ACCORD: Autoregressive Constraint-satisfying Generation for COmbinatorial Optimization with Routing and Dynamic attention

Henrik Abgaryan
LAMSADE, University Paris Dauphine - PSL
Paris, France
henrik.abgaryan@dauphine.eu

Tristan Cazenave
LAMSADE, University Paris Dauphine - PSL
Paris, France
tristan.cazenave@lamsade.dauphine.fr

Ararat Harutyunyan
LAMSADE, University Paris Dauphine - PSL
Paris, France
ararat.harutyunyan@lamsade.dauphine.fr

Abstract

Large Language Models (LLMs) have demonstrated impressive reasoning capabilities, yet their direct application to NP-hard combinatorial problems (CPs) remains underexplored. In this work, we systematically investigate the reasoning abilities of LLMs on a variety of NP-hard combinatorial optimization tasks and introduce **ACCORD**: **A**utoregressive **C**onstraint-satisfying generation for **CO**mbinatorial optimization with **R**outing and **D**ynamic attention. ACCORD features a novel dataset representation and model architecture that leverage the autoregressive nature of LLMs to dynamically enforce feasibility constraints, coupled with attention-based routing to activate problem-specific LoRA modules. We also present the ACCORD-90k supervised dataset, covering six NP-hard combinatorial problems: TSP, VRP, Knapsack, FlowShop, JSSP, and BinPacking. Extensive experiments demonstrate that our ACCORD model, built on an 8B-parameter Llama backbone, consistently outperforms standard prompting and input-output methods, even when compared to much larger LLMs, such as gpt-4. Ablation studies further show that our output structure enhances solution feasibility. To the best of our knowledge, this is the first large-scale, end-to-end framework for exploring the applications of LLMs to a broad spectrum of combinatorial optimization problems. The codes are publicly available at [1]

1 Introduction

Large Language Models (LLMs) have rapidly established themselves as versatile engines for reasoning across a broad spectrum of tasks, encompassing arithmetic, commonsense logic , [27], [7], [6]. Among the prominent strategies enabling such capabilities is the Chain-of-Thought approach, which allows these models to decompose complex problems into sequential, interpretable steps [31].

Recent efforts have sought to adapt these reasoning techniques to address more advanced optimization tasks. Combinatorial optimization problems (CPs) are decision-making challenges where the goal is to select an optimal arrangement or subset from a large, discrete set of possibilities. Classic examples include the Traveling Salesman Problem (TSP), Vehicle Routing Problem (VRP), and Job Shop

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

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

Scheduling Problem (JSSP), which have widespread applications in logistics, manufacturing, and artificial intelligence [18]. Due to their NP-hard nature, even moderately sized instances possess a combinatorial explosion of potential solutions, rendering brute-force approaches infeasible. As a result, practical methods typically rely on heuristics or approximation algorithms to provide near-optimal solutions within reasonable time frames. As NP-hard problems, CPs present huge obstacles in practical settings [24]. Presently, the predominant paradigm in industry relies on metaheuristic algorithms—sophisticated combinations of simple, efficient heuristics—for solving CPs under various constraints. However, the success of these heuristics is often highly sensitive to the specific structure and requirements of each problem, necessitating tailored approaches for optimal results.

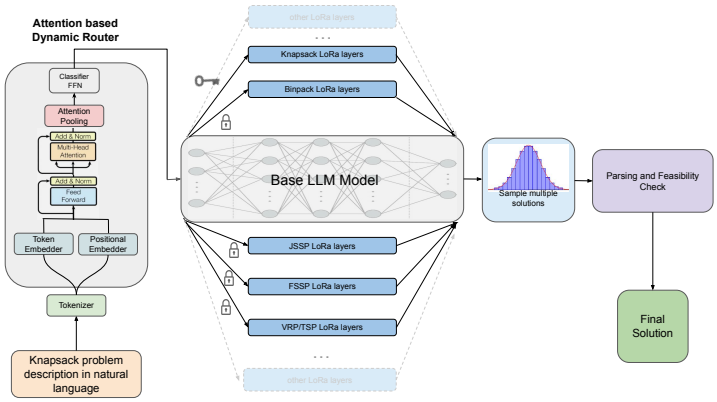


Figure 1: Overview of the ACCORD inference pipeline. As an example, a knapsack problem described in natural language is provided as input, then Attention based Dynamic router3 activates the corresponding LoRA layer specialized for knapsack tasks. Multiple candidate solutions are generated via sampling, each checked for feasibility. The best feasible solution is returned as the final output. Note that the pipeline generalizes to other combinatorial problems in the same manner; knapsack is shown here for illustration.

At the same time, investigations into leveraging LLMs for combinatorial problem solving have revealed significant research gaps. While the latest breakthroughs highlight the promise of LLMs in diverse reasoning scenarios [1], [14], [30], [34], their full potential in the context of combinatorial optimization remains largely untapped. Applying LLMs directly to these problems presents unique challenges: LLMs are trained primarily for natural language generation, not for enforcing strict combinatorial constraints, leading to issues such as hallucinations (plausible but infeasible solutions) [12], lack of optimality, and limited interpretability [28]. Furthermore, the absence of systematic search or explicit constraint mechanisms means LLM outputs can violate feasibility or fail to improve upon prior attempts. Recent advances have begun to explore the application of large language models (LLMs) to combinatorial optimization (CO). Numerous prompting-based approaches have been tested on CO tasks [32, 13, 22, 31, 35, 20, 14], demonstrating progress in solution quality and constraint handling. However, to date, there has been no comprehensive study evaluating a unified fine-tuned LLM-based framework for NP-hard CO problems across multiple domains.

