EHOP: A Dataset of Everyday NP-hard Optimization Problems

Anonymous ACL submission

Abstract

We introduce the dataset of Everyday Hard Optimization Problems (EHOP), a collection of NP-hard optimization problems expressed in natural language. EHOP includes problem formulations that could be found in computer science textbooks, versions that are dressed up as problems that could arise in real life, and variants of well-known problems with inverted rules. We find that state-of-the-art LLMs, across multiple prompting strategies, systematically solve textbook problems more accurately than their real-life and inverted counterparts. We argue that this constitutes evidence that LLMs adapt solutions seen during training, rather than leveraging reasoning abilities that would enable them to generalize to novel problems.

1 Introduction

Many real-world tasks that people face in their personal and professional lives are NP-hard optimization problems. Such problems are as diverse as planning family vacations, scheduling airline crews (Gopalakrishnan and Johnson, 2005), and allocating organ donations (Abraham et al., 2007). People rarely enjoy solving these problems, and they are not particularly good at solving them either (Hidalgo-Herrero et al., 2013).

One of the most exciting promises of large language models (LLMs) is that they can help non-experts solve their real-world computational problems when they express them in natural language (NL). The hope is that a wide range of users across a wide range of tasks will be able to describe their problem to the LLM, and the LLM will handle the difficult task of "problem solving," i.e., recognizing that the real-world problem can be described in terms of a known computational problem and then solving that problem efficiently and optimally. In the case of NP-hard problems, this could potentially be accomplished with the LLM solving the problem by itself, e.g., through chain-of-thought (CoT) reasoning (Fan et al., 2024), or the LLM could convert the NL description into a linear program (LP) and solve it with specialized tools (AhmadiTeshnizi et al., 2024).

Textbook: Given an undirected graph G, color its nodes such that no two adjacent nodes have the same color. Use as few colors as possible.
Costumed (💔 Parties with Exes): Your birthday is coming up, and you want to celebrate with all your friends. You do not want people who used to be in a relationship at the same party. How many parties do you need, and who should be invited to which party?
Inverted: Given an undirected graph G, color its nodes such that no two *non-adjacent* nodes have the same color. Use as few colors as possible.

Figure 1: Three variants of Graph Coloring in EHOP.

However, recent work has raised the question of "reasoning vs. reciting": are LLMs actually carrying out systematic problem-solving, or are they simply adapting solutions for similar problems in their training data (Mirzadeh et al., 2024; Wu et al., 2024)? LLMs that can only solve problems whose solution paths are documented on the Internet will not fulfill the promise of opening general problem-solving to lay users.

In this paper, we contribute to the reasoning vs. reciting debate by introducing the dataset of Everyday Hard Optimization Problems (EHOP), which consists of NP-hard optimization problems presented in both textbook and real-world variants. EHOP is based on three well-studied problems (Graph Coloring, Knapsack, and Traveling Salesman). We "dress up" the instances of each problem with three different *cos-*

tumes (see Figure 1 for an example) that represent real-world situations which require solving the underlying problem. Furthermore, we add *inverted* variants of all problems, which fundamentally distort the solutions of the problem with a small change in problem formulation. If LLMs perform reasoning, they should solve textbook, inverted, and costumed problems at similar levels of accuracy. If they recite, we would expect textbook problems, for which solution mechanisms are presented explicitly on the Internet, to be easier.

In our experiments on EHOP, we find that while both GPT-4o (OpenAI, 2024) and Llama 3.1 70B Instruct (Grattafiori et al., 2024) solve small textbook instances quite well through CoT reasoning, the proportion of textbook problems they solve optimally is substantially higher than for the inverted and costumed variants, often by more than 20 percentage points. When we use these LLMs to convert problems into LPs and solve the LPs with a standalone tool, accuracy on textbook problems is even higher and scales much better to larger instance sizes, but the vulnerability to inversion and costuming persists. This is evidence that LLMs draw their apparent problem-solving capabilities from an ability to adapt solutions seen in training and struggle to generalize to novel problems.

We will make EHOP publically available upon acceptance.

2 Related Work

LLMs have been shown to perform remarkably well on benchmarks for complex problem-solving tasks, such as tool use (Yao et al., 2023), complex gameplay (Wang et al., 2023), and AI planning (Stein et al., 2024). This has been attributed to the ability of iterative prompting strategies such as chain-of-thought (Kojima et al., 2022; Wei et al., 2022) to perform general reasoning and problem solving.

However, recent work has raised the question of whether LLMs actually perform systematic reasoning, or whether they are "reciting" solution paths from their training data by adapting them gracefully to the inference-time problem (Wu et al., 2024; Kambhampati, 2024). The fact that LLM reasoners often degrade in accuracy for larger problem instances is one piece of evidence for the recitation hypothesis. Furthermore, as long as chains of thought are limited to a polynomial number of steps, transformers provably solve exactly the problems that can be solved in polynomial time (Merrill and Sabharwal, 2024), which fails to cover most reasoning problems, for which no optimal polynomial algorithms are known.

In this paper, we focus on NP-hard optimization problems, with particular attention to the difference between textbook and everyday problems. The performance of LLMs on NP-hard problems has been investigated in a number of recent studies. NPHardEval (Fan et al., 2024) looks only at textbook problems, including the three base problems we consider here. GraphArena (Tang et al., 2024) evaluates LLMs on NP-hard graph problems on a variety of large real-world graphs, and is also limited to textbook problems. NL4Opt (Ramamonjison et al., 2022) and NLP4LP (AhmadiTeshnizi et al., 2024) provide evaluation datasets on real-world NP-hard problems, but they are not linked to the underlying textbook problems. EHOP differs from all these datasets in that we present the exact same instances of the base problem both in textbook and real-world variants, making it possible for the first time to measure the impact of this distinction.

3 Everyday optimization problems

An optimization problem is called *NP-hard* if every problem that can be solved in non-deterministic polynomial time can be reduced to it in polynomial time (Garey and Johnson, 1979). While it is generally assumed that deterministic algorithms that solve NP-hard problems must have worst-case exponential runtime, problems in NP are still of lower computational complexity than, e.g., planning or reasoning. In this paper, we focus on three well-known NP-hard optimization problems: GRAPH COLORING, KNAPSACK, and TRAVELING SALESMAN.

To construct EHOP, we first generate a number of random instances for each of the three base problems. Instances are concrete examples of a problem; for example, an instance of the GRAPH COLORING example in Figure 1 consists of a specific graph G. We present each instance in its Textbook form; in addition, we dress it up in real-world *costumes* and *invert* it. This yields a total of eight *variants* of each instance. Table 5 to Table 7 in the appendix show examples of all variants.

Not all instances of an NP-hard problem are equally difficult. We therefore take special care to ensure that experimental results remain comparable across variants, especially when we invert the problems.

3.1 🖌️ Graph Coloring

An instance of the Graph Coloring problem consists of an undirected graph $G = (V, E)$. The task is to assign each node a color such that no two adjacent nodes have the same color, while using the fewest colors possible.

Inverted Graph Coloring asks for color assignments in which no two *non-adjacent* nodes have the same color. For each instance G of the base problem, we take the complement of G as an instance of the inverted problem; it has an edge between two nodes if and only if there is no edge between them in G. Thus, the same coloring will solve the inverted problem on the inverted instance, ensuring identical difficulty.

In addition to the 🖌️ **Textbook** variant, we have constructed three costumes that are not obviously about graph coloring:

🏫 **Student Groups.** V represents a set of students, and E represents friendships. A teacher wants to assign students to as few groups as possible, while ensuring that no student is distracted by a groupmate who is also a friend.

💔 **Parties with Exes.** V represents a person's set of friends, and E represents which friends used to be in a romantic relationship with each other. This person wants to celebrate their birthday with their friends while avoiding any awkwardness arising from exes being at the same party. They want to minimize the number of parties they have to plan.

🥋 **Taekwondo Tournament.** V represents participants in a Taekwondo tournament, and E represents which participants will be fighting one another in the tournament. The tournament organizer wants to assign participants to warm-up rooms without giving opponents the chance to study each other in advance of the competition.

3.2 🎒 Knapsack

An instance of the Knapsack problem consists of a knapsack with some capacity $C \in \mathbb{N}$ and a set of items with weights $w_1, \ldots, w_n \in \mathbb{N}$ and values $v_1, \ldots, v_n \in \mathbb{N}$. The task is to find a subset of items that maximizes the sum of the values of these items, under the constraint that their total weight must not be greater than C.

In inverted Knapsack, the task is to *minimize* the selected items' total value, with the constraint that the items' total weight must be *at least* C. For each instance of the base problem, we construct an instance of the inverted problem by setting the knapsack capacity to $\sum w_i - C$. Then the optimal solution of the inverted instance consists of exactly the items that were *left out* of the knapsack in the original instance, ensuring equal difficulty.

We have constructed the following costumes:

🍋 **Lemonade Stand.** We have C liters of lemonade to sell at our lemonade stand and would like to sell it for as much money as possible. Each of our n customers offers to pay a price v_i for w_i liters of lemonade.

🏛️ **Sightseeing.** We have C hours to spend in Paris and would like to visit attractions that give us maximal total satisfaction. Each of the n possible attractions will give us some satisfaction v_i and take some time w_i to visit.

🎉 **Party Planning.** We have a decoration budget C for the party we are planning, and we wish to maximize the total coolness of our party. Each potential decoration item has a coolness score of v_i and a price of w_i.

3.3 ✈️ Traveling Salesman

An instance of the Traveling Salesman problem consists of a set $C = \{1, \ldots, n\}$ of cities, and for any pair of cities, we have a distance $d(i, j) \in \mathbb{N}$. The task is to find the shortest round trip that visits all the cities. That is, we are looking for a permutation $\pi : C \to C$ that minimizes

$$d(\pi_n, \pi_1) + \sum_{i=1}^{n-1} d(\pi_i, \pi_{i+1}).$$

Inverted Traveling Salesman changes the goal to *maximizing* the sum of the distances rather than minimizing it. For each instance of the base problem, we construct an instance of the inverted problem by converting each distance $d(i, j)$ to $m - d(i, j) + s$, where $m = \max d(i, j)$. We sample a random shift $s \in \{1, \ldots, n\}$ for each instance to maintain some variety of edge weights. This construction ensures that the optimal solutions of an instance and its inverted instance are the same.

We have constructed the following costumes:

📋 **Task Schedule.** C represents a set of tasks that have to be done daily, and d represents the time it takes to modify one's workspace to transition between tasks. Note that the transition from one day to the next captures the term $d(\pi_n, \pi_1)$.

Exercise Schedule. As their New Year's resolution, a person will do a physical activity from a set C every day, never repeating until they've exhausted the set, after which they will go through it again in the same order as before. They want to maximize the day-to-day variety of their activities by minimizing the similarity score d between adjacent activities.

