Scaling Combinatorial Reasoning with Conflict-Driven Clause Learning In-Context

Yoshiki Takashima[1] **Ferhat Erata**[2]
[1]Yale Law School [2]Yale University

Abstract

Recent advances in reasoning models have led to impressive performance in several areas of reason: first-order logic, mathematics, and computer programming to name a few. Yet these models do not scale to combinatorial reasoning tasks. These tasks require the model to find a solution out of a combinatorially large search space while reasoning correctly at each step of the search. Existing techniques for such problems are neuro-symbolic: they translate the problem into a formal representation and use symbolic solvers to conquer the combinatorial search space. These approaches are often limited to rigid reasoning tasks that have exact formal translations and even then, translation incurs an error overhead, leading to lower performance.

We propose *CDCL-IC*, a Chain-of-Thought (CoT) reasoning technique that conquers combinatorial search problems through Conflict-Driven Clause Learning (CDCL). Our technique uses CDCL to learn bad patterns during backtracking CoT reasoning and blocks it using in-context learning to prune the search space. We implement *CDCL-IC* for Sudoku, and show that our approach greatly outperforms both traditional CoT and `o3-mini` on 9x9 Sudoku problems. [1]

1 Introduction

Reasoning models are the driving force behind recent advances in AI [1–5]. These models, which break down problems into steps and solve them incrementally enable natural-language reasoning on topics such as mathematics, computer programming, and logic [6–9]. These models perform filtering and search over these reasoning steps, allowing it to search through multiple chains of thoughts until a suitable candidate is reached.

While reasoning models perform well on standard deductive reasoning tasks that require multiple steps of correct reasoning, combinatorial reasoning problems such as Boolean Satisfiability (SAT) [10] and Integer Linear Programming (ILP) [11] present another challenge beyond step-wise accuracy. In these problems, a series of locally valid steps can lead to a state where the problems are unsolvable. Therefore, these problems require backtracking search to solve. Existing techniques for handling combinatorial reasoning are neuro-symbolic. To avoid searching in natural language CoT, these techniques use language models to convert the problem into a formal representation such as executable programs [12–14] or logic solver constraints. This translation comes at a cost. The problem domain needs to neatly map to existing search paradigms like SAT or ILP, and the translator accuracy seldom rises above 85% in recent works [15].

Conquering combinatorial reasoning is a classic task in symbolic AI [16, 17, 10]. Symbolic AI solvers for combinatorial reasoning problems such as SAT and ILP employ online learning techniques to prune the search space. We focus on CDCL, a technique used in SAT solvers. CDCL works by incrementally solving a problem with locally correct steps. If these steps make the problem unsolvable (conflict), the solver performs an analysis of the steps to identify a pattern of steps that lead to a conflict. This pattern is learned so that the solver can prune a search if this pattern occurs. We take inspiration from CDCL in SAT solvers and implement it in the scope of in-context learning.

[1]`mailto:yoshiki.takashima@yale.edu`

Preprint. Under review.

Our reasoning technique starts off by incrementally solving the problem in steps. The step-generating model labels each step as one of two labels: free choices and deduced steps. A reward model checks each step as valid. While solving combinatorial problems, it is possible to make all valid local steps but end up in an unsolvable state. Once the problem becomes unsolvable, we perform an analysis of the free choices and deduced to learn a pattern of unsolvable choices. This learned pattern is appended to the context of the reward model which is fine-tuned to take these patterns as instructions of which partial boards to prune. With the bad pattern pruning large number of boards, the solver approaches the solution more efficiently.

The rest of the paper proceeds as follows. First, we discuss the background in both neural and symbolic AI on which our approach stands. Then, we provide a description of our technique, followed by an evaluation, and discussion on the applicability of our approach to broader combinatorial reasoning tasks. Our contribution is as follows

- *CDCL-IC*: a technique to scale CoT reasoning to combinatorial search problems.
- An implementation of *CDCL-IC* for Sudoku, an NP-Complete combinatorial reasoning problem.
- A preliminary evaluation of *CDCL-IC* against `o3-mini` and an ablation analysis of *CDCL-IC* demonstrating the effectiveness of the approach at pruning the search space.

