSSP. We introduced a novel supervised dataset called Starjob for solving JSSP tailored for LLM training. Our results on well known benchmark problems(Taillard 1993), (Demirkol, Mehta, and Uzsoy 1998) indicate that with minimal fine-tuning using the RsLoRA method(Kalajdzievski 2023), Llama 8B can effectively schedule, matching or surpassing traditional PDRs and neural network approaches.

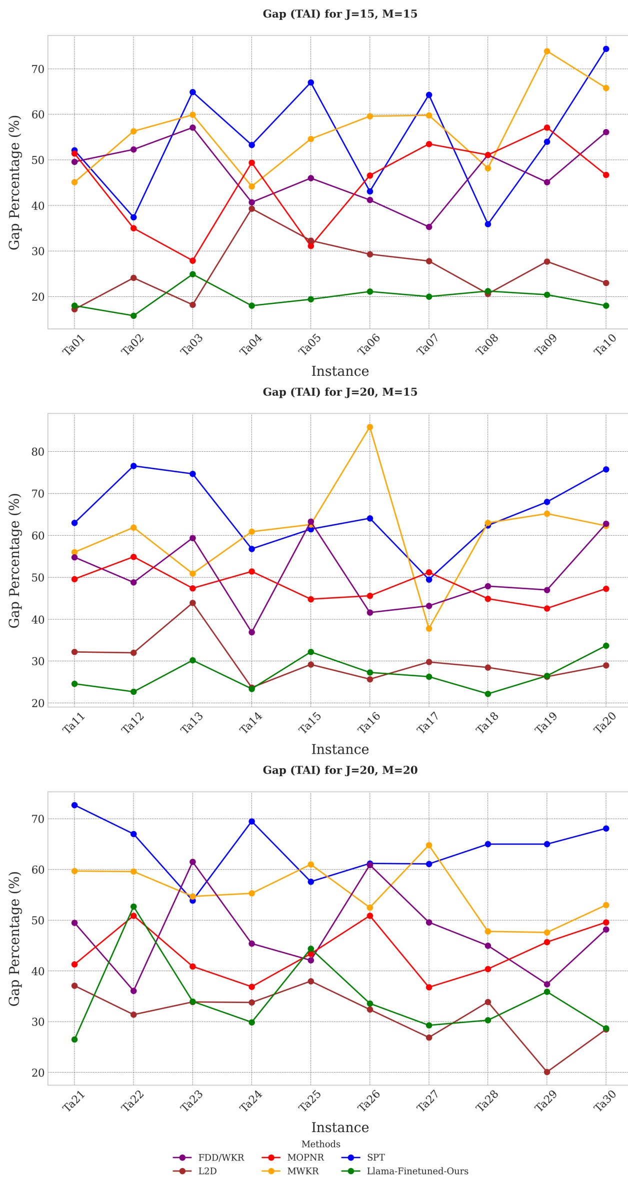

Figure 3: Comparison of TAI metrics. This plot provides insights into the performance of the approaches under consideration.