UN Seating. A staff member at the United Nations needs to figure out how to seat the representatives C from various countries around a circular table. They want to minimize the total political tension d between adjacent representatives.

4 Experiments

4.1 Dataset

The EHOP dataset consists of two parts: EHOP-RANDOM and EHOP-HARD. Each of these two sub-datasets consists of 150 distinct instances of each of the three base problems (100 for GRAPH COLORING in EHOP-HARD, see below), presented in each of the eight variants (Textbook and three costumes $\times$ standard/inverted). In total, EHOP has 6800 NL task descriptions.

To create EHOP-RANDOM, we randomly sampled 25 instances of each base problem for six different instance sizes: for GRAPH COLORING and TRAVELING SALESMAN, we generated instances with 4, 5, 6, 7, 8, and 9 nodes/cities, and for KNAPSACK, we generated instances with 4, 8, 12, 16, 20, and 24 items. We chose these scales to represent a spectrum of difficulties ranging from easy to hard. We determined the optimal solution for each instance with an optimal solver.[1]

EHOP-HARD contains similar instances that are less vulnerable to being solved by greedy heuristics; we will explain it in detail in Section 5.3.

4.2 Models and Prompting

We evaluated GPT-4o (`gpt-4o-2024-08-06`) and Llama-3.1-70B Instruct on EHOP (see Appendix A for model details). For each LLM, we evaluated a number of prompting strategies; the detailed prompts are in Appendix D.

1. **One-Shot**: We prompt the LLM for a solution to the NL task description, with a single example and its optimal answer prepended to the prompt. The example is from the same variant and of the largest input size for the base problem, e.g., a 9-node graph for all GRAPH COLORING instances. This ensures that any reduction in problem-solving accuracy is not caused by length generalization issues, which are a known problem for transformers (Zhou et al., 2024; Anil et al., 2022).

2. **Zero-Shot Chain-of-Thought (CoT)**: The task description is followed by the sentence "Let's think step by step." (Kojima et al., 2022)

3. **One-Shot Chain-of-Thought (CoT)**: We prepend to the prompt the same example used in the one-shot case, this time with an answer text that includes a chain of thought resulting in a solution (Wei et al., 2022).

We also implemented an **ILP Python** prompting strategy, which prompts the LLM to translate the problem instance into Python code that calls the Gurobi solver on an Integer Linear Program (ILP) encoding of the instance (Gurobi Optimization LLC, 2024). Unlike in the first three prompting strategies, ILP Python does not attempt to solve the optimization problems through LLM reasoning; the problem is solved exactly and optimally by Gurobi, and the LLM merely translates the NL specifications to code and then translates the code's output back into NL. If the code generated by the LLM produces an error, we halt the process and count it as a failure.

We chose Python as the ILP specification syntax because this has been shown to outperform LLM translations into domain-specific languages (Bogin et al., 2024). We include results for specifying the ILPs in the domain-specific LP file format in Appendix F. Note that the idea of mapping NP-complete problems into ILPs using LLMs was previously explored by AhmadiTeshnizi et al. (2024), but not evaluated as systematically as in this paper.

Finally, we compare LLM problem-solving accuracy on each problem to **greedy baselines**. For GRAPH COLORING, the greedy heuristic colors

[1]Solvers for GRAPH COLORING and TRAVELING SALESMAN were coded using the `gurobipy` package (Gurobi Optimization LLC, 2024), and KNAPSACK instances were solved using Google OR-Tools.

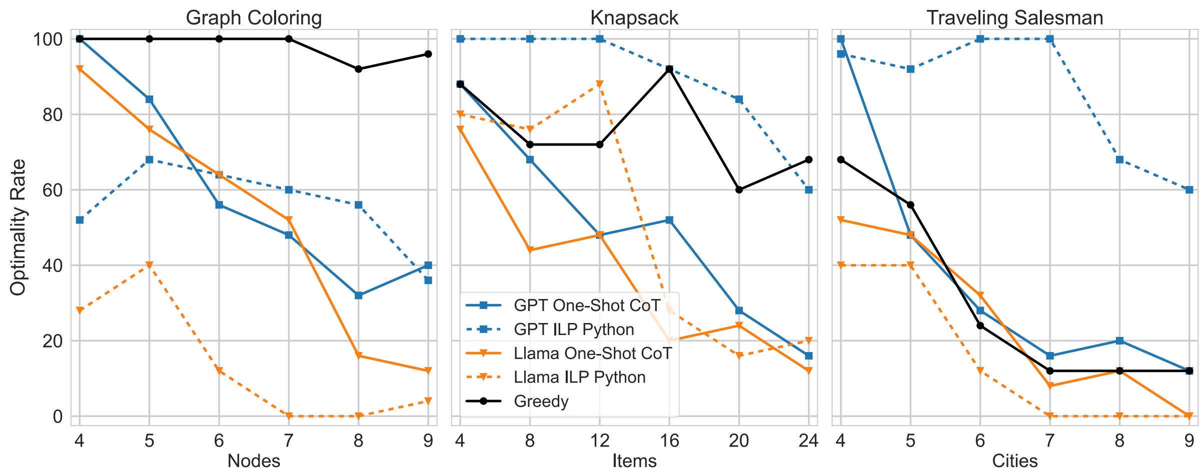

Figure 2: Percentage of instances solved optimally, as a function of instance size. Results are on the textbook variants in EHOP-RANDOM.

each node with the smallest color (where colors are represented by the numbers 1, 2, …) that does not conflict with any neighbors that have already been colored. Nodes are traversed in descending order of degree. For KNAPSACK, the strategy iterates through the items in descending order of density (value divided by weight), adding each item to the knapsack if it still fits in the remaining capacity. For TRAVELING SALESMAN, we use the strategy of always moving to the closest unvisited city. We apply the greedy baselines directly to the original problem instances. Note that all three greedy strategies are linear-time algorithms which always produce valid solutions, but give no guarantee of optimality.

4.3 Evaluation

We run all models with all prompting strategies on all instances in EHOP and classify the correctness of the output using the following scheme. An **incompatible** response is syntactically flawed; it can't be parsed as a solution to the problem. An **erroneous** response can be parsed as a solution, but it violates constraints of the underlying problem; for instance, it assigns adjacent nodes in GRAPH COLORING the same color. Among the remaining responses, we distinguish between **optimal** and **suboptimal** solutions, depending on whether they find a configuration that optimizes the objective as much as possible. ILP Python can additionally produce **ILP code failures** if the LLM-generated code cannot be executed without errors. See Appendix B for examples of each result category.

5 Results

5.1 Scaling to larger instances is hard, except for ILP

Figure 2 gives an overview of the percentage of instances for each textbook problem that were solved optimally, as a function of input size. For readability, we focus on One-Shot CoT since it consistently outperformed One-Shot and Zero-Shot CoT; full results are in Appendix F. We find that as instances are scaled up, the accuracy of most methods drops dramatically. The greedy heuristics outperform all LLM-based methods except ILP Python.

The ILP Python approach with GPT-4o maintains a higher accuracy even for larger instances. In this condition, the LLM is still required to make use of its "world knowledge" to flesh out the textual problem into a fine-grained symbolic ILP specification. However, it is freed up from having to perform complex combinatorial reasoning and keeping track of long chains of intermediate results (Zhang et al., 2024), which becomes exponentially harder as instances scale up. Unlike the other strategies, the ILP approach does not expose the LLM to the NP-hardness of the problem; the complexity of the language-to-ILP translation task grows linearly with input length.

5.2 Textbook is easier than other variants

We next measure whether the Textbook presentations are easier than the costumed and inverted variants, in order to provide new evidence on the reasoning vs. reciting debate. Recall from Section 3 that we carefully designed the instances of

Problem	Variant	One-Shot		Zero-Shot CoT		One-Shot CoT		ILP Python		Greedy
		GPT	Llama	GPT	Llama	GPT	Llama	GPT	Llama	
GCP	Textbook	42.0	9.3	60.7	38.7	60.0	52.0	56.0	14.0	98.0
	Inverted	−39.3	+4.7	−59.4	−38.7	−59.3	−52.0	−41.3	−7.3	
	Costumed	−6.2	−6.2	−6.5	−17.8	−4.7	−19.6	−43.8	+20.7	
KSP	Textbook	22.7	15.3	48.0	37.3	50.0	37.3	89.3	51.3	75.3
	Inverted	+4.6	−7.3	+2.7	−2.6	−4.7	−26.0	−0.6	+6.0	
	Costumed	−2.0	−1.5	−1.8	−4.9	−2.2	−4.4	−7.5	−0.9	
TSP	Textbook	34.7	28.7	31.3	25.3	37.3	25.3	86.0	15.3	30.7
	Inverted	−20.7	−24.0	−14.0	−11.3	−9.3	−15.3	−10.7	−10.6	
	Costumed	−8.3	−14.0	−1.7	−5.5	−9.1	−8.0	−37.1	−11.5	

Table 1: Percentage of instances solved optimally on EHOP-RANDOM, broken down by problem variant. Values from the non-textbook variants are provided as their differences relative to Textbook. "Costumed" is the average over the three costumes of each base problem.

	GPT	Llama
GCP	52.1%	66.4%
KSP	16.8%	37.6%
TSP	4.8%	5.1%

Table 2: The percentages of LLM responses on EHOP-RANDOM that were erroneous or incompatible, averaged across prompting strategies and variants.

each base problem to be of equal difficulty across variants.

As Table 1 shows, the methods we evaluated perform better on the Textbook variant than on the other variants in almost all conditions. The rows labeled "inverted" represent the inverted Textbook variants; the "costumed" rows are averages over all three costumes. Results for the individual variants, including ones that are inverted *and* costumed at the same time, are in Appendix F. The drop is especially pronounced for the inverted problems, which are worded in ways that make them recognizably related to well-documented archetypes of NP-hard problems. This very likely confuses the LLM, which might not register any difference from the standard problem.

While the ILP Python prompting strategy outperforms the others, it is still sensitive to deviations from the textbook presentations. This suggests that while the model no longer struggles to perform the right computation, the task of translating a problem to code is nevertheless affected by the ability to recognize the problem (when it is costumed) or to recognize how it deviates from the standard assumptions (when it is inverted).

5.3 LLMs rarely beat greedy heuristics

One of the most striking findings of Figure 2 is the extent to which the greedy heuristics are competitive with the LLM-based approaches: the greedy approach is near-optimal on Graph Coloring, outperforms CoT reasoning on Knapsack, and is on par with it on Traveling Salesman. This raises the question of whether the LLM-based solvers achieve their relatively high accuracies in Table 1 only because the instances in EHOP-RANDOM are very easy for their size. This echoes the point made by Tedeschi et al. (2023) that the capabilities of a model can only be accurately measured on sufficiently difficult datasets.