2 Background

Implication Graphs. In discrete combinatorial reasoning, we can divide moves into 2 disjoint categories. First of these are choices like filling a Sudoku cell with 5 when filling it with 2 or 1 would have been equally good. Second are deductions, which are forced. For example, if the row of a Sudoku cell contains every number but 9, then the only option for that cell is 9.

Once every move is categorized into choices and deductions, we can construct an implication graph [16]. Implication graphs are directed graphs such that each edge is one constraint that forms part of the deduction and nodes are assignments like filling a cell. All choices have no incoming edges because they are chosen, not deduced. Every deduction will have incoming edges from other nodes that forced that choice. For example, if 9 is the only number to fill a Sudoku cell because all other numbers are used on its row, then the there will be an directed edge from every other assignment on the row to node representing the cell being filled with 9.

Learning Clauses with First Unique Implication Point The instrumental value of implication graphs is in deriving clauses. These clauses are patterns that every solution must obey. Simply blocking the immediate pattern does not prune the search space by much because it would be very specific to that particular set of choices. This is especially true of for combinatorial problems like SAT or Sudoku where most of the choices happen at the early steps and the rest of the steps are deductions that are forced by the early choices. In these cases, we need to traverse up the implication graph until choices are reached. This ensures that we block as general patterns as possible. The first point of encountering choices is called the first unique implication point [16, 18].

In-Context Learning The performance of language models can be improved by giving examples in the context window [19]. In-context learning enables changes to model behavior on the fly without training or fine-tuning. We use this feature of language models to adjust the behavior and enforce the patterns found by clause learning.

3 Technique

We present *CDCL-IC*, the technique for scaling reasoning models to combinatorial problems.

3.1 Overview of the Algorithm

The overview of *CDCL-IC* is given in Figure 1. Since *CDCL-IC* augments reasoning models, the inputs and outputs are the same. The input is a combinatorial reasoning problem specified in natural language. Given the prompt, we construct a tree of natural-language steps while ranking them with a reward model. The reward model ensures that each step is locally valid. The tree is expanded until

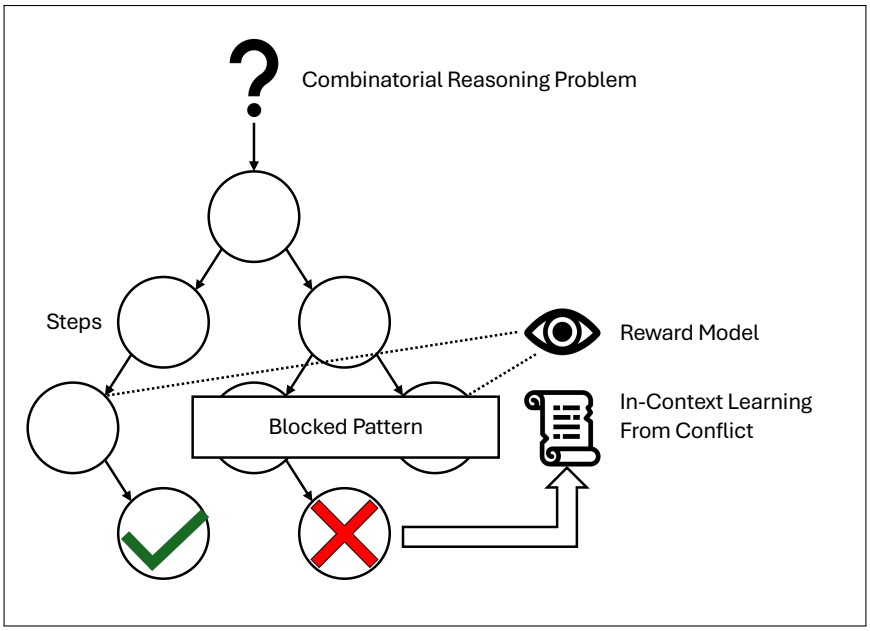

Figure 1:

the solution is found. The difference between traditional reasoning models and *CDCL-IC* is how it deals with failures. Once a failure is encountered, *CDCL-IC* analyzes the reasoning steps taken to reach that point to construct an implication graph. From this implication graph, we derive a pattern that leads to failure. *CDCL-IC* allows the reward model prune the search space further by learning this negative pattern in-context and reflecting it on the reward of the steps.

