LLMs can Schedule

Henrik Abgaryan (corresponding author)[1*], Tristan Cazenave[1] and Ararat Harutyunyan[1]

[1]LAMSADE, CNRS, Université Paris Dauphine–PSL, Paris, Île-de-France, France.

*Corresponding author(s). E-mail(s): henrik.abgaryan@dauphine.eu;
Contributing authors: tristan.cazenave@dauphine.eu;
ararat.harutyunyan@dauphine.eu;

Abstract

The job shop scheduling problem (JSSP) requires assigning operations to machines while respecting job precedence and machine-capacity constraints, typically to minimize makespan. We investigate whether supervised fine-tuning enables large language models (LLMs) to generate complete, feasible schedules. We publicly release 120,000 unique training-source instances paired with feasible solver-generated schedules and fine-tune Phi-3-Mini-128K-Instruct using low-rank adaptation. On a stratified sample of 216 test instances, the final checkpoint produced at least one feasible schedule per instance using ten candidates. Individual candidate validity was 95.51%, and the mean best-of-ten gap to feasible reference schedules was 5.36%. This gap was lower than that of L2D and higher than those of three self-labeling JSSP configurations.

Keywords: Job shop scheduling, Large language models, Supervised fine-tuning, Constraint validation, Training-budget evaluation

1 Introduction

The job shop scheduling problem (JSSP) remains a well-studied and computationally challenging problem in production scheduling and optimization. It involves scheduling a set of N jobs, each consisting of an ordered sequence of operations with specified processing times, on M machines while respecting job precedence and machine-capacity constraints. In this study, the objective is to minimize makespan, $C_{\max}$, the time at which all operations finish; other JSSP formulations may instead optimize measures

such as average job flow time. Classical optimization, local search, dispatching rules, and specialized neural policies offer different tradeoffs between solution quality and computation [1–5]. Generating a schedule as text introduces an additional challenge: a fluent answer may omit operations or violate constraints even when its reported makespan appears plausible.

We study a direct approach in which a large language model (LLM) receives a deterministic natural-language description and emits a complete schedule. We fine-tune Phi-3-Mini-128K-Instruct [6] using supervised low-rank adaptation (LoRA) [7], then assess generated schedules with an independent validator. The decoder generates complete schedules without constraint enforcement, search over partial schedules, or post-generation repair. We evaluate performance across training checkpoints and compare the final checkpoint with released neural schedulers.

We release 120,000 unique training instances paired with solver-generated schedules and 1,800 separate test instances. The training data are divided into 118,000 training and 2,000 validation instances. We assess five training checkpoints on a stratified cohort of 216 test instances and compare the final checkpoint with the released Learning to Dispatch (L2D) policy and three self-labeling JSSP (SLJ) configurations, using ten candidate schedules per instance [4, 5].

Schedule feasibility increases across training checkpoints. On the test cohort, the final checkpoint achieves a 5.36% mean best-of-ten reference gap, compared with 9.86% for L2D and 2.17–2.71% for the SLJ configurations. The evaluation is limited to the released test population and does not measure performance on other benchmark suites.

2 Related Work

Despite the JSSP's worst-case complexity [8], exact methods can solve some JSSP instances, depending on instance size and structure, available bounds, and computational resources. Constraint programming and local search exploit the scheduling structure directly [1, 2], while dispatching rules provide inexpensive construction heuristics [3, 9]. Neural methods include reinforcement-learning dispatching policies, such as Learning to Dispatch (L2D) [4], and self-labeling approaches based on schedule-construction networks [5]. These methods motivate reporting feasibility alongside makespan.

LLMs have been studied as optimizers [10], end-to-end combinatorial optimization solvers [11], assistants for optimization modeling [12], and in iterative model-refinement studies [13]. Program of Thoughts separates generated reasoning programs from their execution [14], whereas Thought of Search and LLM-generated planning heuristics place generated components within explicit search procedures [15, 16]. Here, the model emits all operation timings in one response, without downstream search for constraint checking or feasibility recovery.