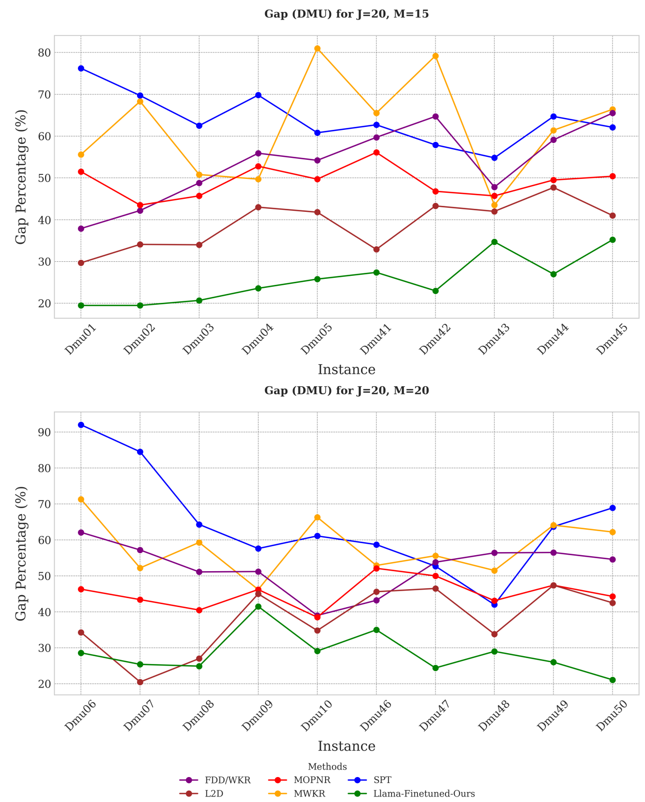

Figure 4: Comparison of DMU metrics. This plot provides insights into the performance of the approaches under consideration.

Limitations and Future Work

This work represents the first exploration of utilizing LLMs for tackling the Job Shop Scheduling Problem (JSSP), demonstrating their significant potential in this complex domain. Our experiments show that LLMs can generate effective scheduling solutions, opening up new avenues for applying natural language processing advances to scheduling problems.

An inherent advantage of LLMs is their ability as language models to facilitate interactive exploration of the JSSP. Users can engage with the LLM to ask questions about the scheduling problem it is attempting to solve, gaining insights into which constraints may hinder finding the optimal solution.

While the results are promising, we acknowledge that the generalizability of LLMs across diverse JSSP instances poses a challenge. The models may require further refinement to consistently perform well on a wider variety of problem sizes and configurations. Addressing this limitation offers an opportunity for future research to enhance the robustness of LLMs in scheduling applications.

By introducing the Starjob dataset and applying LLMs, we have laid the groundwork for future research at the intersection of scheduling and language models. Exploring various LLM architectures, fine-tuning strategies, and integrating LLMs with other artificial intelligence techniques like reinforcement learning and graph neural networks may further enhance performance.

In summary, our study successfully tests the potential of LLMs on the JSSP problem for the first time, indicating a promising direction for future exploration in the field of scheduling.

References

Ahn, J.; Verma, R.; Lou, R.; Liu, D.; Zhang, R.; and Yin, W. 2024. Large Language Models for Mathematical Reasoning: Progresses and Challenges. In Falk, N.; Papi, S.;

and Zhang, M., eds., *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, 225–237. St. Julian's, Malta: Association for Computational Linguistics.

AI, M. 2024a. Llama 3 Model Card. Accessed: 2024-08-10.

AI, U. 2024b. Unsloth: Accelerated Fine-Tuning for Large Language Models. Accessed: 2024-11-19.

Beck, J. C.; Feng, T. K.; and Watson, J.-P. 2010. Combining Constraint Programming and Local Search for Job-Shop Scheduling. *INFORMS Journal on Computing*, 23(1): 1–14.

Bonetta, G.; Zago, D.; Cancelliere, R.; and Grosso, A. 2023. Job Shop Scheduling via Deep Reinforcement Learning: a Sequence to Sequence approach. *Not Specified*.

Cebi, C.; Atac, E.; and Sahingoz, O. K. 2020. Job Shop Scheduling Problem and Solution Algorithms: A Review. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 1–7.

Chaudhry, S. A.; and Khan, S. 2015. Comparison of dispatching rules in job-shop Scheduling problem Using simulation: A case study. *ResearchGate*.

Chen, W.; Ma, X.; Wang, X.; and Cohen, W. W. 2023. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *Transactions on Machine Learning Research*.

Chen, Z.; Mao, H.; Li, H.; Jin, W.; Wen, H.; Wei, X.; Wang, S.; Yin, D.; Fan, W.; Liu, H.; and Tang, J. 2024a. Exploring the Potential of Large Language Models (LLMs) in Learning on Graphs. arXiv:2307.03393.

Chen, Z.; Mao, H.; Li, H.; Jin, W.; Wen, H.; Wei, X.; Wang, S.; Yin, D.; Fan, W.; Liu, H.; and Tang, J. 2024b. Exploring the Potential of Large Language Models (LLMs) in Learning on Graphs. arXiv:2307.03393.