We introduce **ACCORD** (**A**utoregressive **C**onstraint-satisfying generation for **CO**mbinatorial optimization with **R**outing and **D**ynamic attention), a framework for evaluating LLMs on combinatorial optimization. Our contributions are: (i) the ACCORD-90k dataset for TSP, VRP, Knapsack, FlowShop, JSSP, and BinPacking, using an autoregressive constraint-aware representation; (ii) a model architecture with attention-based dynamic routing and task-specific LoRA modules; (iii) extensive ablations showing lower optimality gaps and higher feasibility than list-of-lists and SOTA prompting (e.g., GPT-4 with Code Interpreter). ACCORD yields feasibility gains of 24.86% (FlowShop), 10% (TSP, VRP), 7% (JSSP), 4% (Knapsack), and 2% (BinPacking). This is the first large-scale, end-to-end CO framework with LLMs, opening new directions in symbolic reasoning and optimization.

Table 1: Optimality gap (%) of prompting methods (GPT-4 with code interpreter) vs. ACCORD (Llama 8B). Lower is better. N/A: no feasible solution.

Size	Method	Knapsack	BinPack	TSP	VRP	JSSP
5	IO (GPT-4)	90.1	108.2	100.3	102.0	105.3
	CoT (GPT-4)	66.9	78.2	81.2	78.2	79.4
	SR (GPT-4)	62.0	77.4	71.6	72.5	71.7
	LtM (GPT-4)	21.6	40.0	43.6	40.7	44.1
	SGE (GPT-4)	8.1	9.1	8.3	11.9	9.3
	IO (Llama 8B)	N/A	N/A	N/A	N/A	N/A
	ACCORD (Llama 8B)	**3.9***	**0.0***	**0.6***	**1.0***	**0.0***
8	IO (GPT-4)	103.5	112.8	116.9	116.3	108.2
	CoT (GPT-4)	73.8	85.1	89.0	89.5	85.2
	SR (GPT-4)	72.6	86.3	85.6	83.3	78.4
	LtM (GPT-4)	26.4	52.7	53.5	54.4	49.8
	SGE (GPT-4)	14.9	21.0	15.2	19.7	21.3
	IO (Llama 8B)	N/A	N/A	N/A	N/A	N/A
	ACCORD (Llama 8B)	**7.4***	**0.0***	**1.8***	**1.0***	**5.0***
12	IO (GPT-4)	101.5	120.7	121.6	118.5	117.6
	CoT (GPT-4)	79.3	93.8	86.8	90.1	89.3
	SR (GPT-4)	77.1	82.2	88.6	88.4	87.0
	LtM (GPT-4)	35.8	55.4	57.5	59.2	56.0
	SGE (GPT-4)	16.8	22.4	16.1	24.0	22.9
	IO (Llama 8B)	N/A	N/A	N/A	N/A	N/A
	ACCORD (Llama 8B)	**5.1***	**2.6***	**2.9***	**2.2***	**12.4***

2 Main Method: ACCORD Representation for Feasibility-Aware Solution Generation

A core challenge in applying Large Language Models (LLMs) to combinatorial optimization is the effective encoding of feasibility constraints within the generated solutions. Conventional representations, such as the "list of lists" format, provide direct encodings of solution sets, which are familiar to LLMs due to their prevalence in general-purpose data and code corpora. However, these representations are static—constraints are only checked after solution generation, offering limited guidance for incremental feasibility during the autoregressive decoding process. To address this limitation, we decided to utilize the auto-regressive nature of the LLMs and developed a representation, which is specifically designed to leverage the autoregressive generation paradigm of LLMs. Unlike the list-based format, our representation decomposes solutions into a sequence of state transitions, with each step not only specifying the next element of the solution but also explicitly updating and exposing the relevant feasibility metrics (e.g., cumulative weights, distances, machine usage, or value). This design allows the model to compute and check constraints dynamically as each token is generated, closely mimicking the typical reasoning and verification process of a human solver. ACCORD representation embeds constraint satisfaction directly into the generation process. For instance, in the Knapsack problem, each item addition is accompanied by an explicit update of the running total value and weight, immediately verifying the capacity constraint at each step:

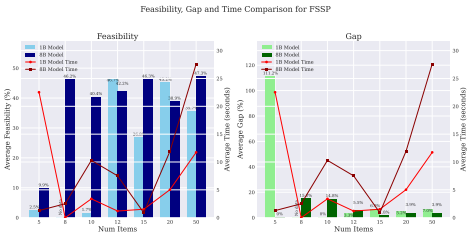
```
[[item_id, weight, value] -> value:  prev_v + value = new_v,
weight:  prev_w + weight = new_w <= capacity], ...
```

Please refer to page 1 in Appendix A for a concrete example. Similarly, for Bin Packing, the incremental assignment of items to bins is annotated with cumulative weights, ensuring that no bin exceeds its capacity as the sequence unfolds. Routing problems (VRP, TSP) and scheduling problems (JSSP) are analogously handled by tracking cumulative distances or machine times within the autoregressive output stream. Example of each of these generates is avaialable in the Appendix A. This approach transforms the constraint satisfaction problem into a stepwise process, where feasibility checks are interleaved with generation. As a result, the LLM is naturally guided away from infeasible sequences, as each decision is immediately contextualized by the current state of the solution.