For scheduling, MASC studies an LLM-based multi-agent scheduling chain for flexible JSSP [17]. Our setting is classical JSSP with fixed machine assignments and a single makespan objective. We study supervised schedule generation for this setting.

LoRA provides a parameter-efficient adaptation mechanism [7], but does not encode scheduling constraints or guarantee valid output.

3 Problem definition

Let J be a set of jobs and M a set of machines. Job j contains an ordered sequence of operations $O_{j0}, \ldots, O_{j,n_j-1}$. Operation O_{jk} has prescribed machine $m_{jk} \in M$, positive processing time p_{jk}, start time $S_{jk} \geq 0$, and completion time $C_{jk} = S_{jk} + p_{jk}$. Operations are non-preemptive. A feasible schedule assigns every operation exactly once, satisfies job precedence,

$$S_{j,k+1} \geq C_{jk}, \tag{1}$$

and prevents overlap between any two operations assigned to the same machine. Intervals are interpreted as $[S_{jk}, C_{jk})$, so one operation may start when another ends. The objective is to minimize

$$C_{\max} = \max_{j \in J} \max_{0 \leq k < n_j} C_{jk}. \tag{2}$$

The released classical instances contain one operation per machine for each job; their size is the number of jobs times the number of machines. This product counts operations and is not a complete description of computational difficulty.

4 Dataset and natural-language representation

4.1 Released data and reference schedules

The public dataset is available at https://github.com/starjob42/datasetjsp. The audited training source contains 120,000 unique instances with job and machine counts between 2 and 20, including rectangular configurations. Observed operation durations range from 1 to 499 time units. The separate released test source contains 1,800 unique instances: 50 instances in each of the 36 groups satisfying $2 \leq |M| \leq |J| \leq 9$, with observed durations from 1 to 98. Canonical identities are derived from dimensions and the ordered job, operation, machine, and duration tuples; they do not depend on prompt wording or reference schedules. No canonical instance is shared between the training and test sources.

Reference schedules were generated with OR-Tools using a 300-second solve budget and checked for feasibility against the underlying instances. Because optimality was not certified, generated schedules are evaluated by their gaps to these feasible reference labels rather than by certified optimality gaps.

A Python shuffle with seed 42 partitions the training source into 118,000 training records and 2,000 validation records. All training records are used for one epoch.

4.2 Machine-centric descriptions and schedule format

Descriptions are produced automatically from deterministic templates. The release provides both job-centric and machine-centric descriptions; this study preserves its verbose machine-centric text. Operations are grouped by machine, and each row includes the job identifier, operation index, and duration. The operation index specifies precedence within a job: the order of rows within a machine group does not

prescribe a processing order. The model receives no reference schedule as part of its problem description.

The following operation rows describe a two-job, two-machine instance.

```
Machine 0 is used for the following Operations:
  Job 0 Operation 0 duration 3 mins.
  Job 1 Operation 1 duration 1 mins.
Machine 1 is used for the following Operations:
  Job 1 Operation 0 duration 2 mins.
  Job 0 Operation 1 duration 2 mins.
```

One feasible schedule, using the released output syntax, is

```
Solution:
Job 0 Operation 0 on Machine 0 : 0 + 3 -> 3
Job 1 Operation 0 on Machine 1 : 0 + 2 -> 2
Job 1 Operation 1 on Machine 0 : 3 + 1 -> 4
Job 0 Operation 1 on Machine 1 : 3 + 2 -> 5
Makespan: 5
```

Each output row specifies a start time, duration, and completion time.

5 Training

5.1 Model training

We fine-tune the 3.8-billion-parameter Phi-3-Mini-128K-Instruct model [6] using low-rank adaptation (LoRA), while keeping the base model frozen. LoRA targets the `qkv_proj` and `o_proj` modules in each transformer layer, with rank 128, scaling factor $\alpha = 256$, and dropout 0.05.

Training comprises one pass through the 118,000 training examples, using a micro-batch size of two and 32 gradient-accumulation steps (nominal effective batch size 64) with a learning rate of 2×10^{-4}. Validation data are used for performance monitoring.