Corsini, A.; Porrello, A.; Calderara, S.; and Dell'Amico, M. 2024. Self-Labeling the Job Shop Scheduling Problem. In *Self-Labeling the Job Shop Scheduling Problem*. Arxiv.

Dai, X. 2024. GPU-Benchmarks-on-LLM-Inference. https://github.com/XiongjieDai/GPU-Benchmarks-on-LLM-Inference. Accessed: 2024-11-26.

Demirkol, E.; Mehta, S.; and Uzsoy, R. 1998. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1): 137–141.

Fisher, H.; and Thompson, G. L. 1963. Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules. In Muth, J. F.; and Thompson, G. L., eds., *Industrial Scheduling*, chapter 3.2, 225–251. Englewood Cliffs, NJ, USA: Prentice-Hall.

Frieder, S.; Berner, J.; Petersen, P.; and Lukasiewicz, T. 2024. Large Language Models for Mathematicians. arXiv:2312.04556.

Garey, M. R.; Johnson, D. S.; and Sethi, R. 1976. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2): 117–129.

Gerganov, G. 2023. llama.cpp: LLM inference in C/C++. https://github.com/ggerganov/llama.cpp. Accessed: 2024-11-26.

Hu, E. J.; yelong shen; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*.

Huang, W.; Abbeel, P.; Pathak, D.; and Mordatch, I. 2022. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. In *Proceedings of the International Conference on Machine Learning*. PMLR. *equal advising.

Kalajdzievski, D. 2023. A Rank Stabilization Scaling Factor for Fine-Tuning with LoRA. arXiv:2312.03732.

Nowicki, E.; and Smutnicki, C. 2005. An Advanced Tabu Search Algorithm for the Job Shop Problem. *Journal of Scheduling*, 8(2): 145–159.

Taillard, E. 1993. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2): 278–285.

Valmeekam, K.; Olmo, A.; Sreedharan, S.; and Kambhampati, S. 2022. 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*.

Veronique Sels, N. G.; and Vanhoucke, M. 2012. A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions. *International Journal of Production Research*, 50(15): 4255–4270.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E. H.; Le, Q. V.; and Zhou, D. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Google Research, Brain Team*.

Yang, C.; Wang, X.; Lu, Y.; Liu, H.; Le, Q. V.; Zhou, D.; and Chen, X. 2023. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*.

Zahmani, M. H.; Atmani, B.; Bekrar, A.; and Aissani, N. 2015. Multiple Priority Dispatching Rules for the Job Shop Scheduling Problem. In *3rd International Conference on Control, Engineering Information Technology (CEIT'2015)*. Tlemcen, Algeria.

Zhang, C.; Song, W.; Cao, Z.; Zhang, J.; Tan, P. S.; and Xu, C. 2020. Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning. In *34th Conference on Neural Information Processing Systems (NeurIPS)*.

Zhao, J.; Zhuo, L.; Shen, Y.; Qu, M.; Liu, K.; Bronstein, M. M.; Zhu, Z.; and Tang, J. 2024. GraphText: Graph Learning in Text Space.

Appendix

Trainig Details

Model Overview

The model being fine-tuned is LLaMA 3.1, an 8 billion parameter model from Meta(AI 2024a), using a 4-bit quantized version to reduce memory usage. Finetning was conducted using Stabilized Low-Rank Adaptation (RsLoRA) ()

with rank $r = 64$ to introduce learnable parameters specifically in targeted layers. (Kalajdzievski 2023) Compared to Lora(Hu et al. 2022) RsLoRa improves the stability of training by modifying the rank during adaptation(Kalajdzievski 2023). The target modules include:

$$\text{target_modules} = \{\texttt{q_proj}, \texttt{k_proj}, \texttt{v_proj}, \texttt{o_proj}, \texttt{gate_proj}, \texttt{up_proj}, \texttt{down_proj}\} \quad (1)$$

The LoRA-specific parameters are configured as follows:

- Rank (r): 64
- LoRA Alpha (α): 64
- LoRA Dropout: 0
- Bias: none

This resulted in number of trainable parameters $= 167,772,160$ or 0.02 % of the entire Llama 8B model's parameters.