3.2 Stepwise Search for a Solution

CDCL-IC is compatible with models that reason step-by-step. The base model is executed to a chain of steps that solve the problem. We continue to execute until it gets stuck with the choices made. At that point, an implication graph is constructed to determine the pattern of choices that led to the model getting stuck.

Constructing the implication graph requires that the steps be labeled to trace the dependencies between them. Choices made from more than one available option should be labeled as such. Deductions, where previous choices or the configuration of the problem force only one option also labeled, with directed edges indicating past choices and inputs that forced this result. We demonstrate this with an example from 9x9 Sudoku.

Choices. Consider the model filling a particular cell whose row contains $\{1, 2\}$, column $\{3, 4\}$ and square contains $\{5, 6\}$. The set of possible numbers for this cell are $\{7, 8, 9\}$. Picking once from the set of $\{7, 8, 9\}$ is a choice. Note that, despite being constrained by other cells, we consider this a free choice. This contrasts with SAT, where boolean variables can only take two value, and thus every restriction on the value will force the other value.

Deductions. Now instead, suppose that the row contains $\{1, 2, 7, 8\}$. The only option left for the cell is 9. When only one option is available, we label this as a deduction. For deductions we keep the list of other choices and variables that constrained the choice. For example, we would keep the row cells $\{1, 2, 7, 8\}$, column cells $\{3, 4\}$ and square cells $\{5, 6\}$ as constraining the value to 9.

Conflicts/Stuck. Sometimes, there are no options. Consider the board from "Deductions" but with the column containing $\{3, 4, 9\}$. In this case, no value can fill the cell because the only option, 9, is no longer available. We detect this when every choice results in an step rejected by the reward model. At this point, we construct the implication graph and resolve the pattern of bad choices that yielded this state. This step is done using the classic technique from SAT, which we elide for brevity.

3.3 In-Context Tunable Reward Model

Now that a new pattern of steps found, we need to allow the reward model to take it into account when computing the reward for the steps. While it would be possible to add one particular bad pattern to the training data of the reward model, the number of bad patters is also exponential and thus it is not possible to cover meaningful set of them by re-training. Instead, we propose to make the reward model tunable to the bad patterns by putting them in the context and asking the reward model to detect that bad pattern. This turns the problem of learning an exponential set of bad patterns into a detection problem given a single bad pattern.

3.4 Adjusting the Reward

Now that we learned a new pattern of bad steps, we need to incorporate it into a way of rewarding the reasoning steps. Using the In-context tunable reward model, we output $n + 1$ scores between 0 and 1. The first score r_0 is estimates the probability that this move obeys the rules of the problem. The next n estimate the probability that the board doesn't contain the i-th bad pattern where $i \in \{1, ldots, n\}$. We discount the initial naive reward r_0 with the blocked patterns. Finally, since only one bad pattern needs to match, we take the minimum reward $\min(r_0 * r_{b_1}, \ldots, r_0 * r_{b_n})$ where $b_1, \ldots, b_n$ are the learned patterns.

4 Evaluation

We present an evaluation of *CDCL-IC* on the following research questions (RQs) using the NP-Complete problem Sudoku [**?**].

RQ1: How does *CDCL-IC* compare against state-of-the-art (SOTA) reasoning models in solving NP-Complete combinatorial search problems?

RQ2: How does the clause-learning feature of *CDCL-IC* contribute to the performance?

RQ3: How does the failure modes of *CDCL-IC* compare with that of SOTA reasoning models?

4.1 Benchmark

We select Sudoku as our benchmark because it satisfies three requirements. First, it is an NP-complete discrete search problem without numerical computation steps. This is important because we do not want to confound our results with issues of mathematical or arithmetic reasoning, which would be required for problems like Integer Linear Programming [16]. Second, Sudoku, especially 9x9, has plenty of training data available for LLMs. In contrast, other NP-Complete problems like Boolean Satisfiability [10] are unlikely to be found in LLM training datasets. Third, Sudoku has canonical display format of n-by-n grids. This avoids the issue of LLM performance being negatively effected by the our choice of formatting the problem into strings.