To examine the effect of training duration, we evaluate the model at initialization and at approximately 25%, 50%, 75%, and 100% of the training run. These checkpoints come from a single training trajectory, allowing us to assess how schedule feasibility and quality change as fine-tuning progresses.

5.2 Training instructions and tokenization

For each training example, preparation chooses one of the following introductory instructions using the recorded random procedure:

`seen_1` Instruct: Provide a solution schedule for the JSSP problem below, also indicate the makespan.
`seen_2` Task: Provide the steps of a solution for the JSSP problem and determine the makespan.

`seen_3` Command: Give a detailed solution to tackle the JSSP problem, focusing on optimizing the makespan.

Checkpoint evaluations use `seen_1`. Complete training conversations use the released wrapper:

```
System: "You are expert in Job Shop Scheduling Problem"
User: " {instruction} \n{problem} "
Assistant: " {solution} "
```

Here `\n` denotes one newline, and the displayed spaces in the user and assistant strings are preserved. Conversations are serialized with the model's native chat template [18], without an additional assistant-generation prefix.

Examples are tokenized without padding and right-padded within each micro-batch using the EOS token. The causal-language-model loss is applied to system, user, and assistant tokens. The data collator assigns label -100 to EOS/padding token positions, including EOS tokens within otherwise unpadded examples. Sequences longer than 40,000 tokens are rejected rather than truncated; the maximum prepared sequence length is 19,198 tokens.

6 Evaluation

6.1 Test cohort and candidate generation

The test cohort comprises 216 instances sampled from the released 1,800-instance test set, with six instances from each of 36 job/machine-size groups.

At inference, the input format and generation settings are held constant across model checkpoints. Ten candidate schedules are generated for each test instance. All outputs are assessed as generated, without post-generation repair or replacement.

6.2 Overview of JSSP Solution Parsing and Validation

An independent parser and validator checks each candidate against the canonical instance. Validity requires every expected operation exactly once; correct machine assignments and durations; finite, nonnegative start times; consistent start-plus-duration arithmetic; nonoverlapping machine intervals; job precedence; and a unique reported makespan equal to the maximum completion time. Numerical equality and overlap checks use an absolute tolerance of 10^{-6} time units. Missing or malformed operations, extra or duplicated operations, and inconsistent makespan declarations invalidate the answer. All generated texts, including failures and token-capped answers, remain in the candidate denominator.

For a candidate set $\mathcal{S}_i$ from instance i, let $V_i \subseteq \mathcal{S}_i$ be the strictly valid schedules. Candidate feasibility is the number of valid candidates divided by all generated candidates. Instance feasibility at budget k is the proportion of instances for which the first k candidates contain at least one valid schedule. When V_i is nonempty, the best

valid makespan is

$$C_i^{\text{best}} = \min_{s \in V_i} C_{\max}(s), \qquad g_i = 100 \frac{C_i^{\text{best}} - C_i^{\text{ref}}}{C_i^{\text{ref}}}. \tag{3}$$

The reference-label gap g_i need not be nonnegative, because C_i^{ref} is not certified optimal. Mean, median, and observed maximum best-of-ten gaps are calculated over instances with at least one valid schedule and interpreted alongside instance feasibility. An observed maximum is descriptive of this test cohort.

Instance feasibility uses all 216 test instances. Rejected-candidate counts and rates use all 2,160 candidates per checkpoint. A candidate may trigger multiple error categories; each candidate or instance is counted at most once within a category, so category totals are not a disjoint partition of rejected candidates. Wilson 95% intervals summarize instance proportions. These uncertainty summaries are conditional on the fixed training run and decoding-seed scheme.

6.3 Matched neural baselines

We evaluate the released L2D policy and three released SLJ configurations [4, 5]; checkpoint files are identified by their hashes. L2D was trained for 20×20 instances with durations from 1 to 99. The SLJ configurations use $\beta = 32, 128, 256$; the $\beta = 256$ checkpoint has a different decoder-head count. Complete training manifests and seeds are unavailable, so the comparison uses the published checkpoints without retraining.