Quantization and Memory Efficiency

The model is loaded in 4-bit precision to reduce memory consumption. Gradient checkpointing is enabled using the `unsloth` (AI 2024b) method, to fit longer sequences by saving memory. This reduces the VRAM usage by approximately 30%, enabling larger batch sizes.

Training Parameters

The fine-tuning process is controlled by the following parameters:

- **Batch size per device**: 4
- **Gradient accumulation steps**: 4
- **Max sequence length**: 10,000 tokens
- **Number of epochs**: 1
- **Warmup steps**: 5
- **Learning rate**: 2×10^{-4}
- **Optimizer**: AdamW with 8-bit precision
- **Weight decay**: 0.01
- **Learning rate scheduler**: Linear decay
- **FP16 precision**:True
- **Number of Epochs**: 1

Data and Dataset Splitting

The dataset used for training is a local version of the proposed Strajob dataset, and it is split into 98% training and 2% evaluation:

$$\text{train} : \text{eval} = 98\% : 2\%$$

The prompts are formatted using a predefined Alpaca-style template, which ensures the model is trained on instruction-following tasks.

Evaluation and Saving Strategy

The best model was loaded at the end of training based on the evaluation loss:

$$\text{Metric for Best Model} = \text{Evaluation Loss}$$

Total number of saved models is limited to 50 to prevent excessive memory usage.

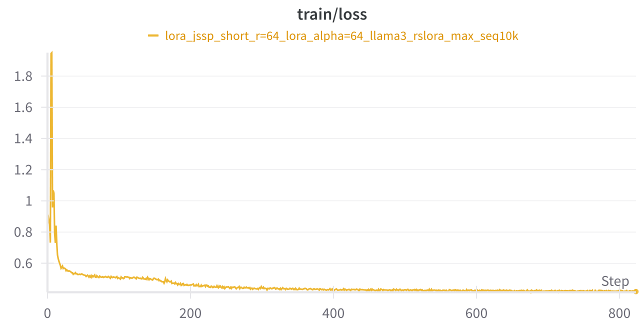

Figure 5: Train loss of Llama 8B 4bt model on Starjob dataset

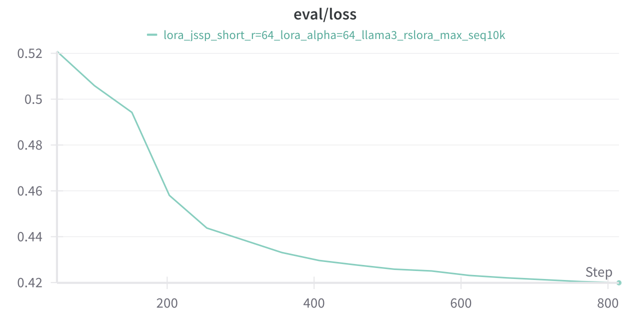

Figure 6: Eval Loss of Llama 8B 4bt model on Starjob dataset

GPU Utilization

The training process takes place on Nvidia A6000 GPU with 48GB of memory. Training took around 70 hours and required 30GB of GPU RAM.

General Statistics about dataset

The dataset comprises 120,000 randomly generated JSSP instances with solutions in natural language, provided in `.json` format with the following columns:

- `num_jobs` (int64): 12 unique values.
- `num_machines` (int64): 12 unique values.
- `instruction` (object): 120,000 unique values. Initial problem description detailing jobs and machines.
- `input` (object): 120,000 unique values. Problem description formatted for LLM.
- `output` (object): 120,000 unique values. Solution in LLM format.
- `matrix` (object): 120,000 unique values. OR-Tool makespan and solution in matrix format.

The `output` column serves as the target or label column, providing the solution to the JSSP problem in natural language and the associated makespan.