The gap between the LLM-based methods and the greedy heuristics is smallest on Traveling Salesman. This may be due to the fact that it is relatively easy for the LLM to generate a valid (if perhaps suboptimal) solution, as illustrated in Table 2. In Traveling Salesman, any answer that consists of 1 followed by a permutation of the numbers $2, \ldots, n$ is a valid solution. On the other hand, Graph Coloring and Knapsack both have constraints which eliminate potential solutions in more unpredictable ways.

We analyze the exact impact of instance difficulty on the performance of the different strategies by constructing a second sub-dataset of EHOP, which we call EHOP-HARD. This dataset is generated similarly to EHOP-RANDOM, except we only retain instances which the greedy heuristics of Section 4.2 do not solve optimally. This results in the Graph Coloring instances being limited to instance sizes 6–9, as virtually all instances with four or five nodes are solved optimally by the greedy heuristic.

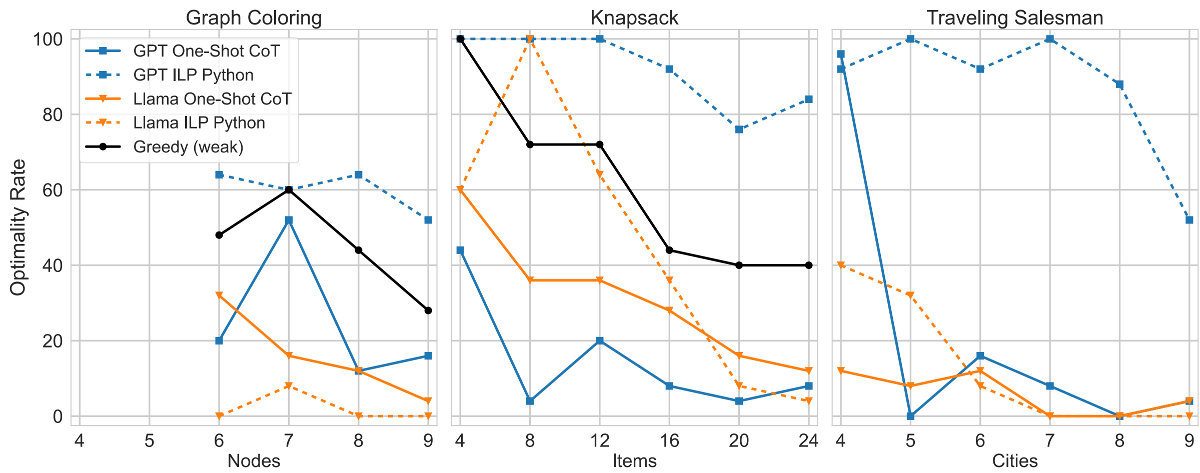

Figure 3: Percentage of instances solved optimally, as a function of instance size. Results are on the Textbook variants in EHOP-HARD. Note that this plot uses different greedy heuristics than Figure 2.

Problem	Variant	One-Shot GPT	One-Shot Llama	Zero-Shot CoT GPT	Zero-Shot CoT Llama	One-Shot CoT GPT	One-Shot CoT Llama	ILP Python GPT	ILP Python Llama	Greedy (weak)
GCP	Textbook	16.0	1.0	25.0	7.0	25.0	16.0	60.0	2.0	45.0
	Inverted	−16.0	+4.0	−25.0	−7.0	−24.0	−16.0	−54.0	−1.0	
	Costumed	+5.3	−1.0	+0.7	−1.0	−0.7	−7.0	−52.7	+19.3	
KSP	Textbook	8.7	5.3	18.0	10.7	14.7	31.3	92.0	45.3	61.3
	Inverted	+11.3	+9.4	+18.7	+15.3	+24.6	−17.3	−4.7	+8.0	
	Costumed	+2.2	+5.1	+3.6	+5.1	+9.5	−5.7	−8.0	−1.1	
TSP	Textbook	15.3	8.0	24.7	12.0	20.7	6.0	87.3	13.3	—
	Inverted	−4.6	−6.7	−6.7	−5.3	−4.7	−2.7	−12.6	−7.3	
	Costumed	−1.7	−3.3	−3.6	−4.4	−11.6	−0.7	−33.7	−9.5	

Table 3: Optimality rates on EHOP-HARD, as a function of problem variant. Formatting matches that of Table 1.

We repeat the analyses of Section 5.1 and Section 5.2 on EHOP-HARD. The results are shown in Figure 3 and Table 3. Note that we use a different set of **weak greedy heuristics** than in Figure 2, because EHOP-HARD is constructed such that the original greedy heuristics solve none of the instances optimally. For GRAPH COLORING, we traverse the nodes in random order, rather than in descending order of degree; for KNAPSACK, we pick the highest-value, rather than the highest-density, items first. We call these heuristics "weak" because they performed worse than the original heuristics on EHOP-RANDOM.

The purely LLM-based approaches perform much more poorly overall than in the experiments on EHOP-RANDOM, giving further evidence to the fact that they primarily follow a greedy strategy. While their accuracy does not drop to zero, they are still being systematically outperformed by the "weak" greedy heuristics. The ILP approach is largely unaffected, illustrating the strength of the neurosymbolic translation-based method. The overall pattern in Table 3 is still that the Textbook variant is easier than the others, except for methods that already perform very poorly on Textbook.

5.4 Generalization over numbers is brittle

While the ILP method outperforms the other prompting strategies in our experiments, it also exposes an interesting weakness of LLMs as translators. Gurobi is an optimal solver for ILPs; thus, those instances of EHOP that are not solved optimally by ILP Python must be due to mistakes that the LLM made in mapping the NL task description to a linear program (or in mapping code output back to NL). The instances of the same size differ only in the values of the parameters; specifically in KNAPSACK and TRAVELING SALESMAN, the prompts have identical length and content, and the only change is in some numbers. Thus, the fact that ILP Python does not yield 100% or 0% optimal accuracy among the instances of the same size in

Figure 2 indicates that changing a number in the prompt can make the difference between the LLM mapping it into the correct ILP or not. Table 9 in Appendix E shows a concrete example of this.

Note that this failure mode is orthogonal to the difficulty for the LLM of actually solving the problem; in the ILP method, the LLM acts as a semantic parser that must simultaneously recognize a textual problem as an instance of a textbook problem and map it to an ILP. This echoes the finding that other forms of semantic parsing are hard for few-shot prompting as well, with LLMs as recent as GPT-4 (Ettinger et al., 2023).

6 Discussion

The results above paint an intricate picture of the features that make it easy or difficult for an LLM to solve NP-hard optimization problems. First, given previous research, it was expected that *instance size* would negatively impact accuracy; we have confirmed this for purely LLM-based methods. Second, we have identified *instance difficulty* as an important factor: among instances of the same size, those that cannot be solved by greedy heuristics are also harder for LLMs. Neurosymbolic methods that combine LLMs as semantic parsers with exact ILP solvers are more robust to both of these factors.

Finally, the *presentation* of a problem instance impacts how difficult it is for LLM-based methods to solve; all methods, including the ILP-based ones, perform much better on the well-established textbook presentations rather than our novel costumed and inverted variants. This sheds doubt on the LLMs having learned general problem-solving skills that would allow them to generalize from one problem formulation to another ("reasoning"). Instead, it seems that they are much better at generalizing existing solution methods for textbook problems that are documented on the Internet to other instances of the same textbook problem ("reciting").

In the experiments above, we have deliberately avoided the use of LLMs that perform uncontrollable and undocumented chain-of-thought reasoning and code generation "behind the scenes" (e.g., o1) in order to obtain interpretable results. Both CoT reasoning and code generation were included in a controlled fashion in the experiments, suggesting that these mechanisms by themselves will not make LLMs general problem-solvers. We conjecture that the ability of LLMs to generalize to novel problems could be increased in the future by explicitly including problem solving (rather than the supervised replication of existing solutions) in the training regime.

Furthermore, we have deliberately designed our prompts such that they do not name the base problem in the dressed-up task descriptions. It is possible that giving such hints to the LLM would substantially improve its ability to recognize the variant as equivalent to the base problem, closing the gap between the variants. However, this sort of hand-holding would greatly undermine the usefulness of LLMs as general problem-solvers. The exciting promise of LLMs is their potential to provide general-purpose assistance to a wide range of people in a wide range of contexts. For this vision to be realized, we require that LLMs are able to recognize the underlying computational problems presented to them, even, and especially, when this is not obvious to their users. We cannot assume that the average user will ask their assistant to view their request for travel advice as an instance of KNAPSACK – the average user has almost certainly never heard of KNAPSACK.

Our results show that LLMs struggle when textbook problems are obscured by even the most superficial of costumes. It follows that they would be even less likely to recognize and correctly solve such optimization problems when embedded in more realistic contexts–e.g., optimizing the energy grid or allocating government resources. If LLMs depend on their human prompters to have done the creative problem solving for them, they will be a far less useful technology than is currently hoped.

7 Conclusion

We have shown that current LLMs are much better at solving NP-hard optimization problems when they are presented in their well-documented textbook form than when they arise as everyday problems or are subtly distorted. This is further evidence that LLMs are often reciting when they appear to be reasoning.

One limitation of EHOP as a dataset of real problem-solving tasks is that real users will often not be able to spell an instance of an everyday problem out in detail, e.g. by assigning a numeric satisfaction value to every museum in Paris. It would be interesting to explore dialogue systems performing actual collaborative problem-solving with the user. The costumes of EHOP could be a good starting point for such work.

Limitations

In addition to the limitations that we already discussed in Section 6, it is worth considering the limitations of the EHOP dataset which we introduce here.

The instances of EHOP cover a limited range of instance sizes. We chose the sizes for each base problem based on the performance of the greedy heuristics, and we believe that they are sufficiently representative to support the claims we make. As LLMs improve, it may become informative to evaluate on larger instances. We will make the code for generating more EHOP-like task descriptions available alongside the dataset itself to facilitate this.

Furthermore, EHOP is based on three well-established textbook problems, and the costumes do not actually cover full-blown real-world use cases like the ones in NL4Opt (Ramamonjison et al., 2022). This is because we did not construct EHOP to be predictive of real-world problem-solving accuracies but instead to permit a targeted comparison of the impact of problem presentation.

References

David J. Abraham, Avrim Blum, and Tuomas Sandholm. 2007. Clearing algorithms for barter exchange markets: enabling nationwide kidney exchanges. In *Proceedings of the 8th ACM Conference on Electronic Commerce*, pages 295–304, San Diego, California, USA. Association for Computing Machinery.

Ali AhmadiTeshnizi, Wenzhi Gao, Herman Brunborg, Shayan Talaei, and Madeleine Udell. 2024. Optimus-0.3: Using Large Language Models to Model and Solve Optimization Problems at Scale. *arXiv preprint arXiv:2407.19633*.

Cem Anil, Yuhuai Wu, Anders Johan Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Venkatesh Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. 2022. Exploring Length Generalization in Large Language Models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*.