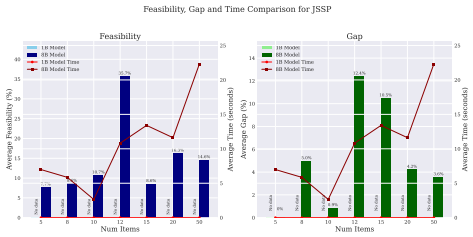
2.1 Dataset Generation

We generated synthetic datasets for several CO problems using Google OR-Tools [10], producing about 15,000 instances per task in both list-of-lists and ACCORD formats. **TSP & VRP:** Instances varied by location count ($N \in \{5, \ldots, 100\}$) and vehicles ($V \in \{1, \ldots, 10\}$), solved via 'PATH_CHEAPEST_ARC'. **Knapsack:** Varied item counts and constraints; instances with OR-Tools timeouts were discarded. **Bin Packing:** Item counts, weights, and bin limits were randomized;

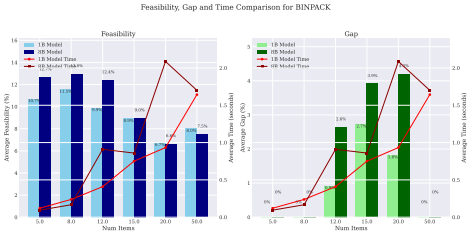
solutions minimized bin usage. **JSSP:** Instances ranged from 10×10 to 100×20 with random job sequences, solved via CP-SAT. **FSSP:** Flowshop instances up to 50×2 used the NEH heuristic [23]. See Appendix D for details.



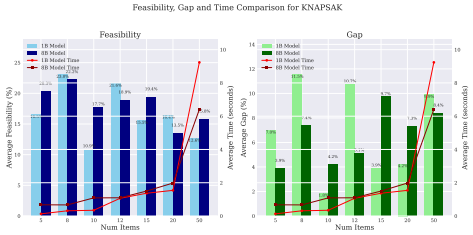
(a) Flow Shop



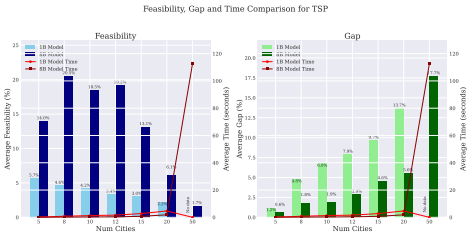
(b) JSSP



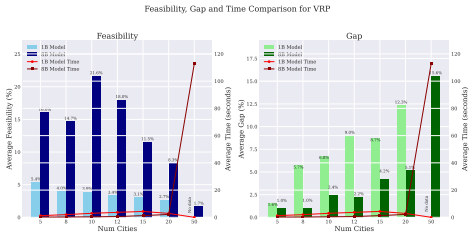
(c) BinPack



(d) Knapsack



(e) TSP



(f) VRP

Figure 2: This figure illustrates the performance of the LLama 3.1 (8B) and LLama 3.2 (1B) models in terms of the average gap percentage compared to the OR-Tools solution, where a lower gap indicates better performance. The left y-axis represents the average gap percentage, while the right y-axis corresponds to the running time in seconds. Bar plots indicate the average gap. The line plots depict the average running time per instance size, with the x-axis showing the problem size in terms of the number of nodes in the graph representation. Instances labeled as "No Data" indicate that, within a sampling budget of 60, the model failed to generate any feasible solution.

3 Model Architecture

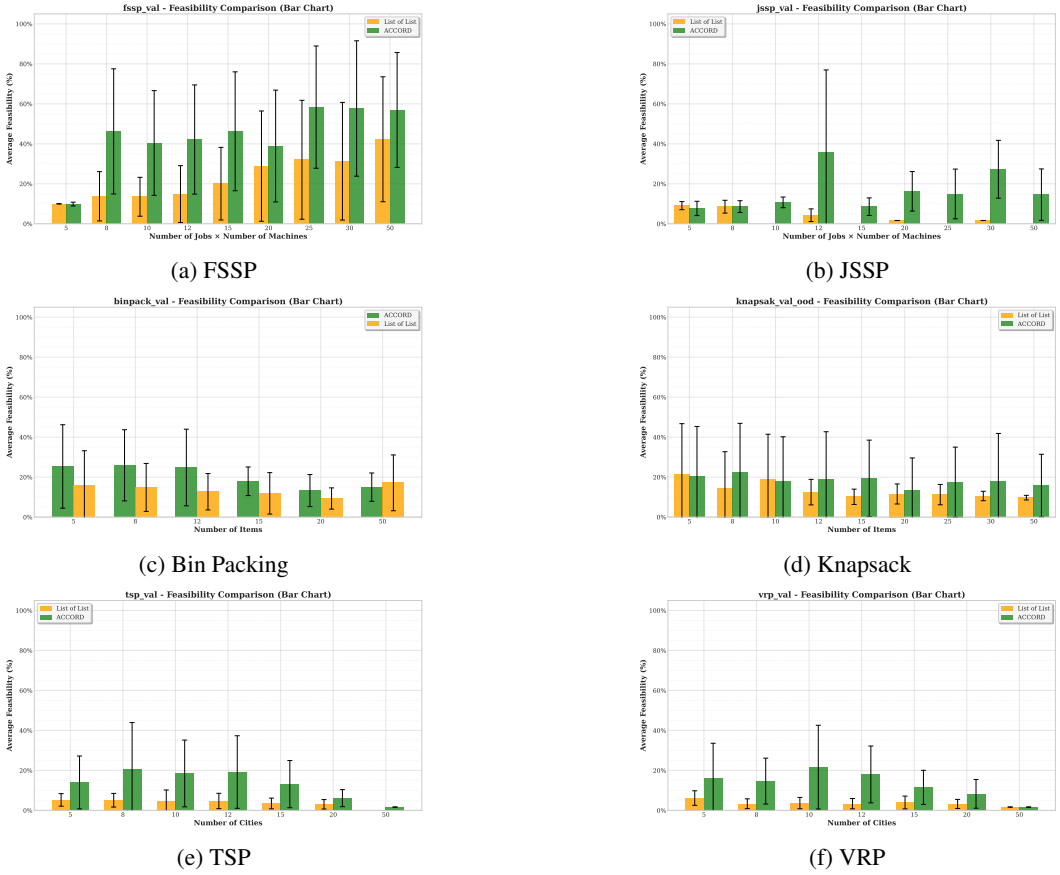
To dynamically activate the correct LoRA layers for each combinatorial optimization problem, we use an attention-based Dynamic Router TextClassifier that selects the appropriate LoRA weights based on the instruction text (see Figure 1). Our model builds on a transformer architecture, enhanced to capture problem-specific features. Each input token x_i is embedded with positional information and normalized:

$$\mathbf{E}' = \text{Dropout}(\text{LayerNorm}(\mathbf{E}_{\text{token}}(\mathbf{x}) + \mathbf{E}_{\text{pos}}(\mathbf{p}))) \tag{1}$$

The embeddings are projected to the hidden dimension and passed through several transformer layers with alternating multi-head attention and feed-forward sublayers, each followed by layer normalization. Token representations from the final transformer layer are pooled using attention-based pooling:

$$\mathbf{r} = \sum_{i=1}^{n} a_i \mathbf{h}_i \qquad \mathbf{y} = \mathbf{W}_2 \cdot \text{LayerNorm}(\text{GELU}(\mathbf{W}_1 \mathbf{r} + \mathbf{b}_1)) + \mathbf{b}_2 \tag{2}$$

Finally, the pooled vector $\mathbf{r}$ is passed through a classification head to produce logits $\mathbf{y}$ for each problem class. This architecture enables dynamic, instruction-based activation of problem-specific LoRA adapters.



(a) FSSP (b) JSSP

(c) Bin Packing (d) Knapsack

(e) TSP (f) VRP

Figure 3: Average feasibility comparison with OR-Tools solution across different problem instance sizes; the higher the feasibility percentage, the better.

3.1 Empirical Comparison with List-of-List Representation

We assess representation impact by fine-tuning LLaMA 3.1 8B on both list-of-lists and ACCORD formats using identical hyperparameters and inputs (Section 2.1). Validation uses 100 out-of-distribution instances for each $n \in \{5, 8, 10, 12, 15, 20, 25, 30, 50\}$. During inference (Fig. 1), an Attention-Based Dynamic Router (Section 3) selects the LoRA branch, generating 60 candidate solutions per instance. The best feasible solution (lowest gap) is selected, where the optimality gap is defined as $\text{Gap} = \frac{\text{Model Value} - \text{OR-Tools Value}}{\text{OR-Tools Value}}$.

where a lower gap indicates a better solution. Feasibility is measured as the percentage of generated solutions that satisfy all constraints. Our results (Fig. 3) show that, although list-of-list representation is familier to LLMs, models trained with this format tend to ignore feasibility constraints, resulting in lower feasibility rates and higher optimality gaps. In contrast, the ACCORD representation explicitly encodes feasibility into the output, enabling the LLM to produce a larger proportion of valid and near-optimal solutions, particularly as the problem size increases. Table 1 further compares our method against various prompting strategies (see Section B for baselines) on both LLama 8B and

GPT-4 with code interpreter enabled. Notably, while GPT-4 can potentially generate and execute solver code, our ACCORD-based method enables the LLM to generate solutions end-to-end without code execution. For both our approach and all prompting baselines, 60 samples per instance are generated, and the best result is selected. ACCORD consistently outperforms prompting strategies across all 6 combinatorial optimization tasks, and achieves optimal solutions on smaller instances. We also assess the impact of model size on average gap, feasibility, and inference time (Fig. 2). The 8B model mostly outperforms the 1B model in feasibility and optimality gap, with only a moderate increase in inference time. For harder instances, such as JSSP, the 1B model fails to find feasible solutions within the sampling limit. Our results demonstrate that scaling from 1B to 8B parameters yields a significant 31.5% relative improvement in solution quality, reducing the average gap from 6.54% to 4.48% (Table 2). The most substantial improvements were observed in routing problems, with TSP and VRP showing 65% and 54% relative gap reductions, respectively. Bin packing problems showed minimal sensitivity to model scale, with only a 1% improvement. In addition to our synthetic OR-Tools instances, we also evaluated ACCORD-8B on Taillard permutation flow-shop benchmarks (50 jobs × 10 machines and 50 jobs × 20 machines; avg. gap $\approx 13.7\%$) and on job-shop benchmarks TAI[26] (15 × 15 to 50 × 20; avg. gap $\approx 21.7\%$) and DMU[8] (20 × 15 to 50 × 15; avg. gap $\approx 22.1\%$) against standard heuristics (MWR/MOR/SPT) and the L2D neural scheduler (see Supplementary Material for full results and runtimes).

3.2 Relationship Between Latent Space Proximity and Solution Feasibility

We analyzed 500 TSP instances using ACCORD and list-of-lists formats to study how latent representations relate to solution feasibility. Hidden states from LLaMA 3.1 8B were reduced via PCA, and Euclidean distances between paired representations were computed. We found a significant negative correlation between latent distance and feasibility ($r = -0.1082$, $p = 0.0155$), with feasibility decreasing as distance from the ACCORD manifold increased. Despite a large performance gap (71.4% feasible for ACCORD vs. 1.6% for list-of-lists), this trend suggests LLMs encode constraint satisfaction geometrically, with latent proximity predicting solution quality.