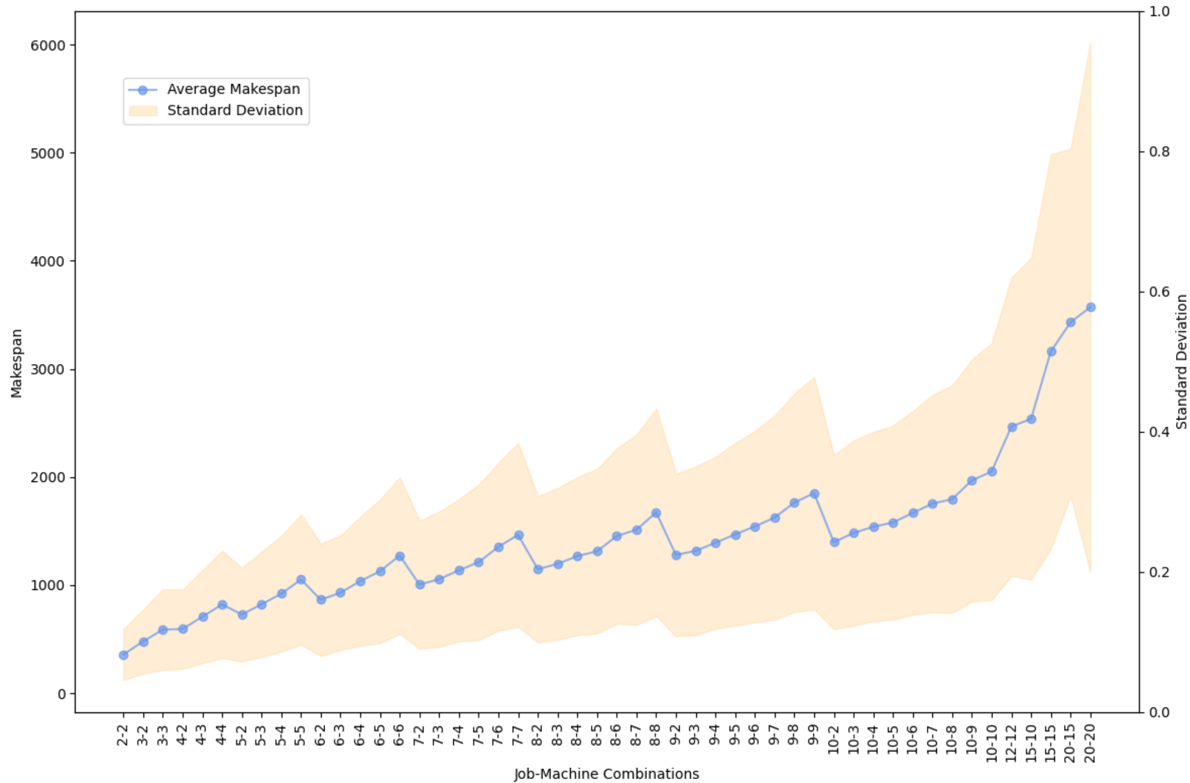

Figure 7: Makespan metrics across different job-machine combinations. The x-axis represents the combinations of jobs and machines (e.g., a 3-2 instance refers to 3 jobs and 2 machines), the right y-axis shows the standard deviation, while the left y-axis shows the makespan values.

```
### Response:
<|endoftext|>The schedule can be optimized using the NEH algorithm. Here is the optimized schedule:

Machine 1: J0, J2, J5, J7, J9, J11, J13, J15, J17, J19
Machine 2: J1, J3, J6, J8, J10, J12, J14, J16, J18, J20
Machine 3: J4, J21, J23, J25, J27, J29, J31, J33, J35, J37
Machine 4: J6, J22, J24, J26, J28, J30, J32, J34, J36, J38
Machine 5: J7, J19, J21, J23, J25, J27, J29, J31, J33, J35
Machine 6: J8, J20, J22, J24, J26, J28, J30, J32, J34, J36
Machine 7: J9, J21, J23, J25, J27, J29, J31, J33, J35, J37
Machine 8: J10, J22, J24, J26, J28, J30, J32, J34, J36, J38
Machine 9: J11, J23, J25, J27, J29, J31, J33, J35, J37, J39
Machine 10: J12, J24, J26, J28, J30, J32, J34, J36, J38, J40
Machine 11: J13, J25, J27, J29, J31, J33, J35, J37, J39, J41
Machine 12: J14, J26, J28, J30, J32, J34, J36, J38, J40, J42
Machine 13: J15, J27, J29, J31, J33, J35, J37, J39, J41, J43
Machine 14: J16, J28, J30, J32, J34, J36, J38, J40, J42, J44
Machine 15: J17, J29, J31, J33, J35, J37, J39, J41, J43, J45
Machine 16: J18, J30, J32, J34, J36, J38, J40, J42, J44, J46
Machine 17: J19, J31, J33, J35, J37, J39, J41, J43, J45, J47
Machine 18: J20, J32, J34, J36, J38, J40, J42, J44, J46, J48
Machine 19: J21, J33, J35, J37, J39, J41, J43, J45, J47, J49