Ben Bogin, Shivanshu Gupta, Peter Clark, and Ashish Sabharwal. 2024. Leveraging Code to Improve In-Context Learning for Semantic Parsing. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4971–5012, Mexico City, Mexico. Association for Computational Linguistics.

Allyson Ettinger, Jena Hwang, Valentina Pyatkin, Chandra Bhagavatula, and Yejin Choi. 2023. "You Are An Expert Linguistic Annotator": Limits of LLMs as Analyzers of Abstract Meaning Representation. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 8250–8263, Singapore. Association for Computational Linguistics.

Lizhou Fan, Wenyue Hua, Lingyao Li, Haoyang Ling, and Yongfeng Zhang. 2024. NPHardEval: Dynamic Benchmark on Reasoning Ability of Large Language Models via Complexity Classes. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4092–4114, Bangkok, Thailand. Association for Computational Linguistics.

Michael R Garey and David S Johnson. 1979. *Computers and Intractability.*volume 174. Freeman San Francisco.

Balaji Gopalakrishnan and Ellis L. Johnson. 2005. Airline Crew Scheduling: State-of-the-Art. *Annals of Operations Research*, 140:305–337.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, et al. 2024. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*.

Gurobi Optimization LLC. 2024. Gurobi Optimizer Reference Manual.

Mercedes Hidalgo-Herrero, Pablo Rabanal, Ismael Rodriguez, and Fernando Rubio. 2013. Comparing Problem Solving Strategies for NP-hard Optimization Problems. *Fundamenta Informaticae*, 124:1–25.

Subbarao Kambhampati. 2024. Can Large Language Models Reason and Plan?. *Annals of the New York Academy of Sciences*, 1534(1):15–18.

Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large Language Models are Zero-Shot Reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pages 22199–22213.

William Merrill and Ashish Sabharwal. 2024. The Expressive Power of Transformers with Chain of Thought. In *The Twelfth International Conference on Learning Representations*.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2024. GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models. *arXiv preprint arXiv:2410.05229*.

OpenAI. 2024. GPT-4o System Card. *arXiv preprint arXiv:2410.21276*.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. 2022. NL4Opt Competition: Formulating Optimization Problems Based on Their Natural Language Descriptions. In Marco Ciccone, Gustavo Stolovitzky, and Jacob Albrecht, editors, *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220, pages 189–203. PMLR.

Katharina Stein, Daniel Fišer, Jörg Hoffmann, and Alexander Koller. 2024. AutoPlanBench: Automatically generating benchmarks for LLM planners from PDDL. *arXiv Preprint arXiv:2311.09830*.

Jianheng Tang, Qifan Zhang, Yuhan Li, and Jia Li. 2024. GraphArena: Benchmarking Large Language Models on Graph Computational Problems. arXiv preprint arXiv:2407.00379.

Simone Tedeschi, Johan Bos, Thierry Declerck, Jan Hajič, Daniel Hershcovich, Eduard Hovy, Alexander Koller, Simon Krek, Steven Schockaert, Rico Sennrich, Ekaterina Shutova, and Roberto Navigli. 2023. What`s the Meaning of Superhuman Performance in Today`s NLU?. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12471–12491, Toronto, Canada. Association for Computational Linguistics.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, Yitao Liang, and Team CraftJarvis. 2023. Describe, explain, plan and select: interactive planning with large language models enables open-world multi-task agents. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, New Orleans, LA, USA. Curran Associates Inc.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *36th Conference on Neural Information Processing Systems*.

Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. 2024. Reasoning or Reciting? Exploring the Capabilities and Limitations of Language Models Through Counterfactual Tasks. In *Proceedings of the NAACL-HLT*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*.

Chunhui Zhang, Yiren Jian, Zhongyu Ouyang, and Soroush Vosoughi. 2024. Working Memory Identifies Reasoning Limits in Language Models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16896–16922, Miami, Florida, USA. Association for Computational Linguistics.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Joshua M. Susskind, Samy Bengio, and Preetum Nakkiran. 2024. What Algorithms can Transformers Learn? A Study in Length Generalization. In *The Twelfth International Conference on Learning Representations*.

<table>
<tr><th>Result</th><th>LLM Response</th><th>Optimal Solution</th></tr>
<tr><td>Optimal</td><td>1,2,3,4,1</td><td>1, 2, 3, 4</td></tr>
<tr><td>Suboptimal</td><td>1,4,3,2,1</td><td>1, 4, 2, 3</td></tr>
<tr><td>Erroneous</td><td>1, 1, 1, 1, 1</td><td>1, 4, 3, 5, 2</td></tr>
<tr><td>Incompatible</td><td>1,4,1,2,3,5,1</td><td>1, 4, 2, 3, 5</td></tr>
<tr><td>ILP Failure</td><td><pre>
from gurobipy import GRB, Model, quicksum

def f():
    # Create the model
    model = Model("Traveling Salesman Problem")

    # Create helper variables
    n = 4 # number of cities
    dist = [[0, 5, 11, 4], [5, 0, 1, 1], [11, 1, 0, 3], [4, 1, 3, 0]] # distance matrix

    # Add variables
     x = model.addVars(n, n, vtype=GRB.BINARY, name="x") # x[i, j] = 1 if we travel from city i to city j
    u = model.addVars(n, vtype=GRB.INTEGER, name="u") # u[i] = order in which we visit city i

    # Add constraints
     model.addConstrs(quicksum(x[i, j] for j in range(n)) == 1 for i in range(n)) # each city is visited exactly once
     model.addConstrs(quicksum(x[j, i] for j in range(n)) == 1 for i in range(n)) # each city is left exactly once
     model.addConstrs(u[i] - u[j] + n * x[i, j] <= n - 1 for i in range(n) for j in range(n) if i != j) # subtour elimination
      model.addConstrs(x[i, i] == 0 for i in range(n)) # we cannot visit the same city twice
    model.addConstr(u[0] == 1) # we start at city 1

    # Set objective
      model.setObjective(quicksum(dist[i][j] * x[i, j] for i in range(n) for j in range(n)), GRB.MINIMIZE)

    # Optimize/solve the model
    model.optimize()

    # Return the optimized model
    return model
</pre>This ILP formulation uses the following variables:
...
The objective function is the total distance traveled, which is minimized.<hr><code>AttributeError at line 117: Unable to retrieve attribute 'X'</code></td><td>1, 2, 3, 4</td></tr>
</table>

Table 4: The following examples are all generated by Llama for textbook TRAVELING SALESMAN with the ILP Python prompting strategy. Except for the code failure instance, there was a code response which was then executed successfully and returned to the model before the final output was produced. The code which produced an error is shown in the ILP Failure case. The error here is indicative of an ILP model which cannot be properly optimized.

A Language Model Details

For both GPT-4o and Llama 3.1 70B Instruct, we use the following sampling parameters for all LLM-only prompting strategies:

```
max_tokens=1024
temperature=0.0
presence_penalty=0.0
frequency_penalty=0.0
seed=1
```

In the case of the ILP LP prompting strategy, `max_tokens` is set to 6000 for the completion that is meant to produce the LP code. We similarly change `max_tokens` to 3072 for the ILP Python prompting strategy in the generation step. After the generation step, `max_tokens` is reset to 1024 (when asking the LLM to translate code output back to NL).

It is not known how many parameters GPT-4o has, and Llama 3.1 70B Instruct has 70 billion parameters. GPT-4o was prompted using API calls, so we do not know the GPU cost associated with running this subset of the experiments, though the API calls took about 50 hours in total to complete (excluding the ILP LP prompting strategy). We estimate that it takes about 240 GPU hours running on NVIDIA H100 PCIe GPUs to run the entire experiment (excluding ILP LP) on Llama 3.1 70B Instruct.

	Standard	Inverted
Textbook	I have a network of 4 nodes, numbered 1 to 4, with various nodes being connected to one another. I want to color the nodes such that no two connected nodes have the same color. The connections are as follows: Node 1 and node 3 are connected. Node 1 and node 4 are connected. Node 2 and node 3 are connected. Node 2 and node 4 are connected. How can I color the nodes using the fewest colors possible? Generate a comma-separated list of the colors for each node, where the colors are represented by integers ranging from 1 to the number of colors used. The colors should be in the order of the vertices, so the first color will correspond to node 1, the second color will correspond to node 2, and so on.	I have a network of 4 nodes, numbered 1 to 4, with various nodes being connected to one another. I want to color the nodes such that no two unconnected nodes have the same color. The connections are as follows: Node 1 and node 2 are connected. Node 3 and node 4 are connected. How can I color the nodes using the fewest colors possible? Generate a comma-separated list of the colors for each node, where the colors are represented by integers ranging from 1 to the number of colors used. The colors should be in the order of the vertices, so the first color will correspond to node 1, the second color will correspond to node 2, and so on.
Student Groups	I am a teacher, and I want to assign my 4 students to different groups. I need the groups to focus, so I need to make sure that no two students who are friends with one another are in the same group, otherwise they may get distracted. I don't need the groups to all be the same size, but I want to minimize the total number of groups. The friendships are as follows: Student 1 and student 3 are friends. Student 1 and student 4 are friends. Student 2 and student 3 are friends. Student 2 and student 4 are friends. Which group should each student be assigned to? Generate a comma-separated list with each student's group, where the groups are represented by integers ranging from 1 to the total number of groups. The groups should be in the order of the students' numbers, so the first group in the list will correspond to student 1, the second group will correspond to student 2, and so on.	I am a teacher, and I want to assign my 4 students to different groups. I want the groups to have fun, so I need to make sure that only students who are friends with one another are in the same group. In other words, no group can have a pair of students who aren't friends with each other. I don't need the groups to all be the same size, but I want to minimize the total number of groups. The friendships are as follows: Student 1 and student 2 are friends. Student 3 and student 4 are friends. Which group should each student be assigned to? Generate a comma-separated list with each student's group, where the groups are represented by integers ranging from 1 to the total number of groups. The groups should be in the order of the students' numbers, so the first group in the list will correspond to student 1, the second group will correspond to student 2, and so on.
Parties with Exes	My birthday is coming up, and I want to celebrate with my 4 friends. Unfortunately, some of my friends used to be in romantic relationships with each other, and they don't get along anymore. I will therefore be having multiple birthday parties. I want to invite each person to one party, and I want to invite exes to different parties so that no two people who used to date one another are at the same party. I have a list of who used to date whom, and I want to host as few parties as possible while avoiding the awkardness of having a pair of exes at the same party. The past relationships are as follows: Friend 1 and friend 3 used to be in a relationship. Friend 1 and friend 4 used to be in a relationship. Friend 2 and friend 3 used to be in a relationship. Friend 2 and friend 4 used to be in a relationship. Which party should each friend be invited to? Generate a comma-separated list with each friend's party, where the parties are represented by integers ranging from 1 to the total number of parties. The parties should be in the order of the friends' numbers, so the first party in the list will correspond to friend 1, the second party will correspond to friend 2, and so on.	My birthday is coming up, and I want to celebrate with my 4 friends. Some of my friends used to be in romantic relationships with each other, and they don't get along anymore. I will therefore be having multiple birthday parties. I want to invite each person to one party, and I want to make things as awkward as possible, so I only want to invite two people to the same party if they used to be in a relationship. I have a list of who used to date whom, and I want to host as few parties as possible while avoiding having a pair of people who haven't dated at the same party. The past relationships are as follows: Friend 1 and friend 2 used to be in a relationship. Friend 3 and friend 4 used to be in a relationship. Which party should each friend be invited to? Generate a comma-separated list with each friend's party, where the parties are represented by integers ranging from 1 to the total number of parties. The parties should be in the order of the friends' numbers, so the first party in the list will correspond to friend 1, the second party will correspond to friend 2, and so on.
Taekwondo Tournament	I am organizing a taekwondo tournament. There are 4 participants, and I need to reserve some rooms in the tournament hall for them to warm up in. I want to make sure that no two participants who are competing against each other are in the same room. This way, no one will learn about an opponent's technique ahead of the actual competition. I have a list of who is competing against whom, and I want to reserve as few rooms as possible while making sure no one is in the same room as any of their opponents. Here are the matchups: Participant 1 and participant 3 are competing against one another. Participant 1 and participant 4 are competing against one another. Participant 2 and participant 3 are competing against one another. Participant 2 and participant 4 are competing against one another. Which room should each participant be assigned to? Generate a comma-separated list with each participant's room, where the rooms are represented by integers ranging from 1 to the total number of rooms. The rooms should be in the order of the participants' numbers, so the first room in the list will correspond to participant 1, the second room will correspond to participant 2, and so on.	I am organizing a taekwondo tournament. There are 4 participants, and I need to reserve some rooms in the tournament hall for them to warm up in. I want to make sure that if two participants are not competing against each other, then they are in different rooms. This way, competitive tension will be as high as possible. I have a list of who is competing against whom, and I want to reserve as few rooms as possible while making sure no one is in the same room as a non-opponent. Here are the matchups: Participant 1 and participant 2 are competing against one another. Participant 3 and participant 4 are competing against one another. Which room should each participant be assigned to? Generate a comma-separated list with each participant's room, where the rooms are represented by integers ranging from 1 to the total number of rooms. The rooms should be in the order of the participants' numbers, so the first room in the list will correspond to participant 1, the second room will correspond to participant 2, and so on.