We randomly generate 25 Sudoku problems. These randomly generated instances may have any number of solutions, including none. These problems are solved using a custom DFS solver to generate the baseline data.

4.2 Setup and Implementation

CDCL-IC: Our instantiation of *CDCL-IC* for Sudoku consists of two parts. First, there is a generator that produces one step given the previous board state. Repeatedly querying this will produce a chain of board states. Second, a reward model is trained to detect improper steps in filling the Sudoku board, such as putting the same number in the same row or producing a board with a blocked pattern. This model is trained to detect board patterns from the context, allowing it to block learned bad board patterns. Qwen 2.5 0.5B [20] is fine-tuned for this purpose.

`o3-mini`: We use `o3-mini-2025-01-31` as our SOTA baseline. This model allows us to select the "reasoning level" used during inference. A preliminary investigation using 4x4 models showed increased error hallucination with the "high" setting. Therefore, we use "medium" for the evaluation.

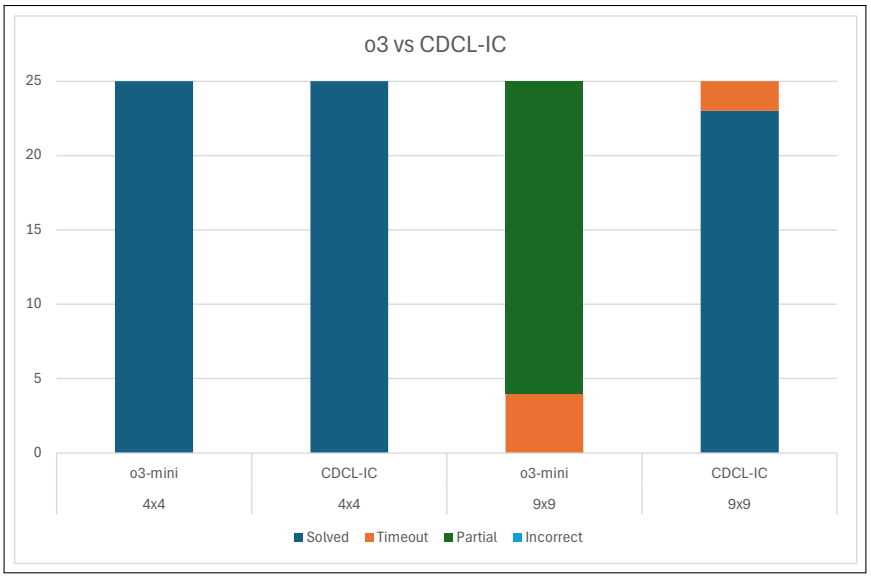

Figure 2: We compare *CDCL-IC* and `o3-mini` on their ability to solve 4x4 and 9x9 Sudoku boards. Our evaluation shows that, while both solve 4x4 Sudoku boards easily, `o3-mini` fails to solve 9x9 Sudoku boards. Instead, partial solutions and timeouts occur.

Table 1: We compare *CDCL-IC* and search without *CDCL-IC* for 4x4 and 9x9 Sudoku boards. We measure the number of boards solved out of 25 and the median number of backtracks per board.

	4x4 Solved	4x4 Median No. Backtracks	9x9 Solved	9x9 Median No. Backtracks
Without *CDCL-IC*	25	3.2	0	>210
With *CDCL-IC*	25	3.2	23	77

4.3 RQ1: *CDCL-IC* v.s. SOTA Reasoners

We present an evaluation of *CDCL-IC* against `o3-mini`. Fig. 2 shows the success rates of *CDCL-IC* and `o3-mini` on 4x4 and 9x9 Sudoku boards. *CDCL-IC* was run on 25 benchmarks, and `o3-mini` on 50 benchmarks due to cost reasons. The green bar indicates the percentage of Sudoku problems solved. Red, grey, and purple indicate non-solutions, timeouts, and incorrect solutions respectively. Each of these failure modes will be analyzed further in RQ3

For 4x4 Sudoku boards, both *CDCL-IC* and `o3-mini` solve them with accuracy above J%. Rates of errors are low for 4x4 boards. 9x9 boards presents a challenge to both tools. *CDCL-IC*'s accuracy lowers to 70% while `o3-mini` solves almost no boards. The respective error categories increase with timeouts accounting for most of *CDCL-IC*'s failures while `o3-mini` refuses to answer most of the problems it fails at.