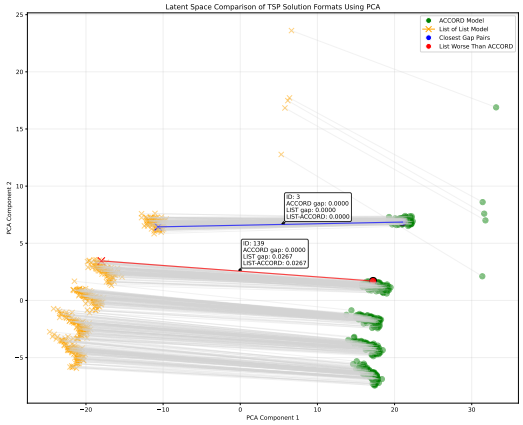


Figure 4: Latent representation distance versus solution feasibility on TSP problems, demonstrating negative correlation between distance and constraint satisfaction.

4 Conclusion, Limitations and Future Work

We introduced ACCORD, a framework that encodes combinatorial constraints into an autoregressive text format and uses dynamic LoRA routing to probe an LLM's end-to-end ability on NP-hard tasks. On six standard benchmarks (TSP, VRP, FlowShop, JSSP, Knapsack, BinPacking), an 8 B-parameter model trained with ACCORD achieves strong feasibility rates and competitive optimality gaps compared to prompting and a naïve list-of-lists format. Our goal is not to supplant specialized solvers but to map out how far small LLMs can go as self-contained combinatorial reasoners. By releasing

ACCORD and its 90K dataset, we offer a reproducible codebase for future work at the intersection of optimization and generative modeling. Despite its strong performance, ACCORD is bounded by the LLM's context window (limiting very large instances) and relies on LoRA adapters on an 8B-parameter model. In future work, we will investigate larger backbones (with full fine-tuning), expand the effective context via external memory or hierarchical encoding, and apply ACCORD to real-world, large-scale optimization scenarios.

References

[1] H. Abgaryan, T. Cazenave, and A. Harutyunyan. Starjob: Dataset for llm-driven job shop scheduling. In *ArXiv Preprint arXiv:2503.01877v1*, 2024.

[2] James Adams, Elias Balas, and David Zawack. Shifting bottleneck procedures for job shop scheduling. In *Management Science*, volume 34, pages 391–401. INFORMS, 1988.

[3] Meta AI. Llama 3 model card, 2024. Accessed: 2024-08-10.

[4] Unsloth AI. Unsloth: Accelerated fine-tuning for large language models, 2024. Accessed: 2024-11-19.

[5] Pranjal Awasthi, Sreenivas Gollapudi, Ravi Kumar, and Kamesh Munagala. Combinatorial optimization via llm-driven iterated fine-tuning. *arXiv preprint arXiv:2503.06917*, 2025.

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[7] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

[8] Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, 1998.

[9] M Goel et al. Genetic algorithms in scheduling. In *International Conference on Genetic Algorithms*, 1996.

[10] Google. Google's or-tools. `https://developers.google.com/optimization/`. Accessed: 2024-05-07.

[11] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[12] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Improving the reasoning capabilities of large language models in complex tasks. In *Proceedings of the International Conference on Machine Learning*, 2022.

[13] W. Huang et al. Large language models for vehicle routing: A prompting-based approach. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2024.

[14] Zangir Iklassov, Yali Du, Farkhad Akimov, and Martin Takáč. Self-guiding exploration for combinatorial problems. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, 2024.

[15] Damjan Kalajdzievski. A rank stabilization scaling factor for fine-tuning with lora, 2023.

[16] Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, 2017.

[17] Wouter Kool, Holger van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

[18] Jan K Lenstra, A Rinnooy Kan, and P Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1979.

[19] Shengcai Liu, Caishun Chen, Xinghua Qu, Ke Tang, and Yew-Soon Ong. Large language models as evolutionary optimizers. *arXiv preprint arXiv:2310.19046*, 2023.

[20] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

[21] Mahmoud Masoud, Ahmed Abdelhay, and Mohammed Elhenawy. Exploring combinatorial problem solving with large language models: A case study on the traveling salesman problem using gpt-3.5 turbo. *arXiv preprint arXiv:2405.01997*, 2024.

[22] Chinmay Mittal, Krishna Kartik, Mausam, and Parag Singla. Puzzlebench: Can llms solve challenging first-order combinatorial reasoning problems? *arXiv preprint arXiv:2402.02611*, 2024.

[23] Muhammad Nawaz, E. Emory Enscore, and Inyong Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

[24] Afshin Oroojlooyjadid, Lawrence V Snyder, and Martin Takáč. Applying deep learning to the newsvendor problem. *Iise Transactions*, 52(4):444–463, 2020.

[25] R Roy and G Sussmann. Machine scheduling by mathematical programming. In *Journal of the Operational Research Society*, volume 15, pages 352–362. JORS, 1964.

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

[27] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

[28] K. Valmeekam, A. Olmo, S. Sreedharan, and S. Kambhampati. A benchmark for evaluating planning and reasoning in large language models. In *NeurIPS Foundation Models for Decision Making Workshop*, 2022.

[29] Fang Wan, Julien Fondrevelle, Tao Wang, Kezhi Wang, and Antoine Duclos. Optimizing small-scale surgery scheduling with large language model. In *Proceedings of the 21st International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, pages 223–228, Lisbon, Portugal, 2024.

[30] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[31] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[32] C. Yang, X. Wang, Y. Lu, H. Liu, Q.V. Le, and X. Chen. Optimization by prompting: Leveraging large language models for combinatorial optimization. *arXiv preprint arXiv:2309.03409*, 2023.