Each configuration generates ten candidates for each of the same 216 canonical instances using the original policy probabilities, without greedy sampling, early stopping, or candidate replenishment. Models are evaluated in inference mode, with L2D BatchNorm statistics frozen. Native schedules are reconstructed and checked with the shared validator. For SLJ, the uniquely determined final operation is appended after verifying the captured final decision and machine assignment.

We report feasibility over the full test population and reference gaps conditional on validity. Paired gap differences use the same 216 instances. Training data, architectures, hardware, and computational costs differ across methods, so the comparison is not resource-matched.

7 Results

7.1 Matched gap distributions on the frozen test population

Figure 1 and Table 1 summarize the best valid schedule among ten candidates for each of the same 216 test instances, comparing the fully trained Phi-3 checkpoint with four released neural baselines. All five methods are feasible on all 216 instances. Of the 2,160 Phi-3 candidates, 2,063 pass strict validation; all 2,160 baseline candidates pass.

All five methods yielded feasible schedules for every test instance, providing a common 216-instance population for paired comparisons.

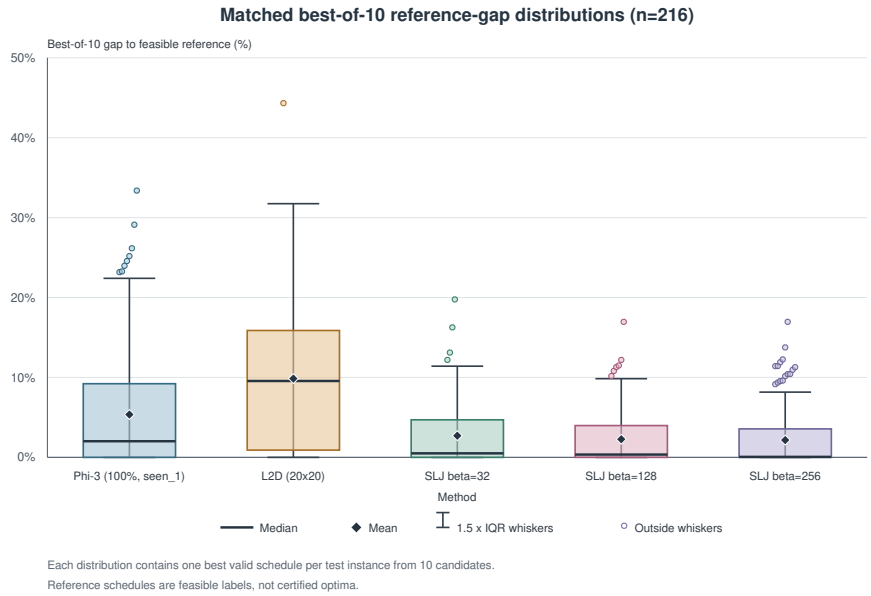


Fig. 1 Distribution of best-of-ten reference gaps on the same 216 test instances. Each box shows the interquartile range, the horizontal bar marks the median, the diamond marks the mean, and whiskers extend to the most distant observations within 1.5 interquartile ranges; points beyond the whiskers are shown individually. Every method found a feasible schedule on all 216 instances. Reference schedules are feasible labels, not certified optima.

7.2 Training-budget evaluation

Each checkpoint was evaluated with ten candidates per test instance. The training framework recorded 98.24 hours of active run time, including training, validation, and checkpointing but excluding setup and downtime between resumed processes.

Figure 2 summarizes model performance across training checkpoints.

Instance feasibility increased with training. At the final checkpoint, ten candidates yielded at least one feasible schedule for each test instance; 95.51% of individual candidates were valid, and the mean best-of-ten gap to the feasible reference schedules was 5.36%.

7.3 Paired comparison with neural schedulers

Table 2 compares the fully trained Phi-3 checkpoint, using the fixed `seen_1` instruction, with released L2D and SLJ configurations. Each method generated ten candidates for the same 216 test instances, sampled from the released test set with six instances from each of 36 job/machine-size groups. All five methods produced at least one feasible schedule for every instance. Table 1 and Figure 1 summarize their best-of-ten gap distributions.