Machine 20: J22, J34, J36, J38, J40, J42, J44, J46, J48, J50

This schedule minimizes the makespan by assigning the jobs to machines in a way that maximizes the utilization of each machine and mi
nimizes the idle time between jobs. The schedule can be further improved by considering other optimization criteria, such as minimizi
ng the maximum completion time of any machine. However, this schedule provides a good starting point for further optimization. |<|end
oftext|>## Step 1: Understand the problem and the input data.
The problem is to optimize the schedule for 100 jobs (denoted as J) across 20 machines (denoted as M) to minimize the makespan, which
 is the completion time of the last operation in the schedule. Each M can process only one J at a time, and once started, J cannot be
 interrupted.

## Step 2: Read and understand the input data.
The input data consists of 100 jobs, each represented by a list of machines and their corresponding processing times. The jobs are de
noted as J0 to J99, and the machines are denoted as M1 to M20.

## Step 3: Choose an optimization algorithm.
The NEH (Non-Enumerative Heuristic) algorithm is a popular choice for solving the flow shop scheduling problem. It works by iterative
ly constructing a feasible schedule and improving it through a series of exchanges.
```

Figure 8: Zero Shot inference on LLama 8B 4bt

Table 3: Comparison of PDRs against L2D gainist Finetuned Llama on Starjob dataset and the average Gaps on Tai Benchmark Dataset. The lower the value, the closer the schedule is to the optimal solution, thus representing better performance.