Table 5: Examples of the four Graph Coloring costumes, both standard (textbook rules) and inverted, all generated using the same problem instance.

B Result Category Examples

Table 4 shows examples of each result type within the context of Traveling Salesman. It should be noted that since models often would repeat the first node at the end of a tour (as seen in all of the responses in this table), we treated both "1, 2, 3, 4" and "1, 2, 3, 4, 1" as proper formatting. The response "1, 1, 1, 1, 1" is classified as erroneous since it has the right length (5 locations) but it does not meet the constraint of visiting each location exactly once. The response "1,4,1,2,3,5,1", on the other hand, is classified as incompatible since it has 7 locations (6 after removing the redundant 1 at the end), which is more than the expected format (5 locations).

	Standard	Inverted
Textbook	I am trying to fill a bag with valuable items. Each item has a weight and a value. Here are the items I have: Item 1 has a weight of 1 kg and a value of 2 €. Item 2 has a weight of 1 kg and a value of 2 €. Item 3 has a weight of 3 kg and a value of 3 €. Item 4 has a weight of 3 kg and a value of 4 €. Which items should I pack to get the most value possible while also making sure the total weight of the items does not exceed the bag's capacity of 1 kg? Generate a comma-separated list of the items I should put in the bag, where each item is represented by its number.	I am trying to fill a bag with worthless items. Each item has a weight and a value. Here are the items I have: Item 1 has a weight of 1 kg and a value of 2 €. Item 2 has a weight of 1 kg and a value of 2 €. Item 3 has a weight of 3 kg and a value of 3 €. Item 4 has a weight of 3 kg and a value of 4 €. Which items should I pack to get the least value possible while also making sure the total weight of the items is at least 7 kg? Generate a comma-separated list of the items I should put in the bag, where each item is represented by its number.
Lemonade Stand	I am running a lemonade stand where I don't set a single price but rather let the customers make custom offers. Each customer is offering a specific amount of money for a specific amount of lemonade. Each offer is rigid, so I can only fulfill it exactly as stated or not fulfill it at all. I have the following offers: Customer 1 is offering \$2 for 1 gallon of lemonade. Customer 2 is offering \$2 for 1 gallon of lemonade. Customer 3 is offering \$3 for 3 gallons of lemonade. Customer 4 is offering \$4 for 3 gallons of lemonade. Which customers' offers should I take up to make my revenue as large as possible given that I can't sell more than 1 total gallons of lemonade? Generate a comma-separated list of the customers whose offers I should take up, where each customer is represented by their number.	I am running a lemonade stand where I don't set a single price but rather let the customers make custom offers. Each customer is offering a specific amount of money for a specific amount of lemonade. Each offer is rigid, so I can only fulfill it exactly as stated or not fulfill it at all. I have the following offers: Customer 1 is offering \$2 for 1 gallon of lemonade. Customer 2 is offering \$2 for 1 gallon of lemonade. Customer 3 is offering \$3 for 3 gallons of lemonade. Customer 4 is offering \$4 for 3 gallons of lemonade. I don't want to seem greedy. Which customers' offers should I take up to make my total revenue as small as possible while selling at least 7 gallons of lemonade? Generate a comma-separated list of the customers whose offers I should take up, where each customer is represented by their number.
Sightseeing	I am going to be visiting Paris tomorrow, and I want to make the most of my time there. I have a list of attractions I want to visit, but I don't have enough time to visit all of them. I have given each attraction a point value and determined how many minutes I would need to spend on it. Here are the attractions: Attraction 1 has a score of 2 points and would require 10 minutes. Attraction 2 has a score of 2 points and would require 10 minutes. Attraction 3 has a score of 3 points and would require 30 minutes. Attraction 4 has a score of 4 points and would require 30 minutes. Which attractions should I visit to make the total point value as high as possible while not having the total time required go over my sightseeing limit of 10 minutes? Generate a comma-separated list of the attractions I should visit, where each attraction is represented by its number.	I am going to be visiting Paris tomorrow with a friend. I need to go through some emails at the start of the trip while my friend gets a head start on the sightseeing. I want to tell him which attractions he can visit before I join him so that I miss out as little as possible. I have given each attraction on our list a point value and determined how many minutes one would need to spend on it. Here are the attractions: Attraction 1 has a score of 2 points and would require 10 minutes. Attraction 2 has a score of 2 points and would require 10 minutes. Attraction 3 has a score of 3 points and would require 30 minutes. Attraction 4 has a score of 4 points and would require 30 minutes. Which attractions should I tell my friend to visit to make the total score of the attractions he sees without me as low as possible while ensuring that the total time required to visit them is at least 70 minutes? Generate a comma-separated list of the attractions I should suggest to my friend, where each attraction is represented by its number.
Party Planning	I am planning a party, and I need to buy some decorations. Each decoration has a cost and a point value I've assigned in terms of its worth as a decoration. Here are the decorations I can buy: Decoration 1 has a cost of \$10 and a point value of 2. Decoration 2 has a cost of \$10 and a point value of 2. Decoration 3 has a cost of \$30 and a point value of 3. Decoration 4 has a cost of \$30 and a point value of 4. I can buy at most one of each decoration. Which decorations should I purchase to make the total point value as high as possible without going over my budget of \$10? Generate a comma-separated list of the decorations I should buy, where each decoration is represented by its number.	I am planning a party, and I need to buy some decorations. I don't want the decorations to be the focus of the party, so I wan't to pick the worst ones, but I still need to spend the decorations budget. Each decoration has a cost and a point value I've assigned in terms of its worth as a decoration. Here are the decorations I can buy: Decoration 1 has a cost of \$10 and a point value of 2. Decoration 2 has a cost of \$10 and a point value of 2. Decoration 3 has a cost of \$30 and a point value of 3. Decoration 4 has a cost of \$30 and a point value of 4. I can buy at most one of each decoration. Which decorations should I purchase to make the total point value as low as possible while spending at least \$70? Generate a comma-separated list of the decorations I should buy, where each decoration is represented by its number.

Table 6: Examples of the four KNAPSACK costumes, both standard (textbook rules) and inverted, all generated using the same problem instance.

C Costumes

Table 5, Table 6, and Table 7 display examples of how problem instances were presented to the LLM. The instances used to generate all examples were of the smallest scale used in the EHOP dataset (4 nodes/4 items/4 cities).

D Prompting Strategies

Table 8 presents the overall structure of each prompting strategy. The BASE PROMPT would be of the form of one of the examples seen in Appendix C. It is also worth noting that the DEMO PROMPT and DEMO GREEDY CoT were always formatted to match the variant of the BASE PROMPT.

In the One-Shot strategies, the `Assistant` response was provided by us to emulate a past response in the conversational context. In the ILP cases, on the other hand, the `Assistant` response was in fact generated by the LLM, and the following `User` response would depend on its content. If the code ran successfully, its output would be inserted in the format of the response shown, and if the LLM's code produced an error, the instance would be marked as a code failure, and there would be no follow-up. For full implementation details, see our codebase.

