Holy Grail 2.0: From Natural Language to Constraint Models

Dimos Tsouros
KU Leuven, Belgium

Hélène Verhaeghe
KU Leuven, Belgium

Serdar Kadıoğlu
AI Center of Excellence, Fidelity Investments, USA
Department of Computer Science, Brown University, USA

Tias Guns
KU Leuven, Belgium

Abstract

Twenty-seven years ago, E. Freuder highlighted that "Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it". Nowadays, CP users have great modeling tools available (like Minizinc and CPMpy), allowing them to formulate the problem and then let a solver do the rest of the job, getting closer to the stated goal. However, this still requires the CP user to know the formalism and respect it. Another significant challenge lies in the expertise required to effectively model combinatorial problems. All this limits the wider adoption of CP. In this position paper, we investigate a possible approach to leverage pre-trained Large Language Models to extract models from textual problem descriptions. More specifically, we take inspiration from the Natural Language Processing for Optimization (NL4OPT) challenge and present early results with a decomposition-based prompting approach to GPT Models.

2012 ACM Subject Classification Computing methodologies → Discrete space search; Computing methodologies → Supervised learning by classification; Theory of computation → Constraint and logic programming

Keywords and phrases Model learning, Constraint learning, Modelling, NLP, NL4CP, NL4OPT, NER4OPT

Funding This research received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant No. 101002802, CHAT-Opt)

1 Introduction

In 1996, Eugene Freuder highlighted that "*Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.*" [5]. Twenty-seven years later, current technologies like modeling languages (such as Minizinc [12] or Essence [6]) and modeling libraries (like CPMpy [8]) have proven that indeed, constraint programming has the capability to achieve this holy grail of programming.

However, there is still a gap between a natural formulation of the problem (in Natural Language) and a CP model. The user is still responsible for transforming the problem at hand as an optimization model, to model it as a set of variables with their domains, a set of constraints, and an objective function. This is not always trivial and requires quite some expertise, and it is considered a bottleneck for the wider adoption of CP.

Today, with the astonishing recent advances in Large Language Models (LLMs), isn't it the time to think ahead, toward the holy grail 2.0, where the user could use natural language to define a problem, and the system would be able to understand and automatically extract the formal model required by the solvers?

Inspired by the developments in the area of NL4OPT, and especially the recent advancements in the use of LLMs to process natural language, recent works have started investigating methods for

automating the "natural language to optimization model" process using LLMs, to be more efficient and accessible to non-experts. The whole process of modeling a problem is usually split into different subtasks, as it is a multi-step process, and LLMs usually find it hard to tackle such tasks.

In [3], the NER4OPT subtask is formalized, as an interdisciplinary problem at the intersection of Named Entity Recognition (NER) and Combinatorial Optimization. The differences from standard NER tasks are discussed, and a method to tackle this subtask is proposed, using classical techniques based on morphological and grammatical properties combined with modern methods leveraging pre-trained LLMs. A system that tries to close the modeling loop and automatically formulate optimization models from problem descriptions is proposed in [15]. This system uses a two-step approach. First, the problem description is transformed into an intermediate representation, which is very similar to the NER4OPT task, i.e., labeling the entities of the problem. Then, in the second step, this intermediate representation is used to formally formulate the optimization problem. This system is built upon the BART language model [10].

In 2022, the NL4Opt competition [16] took place at the NeurIPS conference. They targeted the two tasks described above, i.e., recognizing the semantic entities corresponding to the optimization problem (i.e., variables, values, objective) and generating a meaningful representation (i.e., identifying the actual relations between variables, the associations of the domains and variables,...). For tackling the second subtask, the system from [7] managed to achieve the best results by splitting it again into subtasks, i.e. first find the relations between the entities of the problem, and only then continue with the *formulation* task, as it was shown that LLMs can exploit this information to improve their performance. Again, BART was used in this work.