4.4 RQ2: Scaling Search with Clause Learning

We compare *CDCL-IC* with and without clause learning. The results are provided in Table 1. The first row is the data for 4x4 boards while the second is for 9x9 boards. Each column is a statistic, with the left half of columns being the ablation variant without clause learning and the right half being *CDCL-IC*. For both 4x4 and 9x9, *CDCL-IC* performs strictly better than the ablation variant. In 4x4, both *CDCL-IC* and search without *CDCL-IC* are identical. In 9x9, none of the boards get solved without *CDCL-IC* while search with *CDCL-IC* solves 23 problems out of 25 while reducing backtracking by a factor of at least 3 times per problem. Combined, we show clause learning helps scale search for larger problems.

4.5 RQ3: Failure Analysis

We present a qualitative analysis of the errors encountered by SOTA reasoning models and *CDCL-IC* over the evaluation. We manually inspect a set of 50 benchmarks and answers for `o3-mini` and *CDCL-IC*.

Non-Answers For `o3-mini`, the model often fails to answer with a valid Sudoku board. There are roughly 2 modes to this failure. First, `o3-mini` will answer that "9x9 Sudoku is a difficult problem," and goes on to explain its search complexity. Second, `o3-mini` partially solves the Sudoku board with about a quarter of cells left open with "?" in the place of a number. `o3-mini` claims that "?" has a satisfying solution even when it does not.

We note that "non-answer" errors account for most of the increase in errors between 4x4 and 9x9 Sudoku. As the board size increases, `o3-mini` resorts to giving partial answers and explanations of why it failed to solve rather than actual answers. If we assume that `o3-mini` has uses a reward model, we consider these answers are the result of reward-hacking.

Timeouts For both *CDCL-IC* and `o3-mini`, we encountered timeouts. For *CDCL-IC*, we set a manual timeout of 100 backtracking operations, which it exceeds on 2 out of 25 problems. For `o3-mini`, 20 problems caused the API to return a `503` error, which we consider to be a timeout.

Error Hallucination per Step We note that *CDCL-IC* has a higher failure rate for 9x9 boards. We note that these are due to compounding error probabilities when running at longer steps. For each step, the test-time accuracy of the reward model is above 98%. While the reward model's errors are evenly distributed with respect to how much the board has been filled, compounding these across nearly 80 steps cause all but 15% of solutions to have stepwise errors. This does not automatically imply that the accuracy is, or should be, 15%; if a reward model negatively decides a valid path in a problem that turns out to be unsatisfiable, accuracy will not be effected.

`o3-mini` is proprietary and we have no insight into the nature of its failures as we have no access to its full chain-of-thought. However, our issues with "high effort" settings on 4x4 boards suggests similar factors are at play. As the chain length increases, hallucination in the reward model compounds. We note that these problems are fundamental to any reward model that needs to operate over large number of steps.

5 Discussion: General Applicability

We take this section to discuss the general applicability of out approach. While learning from failure is a general framework that guides search-based approaches for NP-Complete problems, we note three restrictions for *CDCL-IC* to apply: the reasoning steps need to be discrete, the steps need be categorized into "choices" and "deductions", and every "deduction" needs to keep track of which previous "choices" and "deductions" which forced this step. We go over the requirements in detail.

The discrete step requirement is general to any incrementally backgracking reasoning model. If the steps were not discrete, it would be difficult to backtrack to the "last correct" step. For logical, programming, and mathematical problems, the output, and thus the steps are generally discrete.

Our system requires that the steps fall into exactly one of two categories: "choices" and "deductions." For example, the generator can fill a particular cell with the number "5" under exactly two circumstances. First, it can choose to do so when there are other options, or "5" is the only number not used in the row. In the former case, the tool chose "5" out of multiple options. In the latter, the choice of "5" is deduced. This distinction matters because when a particular path of cell assignments fail, we do not want to block its nearest causes, but instead, the broadest set causes, or 1UIP cut in SAT literature [10]. *CDCL-IC* traverses the set of "deductions" until a broader set is reached.