Across the matched instances, the Phi-3 mean gap was lower than L2D's and higher than that of each SLJ configuration.

Table 1 Descriptive statistics for each method's best valid candidate among ten on the same 216 test instances. For Phi-3, "100% training" denotes the final checkpoint, and `seen_1` is the fixed instruction. The Feasible instances column counts instances with at least one valid schedule (out of 216); Valid candidates counts individual schedules that pass validation (out of 2,160). All methods yield a feasible schedule for every instance. Mean and sample standard deviation (SD), minimum, quartiles, median, and maximum are reported as reference gaps in percent; quartiles use linear interpolation.

Method	Feasible instances	Valid candidates	Mean (SD)	Min	Q_1	Median	Q_3	Max
Phi-3 (100% training)								
`seen_1`	216	2,063 (95.51%)	5.36 (7.02)	0.00	0.00	2.02	9.22	33.38
L2D (20×20)	216	2,160 (100%)	9.86 (8.54)	0.00	0.91	9.56	15.88	44.32
SLJ ($\beta = 32$)	216	2,160 (100%)	2.71 (3.66)	0.00	0.00	0.51	4.70	19.76
SLJ ($\beta = 128$)	216	2,160 (100%)	2.27 (3.19)	0.00	0.00	0.35	3.98	16.96
SLJ ($\beta = 256$)	216	2,160 (100%)	2.17 (3.32)	0.00	0.00	0.07	3.57	16.96



Fig. 2 Training-budget results on 216 test instances. (A) Instance feasibility (at least one feasible schedule among ten) and candidate validity, with denominators of 216 instances and 2,160 candidates. Both panels show the percentage of planned optimizer updates on the horizontal axis, with update counts in parentheses. (B) Mean reference gap of the best valid candidate among ten, averaged over instances with a feasible candidate; lower values are better. The initialized checkpoint yielded no feasible schedules, so its gap is undefined. Reference schedules are feasible labels, not certified optima. All checkpoints use the fixed `seen_1` instruction and one training seed.

Table 2 Paired best-of-ten reference-gap comparison on 216 instances. The mean difference is Phi-3 minus the baseline in percentage points; negative values favor Phi-3. The count triplet gives instances where Phi-3's gap is lower, equal, and higher, respectively. All methods are feasible on every instance; each baseline yields 2,160 valid candidates. Phi-3 uses the final checkpoint and fixed `seen_1` instruction.

Baseline	Baseline mean gap (%)	Phi-3 minus baseline (pp)	Phi-3 lower, equal, higher	Paired instances
L2D (20×20)	9.86	−4.50	145, 42, 29	216
SLJ ($\beta = 32$)	2.71	+2.65	39, 85, 92	216
SLJ ($\beta = 128$)	2.27	+3.09	25, 88, 103	216
SLJ ($\beta = 256$)	2.17	+3.19	27, 88, 101	216

8 Conclusion

This study evaluates direct generation of job-shop schedules by a fine-tuned language model, with feasibility checked independently of generation. On a stratified sample of 216 test instances, ten candidates from the final Phi-3 checkpoint yield at least one feasible schedule for every instance; 95.51% of individual candidates are valid, and the mean best-of-ten reference gap is 5.36%. This gap is lower than L2D's 9.86% and higher than the SLJ configurations' 2.17–2.71%. The results characterize one training run on the released test population and show that candidate validity and schedule quality should be considered together.

9 Limitations and Future Work

The study uses one fine-tuning run and one decoding-seed scheme; it therefore does not measure variability across training or sampling runs. Checkpoints follow a single learning-rate schedule, and equal candidate counts do not equalize computational cost. Model capacity and the machine-centric representation are held fixed.

Evaluation uses small instances from one generator ($2 \leq |M| \leq |J| \leq 9$) and feasible reference schedules without proven optimality. The results do not establish generalization to larger or different benchmarks, flexible or dynamic scheduling, or objectives beyond makespan. Reference gaps are conditional on valid outputs, and baseline architectures and training procedures are not controlled.