J	M	Instance	SPT	MWKR	FDD/WKR	MOPNR	L2D	Optimal	Llama-Finetuned-Ours
15	15	Ta01	1872 (52.1%)	1786 (45.1%)	1841 (49.6%)	1864 (51.4%)	1443 (17.2%)	1231.0	1453.0 (18.0%)
15	15	Ta02	1709 (37.4%)	1944 (56.3%)	1895 (52.3%)	1680 (35.0%)	1544 (24.1%)	1244.0	1440.0 (15.8%)
15	15	Ta03	2009 (64.9%)	1947 (59.9%)	1914 (57.1%)	1558 (27.9%)	1440 (18.2%)	1218.0	1521.0 (24.9%)
15	15	Ta04	1825 (53.3%)	1694 (44.2%)	1653 (40.7%)	1755 (49.4%)	1637 (39.3%)	1175.0	1387.0 (18.0%)
15	15	Ta05	2044 (67.0%)	1892 (54.6%)	1787 (46.0%)	1605 (31.1%)	1619 (32.3%)	1224.0	1461.0 (19.4%)
15	15	Ta06	1771 (43.1%)	1976 (59.6%)	1748 (41.2%)	1815 (46.6%)	1601 (29.3%)	1238.0	1499.0 (21.1%)
15	15	Ta07	2016 (64.3%)	1961 (59.8%)	1660 (35.3%)	1884 (53.5%)	1568 (27.8%)	1227.0	1473.0 (20.0%)
15	15	Ta08	1654 (35.9%)	1803 (48.2%)	1839 (51.1%)	1839 (51.1%)	1468 (20.6%)	1217.0	1475.0 (21.2%)
15	15	Ta09	1962 (54.0%)	2215 (73.9%)	1848 (45.1%)	2002 (57.1%)	1627 (27.7%)	1274.0	1534.0 (20.4%)
15	15	Ta10	2164 (74.4%)	2057 (65.8%)	1937 (56.1%)	1821 (46.7%)	1527 (23.0%)	1241.0	1465.0 (18.0%)
20	15	Ta11	2212 (63.0%)	2117 (56.0%)	2101 (54.8%)	2030 (49.6%)	1794 (32.2%)	1357.0	1691.0 (24.6%)
20	15	Ta12	2414 (76.6%)	2213 (61.9%)	2034 (48.8%)	2117 (54.9%)	1805 (32.0%)	1367.0	1677.0 (22.7%)
20	15	Ta13	2346 (74.7%)	2026 (50.9%)	2141 (59.4%)	1979 (47.4%)	1932 (43.9%)	1343.0	1749.0 (30.2%)
20	15	Ta14	2190 (56.8%)	2164 (60.9%)	1841 (36.9%)	2036 (51.4%)	1664 (23.7%)	1345.0	1660.0 (23.4%)
20	15	Ta15	2163 (61.5%)	2180 (62.6%)	2187 (63.3%)	1939 (44.8%)	1730 (29.2%)	1339.0	1770.0 (32.2%)
20	15	Ta16	2232 (64.1%)	2528 (85.9%)	1926 (41.6%)	1980 (45.6%)	1710 (25.7%)	1360.0	1731.0 (27.3%)
20	15	Ta17	2185 (49.5%)	2015 (37.8%)	2093 (43.2%)	2211 (51.2%)	1897 (29.8%)	1462.0	1846.0 (26.3%)
20	15	Ta18	2267 (62.4%)	2275 (63.0%)	2064 (47.9%)	1981 (44.9%)	1794 (28.5%)	1396.0	1706.0 (22.2%)
20	15	Ta19	2238 (68.0%)	2201 (65.2%)	1958 (47.0%)	1899 (42.6%)	1682 (26.3%)	1332.0	1685.0 (26.5%)
20	15	Ta20	2370 (75.8%)	2188 (62.3%)	2195 (62.8%)	1986 (47.3%)	1739 (29.0%)	1348.0	1802.0 (33.7%)
20	20	Ta21	2836 (72.7%)	2622 (59.7%)	2455 (49.5%)	2320 (41.3%)	2252 (37.1%)	1642.0	2077.0 (26.5%)
20	20	Ta22	2672 (67.0%)	2554 (59.6%)	2177 (36.1%)	2415 (50.9%)	2102 (31.4%)	1600.0	2443.0 (52.7%)
20	20	Ta23	2397 (53.9%)	2408 (54.7%)	2514 (61.5%)	2194 (40.9%)	2085 (33.9%)	1557.0	2086.0 (34.0%)
20	20	Ta24	2787 (69.5%)	2553 (55.3%)	2391 (45.4%)	2250 (36.9%)	2200 (33.8%)	1644.0	2135.0 (29.9%)
20	20	Ta25	2513 (57.6%)	2582 (61.0%)	2267 (42.1%)	2146 (43.4%)	2201 (38.0%)	1595.0	2304 (44.4%)
20	20	Ta26	2649 (61.2%)	2506 (52.5%)	2484 (60.9%)	2284 (50.9%)	2176 (32.4%)	1643.0	2195.0 (33.6%)
20	20	Ta27	2707 (61.1%)	2768 (64.8%)	2514 (49.6%)	2298 (36.8%)	2132 (26.9%)	1680.0	2172.0 (29.3%)
20	20	Ta28	2654 (65.0%)	2370 (47.8%)	2330 (45.0%)	2259 (40.4%)	2146 (33.9%)	1603.0	2088.0 (30.3%)
20	20	Ta29	2681 (65.0%)	2399 (47.6%)	2322 (37.4%)	2367 (45.7%)	1952 (20.1%)	1625.0	2209 (35.9%)
20	20	Ta30	2662 (68.1%)	2424 (53.0%)	2348 (48.2%)	2370 (49.6%)	2035 (28.5%)	1584.0	2038.0 (28.7%)

Table 4: Comparison of PDRs against L2D gainist Finetuned Llama on Starjob dataset and the average Gaps on DMU Benchmark Dataset. The lower the value, the closer the schedule is to the optimal solution, thus representing better performance.