The target formulation for the mentioned works focused mainly on creating linear programming problems. A first approach for modeling constraint problems using CP, involving also the translation to the Minizinc modeling language [12], was presented in [1]. The described system uses a single-step approach for the modeling, followed by an automated fixing process, that compiles and debugs the generated model in the Minizinc language. The early results are promising, however, in some cases the specifications of the optimization problem are not achieved, due to the one-step modeling method.

Inspired by the developments in the area of NL4OPT, the recent advancements in the use of LLMs (such as GPT-3.5), and the recent works that show promising results, in this position paper we propose a modular step-by-step framework to model a problem based on the text description. In this framework, we combine the benefits of the mentioned systems, by splitting the modeling task into four subtasks that have been shown to boost the performance of LLMs. The system takes as input the description in natural language, and then the modeling process starts. First extract the problem's entities (variables, domains, constraints, objective), then find the relations between the entities, use the entities and their relations to formalize it as an optimization problem, and translate it into a constraint model language (such as CPMpy). After the modeling is finished, the resulting code is compiled and run, and automatically debugged if necessary. Finally, the system will interact with the user to refine the model.

The rest of the paper is structured as follows:In Section 2 we discuss our proposed method for automatic modeling by exploiting the emerging abilities of LLMs and we describe the different subtasks. Section 3 focuses on how LLMs can be leveraged and discusses techniques to boost them through prompt engineering. Then, we discuss the different levels of abstraction of problem descriptions that we want to tackle (Section 4). In Section 5 we give an example of how our system works. Section 6 summarizes our paper and gives some directions for future work.

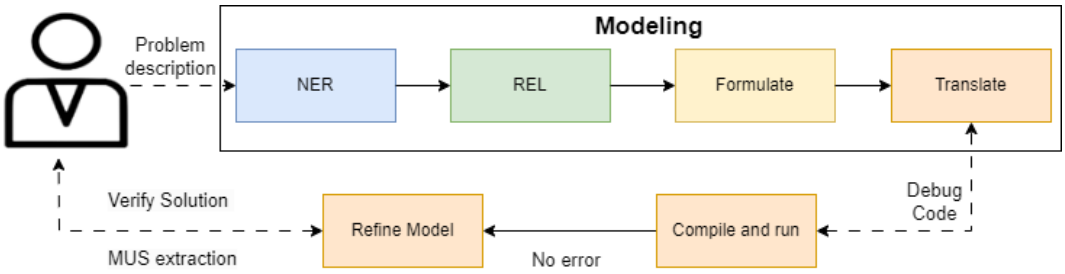

Figure 1 The framework proposed

2 Exploiting LLMs for converting natural language to constraint models

Figure 1 shows the high-level loop for modeling constraint problems from text descriptions. The modeling task can of course be done altogether using LLMs, as in [1]. However, the modeling task is not a trivial task and consists of many subtasks, as discussed in [13, 3], and LLMs find it difficult to handle difficult multi-step tasks in one step [14]. This is confirmed by early experiments with the one-step approach. We hence propose a decomposition-based approach, inspired by recent work in NL4OPT. In our proposed approach the system splits the modeling task into the four following steps:

1. **NER4OPT** The modeling system has to extract the semantic entities of the optimization problem, i.e. the parameters, variables, domains, constraints, and objective. Note that this task differs significantly from the classical NER in NLP due to its multi-sentence dependency with high-level of ambiguity, low data regime with high-cost of annotation, and inherent aleatoric uncertainty. It is coined as Ner4Opt, an interdisciplinary problem at the intersection of NLP and Optimization and studied in detail for its lexical, semantic, and hybrid solutions [3] using large-language models as well as fine-tuning with optimization corpora.
2. **REL** Note that, in NER4OPT the goal is to label words in the description using predefined optimization entities. However, the relations between the entities are not yet found, e.g. which variables are in the scope of each constraint. In the second task, the system has to identify the relations between the various entities extracted in the previous step, link the values with the type of parameters, find the domains associated with the variables, and identify the scopes of the constraints. This subtask is similar to the first step from [7], which takes as input the labeled entities and splits the process of formulating the optimization problem into two steps.
3. **Formulation** The third subtask of our framework is to formulate the problem as a constraint problem, given the list of labeled entities and their relation, in a formal way. This subtask is again inspired by [7], as it follows their step of finding the relations of the constraints.
4. **Translation** Then, fourth, the system would take as input the problem formulation from the previous step and translate it into code, following the syntax of the desired modeling language, as in [1]. Given a clear description of the coding task, with the required entities, constraints etc., LLMs have been shown to be effective in code-writing [22].