The model generates complete schedules without constraint enforcement, search, or repair; invalid outputs are excluded from schedule-quality summaries but retained in candidate-validity denominators. Further evaluation across multiple seeds, broader benchmarks and objectives, and constraint-aware hybrid methods would test the scope of these findings.

Statements and Declarations

Funding

This work received support from LAMSADE (Laboratoire d'Analyse et de Modélisation de Systèmes d'Aide à la Décision) and Université Paris Dauphine–PSL.

Competing interests

The authors declare no competing interests relevant to this work.

Ethics and consent

The study uses generated scheduling instances and does not involve human participants or animals. Ethics approval and participant consent are not applicable.

Data and code availability

The code repository for this study is available at https://github.com/starjob42/llms_can_schedule. The public scheduling dataset is available at https://github.com/starjob42/datasetjsp.

Author contributions

Henrik Abgaryan designed and conducted the research, developed the software, performed the experiments and analysis, and wrote the manuscript. Tristan Cazenave and Ararat Harutyunyan provided PhD supervision and general guidance.

References

[1] 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) https://doi.org/10.1287/ijoc.1100.0388

[2] 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

[3] Kaban, A.K., Othman, Z., Rohmah, D.S.: Comparison of dispatching rules in job-shop scheduling problem using simulation: A case study. International Journal of Simulation Modelling **11**(3), 129–140 (2012) https://doi.org/10.2507/IJSIMM11(3)2.201

[4] 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: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems, vol. 33, pp. 1621–1632. Curran Associates, Inc. (2020). https://proceedings.neurips.cc/paper_files/paper/2020/file/11958dfee29b6709f48a9ba0387a2431-Paper.pdf

[5] Corsini, A., Porrello, A., Calderara, S., Dell'Amico, M.: Self-labeling the job shop scheduling problem. In: Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., Zhang, C. (eds.) Advances in Neural Information Processing Systems, vol. 37, pp. 105528–105551. Curran Associates, Inc. (2024). https://doi.org/10.52202/079017-3349

[6] Abdin, M., Jacobs, S.A., Awan, A.A., Aneja, J., Awadallah, A., Awadalla, H., Bach, N., Bahree, A., Bakhtiari, A., Bao, J., Behl, H., Benhaim, A., Bilenko, M., Bjorck, J., Bubeck, S., Cai, Q., Cai, M., Mendes, C.C.T., Chen, W., Chaudhary, V., Chen, D., Chen, D., Chen, Y.-C., Chen, Y.-L., Chopra, P., Dai, X., Del Giorno, A., de Rosa, G., Dixon, M., Eldan, R., Fragoso, V., Iter, D., Gao, M., Gao, M., Gao, J., Garg, A., Goswami, A., Gunasekar, S., Haider, E., Hao, J., Hewett, R.J., Huynh, J., Javaheripi, M., Jin, X., Kauffmann, P., Karampatziakis, N., Kim, D., Khademi, M., Kurilenko, L., Lee, J.R., Lee, Y.T., Li, Y., Li, Y., Liang, C., Liden, L., Liu, C., Liu, M., Liu, W., Lin, E., Lin, Z., Luo, C., Madan, P., Mazzola, M., Mitra, A., Modi, H., Nguyen, A., Norick, B., Patra, B., Perez-Becker, D., Portet, T., Pryzant, R., Qin, H., Radmilac, M., Rosset, C., Roy, S., Ruwase, O., Saarikivi, O., Saied, A., Salim, A., Santacroce, M., Shah, S., Shang, N., Sharma, H., Shukla, S., Song, X., Tanaka, M., Tupini, A., Wang, X., Wang, L., Wang, C., Wang, Y., Ward, R., Wang, G., Witte, P., Wu, H., Wyatt, M., Xiao, B., Xu, C., Xu, J., Xu, W., Yadav, S., Yang, F., Yang, J., Yang, Z., Yang, Y., Yu, D., Yuan, L., Zhang, C., Zhang, C., Zhang, J., Zhang, L.L., Zhang, Y., Zhang, Y., Zhang, Y., Zhou, X.: Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone. Technical report, arXiv version 3 (23 May 2024) (2024). https://doi.org/10.48550/arXiv.2404.14219 . https://arxiv.org/abs/2404.14219v3