	Standard	Inverted
Textbook	I am planning a trip to visit several cities. Here are the distances between each pair of cities: City 1 and city 2 are 8 miles apart. City 1 and city 3 are 14 miles apart. City 1 and city 4 are 13 miles apart. City 2 and city 3 are 6 miles apart. City 2 and city 4 are 15 miles apart. City 3 and city 4 are 3 miles apart. What is the shortest possible route that starts at city 1, visits each city exactly once, and returns to city 1? Please generate a comma-separated list of the cities in the order I should visit them, where the cities are represented by their respective numbers.	I am planning a trip to visit several cities. Here are the distances between each pair of cities: City 1 and city 2 are 11 miles apart. City 1 and city 3 are 5 miles apart. City 1 and city 4 are 6 miles apart. City 2 and city 3 are 13 miles apart. City 2 and city 4 are 4 miles apart. City 3 and city 4 are 16 miles apart. What is the longest possible route that starts at city 1, visits each city exactly once, and returns to city 1? Please generate a comma-separated list of the cities in the order I should visit them, where the cities are represented by their respective numbers.
Task Schedule	I have a set of tasks that I have to complete every day. My boss always makes me start with task 1, but the order in which I complete the rest is up to me. It takes me a certain amount of time to modify my workspace to transition from one task to another, and at the end of the day, I'll need to set up my space for task 1 so that I'm ready the next morning. Here is the time it takes me to transition from one task to another: It takes 8 minutes to transition between task 1 and task 2. It takes 14 minutes to transition between task 1 and task 3. It takes 13 minutes to transition between task 1 and task 4. It takes 6 minutes to transition between task 2 and task 3. It takes 15 minutes to transition between task 2 and task 4. It takes 3 minutes to transition between task 3 and task 4. It takes me the same amount of time to transition between one task and another, regardless of which task I'm transitioning from and which task I'm transitioning to. In what order should I complete the tasks every day to minimize the total time spent transitioning between tasks? Please generate a comma-separated list of the tasks in the order I should complete them, where the tasks are represented by their respective numbers.	I have a set of tasks that I have to complete every day. My boss always makes me start with task 1, but the order in which I complete the rest is up to me. It takes me a certain amount of time to modify my workspace to transition from one task to another, and at the end of the day, I'll need to set up my space for task 1 so that I'm ready the next morning. Here is the time it takes me to transition from one task to another: It takes 11 minutes to transition between task 1 and task 2. It takes 5 minutes to transition between task 1 and task 3. It takes 6 minutes to transition between task 1 and task 4. It takes 13 minutes to transition between task 2 and task 3. It takes 4 minutes to transition between task 2 and task 4. It takes 16 minutes to transition between task 3 and task 4. It takes me the same amount of time to transition between one task and another, regardless of which task I'm transitioning from and which task I'm transitioning to, and the only time I get to relax during the day is during these transitions. In what order should I complete the tasks every day to maximize the total time spent transitioning between tasks? Please generate a comma-separated list of the tasks in the order I should complete them, where the tasks are represented by their respective numbers.
Exercise Schedule	My New Year's resolution is to be more physically active. I've made a list of 4 activities, and I want to do one of them every day. After I do an activity, I can't do it again until I've done everything else on the list. I'm going to start with activity 1 on January first, but the order in which I complete the rest is up in the air. Then, when I'm done with the list, I want to go through the activities again in the same order I used before. I've scored each pair of activities based on how similar they are, with more similar activities getting higher scores. Here are the scores: Activity 1 and activity 2 have a similarity of 8. Activity 1 and activity 3 have a similarity of 14. Activity 1 and activity 4 have a similarity of 13. Activity 2 and activity 3 have a similarity of 6. Activity 2 and activity 4 have a similarity of 15. Activity 3 and activity 4 have a similarity of 3. I want to have a lot of variety from day to day. What is the best order in which to do the activities to minimize the total similarity between activities on adjacent days, including between the last activity and activity 1 (when starting the next round)? Please generate a comma-separated list of the activities in the order I should complete them, where the activities are represented by their respective numbers.	My New Year's resolution is to be more physically active. I've made a list of 4 activities, and I want to do one of them every day. After I do an activity, I can't do it again until I've done everything else on the list. I'm going to start with activity 1 on January first, but the order in which I complete the rest is up in the air. Then, when I'm done with the list, I want to go through the activities again in the same order I used before. I've scored each pair of activities based on how similar they are, with more similar activities getting higher scores. Here are the scores: Activity 1 and activity 2 have a similarity of 11. Activity 1 and activity 3 have a similarity of 5. Activity 1 and activity 4 have a similarity of 6. Activity 2 and activity 3 have a similarity of 13. Activity 2 and activity 4 have a similarity of 4. Activity 3 and activity 4 have a similarity of 16. I want to have smooth transitions from one day to the next. What is the best order in which to do the activities to maximize the total similarity between activities on adjacent days, including between the last activity and activity 1 (when starting the next round)? Please generate a comma-separated list of the activities in the order I should complete them, where the activities are represented by their respective numbers.
UN Seating	I am responsible for the seating assignments at an upcoming UN meeting. There will be representatives from 4 nations sitting at a round table. The representative from nation 1 will be leading the discussion, so they will be sitting in the designated "Director Seat," but nothing else is decided yet. There is some amount of political tension between each pair of nations, and I've been given a list of tension scores for each pair of representatives, with higher scores indicating higher tension. Here are the tension levels between each pair of representatives: Representative 1 and representative 2 have tension score 8. Representative 1 and representative 3 have tension score 14. Representative 1 and representative 4 have tension score 13. Representative 2 and representative 3 have tension score 6. Representative 2 and representative 4 have tension score 15. Representative 3 and representative 4 have tension score 3. I want to minimize the total tension between adjacent pairs of representatives to prevent the discussion from getting heated. What should the seating order be, starting at the Director Seat and continuing clockwise? Note that the last person in the ordering will also be sitting next to the Director Seat. Please generate a comma-separated list of the representatives in the order they should be seated, where the representatives are represented by their respective numbers.	I am responsible for the seating assignments at an upcoming UN meeting. There will be representatives from 4 nations sitting at a round table. The representative from nation 1 will be leading the discussion, so they will be sitting in the designated "Director Seat," but nothing else is decided yet. There is some amount of political tension between each pair of nations, and I've been given a list of tension scores for each pair of representatives, with higher scores indicating higher tension. Here are the tension levels between each pair of representatives: Representative 1 and representative 2 have tension score 11. Representative 1 and representative 3 have tension score 5. Representative 1 and representative 4 have tension score 6. Representative 2 and representative 3 have tension score 13. Representative 2 and representative 4 have tension score 4. Representative 3 and representative 4 have tension score 16. I want to maximize the total tension between adjacent pairs of representatives to encourage discussion and progress. What should the seating order be, starting at the "Director Seat" and continuing clockwise? Note that the last person in the ordering will also be sitting next to the Director Seat. Please generate a comma-separated list of the representatives in the order they should be seated, where the representatives are represented by their respective numbers.

Table 7: Examples of the four TRAVELING SALESMAN costumes, both standard (textbook rules) and inverted, all generated using the same problem instance.

ILP LP. The ILP LP prompting strategy is very similar to ILP Python, with the exception that the LLM is asked to express the ILP program in the LP file format instead of as a Python program. We use the Gurobi solver (Gurobi Optimization LLC, 2024) to evaluate the code generated by the LLM, and we return the variable assignments generated by Gurobi in our follow-up message to the LLM. See our codebase for more details.

Zero-Shot CoT	`User:`	<BASE PROMPT> You may explain your reasoning, but do not add any more explanations once you have produced the comma-separated list. Let's think step by step.
One-Shot	`User:`	<DEMO PROMPT>
	`Assitant:`	<DEMO ANSWER>
	`User:`	<BASE PROMPT>
One-Shot CoT	`User:`	<DEMO PROMPT>
	`Assitant:`	<DEMO GREEDY CoT> <DEMO ANSWER>
	`User:`	<BASE PROMPT>
ILP LP	`User:`	<BASE PROMPT> Instead of solving the problem, please express it as an Integer Linear Programming (ILP) problem in the LP file format. Here is an example of the LP file format: *LP EXAMPLE* Start by thinking step by step about the variables and constraints you'll need in order to express the problem fully, and then create the specification in the LP format. <CAUTION AGAINST COMMON MISTAKES> Please provide the ILP problem in the LP format and do not solve the problem yourself.
	`Assistant:`	**<LLM GENERATED CODE>**
	`User:`	Your ILP problem was successfully solved. Here is the solution: <ILP MODEL PARAMETER VALUES> Translate this solution back to the original problem and provide it as originally specified. Do not add any more explanation once you've provided the solution.
ILP Python	`User:`	<BASE PROMPT> Please express this as an Integer Linear Programming (ILP) problem using Python with the gurobipy library. Specifically, define a function named f that returns an optimized \`gurobipy.Model\` object which represents the problem. Here is an example of the format you should use for your answer: *PYTHON EXAMPLE* Start by thinking step by step about the variables and constraints you'll need in order to express the problem fully, and then define the Python function f. <CAUTION AGAINST COMMON MISTAKES>
	`Assistant:`	**<LLM GENERATED CODE>**
	`User:`	Your code was executed successfully. Here are all the variables of the model and their optimal values: <ILP MODEL PARAMETER VALUES> Translate this solution back to the original problem and provide it as originally specified. Do not add any more explanation once you've provided the solution.

Table 8: The structures of each prompting strategy.

Prompt	I am planning a trip to visit several cities. Here are the distances between each pair of cities: City 1 and city 2 are 8 miles apart. City 1 and city 3 are 1 miles apart. City 1 and city 4 are 1 miles apart. City 2 and city 3 are 2 miles apart. City 2 and city 4 are 13 miles apart. City 3 and city 4 are 8 miles apart. What is the shortest possible route that…	I am planning a trip to visit several cities. Here are the distances between each pair of cities: City 1 and city 2 are 15 miles apart. City 1 and city 3 are 14 miles apart. City 1 and city 4 are 14 miles apart. City 2 and city 3 are 16 miles apart. City 2 and city 4 are 1 miles apart. City 3 and city 4 are 16 miles apart. What is the shortest possible route that…
Result	Optimal: 1, 3, 2, 4	Code Failure: `KeyError at line 28: (0, 0)`

Table 9: The beginnings of two ILP Python prompts that are identical except for the numeric details of the instances (as indicated by the highlighting), yet have quite different results when presented to GPT-4o.

E Generalization over numbers is brittle: Example

We show an example where changing some numbers makes the difference between a correct ILP translation and one that generates invalid Python code in Table 10.

F Full Results

Table 10 and Table 11 present full de-aggregated results from our experiments. They break down results using the result categories discussed in Section 4.3.