J	M	Instance	SPT	MWKR	FDD/WKR	MOPNR	L2D	Optimal	Llama-Finetuned-Ours
20	15	Dmu01	4516 (76.2%)	3988 (55.6%)	3535 (37.9%)	3882 (51.5%)	3323 (29.7%)	2563.0	3064 (19.5%)
20	15	Dmu02	4593 (69.7%)	4555 (68.3%)	3847 (42.2%)	3884 (43.5%)	3630 (34.1%)	2706.0	3233 (19.5%)
20	15	Dmu03	4438 (62.5%)	4117 (50.8%)	4063 (48.8%)	3979 (45.7%)	3660 (34.0%)	2731.0	3296 (20.7%)
20	15	Dmu04	4533 (69.8%)	3995 (49.7%)	4160 (55.9%)	4079 (52.8%)	3816 (43.0%)	2669.0	3299 (23.6%)
20	15	Dmu05	4420 (60.8%)	4977 (81.0%)	4238 (54.2%)	4116 (49.7%)	3897 (41.8%)	2749.0	3458 (25.8%)
20	15	Dmu41	5283 (62.7%)	5377 (65.5%)	5187 (59.7%)	5070 (56.1%)	4316 (32.9%)	3248.0	4137 (27.4%)
20	15	Dmu42	5354 (57.9%)	6076 (79.2%)	5583 (64.7%)	4976 (46.8%)	4858 (43.3%)	3390.0	4169 (23.0%)
20	15	Dmu43	5328 (54.8%)	4938 (43.5%)	5086 (47.8%)	5012 (45.7%)	4887 (42.0%)	3441.0	4634 (34.7%)
20	15	Dmu44	5745 (64.7%)	5630 (61.4%)	5550 (59.1%)	5213 (49.5%)	5151 (47.7%)	3488.0	4429 (27.0%)
20	15	Dmu45	5305 (62.1%)	5446 (66.4%)	5414 (65.5%)	4921 (50.4%)	4615 (41.0%)	3272.0	4423 (35.2%)
20	20	Dmu06	6230 (92.0%)	5556 (71.3%)	5258 (62.1%)	4747 (46.3%)	4358 (34.3%)	3244.0	4173 (28.6%)
20	20	Dmu07	5619 (84.5%)	4636 (52.2%)	4789 (57.2%)	4367 (43.4%)	3671 (20.5%)	3046.0	3821 (25.4%)
20	20	Dmu08	5239 (64.3%)	5078 (59.3%)	4817 (51.1%)	4480 (40.5%)	4048 (27.0%)	3188.0	3982 (24.9%)
20	20	Dmu09	4874 (57.6%)	4519 (46.2%)	4675 (51.2%)	4519 (46.2%)	4482 (45.0%)	3092.0	4376 (41.5%)
20	20	Dmu10	4808 (61.1%)	4963 (66.3%)	4149 (39.0%)	4133 (38.5%)	4021 (34.8%)	2984.0	3853 (29.1%)
20	20	Dmu46	6403 (58.7%)	6168 (52.9%)	5778 (43.2%)	6136 (52.1%)	5876 (45.6%)	4035.0	5447 (35.0%)
20	20	Dmu47	6015 (52.7%)	6130 (55.6%)	6058 (53.8%)	5908 (50.0%)	5771 (46.5%)	3939.0	4899 (24.4%)
20	20	Dmu48	5345 (42.0%)	5701 (51.5%)	5887 (56.4%)	5384 (43.1%)	5034 (33.8%)	3763.0	4854 (29.0%)
20	20	Dmu49	6072 (63.7%)	6089 (64.1%)	5807 (56.5%)	5469 (47.4%)	5470 (47.4%)	3710.0	4674 (26.0%)
20	20	Dmu50	6300 (68.9%)	6050 (62.2%)	5764 (54.6%)	5380 (44.3%)	5314 (42.5%)	3729.0	4515 (21.1%)