After the modeling process is finished, with the output being the code in any modeling language desired, we still cannot be sure that the extracted model is correct. First of all, the code given could contain errors either in compilation or in runtime. In addition, the model could have missed some constraints or modeled some wrongly. Finally, the user may have given an unsatisfiable problem and wants to refine it. Hence, we propose two additional steps to close the loop.

1. **Fixing the output** The fifth step would be compiling (and running) the code, and identifying any errors. In this case, the system automatically corrects the errors in a loop, until a bug-free code is extracted. This step is inspired by the method presented in [1]. It has been shown that LLMs can be exploited for bug-fixing with good results [19].
2. **Refining the model** Finally, the last step would be the presentation of the final model and potential solution(s) to the user. The user would then be able to verify the solution or request Minimal Unsatisfiable Subset(s) (MUS) and/or explanations in case there is no solution. Interactions with the user at that step would be done in order to provide corrections or precisions to the model in order for the user to obtain the desired solution.

With such a modular process, our goal is also to allow people to adapt our process to their needs. One of the use cases is for the 4th step. We plan on targeting the CPMpy modeling language to execute the transformation from the formal model info code. However, another user could want the system to output Minizinc code. In that case, they would have just to replace this module. Also, we should be able to change the LLM used in an easy way, or use hybrid methods, e.g. using the method from [3] in the first step.

3 Leveraging LLMs

Our framework will be highly using LLMs, being also capable to use specialized tools to tackle some subtasks, like for the NER4OPT task [3] Multiple LLMs will be tested such as GPT variations or LLAMA. Prompt engineering (also known as in-context learning), which is the task of carefully designing prompts to better leverage LLMs capabilities, has been shown to significantly boost performance on several tasks [17]. There are many different ways to use prompt engineering for a task on hand. We mainly experimented and included in our system the following:

- **Roles and goals** [18]. One efficient way to get better results using prompt engineering is to specify the role the LLM has to play and the goal to achieve. The intent of this technique is to localize the training of the LLM for the specific task on hand, selecting what types of output to generate and what details to focus on. Some LLMs have also specific internal tools that can help to achieve that in a better way. For example, for the GPT models, we can make use of the system/user prompt duality. The system prompt allows us to set up the role and goals, i.e., we can give a statement such as "Assume you are a combinatorial optimization expert and you need to model a combinatorial problem as an optimization problem". Then the user prompts are used to converse with it.
- **Few-shot learning** [2] will be very useful to boost the performance of LLMs, by giving examples of the task on hand in each step and how to solve it. In addition, it can be used for specifying the formatting we require for the output when formulating the formal model, for example. For instance, the extractions from pre-trained Ner4Opt models can be used to automate the few-shot example generation for in-context learning.
- **Chain-of-thought** [21] prompting techniques take prompting one step further, allowing models to decompose multi-step problems into intermediate steps. This means that additional computation can be allocated to problems that require more reasoning steps than simple input-process-output scenarios. It has been shown that even zero-shot Chain-of-thought prompts can be very efficient, especially in symbolic tasks.
- **Tree of Thoughts** [11, 23] techniques are inspired by the human mind's approach to solving complex reasoning tasks through trial and error. In this process, the system explores the solution space through a tree-like process, using backtracking when needed. This helps to generate alternative outputs for a given prompt, choosing the best one, in order to ensure better results.

