TRACE: Tree-search Reasoning Agent with Constraint-based Enhancement for Grid Puzzle Solving in LLMs

Anonymous ACL submission

Abstract

Inference-time algorithms have recently shown promising results on various Natural Language Understanding (NLU) tasks, such as mathematical reasoning and code synthesis. However, their potential to improve LLMs' logical reasoning in complex problem-solving scenarios like grid-based puzzles, remains underexplored. To this end, we introduce TRACE (**T**ree-search **R**easoning **A**gent with **C**onstraint-based **E**nhancements), a novel method that leverages an agentic approach combining tree-search with a constraint verification module to improve LLMs' performance on logic-intensive, grid-based puzzle solving. TRACE iteratively explores solutions using a step-search agent and enforces logical consistency through constraint satisfaction to eliminate invalid paths. We evaluate TRACE on the most recent and widely used LLMs, including GPT-4o, Gemini-flash, and Llama-3.1-8B. Experimental results indicate that TRACE outperforms the zero-shot Chain-of-Thought (CoT) baseline by 11.2% (on average) in terms of *PuzzleEval*. Further analysis of results reveals that integrating the constraint module within the agentic framework significantly reduces logical inconsistencies. Our study highlights the potential of inference-time and agent-based methods to enhance the logical reasoning abilities of LLMs[1].

1 Introduction

The improved performance of LLMs in the various NLU tasks (Achiam et al., 2023) increased focus on enhancing their logical reasoning abilities to solve complex problems (Pan et al., 2023; Olausson et al., 2023). Grid-based puzzles represent one of the most challenging areas for LLMs, as they require both step-by-step reasoning and the ability to maintain logical consistency across interconnected decision steps. While LLMs with inference-time algorithms perform well in tasks such as mathematical reasoning (Zhang et al., 2024b) and code generation (Wang et al., 2024a), their performance in logic-intensive puzzles remains relatively underexplored. Prior works such as Ishay et al. (2023) have evaluated LLMs on grid puzzles using neuro-symbolic methods that rely on external solvers, but not independent LLM performance. Additionally, recent studies (Tyagi et al., 2024; Dziri et al., 2023) highlight LLMs' difficulties in maintaining logical consistency throughout the puzzle-solving process. These findings motivated us to develop a more structured approach to improve the performance of LLMs at inference time, without relying on external solvers.

Previous works on LLM self-verification and self-correction have demonstrated that step-level refinements can enhance reasoning abilities (Li et al., 2024; Jiang et al., 2024). Motivated by this, we introduce TRACE (Tree-search Reasoning Agent with Constraint-based Enhancements), a dual-agent framework designed to improve LLM performance on grid puzzles (as shown in Figure 1). TRACE integrates an agentic approach, where a step-search agent employs a tree-search algorithm to explore possible solution paths, and a verifier agent, enhanced with a constraint verification module, ensures logical consistency. In this dual-agent framework, the step-search agent generates potential solutions by exploring different paths in the search tree, while the verifier agent checks these solutions against the constraints, pruning invalid paths and guiding the search toward feasible solutions.

Our evaluation of TRACE on widely used recent LLMs—GPT-4o, Gemini-flash, and Llama-3.1—demonstrates its effectiveness. TRACE outperforms the existing baseline, zero-shot CoT, by achieving improvements of 11.2% (on average) in *PuzzleEval* and an increase in overall accuracy. More importantly, the integration of the constraint verification module reduces logical inconsistencies,