[33] C. Zhang, W. Song, Z. Cao, J. Zhang, P.S. Tan, and C. Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

[34] Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*, 2022.

[35] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

A Technical Appendices and Supplementary Material

B Related Work

B.1 Heuristic and Machine Learning Approaches on CO problems

Combinatorial optimization has been tackled with both heuristic and exact methods. Simple priority dispatching rules (PDRs), such as shortest processing time or earliest due date, are computationally efficient but often yield suboptimal solutions due to their greedy nature [18]. Metaheuristics (e.g., simulated annealing, tabu search, genetic algorithms) offer improved solution quality, and exact approaches like the shifting bottleneck procedure [2], mixed-integer programming, and constraint programming can find optimal solutions for small instances, though at high computational cost [25, 9]. Recently, machine learning, particularly deep reinforcement learning (RL) and graph neural networks (GNNs) have advanced combinatorial optimization [33, 16, 17]. RL methods treat scheduling as sequential decision making, learning dispatching policies via environment interaction [33]. GNNs encode jobs and machines as nodes, enabling permutation-invariant representations and, when combined with RL, can model complex dependencies [16]. Attention-based and sequence-to-sequence models further enhance performance on tasks like TSP and VRP, often utilizing iterative refinement [17].

B.2 Large Language Models in Combinatorial Optimization

The advent of LLMs has introduced new paradigms for CO. Early work explored whether LLMs could generate solutions through prompting [32], [13], [22], [31] [35], [20], [14]. Prompting-based strategies, such as OPRO, involve iterative refinement based on feedback, while methods for VRP employ self-debugging and verification to enhance feasibility [13]. However, scalability remains a challenge, as even strong prompting techniques struggle on larger or more complex instances [22]. Recent research has explored a variety of prompting strategies to leverage LLMs for solving combinatorial optimization (CO) problems. The **Input-Output (IO)** method presents the LLM with multiple examples of input and corresponding output solution pairs. The LLM is then prompted to generate an output solution in the same format as the provided examples. This approach relies on the LLM's ability to generalize the mapping from input to output based on observed patterns. In **Chain-of-Thought (CoT)** prompting, the LLM is guided to produce a sequence of intermediate reasoning steps, or "thoughts," before arriving at the final answer [31]. This technique encourages the model to break down complex CO tasks into structured, stepwise reasoning, improving both transparency and solution quality. **Least-to-Most (LtM)** prompting strategy aims to decompose a complex problem into a sequence of simpler subproblems, solving them incrementally [35]. Each subproblem builds upon the solutions of previous ones, enabling the LLM to tackle challenging CO tasks through a series of manageable steps. **Self-Refinement (SR)** is an iterative prompting technique wherein the LLM first generates an initial solution, then provides feedback on its own output, and finally refines the solution based on this feedback [20]. The process repeats until a satisfactory solution is reached. **Self-Guiding Exploration for Combinatorial Problems (SGE)** autonomously generates multiple thought trajectories for a given CO task [14]. Each trajectory represents a distinct heuristic approach, inspired by metaheuristics. SGE decomposes these trajectories into actionable subtasks, executes them sequentially, and refines the results to ensure optimal solutions. Fine-tuning LLMs for CO tasks is another active area [1],[21] . [1] showed that fine-tuned LLM on job-shop scheduling, demonstrates significant improvements in solution quality. Similarly, [21] applied fine-tuning to TSP instances with promising but size-limited results. Hybrid methods integrate LLMs into evolutionary or search frameworks, where the LLM guides genetic operations or receives feedback from constraint solvers to iteratively improve solutions [19, 29, 5]. While promising, these approaches often entail significant computational overhead and still face scaling hurdles.

C Preliminaries: Overview of Classic Combinatorial Optimization Problems

In this section, we introduce several foundational combinatorial optimization problems, explaining their goals and constraints in accessible terms while also providing their standard mathematical formulations. **General Combinatorial Optimization Problem** Combinatorial optimization involves searching for the best solution from a finite set of possibilities. Formally, given a set of feasible

solutions $\mathcal{S}$ and an objective function $f : \mathcal{S} \to \mathbb{R}$, the goal is to find

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

or, in some cases, to maximize $f(s)$ depending on the problem.

Traveling Salesman Problem (TSP) Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the starting point. Mathematically, for n cities $V = \{1, 2, \ldots, n\}$ and a distance matrix $D \in \mathbb{R}^{n \times n}$, we seek a tour (a permutation π of all cities) that minimizes the total travel distance, where $\pi(n+1) = \pi(1)$ to ensure the tour closes:

$$\min_{\pi \in \mathcal{P}_n} \sum_{i=1}^{n} D_{\pi(i), \pi(i+1)}$$

Vehicle Routing Problem (VRP) The VRP extends the TSP to multiple vehicles. Given a depot, n customers (with demands q_i), and a fleet of vehicles each with capacity Q, the goal is to design routes—each starting and ending at the depot—so that every customer is visited exactly once, no vehicle exceeds its capacity, and the total travel distance is minimized:

$$\min \sum_{k=1}^{m} \sum_{j=0}^{\ell_k} D_{v_j^k, v_{j+1}^k}$$

subject to

$$\bigcup_{k=1}^{m} \{v_1^k, \ldots, v_{\ell_k}^k\} = V \quad \text{(All customers served)}$$

$$\sum_{j=1}^{\ell_k} q_{v_j^k} \leq Q \quad \forall k \quad \text{(Capacity constraint)}$$