				One-Shot				Zero-Shot CoT				One-Shot CoT				ILP LP					ILP Python				
				O	S	E	I	O	S	E	I	O	S	E	I	O	S	E	I	F	O	S	E	I	F
🖌️ GCP	GPT	➡️	🖌️	42.0	9.3	48.7	0.0	60.7	4.0	34.7	0.7	60.0	2.7	37.3	0.0	42.0	7.3	48.0	0.0	2.7	56.0	14.0	25.3	4.7	0.0
			🏫	37.3	10.7	52.0	0.0	55.3	9.3	34.7	0.7	57.3	5.3	37.3	0.0	38.0	6.7	54.7	0.7	0.0	26.0	46.0	24.0	0.7	3.3
			💔	38.7	4.7	56.7	0.0	54.0	6.0	38.0	2.0	52.0	4.0	43.3	0.7	44.7	18.7	26.7	3.3	6.7	10.0	51.3	25.3	1.3	12.0
			🥋	31.3	18.7	50.0	0.0	53.3	14.0	30.0	2.7	56.7	3.3	40.0	0.0	19.3	13.3	58.0	0.7	8.7	0.7	0.0	0.7	0.0	98.7
		🔄	🖌️	2.7	1.3	96.0	0.0	1.3	5.3	90.7	2.7	0.7	4.7	94.7	0.0	17.3	10.0	65.3	0.0	7.3	14.7	5.3	68.7	8.0	3.3
			🏫	27.3	8.0	64.7	0.0	46.0	4.0	50.0	0.0	47.3	8.0	44.7	0.0	10.0	4.7	80.7	0.0	4.7	40.7	19.3	32.7	5.3	2.0
			💔	22.0	9.3	68.7	0.0	15.3	8.0	74.0	2.7	26.7	10.0	63.3	0.0	18.0	15.3	50.7	4.7	11.3	34.0	29.3	27.3	4.7	4.7
			🥋	10.0	6.7	83.3	0.0	4.0	2.0	93.3	0.7	14.0	6.7	79.3	0.0	7.3	18.7	68.0	2.7	3.3	0.0	0.0	8.7	10.0	81.3
	Llama	➡️	🖌️	9.3	2.7	88.0	0.0	38.7	14.0	36.7	10.7	52.0	15.3	29.3	3.3	1.3	12.7	56.0	1.3	28.7	14.0	8.7	30.7	0.0	46.7
			🏫	0.7	4.0	95.3	0.0	21.3	42.0	30.7	6.0	28.7	35.3	32.7	3.3	1.3	11.3	48.0	0.0	39.3	38.0	6.7	44.7	2.0	8.7
			💔	4.7	0.7	94.7	0.0	18.7	9.3	49.3	22.7	34.7	16.0	42.0	7.3	4.0	8.7	25.3	32.0	30.0	26.0	10.0	45.3	7.3	11.3
			🥋	4.0	1.3	94.0	0.7	22.7	27.3	40.0	10.0	34.0	23.3	37.3	5.3	2.0	11.3	44.7	0.0	42.0	40.0	2.7	22.7	2.0	32.7
		🔄	🖌️	14.0	2.0	84.0	0.0	0.0	2.0	90.7	7.3	0.0	3.3	86.7	10.0	1.3	8.0	50.0	0.7	40.0	6.7	3.3	59.3	0.0	30.7
			🏫	13.3	0.0	86.7	0.0	10.0	0.0	56.7	33.3	13.3	0.0	86.0	0.7	1.3	6.0	42.0	0.0	50.7	10.0	5.3	50.0	2.0	32.7
			💔	20.0	8.7	71.3	0.0	8.0	6.0	66.0	20.0	18.0	2.0	70.7	9.3	2.0	10.0	22.0	21.3	44.7	0.0	3.3	50.0	10.7	36.0
			🥋	19.3	3.3	77.3	0.0	8.0	4.0	78.7	9.3	11.3	2.0	79.3	7.3	0.0	6.0	57.3	0.0	36.7	0.7	0.0	26.0	0.0	73.3
🎒 KSP	GPT	➡️	🎒	22.7	68.0	9.3	0.0	48.0	44.0	2.0	6.0	50.0	35.3	14.0	0.7	98.7	0.7	0.7	0.0	0.0	89.3	3.3	7.3	0.0	0.0
			🍋	23.3	63.3	13.3	0.0	49.3	35.3	13.3	2.0	52.7	35.3	10.7	1.3	99.3	0.7	0.0	0.0	0.0	84.7	5.3	10.0	0.0	0.0
			🏛️	21.3	72.0	6.7	0.0	45.3	49.3	5.3	0.0	48.7	42.7	7.3	1.3	99.3	0.7	0.0	0.0	0.0	76.7	7.3	16.0	0.0	0.0
			🎉	17.3	62.7	20.0	0.0	44.0	52.0	2.7	1.3	42.0	52.0	5.3	0.7	100.0	0.0	0.0	0.0	0.0	84.0	4.0	12.0	0.0	0.0
		🔄	🎒	27.3	23.3	49.3	0.0	50.7	42.7	5.3	1.3	45.3	43.3	11.3	0.0	98.0	1.3	0.7	0.0	0.0	88.7	3.3	8.0	0.0	0.0
			🍋	12.0	37.3	50.7	0.0	52.7	38.0	8.0	1.3	47.3	40.7	12.0	0.0	99.3	0.0	0.0	0.0	0.7	78.7	7.3	14.0	0.0	0.0
			🏛️	9.3	23.3	67.3	0.0	27.3	50.7	17.3	4.7	24.7	58.0	17.3	0.0	98.0	0.7	1.3	0.0	0.0	74.0	8.0	18.0	0.0	0.0
			🎉	10.0	42.0	48.0	0.0	38.0	44.7	16.7	0.7	29.3	62.0	7.3	1.3	98.0	1.3	0.7	0.0	0.0	86.0	3.3	10.7	0.0	0.0
	Llama	➡️	🎒	15.3	58.7	24.0	2.0	37.3	42.7	6.7	13.3	37.3	50.0	12.0	0.7	92.0	6.0	2.0	0.0	0.0	51.3	18.7	29.3	0.7	0.0
			🍋	14.0	62.7	20.0	3.3	31.3	36.7	23.3	8.7	37.3	36.7	24.0	2.0	76.7	20.7	2.7	0.0	0.0	46.0	14.7	34.0	0.0	5.3
			🏛️	14.7	53.3	32.0	0.0	32.7	46.7	4.7	16.0	33.3	48.7	18.0	0.0	91.3	1.3	6.0	0.0	1.3	52.0	13.3	33.3	0.0	1.3
			🎉	12.7	48.7	38.7	0.0	33.3	50.7	8.7	7.3	28.0	47.3	24.0	0.7	94.0	5.3	0.7	0.0	0.0	53.3	11.3	35.3	0.0	0.0
		🔄	🎒	8.0	24.7	56.7	10.7	34.7	39.3	22.0	4.0	11.3	46.0	34.7	8.0	90.7	0.7	5.3	0.0	3.3	57.3	4.7	38.0	0.0	0.0
			🍋	8.7	22.0	64.0	5.3	29.3	42.0	20.7	8.0	13.3	34.7	45.3	6.7	77.3	2.7	10.7	0.0	9.3	47.3	7.3	38.7	0.7	6.0
			🏛️	4.7	35.3	60.0	0.0	19.3	27.3	33.3	20.0	5.3	41.3	48.0	5.3	82.7	2.7	2.0	0.0	12.7	50.0	7.3	40.0	0.0	2.7
			🎉	2.7	42.7	50.7	4.0	20.0	39.3	24.7	16.0	9.3	36.7	45.3	8.7	90.7	1.3	7.3	0.0	0.7	33.3	6.0	60.7	0.0	0.0
✈️ TSP	GPT	➡️	✈️	34.7	65.3	0.0	0.0	31.3	67.3	0.7	0.7	37.3	62.7	0.0	0.0	16.7	11.3	1.3	10.0	60.7	86.0	9.3	0.0	2.7	2.0
			📋	27.3	72.7	0.0	0.0	30.7	68.7	0.0	0.7	22.7	77.3	0.0	0.0	4.7	32.7	2.7	14.0	46.0	60.0	10.0	1.3	17.3	11.3
			📅	22.7	77.3	0.0	0.0	32.0	68.0	0.0	0.0	30.7	69.3	0.0	0.0	2.0	31.3	9.3	4.0	53.3	32.7	30.7	19.3	6.0	11.3
			🪑	29.3	70.7	0.0	0.0	26.0	72.7	0.0	1.3	31.3	68.7	0.0	0.0	2.7	40.7	0.0	0.0	56.7	54.0	44.0	0.7	0.0	1.3
		🔄	✈️	14.0	86.0	0.0	0.0	17.3	68.0	2.0	12.7	28.0	72.0	0.0	0.0	14.7	6.7	0.7	6.0	72.0	75.3	20.7	1.3	2.0	0.7
			📋	14.7	85.3	0.0	0.0	24.7	75.3	0.0	0.0	18.7	80.7	0.0	0.7	2.7	14.7	1.3	6.0	75.3	46.7	22.0	2.7	18.0	10.7
			📅	30.0	70.0	0.0	0.0	34.0	63.3	0.0	2.7	28.0	72.0	0.0	0.0	2.7	11.3	12.0	4.7	69.3	16.0	12.7	44.0	11.3	16.0
			🪑	27.3	72.7	0.0	0.0	27.3	72.7	0.0	0.0	28.0	72.0	0.0	0.0	5.3	11.3	0.0	0.7	82.7	50.7	42.0	4.7	2.0	0.7
	Llama	➡️	✈️	28.7	71.3	0.0	0.0	25.3	52.7	1.3	20.7	25.3	74.7	0.0	0.0	0.7	2.7	0.0	1.3	95.3	15.3	33.3	14.0	6.7	30.7
			📋	18.7	81.3	0.0	0.0	23.3	61.3	0.0	15.3	19.3	80.7	0.0	0.0	0.0	1.3	0.0	0.7	98.0	7.3	16.0	4.7	7.3	64.7
			📅	8.7	91.3	0.0	0.0	17.3	74.7	2.0	6.0	12.0	87.3	0.0	0.7	0.7	5.3	2.7	3.3	88.0	4.0	18.0	0.0	0.0	78.0
			🪑	16.7	83.3	0.0	0.0	18.7	76.7	0.0	4.7	20.7	78.0	0.0	1.3	0.7	8.7	2.7	0.0	88.0	0.0	5.3	0.0	0.0	94.7
		🔄	✈️	4.7	95.3	0.0	0.0	14.0	63.3	0.0	22.7	10.0	89.3	0.0	0.7	1.3	3.3	0.0	0.7	94.7	4.7	21.3	10.0	5.3	58.7
			📋	8.0	92.0	0.0	0.0	18.0	69.3	2.0	10.7	16.7	82.7	0.0	0.7	0.0	0.7	0.7	0.0	98.7	1.3	26.7	4.0	10.0	58.0
			📅	14.0	86.0	0.0	0.0	23.3	69.3	1.3	6.0	21.3	78.7	0.0	0.0	2.0	5.3	1.3	0.7	90.7	11.3	40.7	2.7	0.0	45.3
			🪑	14.0	86.0	0.0	0.0	20.7	76.0	0.7	2.7	22.0	78.0	0.0	0.0	0.7	0.7	3.3	0.0	95.3	0.0	0.0	0.0	0.0	100.0

Table 10: Full results for EHOP-RANDOM, including the ILP LP prompting strategy and a breakdown of result categories (➡️: standard, 🔄: inverted; O: optimal, S: suboptimal, E: erroneous, I: incompatible, F: ILP code failure). Costumes are represented by their emojis (established in Section 3). Greedy results do not vary by condition, and were provided in Table 1.