[1]Source code is available at https://anonymous.4open.science/r/TRACE-757B

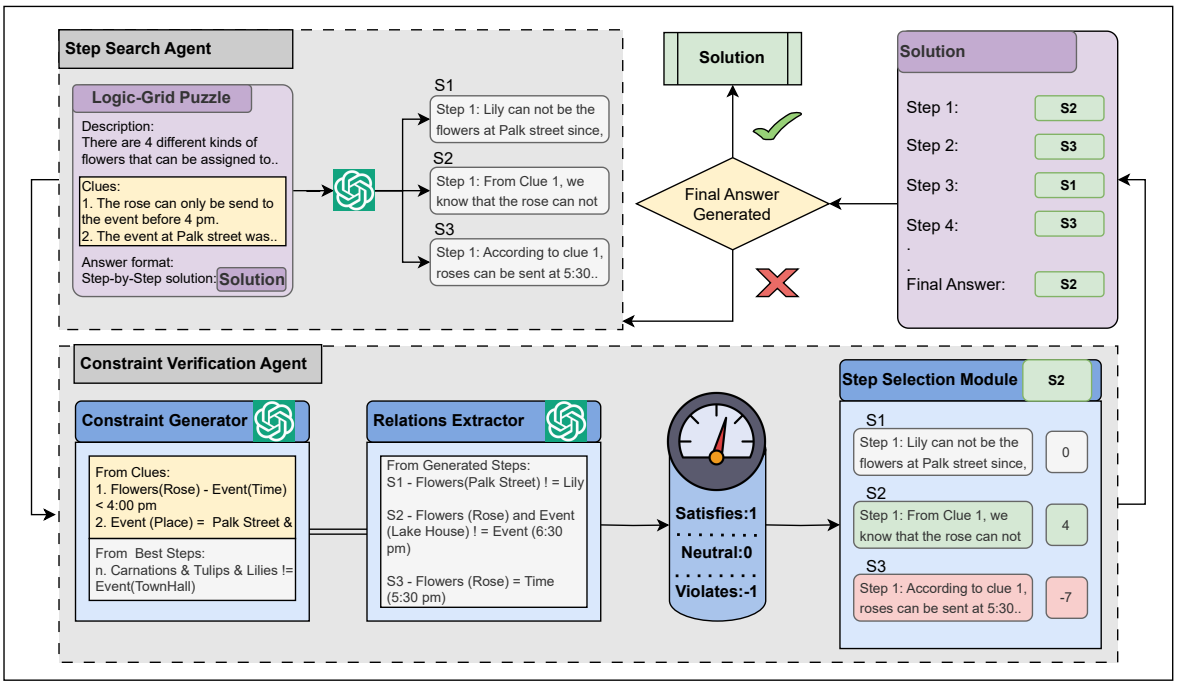

Figure 1: Schematic representation of TRACE. Begins with the Step Search agent (**top-left**) that prompts the LLM to generate a step-wise solution to solve the puzzle. For every step, three instances are generated and passed on to the Constraint Verification Agent (**bottom**). This agent generates constraints from the clues given in the puzzle and extracts relations from each step. It scores the steps based on constraint and the best scoring step is appended to the solution (**top-right**). This process is repeated until the final solution is generated or reached a terminating condition.

a critical issue identified in previous works (Dziri et al., 2023; Tyagi et al., 2024). These results highlight the potential of agent-based frameworks to significantly improve LLMs' logical reasoning abilities, especially in tasks requiring complex problem-solving. We hope that our study contributes valuable findings for future research exploring the integration of inference-time algorithms and agent-based frameworks for the enhancement of logical reasoning abilities of LLMs.

2 TRACE: Tree-search Reasoning Agent with Constraint-based Enhancements

We formally introduce TRACE, our method for solving logic-grid puzzles, and outline the multi-agent process pipeline, as depicted in Figure 1. Our approach is based on the hypothesis that logic-grid puzzles containing sufficient clues to solve can be reformulated as constraint satisfaction problems (CSPs). In a CSP, the constraints determine a fixed number of possible solutions, and for grid puzzles, there is only one unique solution. Leveraging this fact, we transform these constraints into the foundation of our verification agent and design a step-by-step search algorithm to solve grid puzzles.

2.1 Constraint Verification Agent

This agent verifies the steps generated to solve the puzzle by utilizing the clues provided in the puzzle and scoring the steps accordingly. The agent comprises three components: the Constraint Generator, the Relation Extractor, and the Step-selection Module. The prompt structures and additional details of this agent are provided in Appendix C.

Constraint Generator This component converts the given clues from verbose steps to semi-formal logic predicates. It consists of both a static and a dynamic component. The static component transforms the initial puzzle clues into logic predicates, which are stored without any change. The dynamic component updates the constraint list by converting the best step, selected during verification in each iteration, into a new constraint. This dynamic update mitigates the issue of forgetfulness in long reasoning chains. Consequently, the constraint list grows with each step and remains consistent with the original clues and the solution generated so far.

Relation Extractor This component formalizes the steps generated during the puzzle-solving process. In grid puzzles, each step concludes with an inference based on the given clues and the solu-

Algorithm Iterative Greedy Search

Input: Puzzle prompt P, Verifier V
Output: Final solution S
$S \leftarrow$ ""
StepCounter $\leftarrow 1$
V.add_initialConstraints(P)
for *StepCounter* **in** 1...T **do**
    Generate step candidates: S_1, S_2, S_3
    if (S_1 **and** S_2 **and** S_3) == "" **then**
        break
    if (S_1 **or** S_2 *or* S_3) == "Final Answer" **then**
        break
    Score steps using verifier V:
        BestStep $\leftarrow$ *V.score_steps(*S_1, S_2, S_3*)*
        $S \leftarrow S$ + *BestStep*
    StepCounter $\leftarrow$ *StepCounter* + 1
end for
FinalAnswer $\leftarrow$ *V.score_steps(*F_1, F_2, F_3*)*
$S \leftarrow S$ + *FinalAnswer*
return S

Table 1: The Algorithm for TRACE

tion so far. These conclusions can be formulated as pairwise relations between the entities in the puzzle. For example, S1: "The TownHall event had Tulips delivered..." can be formalized as the relation R1: "TownHall – Tulip". This module extracts all such pairwise relations from each step.