- **Plan-and-Solve** [20]. Despite the remarkable success of Zero-shot Chain of Thoughts in solving multi-step reasoning tasks, there are still many cases where Intermediate reasoning step(s) are missed, leading to worse performance. This happens especially when there are many steps involved in the task on hand. Plan-and-Solve was proposed to alleviate this issue. This method consists of 2 different steps: First, the system is asked to focus on devising a plan dividing the entire task, without the need to solve the problem. Then, in a separate step, it is asked to carry out the subtasks according to the plan.

Note that, as the ways these techniques are used are not task-specific, they can be included in a system that automatically transforms the prompt given by the user to a new prompt using the above techniques. So, the real user of such a system does not have to focus on prompt engineering and just on the task on hand, in our case on modeling the problem.

4 Levels of abstraction

In addition to using existing datasets, such as the one from the NL4Opt competition [16], one of our goals is also to measure how well the framework is able to model and recognize problems. To evaluate this, our plan is to evaluate the system in various levels of abstraction.

1. The first level of abstraction would be the easiest, where the problem name would be clearly stated (i.e. use of the words tsp, knapsack, graph coloring,...), and the constraints and variables already identified using tokens. This is the baseline.
2. In the second level, we would omit the name of the problem while still explicitly describing the variables, constraints, and parameters of the problem. This level will help us see if problems are recognized.
3. In the third level, the known lexical of modelization (i.e. words like constraint, variables, domains,...) would be removed. Here, our wish is to measure how well the components of the model can be without being pre-identified using tokens.
4. The fourth level would be the more abstract one. Some parameters could be implicit and not numerical for example. This would be the closest to what any human unaware of what an optimization problem is would do. This final level allows us to really see whether LLMs can do something when the structure of the problem disappears totally.

It is to be expected that the more abstract we are in the problem descriptions, the more difficult it would be for the LLMs and the more interactions it would need with the user to get the perfect model and solution. Simple examples of the different levels of abstraction are given in Example 1.

▶ **Example 1.** Problem descriptions with different levels of abstraction for the same model:

(L1) *I wish to solve a Knapsack problem, where I have 5 items, and so 5 binary variables to tell me which item is in or not. The weights of my items are 2, 3, 7, 4, and 1. The utilities of my items are 2, 3, 1, 2, and 3. The limit of weight is 10. Can you model it?*

(L2) *I wish to solve a problem, where I have 5 items, and so 5 binary variables to tell me which item is in or not. The weights of my items are 2, 3, 7, 4, and 1. The utilities of my items are 2, 3, 1, 2, and 3. I wish to limit the total weight to 10. Also, I wish to maximize the utility of the objects I take. Can you model it?*

(L3) *I wish to go on vacation. The airport only allows 10 kg for my suitcase. I have 5 items. The weights of my items are 2, 3, 7, 4, and 1. The utilities of my items are 2, 3, 1, 2, and 3. Can you tell me what items I should take in order to pass the best vacation?*

(L4) *I wish to go on vacation. The airport only allows 10 kg for my suitcase. I have 5 items: my ski combination, weighing 7 kg, some warm clothes, weighing 4 kg, some hiking boots, weighing 3 kg, a book on hiking, of 1 kg, and some umbrella, of 2 kg. As I'm going hiking, I think my boots and my book are really important, while the ski combination would not help me well. Can you tell me what items I should take in order to pass the best vacation?*

5 Usage Example

Here we provide examples of the input-output of our system. The system utilized here uses GPT-3.5, with prompt engineering for all different subtasks. We use CPMpy as the modeling language to translate in the 4th step. We will use as examples the descriptions used in Example 1 for the first (Figure 2) and fourth (Figure 3) levels of abstraction, so we can examine the resulting models in the two extreme cases.