Job Shop Scheduling Problem (JSSP) JSSP schedules n jobs, each as a sequence of operations on specific machines. Each operation $O_{j,k}$ requires machine $M_{j,k}$ for $p_{j,k}$ time units, following job order. Let $S_{j,k}$ and $C_{j,k}$ be the start and completion times. The objective is to minimize makespan:

$$\min C_{\max} = \max_j C_{j,\ell_j}$$

subject to:

$$\begin{aligned}
&\text{(Precedence)} \quad S_{j,k+1} \geq C_{j,k} \\
&\text{(No machine conflicts)} \quad S_{j,k} \geq C_{j',k'} \text{ or } S_{j',k'} \geq C_{j,k}, \\
&\qquad \forall (j,k) \neq (j',k') \text{ with } M_{j,k} = M_{j',k'}
\end{aligned}$$

Knapsack Problem (KP) Given a set of items, each with a value and weight, what is the most valuable combination of items you can carry without exceeding the weight limit of your knapsack. With n items (weights w_i, values v_i) and capacity W, choose $x_i \in \{0, 1\}$ (item picked or not) to solve:

$$\max \sum_{i=1}^{n} v_i x_i \qquad \text{s.t.} \qquad \sum_{i=1}^{n} w_i x_i \leq W$$

Bin Packing Problem (BPP) Given a set of items of varying sizes, how can you pack them into the fewest number of fixed-size bins. For n items of sizes $s_i \in (0, 1]$, assign them to bins of capacity 1 so as to minimize the total number of bins K:

$$\min K$$

subject to:

$$\sum_{i \in B_k} s_i \leq 1 \quad \forall k, \qquad \bigcup_{k=1}^{K} B_k = \{1, \ldots, n\}, \qquad B_k \cap B_{k'} = \emptyset \; \forall k \neq k'$$

where B_k is the set of items in bin k.

Flow Shop Scheduling Problem (FSSP)

We consider the permutation flow shop scheduling problem (FSSP), where all jobs are processed in the same order on each of m machines. Given n jobs $J = \{1, 2, \ldots, n\}$ and processing times $p_{j,k}$ for job j on machine k, the goal is to find a job sequence π that minimizes the makespan.

Let $C_{j,k}$ denote the completion time of job j on machine k. The constraints are:

$$\text{(Machine order)} \quad C_{\pi(i),k} \geq C_{\pi(i),k-1} + p_{\pi(i),k} \quad \forall i = 1, \ldots, n,\ k = 2, \ldots, m$$
$$\text{(Job sequence)} \quad C_{\pi(i),k} \geq C_{\pi(i-1),k} + p_{\pi(i),k} \quad \forall i = 2, \ldots, n,\ k = 1, \ldots, m$$

The objective is to find the permutation π^* that minimizes the makespan:

$$\min_{\pi \in \mathcal{P}_n} C_{\max} = \min_{\pi \in \mathcal{P}_n} \max_i C_{\pi(i),m}$$

where $C_{\pi(i),m}$ is the completion time of job $\pi(i)$ on the last machine.

Below, we provide representative examples from each of the major combinatorial optimization problems in our dataset. Each example is shown in both the standard "list of list" representation and the "Accord" (arrow-based) representation.

Knapsack Problem

Instruction: You are given a paired representation (value, weight): Find a set of items to pack into a container with a maximum weight capacity $= 20$ that maximizes total value of packed items.

- **Input:**

$$\begin{bmatrix} 6 & 5 \\ 10 & 8 \\ 7 & 4 \\ 6 & 2 \\ 1 & 1 \end{bmatrix}$$

- **Output (List of List):**

```
Solution: [(6, 5), (10, 8), (7, 4), (6, 2), (1, 1)]
  Value: 6+10+7+6+1=30
  Weight: 5+8+4+2+1=20<=20
```

- **Output (Accord):**

```
Solution:
[[6, 5] -> value:0+6=6, weight:0+5=5<=20],
[[10, 8] -> value:6+10=16, weight:5+8=13<=20],
[[7, 4] -> value:16+7=23, weight:13+4=17<=20],
[[6, 2] -> value:23+6=29, weight:17+2=19<=20],
[[1, 1] -> value:29+1=30, weight:19+1=20<=20]

Total Value: 30
Total Weight: 20<=20
```

Bin Packing Problem

Instruction: Given a list of items (id, weight), determine the minimum number of bins (capacity=77) needed to pack all items without exceeding the capacity.

- **Input:**

$$[(0, 17), (1, 24), (2, 19), (3, 11), (4, 27)]$$

- **Output (List of List):**

```
The minimum number of bins required is 2. The bin assignments are: [[0, 1, 2, 3], [4]].
```

- **Output (Accord):**

```
Bin 1:
(0, 17)->17 (1, 24)->41 (2, 19)->60 (3, 11)->71<=77
Bin 2:
(4, 27)->27<=77
Total bins required: 2
```

Vehicle Routing Problem (VRP)

Instruction: Given customers with coordinates and a depot, and multiple vehicles of capacity 85, find the minimum-length routes serving all customers.

- **Coords:** 0:(34, 42), 1:(39, 58), 2:(46, 48), 3:(57, 49), 4:(45, 16)
- **Output (List of List):**

```
[(0): (34, 42), (0): (34, 42)]
[(0): (34, 42), (0): (34, 42)]
[(0): (34, 42), (0): (34, 42)]
[(0): (34, 42), (0): (34, 42)]
[(0): (34, 42), (1): (39, 58), (2): (46, 48),
(3): (57, 49), (4): (45, 16), (0): (34, 42)]
Overall Total Distance: 102
```

- **Output (Accord):**

```
Vehicle Route: (0): (34, 42) -> (0): (34, 42) + 0
Vehicle Route: (0): (34, 42) -> (0): (34, 42) + 0
Vehicle Route: (0): (34, 42) -> (0): (34, 42) + 0
Vehicle Route: (0): (34, 42) -> (0): (34, 42) + 0
Vehicle Route: (0): (34, 42) -> (1): (39, 58) + 16 -> (2): (46, 48) + 12 ->
(3): (57, 49) + 11 -> (4): (45, 16) + 35 -> (0): (34, 42) + 28
Overall Total Distance: 102
```

Traveling Salesman Problem (TSP)

Instruction: Given customers with coordinates and a depot, and 1 vehicle, find the minimum-length route serving all customers.