Step-selection Module This is the central component of the constraint-verification agent. The extracted relations from each step are compared against the list of constraints and scored. A relation that satisfies a constraint is awarded +1, while a violation results in a -1 penalty. If the relation is unrelated to any constraint, it is considered neutral and scored as 0. After scoring all the relations, the step with the highest score is chosen as the best step and added to the solution.

2.2 Step-Search Agent

This agent generates steps to solve the puzzle sequentially while updating the constraint list and solution throughout the process. The algorithm used to carry out the step-wise search is presented in Table 1. The initial prompt structure to generate step-wise solution is provided in Appendix B.

TRACE Algorithm In the algorithm, the key inputs are a puzzle prompt P and the verifier V. The algorithm iteratively builds a solution S by generating candidate steps and scoring them using the verifier. The verifier V plays a crucial role in determining the correctness of each candidate step by evaluating the steps against the logical constraints derived from the puzzle clues. For instance, if the puzzle clue states, "Cal Chandler was from Arkansas," the constraint generator in the verifier V converts this clue into a constraint $CalChandler_State = Arkansas$ during the initialization phase. The verifier then checks each step against this and other constraints as the solution is constructed. At each iteration, the algorithm generates candidate steps S_1, S_2, and S_3, which are evaluated by the verifier based on how well they satisfy the existing constraints. For example, if the relation from step S_1 = "Cal Chandler is from Arkansa" directly matches the constraint, it receives a positive score. This process continues iteratively, adding the best step to the solution. The algorithm stops when either a final answer is found or a terminating condition (all steps are empty or the number of steps >25) is reached. In the final stage, the verifier ensures that all accumulated constraints are satisfied. Through this iterative process, TRACE constructs a solution by balancing logical constraints and the verifier's scoring of each step.

Dual Updation To maintain consistency and address potential forgetfulness in LLMs during long reasoning chains, we use dual updates in the search process. The first update occurs with the solution itself, which is incrementally constructed step by step. This is in contrast to vanilla CoT which generates the whole solution at once. The second update happens in the Constraint Generator module, where the best step from each iteration is converted into a new constraint and added to the constraint list. This dual update mechanism helps eliminate errors caused by inconsistency and error propagation in the reasoning process.

3 Experiments and Results

Experiment Setup We evaluate our method using the GridPuzzle Dataset across all five grid sizes. The primary experiments are conducted on 274 puzzles using the Gemini-Flash model. Zero-shot Chain-of-Thought (CoT) prompting serves as the baseline, and we compare its results with TRACE. Additionally, we perform a case study on 3x4 puzzles using OpenAI's GPT-4o and Meta's Llama-3-8B models. For baselines and the Verification

Gemini-Flash	3 x 4	3 x 5	4 x 4	4 x 5	4 x 6
CoT Accuracy	2	0	0	0	0
Trace Accuracy	3	0	1	1	1
CoT *PuzzleEval*	0.52	0.56	0.55	0.55	0.53
Trace *PuzzleEval*	0.59	0.59	0.58	0.61	0.74

Table 2: The results for Accuracy and *PuzzleEval* on the different grid sizes available in *GridPuzzle* dataset for Gemini-Flash.

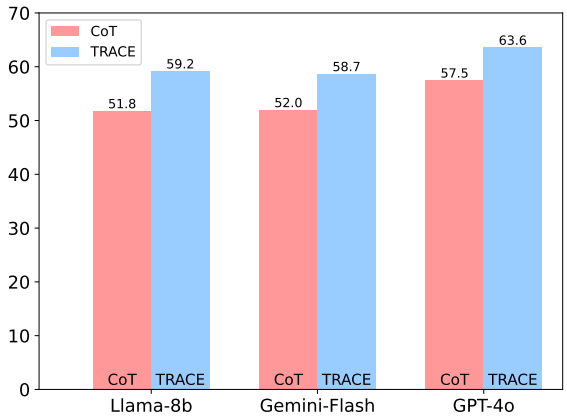

Figure 2: Performance (CoT and TRACE) of three different LLMs in terms of *PuzzleEval* on the 60 3x4 puzzles from the *GridPuzzle* dataset.

agent, the temperature is set to 0, while for the Step-search agent, it is 0.7. All open-source model inferences are run on NVIDIA A100 GPUs.