				One-Shot				Zero-Shot CoT				One-Shot CoT				ILP LP					ILP Python				
				O	S	E	I	O	S	E	I	O	S	E	I	O	S	E	I	F	O	S	E	I	F
🖌️ GCP	GPT	➡️	🖌️	16.0	15.0	69.0	0.0	25.0	18.0	53.0	4.0	25.0	14.0	61.0	0.0	40.0	5.0	49.0	0.0	6.0	60.0	7.0	30.0	3.0	0.0
			🏥	24.0	13.0	63.0	0.0	28.0	16.0	55.0	1.0	26.0	12.0	60.0	2.0	39.0	0.0	59.0	0.0	2.0	15.0	50.0	28.0	4.0	3.0
			💔	19.0	10.0	71.0	0.0	28.0	13.0	57.0	2.0	22.0	10.0	68.0	0.0	34.0	12.0	35.0	6.0	13.0	7.0	48.0	25.0	2.0	18.0
			🧑‍🎓	21.0	22.0	57.0	0.0	21.0	31.0	46.0	2.0	25.0	9.0	66.0	0.0	20.0	6.0	68.0	3.0	3.0	0.0	0.0	1.0	0.0	99.0
		🔄	🖌️	0.0	0.0	100.0	0.0	0.0	1.0	98.0	1.0	1.0	2.0	97.0	0.0	4.0	4.0	81.0	0.0	11.0	6.0	3.0	86.0	5.0	0.0
			🏥	8.0	8.0	84.0	0.0	23.0	9.0	68.0	0.0	33.0	15.0	52.0	0.0	4.0	1.0	94.0	0.0	1.0	42.0	10.0	36.0	11.0	1.0
			💔	6.0	7.0	87.0	0.0	2.0	5.0	93.0	0.0	3.0	18.0	77.0	2.0	10.0	11.0	58.0	10.0	11.0	37.0	34.0	28.0	1.0	0.0
			🧑‍🎓	1.0	11.0	88.0	0.0	0.0	2.0	98.0	0.0	0.0	10.0	90.0	0.0	7.0	3.0	84.0	3.0	3.0	0.0	0.0	11.0	9.0	80.0
	Llama	➡️	🖌️	1.0	1.0	98.0	0.0	7.0	32.0	48.0	13.0	16.0	33.0	44.0	7.0	1.0	11.0	56.0	3.0	29.0	2.0	9.0	25.0	1.0	63.0
			🏥	0.0	0.0	100.0	0.0	9.0	40.0	40.0	11.0	5.0	55.0	39.0	1.0	0.0	12.0	46.0	0.0	42.0	31.0	4.0	59.0	3.0	3.0
			💔	0.0	2.0	97.0	1.0	4.0	9.0	57.0	30.0	13.0	34.0	47.0	6.0	0.0	4.0	23.0	40.0	33.0	9.0	11.0	56.0	8.0	16.0
			🧑‍🎓	0.0	1.0	99.0	0.0	5.0	41.0	49.0	5.0	9.0	28.0	55.0	8.0	1.0	22.0	38.0	0.0	39.0	24.0	3.0	36.0	1.0	36.0
		🔄	🖌️	5.0	2.0	93.0	0.0	0.0	0.0	83.0	17.0	0.0	4.0	83.0	13.0	0.0	3.0	43.0	4.0	50.0	1.0	5.0	47.0	1.0	46.0
			🏥	1.0	0.0	99.0	0.0	3.0	0.0	70.0	27.0	0.0	0.0	98.0	2.0	0.0	12.0	51.0	0.0	37.0	14.0	7.0	48.0	0.0	31.0
			💔	5.0	10.0	85.0	0.0	3.0	4.0	65.0	28.0	4.0	2.0	85.0	9.0	0.0	7.0	18.0	30.0	45.0	0.0	6.0	42.0	12.0	40.0
			🧑‍🎓	5.0	2.0	93.0	0.0	2.0	7.0	80.0	11.0	0.0	2.0	95.0	3.0	0.0	4.0	59.0	1.0	36.0	0.0	0.0	11.0	2.0	87.0
🎒 KSP	GPT	➡️	🎒	8.7	67.3	24.0	0.0	18.0	72.0	2.0	8.0	14.7	68.7	16.0	0.7	99.3	0.0	0.7	0.0	0.0	92.0	3.3	4.7	0.0	0.0
			🍋	11.3	66.0	22.7	0.0	14.7	60.0	21.3	4.0	24.0	60.0	14.7	1.3	100.0	0.0	0.0	0.0	0.0	82.0	6.0	12.0	0.0	0.0
			🏛️	8.0	77.3	14.7	0.0	22.0	74.0	2.7	1.3	16.0	72.7	10.0	1.3	98.7	0.7	0.7	0.0	0.0	84.0	6.7	9.3	0.0	0.0
			🎉	13.3	60.0	26.7	0.0	28.0	64.0	4.0	4.0	32.7	59.3	8.0	0.0	99.3	0.7	0.0	0.0	0.0	86.0	4.0	10.0	0.0	0.0
		🔄	🎒	20.0	29.3	50.7	0.0	36.7	58.7	4.0	0.7	39.3	53.3	7.3	0.0	98.7	0.7	0.7	0.0	0.0	87.3	5.3	7.3	0.0	0.0
			🍋	14.7	37.3	48.0	0.0	30.0	60.7	8.0	1.3	26.7	64.0	8.7	0.7	98.0	0.0	1.3	0.0	0.7	77.3	6.0	14.7	2.0	0.0
			🏛️	13.3	14.0	72.7	0.0	26.0	54.0	15.3	4.7	28.0	62.0	10.0	0.0	98.7	0.0	1.3	0.0	0.0	78.0	5.3	16.7	0.0	0.0
			🎉	14.0	39.3	46.7	0.0	31.3	52.7	13.3	2.7	34.7	58.0	7.3	0.0	93.3	2.7	4.0	0.0	0.0	82.0	5.3	11.3	0.7	0.7
	Llama	➡️	🎒	5.3	68.0	25.3	1.3	10.7	72.7	4.7	12.0	31.3	60.0	7.3	1.3	92.7	6.7	0.7	0.0	0.0	45.3	19.3	35.3	0.0	0.0
			🍋	10.7	71.3	12.0	6.0	17.3	43.3	32.0	7.3	28.7	49.3	18.7	3.3	68.7	24.0	5.3	0.7	1.3	36.7	17.3	40.7	0.0	5.3
			🏛️	9.3	57.3	33.3	0.0	13.3	64.0	4.7	18.0	21.3	64.0	14.7	0.0	92.0	4.0	4.0	0.0	0.0	49.3	16.0	33.3	0.0	1.3
			🎉	11.3	52.0	36.7	0.0	16.7	68.7	6.7	8.0	26.7	50.7	22.0	0.7	92.0	6.7	1.3	0.0	0.0	46.7	10.7	42.7	0.0	0.0
		🔄	🎒	14.7	18.7	54.0	12.7	26.0	54.0	14.7	5.3	14.0	42.7	36.0	7.3	88.0	0.7	7.3	0.0	4.0	53.3	11.3	35.3	0.0	0.0
			🍋	19.3	8.7	68.7	3.3	22.0	56.7	17.3	4.0	16.7	32.0	49.3	2.0	68.7	6.7	10.7	0.0	14.0	45.3	11.3	36.7	0.7	6.0
			🏛️	11.3	23.3	64.7	0.7	20.7	35.3	29.3	14.7	10.0	32.7	55.3	2.0	83.3	4.7	2.7	0.0	9.3	56.0	8.0	35.3	0.0	0.7
			🎉	8.7	28.7	54.7	8.0	26.7	32.7	26.0	14.7	13.3	33.3	45.3	8.0	87.3	0.7	10.0	0.0	2.0	36.0	14.7	49.3	0.0	0.0
✈️ TSP	GPT	➡️	✈️	15.3	84.7	0.0	0.0	24.7	74.0	0.0	1.3	20.7	78.0	1.3	0.0	12.7	10.7	1.3	12.7	62.7	87.3	11.3	0.7	0.7	0.0
			📋	13.3	86.7	0.0	0.0	22.7	77.3	0.0	0.0	8.0	92.0	0.0	0.0	6.0	30.0	4.0	12.0	48.0	59.3	13.3	3.3	11.3	12.7
			📅	18.0	82.0	0.0	0.0	15.3	82.7	0.7	1.3	14.0	86.0	0.0	0.0	5.3	28.7	7.3	4.7	54.0	34.7	24.7	18.7	7.3	14.7
			🪑	9.3	90.7	0.0	0.0	25.3	74.0	0.0	0.7	5.3	94.7	0.0	0.0	3.3	40.7	0.0	0.0	56.0	66.7	29.3	4.0	0.0	0.0
		🔄	✈️	10.7	89.3	0.0	0.0	18.0	70.0	0.7	11.3	16.0	84.0	0.0	0.0	14.0	6.7	0.0	6.0	73.3	74.7	20.0	3.3	2.0	0.0
			📋	8.0	92.0	0.0	0.0	21.3	76.0	0.0	2.7	8.7	90.0	0.0	1.3	1.3	7.3	3.3	3.3	84.7	35.3	26.0	2.7	21.3	14.7
			📅	8.7	91.3	0.0	0.0	15.3	82.7	0.7	1.3	3.3	96.0	0.0	0.7	3.3	8.0	15.3	2.0	71.3	19.3	14.0	40.7	12.0	14.0
			🪑	10.7	89.3	0.0	0.0	19.3	78.0	0.0	2.7	3.3	96.7	0.0	0.0	3.3	10.7	0.7	0.0	85.3	58.7	34.7	3.3	2.0	1.3
	Llama	➡️	✈️	8.0	92.0	0.0	0.0	12.0	62.7	1.3	24.0	6.0	94.0	0.0	0.0	1.3	2.7	0.0	0.0	96.0	13.3	36.0	8.0	6.7	36.0
			📋	5.3	94.7	0.0	0.0	7.3	72.7	0.0	20.0	6.7	93.3	0.0	0.0	0.0	2.7	0.0	0.0	97.3	8.0	18.0	2.7	9.3	62.0
			📅	3.3	96.7	0.0	0.0	9.3	80.7	2.0	8.0	4.7	94.7	0.0	0.7	0.0	4.7	2.0	2.7	90.7	3.3	20.0	0.0	0.7	76.0
			🪑	5.3	94.7	0.0	0.0	6.0	90.7	0.0	3.3	4.7	95.3	0.0	0.0	0.7	7.3	0.7	0.0	91.3	0.0	4.7	0.0	0.0	95.3
		🔄	✈️	1.3	98.7	0.0	0.0	6.7	74.0	0.7	18.7	3.3	96.0	0.0	0.7	0.7	5.3	0.0	0.7	93.3	6.0	14.0	8.0	8.7	63.3
			📋	5.3	94.7	0.0	0.0	5.3	75.3	0.7	18.7	8.0	90.7	0.0	1.3	0.0	0.0	0.7	0.7	98.7	2.7	22.0	5.3	7.3	62.7
			📅	5.3	94.7	0.0	0.0	9.3	80.7	2.7	7.3	4.7	95.3	0.0	0.0	0.0	4.0	1.3	0.0	94.7	14.0	40.0	1.3	0.7	44.0
			🪑	7.3	92.7	0.0	0.0	6.0	90.0	0.7	3.3	4.7	95.3	0.0	0.0	0.7	2.7	1.3	0.7	94.7	0.0	0.7	0.0	0.0	99.3

Table 11: Full results for EHOP-HARD, with formatting matching that of Table 10. Greedy results do not vary by condition, and were provided in Table 3.