Third requirement for every "deduction" having a set of causes arises out of this need to traverse the implications back to root causes. This Implication Graph [10] enables us to the most general cause of the failure. These graphs of reasoning are an extended version of "previous steps" labels in existing reasoning models [21].

6 Conclusion

We presented *CDCL-IC*, a technique for scaling CoT reasoning to combinatorial problems. *CDCL-IC* allows reasoning models to prune their search space while facing combinatorial explosions. We demonstrate its effectiveness for 9x9 Sudoku problems, enabling performance beyond SOTA reasoning models.

References

[1] DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, X. Zhang, X. Yu, Y. Wu, Z. F. Wu, Z. Gou, Z. Shao, Z. Li, Z. Gao, A. Liu, B. Xue, B. Wang, B. Wu, B. Feng, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Ding, H. Xin, H. Gao, H. Qu, H. Li, J. Guo, J. Li, J. Wang, J. Chen, J. Yuan, J. Qiu, J. Li, J. L. Cai, J. Ni, J. Liang, J. Chen, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Zhao, L. Wang, L. Zhang, L. Xu, L. Xia, M. Zhang, M. Zhang, M. Tang, M. Li, M. Wang, M. Li, N. Tian, P. Huang, P. Zhang, Q. Wang, Q. Chen, Q. Du, R. Ge, R. Zhang, R. Pan, R. Wang, R. J. Chen, R. L. Jin, R. Chen, S. Lu, S. Zhou, S. Chen, S. Ye, S. Wang, S. Yu, S. Zhou, S. Pan, S. S. Li, S. Zhou, S. Wu, S. Ye, T. Yun, T. Pei, T. Sun, T. Wang, W. Zeng, W. Zhao, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, W. L. Xiao, W. An, X. Liu, X. Wang, X. Chen, X. Nie, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yang, X. Li, X. Su, X. Lin, X. Q. Li, X. Jin, X. Shen, X. Chen, X. Sun, X. Wang, X. Song, X. Zhou, X. Wang, X. Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. Zhang, Y. Xu, Y. Li, Y. Zhao, Y. Sun, Y. Wang, Y. Yu, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Ou, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Xiong, Y. Luo, Y. You, Y. Liu, Y. Zhou, Y. X. Zhu, Y. Xu, Y. Huang, Y. Li, Y. Zheng, Y. Zhu, Y. Ma, Y. Tang, Y. Zha, Y. Yan, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Xie, Z. Zhang, Z. Hao, Z. Ma, Z. Yan, Z. Wu, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Pan, Z. Huang, Z. Xu, Z. Zhang, and Z. Zhang, "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning," Jan. 2025. arXiv:2501.12948 [cs].

[2] J. Pan, J. Zhang, X. Wang, L. Yuan, H. Peng, and A. Suhr, "TinyZero," 2025.

[3] B. Brown, J. Juravsky, R. Ehrlich, R. Clark, Q. V. Le, C. Ré, and A. Mirhoseini, "Large Language Monkeys: Scaling Inference Compute with Repeated Sampling," Dec. 2024. arXiv:2407.21787 [cs].

[4] C. Snell, J. Lee, K. Xu, and A. Kumar, "Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters," Aug. 2024. arXiv:2408.03314 [cs].

[5] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan, "Tree of Thoughts: Deliberate Problem Solving with Large Language Models," Dec. 2023. arXiv:2305.10601 [cs].

[6] S. Han, H. Schoelkopf, Y. Zhao, Z. Qi, M. Riddell, W. Zhou, J. Coady, D. Peng, Y. Qiao, L. Benson, L. Sun, A. Wardle-Solano, H. Szabó, E. Zubova, M. Burtell, J. Fan, Y. Liu, B. Wong, M. Sailor, A. Ni, L. Nan, J. Kasai, T. Yu, R. Zhang, A. Fabbri, W. M. Kryscinski, S. Yavuz, Y. Liu, X. V. Lin, S. Joty, Y. Zhou, C. Xiong, R. Ying, A. Cohan, and D. Radev, "FOLIO: Natural Language Reasoning with First-Order Logic," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing* (Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, eds.), (Miami, Florida, USA), pp. 22017–22031, Association for Computational Linguistics, Nov. 2024.