As we can see, in both cases the resulting model is correct, and the code provided is working. In the first case, where the description of the problem directly describes the problem as a knapsack problem, giving the specific weights and utilities, it was easier to extract the respective parameters of the model. However, in the second case, the problem type is not directly given, and neither are the utility values. The system is still able to extract the entities of the problem correctly, connect the variables with the respective weights, build the knapsack constraint correctly, and, importantly, infer utility values to build an objective function.

6 Summary and future work

The Holy Grail of computer science has been defined as "the user states the problem, the computer solves it". With the help of the rich modeling languages developed recently, we are now closer to the stated goal. However, the gap between the natural description of a problem and its formal formulation as an optimization problem is still there. The goal of this position paper is to outline our project of developing a framework able to take a description of a problem in natural language, model the problem, and finally turn it into an actual runnable constraint programming model. This framework would make use, among other things, of large language models such as GPT-3.5 or LLaMa in order to extract the features, identify the relation, formulate the problem, and then run it. The first tests show great potential for formalizing good constraint programming models, even when some parameters of the problem are only given in an abstract way.

Future work should explore the usage of other LLMs, and also specialized methods for each subtask, like the ones in [3, 7]. We can also exploit more optimization domain knowledge in prompt tuning, in-context learning, and fine-tuning. Fine-tuning LLMs for specific tasks has been shown to improve them [4]. Initial examples of fine-tuning LLMs with optimization-specific corpora show great potential [3] and we would like to extend those results across the entire pipeline from text to final model. In addition, (soft) prompt tuning [9] can significantly improve the results, having been shown to outperform few-shot learning. Finally, different ways of interacting with the user should be examined, to correct system or user mistakes in a minimum number of interactions.

References

1 Boris Almonacid. Towards an automatic optimisation model generator assisted with generative pre-trained transformer. *arXiv preprint arXiv:2305.05811*, 2023.

2 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.

I wish to solve a Knapsack problem, where I have 5 items, and so 5 binary variables to tell me which item is in or not. The weights of my items are 2, 3, 7, 4, and 1. The utilities of my items are 2, 3, 1, 2, and 3. The limit of weight is 10

Parameters:
- n = 5 (number of items)
- weights = [2, 3, 7, 4, 1] (list of weights of each item)
- utilities = [2, 3, 1, 2, 3] (list of utilities of each item)
- limit = 10 (maximum weight allowed in the knapsack)

Variables:
- x[i] for i in range(n) (binary variable indicating whether item i is selected or not) with domain x[i] in 0, 1 for all i in range(n)

Constraints:
- The sum of the weights of the selected items must be less than or equal to the limit: sum(weights[i] * x[i] for i in range(n)) <= limit

Objective function:
- Maximize the total utility of the selected items: sum(utilities[i] * x[i] for i in range(n))

```
from cpmpy import *

# Parameters
n = 5
weights = [2, 3, 7, 4, 1]
utilities = [2, 3, 1, 2, 3]
limit = 10

# Variables
x = boolvar(shape=n)

# Constraints
model = Model(
   sum(weights[i] * x[i] for i in range(n)) <= limit
)

# Objective function
objective = sum(utilities[i] * x[i] for i in range(n))
model.maximize(objective)

model.solve()
```

Figure 2 Example with the 1st level of abstraction

I wish to go on vacation. The airport only allows 10 kg for my suitcase. I have 5 items: my ski combination, weighing 7 kg, some warm clothes, weighing 4 kg, some hiking boots, weighing 3 kg, a book on hiking, of 1 kg, and some umbrella, of 2 kg. As I'm going hiking, I think my boots and my book are really important, while the ski combination would not help me well. Can you tell me what items I should take in order to pass the best vacation?