- **Coords:** 0:(17, 22), 1:(63, 8), 2:(22, 60), 3:(3, 29), 4:(7, 12)
- **Output (List of List):**

```
[(0): (17, 22), (4): (7, 12), (3): (3, 29), (2): (22, 60), (1): (63, 8), (0): (17, 22)]
Overall Total Distance: 181
```

- **Output (Accord):**

```
Vehicle Route: (0): (17, 22) -> (4): (7, 12) + 14 -> (3): (3, 29) + 17 ->
(2): (22, 60) + 36 -> (1): (63, 8) + 66 -> (0): (17, 22) + 48
Overall Total Distance: 181
```

Job Shop Scheduling Problem (JSSP)

Instruction: Optimize schedule for 2 Jobs (J) across 6 Machines (M) to minimize makespan. Each M can process only one J at a time, and once started, J cannot be interrupted.

- **Input:**

```
J0:
M2:205 M1:157 M0:198 M5:79 M3:110 M4:32
J1:
M3:179 M4:108 M2:82 M5:112 M1:136 M0:27
```

- **Output (List of List):**

```
[[0, 2, 0, 205], [1, 3, 0, 179], [1, 4, 179, 108], [0, 1, 205, 157], [1, 2, 287, 82],
[0, 0, 362, 198], [1, 5, 369, 112], [1, 1, 481, 136], [0, 5, 560, 79], [1, 0, 617, 27],
[0, 3, 639, 110], [0, 4, 749, 32]]
Maximum end completion time or Makespan: 781
```

- **Output (Accord):**

```
Solution:
J0-M2: 0+205 -> 205,
J1-M3: 0+179 -> 179,
J1-M4: 179+108 -> 287,
J0-M1: 205+157 -> 362,
J1-M2: 287+82 -> 369,
J0-M0: 362+198 -> 560,
J1-M5: 369+112 -> 481,
J1-M1: 481+136 -> 617,
J0-M5: 560+79 -> 639,
J1-M0: 617+27 -> 644,
J0-M3: 639+110 -> 749,
J0-M4: 749+32 -> 781,
Maximum end completion time or Makespan: 781
```

Flow Shop Scheduling Problem (FSSP)

Input:

```
J1:
M1:12 M2:7
J2:
M1:8 M2:4
J3:
M1:4 M2:15
J4:
M1:5 M2:9
```

Output (List of List):

```
[[3, 1, 0, 4], [3, 2, 4, 15], [2, 1, 4, 8], [4, 1, 12, 5],
[1, 1, 17, 12], [2, 2, 19, 4], [4, 2, 23, 9], [1, 2, 32, 7]]
Maximum end completion time or Makespan: 39
```

Output (Accord):

```
J3: M1(0+4=4) -> M2(4+15=19)
J2: M1(4+8=12) -> M2(19+4=23)
J4: M1(12+5=17) -> M2(23+9=32)
J1: M1(17+12=29) -> M2(32+7=39)

Maximum end completion time or Makespan: 39
```

D Training Details

We conducted supervised fine-tuning using input-output pairs for two models from Meta: Llama 3.1 8B and Llama 3.2 1B. To minimize memory usage during training, we employed 4-bit quantized versions of these models and trained each for 2 epochs. For a fair comparison, we fine-tuned each model with the same hyperparameters, varying only the output representation: once using the list-of-lists format and once using the ACCORD format, while keeping the input and all other hyperparameters identical. We used Rank-Stabilized Low-Rank Adaptation (RSLoRA) [15] with a rank of $r = 64$ and $\alpha = 64$. The two epochs, training required roughly 40 hours and about 30GB of GPU memory on Nvdidia RTX A6000 GPU. We limited the context length of the model to 40k instead of the original 128k, 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. More training details and curves are available in D.

Training details

The model being fine-tuned is LLaMA 3.1, an 8 billion parameter model from Meta[3], using a 4-bit quantized version to reduce memory usage. Finetning was conducted using Stabilized Low-Rank Adaptation (RsLoRA) [15] with rank $r = 64$ to introduce learnable parameters specifically in targeted layers. [15] Compared to Lora[11] RsLoRa improves the stability of training by modifying the rank during adaptation[15]. The target modules include:

$$\text{target\_modules} = \{\texttt{q\_proj}, \texttt{k\_proj}, \texttt{v\_proj}, \texttt{o\_proj}, \texttt{gate\_proj}, \texttt{up\_proj}, \texttt{down\_proj}\} \quad (3)$$

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` [4] method, to fit longer sequences by saving memory. This reduces the VRAM usage by approximately 30%, enabling larger batch sizes.

Table 2: The effect of the model size on Average Gap (%): Comparison Across CO Problems

Problem	1B Model	8B Model
BINPACK	1.01%	1.00%
FSSP	7.92%	7.17%
JSSP	N/A	6.08%
KNAPSAK	5.90%	5.33%
TSP	8.11%	2.84%
VRP	9.74%	4.50%
AVERAGE	6.54%	4.48%