[7] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan, "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?," Nov. 2024. arXiv:2310.06770 [cs].

[8] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt, "Measuring Mathematical Problem Solving With the MATH Dataset," Nov. 2021. arXiv:2103.03874 [cs].

[9] S. Miner, Y. Takashima, S. Han, F. Erata, T. Antonopoulos, R. Piskac, and S. J. Shapiro, "Scheherazade: Evaluating Chain-of-Thought Math Reasoning in LLMs with Chain-of-Problems," Oct. 2024. arXiv:2410.00151 [cs].

[10] A. Biere, *Handbook of Satisfiability*. Frontiers in artificial intelligence and applications, IOS Press, 2009.

[11] G. Kendall, A. J. Parkes, and K. Spoerer, "A Survey of NP-Complete Puzzles," *J. Int. Comput. Games Assoc.*, vol. 31, no. 1, pp. 13–34, 2008.

[12] N. Borazjanizadeh, R. Herzig, T. Darrell, R. Feris, and L. Karlinsky, "Navigating the Labyrinth: Evaluating and Enhancing LLMs' Ability to Reason About Search Problems," June 2024. arXiv:2406.12172 [cs].

[13] L. Pan, A. Albalak, X. Wang, and W. Y. Wang, "Logic-LM: Empowering Large Language Models with Symbolic Solvers for Faithful Logical Reasoning," Oct. 2023. arXiv:2305.12295 [cs].

[14] W. Chen, X. Ma, X. Wang, and W. W. Cohen, "Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks," Oct. 2023. arXiv:2211.12588 [cs].

[15] X. Ye, Q. Chen, I. Dillig, and G. Durrett, "SATLM: satisfiability-aided language models using declarative prompting," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, (Red Hook, NY, USA), pp. 45548–45580, Curran Associates Inc., Dec. 2023.

[16] S. Russell, P. Norvig, and E. Davis, *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence, Prentice Hall, 2010.

[17] H. Morais, P. Kádár, P. Faria, Z. A. Vale, and H. M. Khodr, "Optimal scheduling of a renewable micro-grid in an isolated load area using mixed-integer linear programming," *Renewable Energy*, vol. 35, no. 1, pp. 151–156, 2010.

[18] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: engineering an efficient SAT solver," in *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, pp. 530–535, 2001.

[19] J. v. Oswald, E. Niklasson, E. Randazzo, J. Sacramento, A. Mordvintsev, A. Zhmoginov, and M. Vladymyrov, "Transformers learn in-context by gradient descent," May 2023. arXiv:2212.07677 [cs].

[20] A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, C. Li, C. Li, D. Liu, F. Huang, G. Dong, H. Wei, H. Lin, J. Tang, J. Wang, J. Yang, J. Tu, J. Zhang, J. Ma, J. Yang, J. Xu, J. Zhou, J. Bai, J. He, J. Lin, K. Dang, K. Lu, K. Chen, K. Yang, M. Li, M. Xue, N. Ni, P. Zhang, P. Wang, R. Peng, R. Men, R. Gao, R. Lin, S. Wang, S. Bai, S. Tan, T. Zhu, T. Li, T. Liu, W. Ge, X. Deng, X. Zhou, X. Ren, X. Zhang, X. Wei, X. Ren, X. Liu, Y. Fan, Y. Yao, Y. Zhang, Y. Wan, Y. Chu, Y. Liu, Z. Cui, Z. Zhang, Z. Guo, and Z. Fan, "Qwen2 technical report," 2024.

[21] D. Zhang, J. Wu, J. Lei, T. Che, J. Li, T. Xie, X. Huang, S. Zhang, M. Pavone, Y. Li, W. Ouyang, and D. Zhou, "LLaMA-Berry: Pairwise Optimization for O1-like Olympiad-Level Mathematical Reasoning," Nov. 2024. arXiv:2410.02884 [cs].