Parameters:

- Maximum weight allowed for the suitcase: 10 kg
- Weight of the ski combination: 7 kg
- Weight of the warm clothes: 4 kg
- Weight of the hiking boots: 3 kg
- Weight of the book on hiking: 1 kg
- Weight of the umbrella: 2 kg

Variables:

- Binary variable x1: 1 if the ski combination is selected, 0 otherwise
- Binary variable x2: 1 if the warm clothes are selected, 0 otherwise
- Binary variable x3: 1 if the hiking boots are selected, 0 otherwise
- Binary variable x4: 1 if the book on hiking is selected, 0 otherwise
- Binary variable x5: 1 if the umbrella is selected, 0 otherwise

Constraints:

- The sum of the weights of the selected items cannot exceed the maximum weight allowed for the suitcase: x1*7 + x2*4 + x3*3 + x4*1 + x5*2 <= 10

Objective function:

- The objective is to maximize the importance of the selected item. We can introduce a weight for each item to represent its importance. Let w1 = 1 for the ski combination, w2 = 2 for the warm clothes, w3 = 3 for the hiking boots, w4 = 4 for the book on hiking, and w5 = 2 for the umbrella. Then, the objective is to maximize the weighted sum of the selected items: w1*x1 + w2*x2 + w3*x3 + w4*x4 + w5*x5.

```
from cpmpy import *

# Parameters
max_weight = 10
weights = [7, 4, 3, 1, 2]
importance = [1, 2, 3, 4, 2]

# Variables
x = boolvar(shape=5)

# Constraints
model = Model(sum(x * weights) <= max_weight)

# Objective function
objective = sum(x * importance)
model.maximize(objective)

# Solve the problem
solution = model.solve()
```

Figure 3 Example with the 4th level of abstraction

3 Parag Pravin Dakle, Serdar Kadıoğlu, Karthik Uppuluri, Regina Politi, Preethi Raghavan, SaiKrishna Rallabandi, and Ravisutha Srinivasamurthy. Ner4opt: Named entity recognition for optimization modelling from natural language. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 299–319. Springer, 2023.

4 Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*, 2020.

5 Eugene Freuder. In pursuit of the holy grail. *ACM Computing Surveys (CSUR)*, 28(4es):63–es, 1996.

6 Alan M Frisch, Warwick Harvey, Chris Jefferson, Bernadette Martínez-Hernández, and Ian Miguel. E ssence: A constraint language for specifying combinatorial problems. *Constraints*, 13:268–306, 2008.

7 Neeraj Gangwar and Nickvash Kani. Highlighting named entities in input for auto-formulation of optimization problems. *arXiv preprint arXiv:2212.13201*, 2022.

8 Tias Guns. Increasing modeling language convenience with a universal n-dimensional array, cppy as python-embedded example. In *Proceedings of the 18th workshop on Constraint Modelling and Reformulation at CP (Modref 2019)*, volume 19, 2019.

9 Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

10 Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

11 Jieyi Long. Large language model guided tree-of-thought. *arXiv preprint arXiv:2305.08291*, 2023.

12 Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

13 Yuting Ning, Jiayu Liu, Longhu Qin, Tong Xiao, Shangzi Xue, Zhenya Huang, Qi Liu, Enhong Chen, and Jinze Wu. A novel approach for auto-formulation of optimization problems. *arXiv preprint arXiv:2302.04643*, 2023.

14 Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

15 Rindranirina Ramamonjison, Haley Li, Timothy T Yu, Shiqi He, Vishnu Rengan, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. Augmenting operations research with auto-formulation of optimization models from problem descriptions. *arXiv preprint arXiv:2209.15565*, 2022.

16 Rindranirina Ramamonjison, Timothy T Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, et al. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. *arXiv preprint arXiv:2303.08233*, 2023.

17 Laria Reynolds and Kyle McDonell. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–7, 2021.

18 Douglas C Schmidt, Jesse Spencer-Smith, Quchen Fu, and Jules White. Cataloging prompt patterns to enhance the discipline of prompt engineering.

19 Dominik Sobania, Martin Briesch, Carol Hanna, and Justyna Petke. An analysis of the automatic bug fixing performance of chatgpt. *arXiv preprint arXiv:2301.08653*, 2023.

20 Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*, 2023.

21 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.

22 Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, pages 1–10, 2022.

23 Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.