[7] Hu, E.J., Shen, Y., 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

[8] Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research **1**(2), 117–129 (1976) https://doi.org/10.1287/moor.1.2.117

[9] Zahmani, M.H., Atmani, B., Bekrar, A., Aissani, N.: Multiple priority dispatching rules for the job shop scheduling problem. In: 2015 3rd International Conference on Control, Engineering & Information Technology (CEIT), pp. 1–6. IEEE, Tlemcen, Algeria (2015). https://doi.org/10.1109/ceit.2015.7232991

[10] Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q.V., Zhou, D., Chen, X.: Large language models as optimizers. In: Kim, B., Yue, Y., Chaudhuri, S., Fragkiadaki, K., Khan, M., Sun, Y. (eds.) International Conference on Learning Representations, pp. 12028–12068 (2024). https://proceedings.iclr.cc/paper_files/paper/2024/file/3339f19c5fcee3ad74502947a32be9e6-Paper-Conference.pdf

[11] Jiang, X., Wu, Y., Li, M., Cao, Z., Zhang, Y.: Large language models as end-to-end combinatorial optimization solvers. In: Belgrave, D., Zhang, C., Lin, H., Pascanu, R., Koniusz, P., Ghassemi, M., Chen, N. (eds.) Advances in Neural Information Processing Systems, vol. 38, pp. 164787–164826. Curran Associates, Inc. (2025). https://doi.org/10.52202/085713-5495

[12] Jiang, C., Shu, X., Qian, H., Lu, X., Zhou, J., Zhou, A., Yu, Y.: LLMOPT: Learning to define and solve general optimization problems from scratch. In: Yue, Y., Garg, A., Peng, N., Sha, F., Yu, R. (eds.) International Conference on Learning Representations, pp. 101580–101606 (2025). https://proceedings.iclr.cc/paper_files/paper/2025/file/fbe6dd68b0cf2b1d43b458d2b8ca31b0-Paper-Conference.pdf

[13] Awasthi, P., Gollapudi, S., Kumar, R., Munagala, K.: Sample-efficient optimization over generative priors via coarse learnability. In: Advances in Neural Information Processing Systems (NeurIPS 2026) (2026). Accepted for publication; manuscript version 7, 8 October 2026, available at https://arxiv.org/abs/2503.06917v7. https://neurips.cc/virtual/2026/poster/150847

[14] 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

[15] Katz, M., Kokel, H., Srinivas, K., Sohrabi, S.: Thought of search: Planning with language models through the lens of efficiency. In: Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., Zhang, C. (eds.) Advances in Neural Information Processing Systems, vol. 37, pp. 138491–138568. Curran Associates, Inc. (2024). https://doi.org/10.52202/079017-4395

[16] Corrêa, A.B., Pereira, A.G., Seipp, J.: Classical planning with LLM-generated heuristics: Challenging the state of the art with Python code. In: Belgrave, D., Zhang, C., Lin, H., Pascanu, R., Koniusz, P., Ghassemi, M., Chen, N. (eds.) Advances in Neural Information Processing Systems, vol. 38, pp. 42070–42108. Curran Associates, Inc. (2025). https://doi.org/10.52202/085713-1401

[17] Wang, Z., Wan, C., Liu, J., Zhang, X., Wang, H., Hu, Y., Hu, Z.: MASC: Large language model-based multi-agent scheduling chain for flexible job shop scheduling problem. Advanced Engineering Informatics **67**, 103527 (2025) https://doi.org/10.1016/j.aei.2025.103527

[18] Hugging Face: Chat templates. Official online documentation. Accessed 17 October 2026. https://huggingface.co/docs/transformers/en/chat_templating