Evaluation and Analysis We assess the performance of TRACE using Accuracy and *PuzzleEval* metrics (Tyagi et al., 2024). Accuracy measures the correctness of the final answer table, while PuzzleEval tracks the average correctness of the solution's reasoning steps. The accuracy results for CoT were low, as expected given the complexity of the task. However, TRACE showed improvement across all grid sizes except for 3x5. *PuzzleEval* scores for CoT remain steady across puzzles, but TRACE's performance improves as the grid size increases, with the largest gains observed in 4x5 and 4x6 puzzles. The improvements in the reasoning chains can be attributed to the reductions in error propagation and reasoning hallucinations.

Case Study We further analyze the 3x4 puzzles using GPT-4o and Llama-8B models. Both Gemini and Llama models perform similarly with the CoT baseline, while GPT-4o shows significantly higher performance. Notably, the improvement with TRACE is more pronounced in the smaller Llama model compared to the other two models. This is likely due to TRACE's ability to enhance the overall structure of the solution. All agents in the experiments use the same model for consistency.

4 Related Work

In recent years, several methods have been proposed for puzzle-solving, primarily in three categories: (i) prompting techniques, (ii) neuro-symbolic approaches, and (iii) fine-tuning strategies (Giadikiaroglou et al., 2024). In the prompting, a variety of techniques have been explored, including Chain of Thought (CoT) (Wei et al., 2022), Tree of Thoughts (ToT) (Yao et al., 2024), Self-refine (Madaan et al., 2024), plan-and-solve (Wang et al., 2023), and self-consistency (Wang et al.), to name a few. In neuro-symbolic, Ishay et al. (2023) utilize GPT-3/4 to translate puzzles into Answer Set Programming (ASP) for enhancing puzzle-solving capabilities. Fine-tuning approaches have also been explored, involving models such as GPT-2 (Noever and Burdick, 2021) and BERT-based models (Zhang and Wan, 2022), among others (Kazemi et al., 2024; Schuster et al., 2021). However, the exploration of agent-based systems for puzzle-solving remains under-explored. A recent study (Berman et al., 2024) explores the multi-agent systems for this task. However, no prior work has investigated the efficiency of inference algorithms in the puzzle-solving. Thus, we propose TRACE, a tree-search-based agentic framework specifically designed for this task. Extended related work on inference-time algorithms is presented in Appendix A.

5 Conclusions

In this paper, we introduced TRACE, an agentic approach that integrates an inference-time search algorithm with constraint verification to solve logic-grid puzzles. Our experiments using Gemini, Llama, and GPT-4o models demonstrated that TRACE improves LLMs' ability to solve complex puzzles. The experiments were conducted within a model self-refinement framework, where both the verifier and search module utilize the same LLM. Future research could explore using larger models as verifiers for smaller open-source models. Additionally, this approach could be extended to other types of constraint satisfaction problems which produce a unique solution. Our work offers an effective agentic method for enhancing LLMs' reasoning capabilities in complex logical tasks.

Limitations

While TRACE shows the effectiveness of an agentic approach to solving logic-grid puzzles, this framework could be extended to other Constraint Satisfaction Problems (CSPs), broadening its scope. Our study is limited to English, but TRACE could be extended to multilingual settings to evaluate its broader applicability. Due to resource constraints, we also restricted our experiments to two proprietary models and one open-source LLM; evaluating a wider range of models could offer more insights into its generalizability. Additionally, we used a simple greedy tree search to showcase the method's effectiveness, but more advanced algorithms such as Monte Carlo Tree Search (MCTS) could be explored in future work.

Ethics Statement

We have used AI assistants (Grammarly and ChatGPT) to address the grammatical errors and rephrase the sentences. Also, the use of proprietary LLMs such as GPT-4o and Gemini, in this study adheres to their policies of usage.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Shmuel Berman, Baishakhi Ray, and Kathleen McKeown. 2024. Solving zebra puzzles using constraint-guided multi-agent systems. *arXiv preprint arXiv:2407.03956*.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D Hwang, et al. 2023. Faith and fate: limits of transformers on compositionality. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 70293–70332.

Panagiotis Giadikiaroglou, Maria Lymperaiou, Giorgos Filandrianos, and Giorgos Stamou. 2024. Puzzle solving using reasoning of large language models: A survey. *arXiv preprint arXiv:2402.11291*.

Adam Ishay, Zhun Yang, and Joohyung Lee. 2023. Leveraging large language models to generate answer set programs. *ArXiv*, abs/2307.07699.

Dongwei Jiang, Jingyu Zhang, Orion Weller, Nathaniel Weir, Benjamin Van Durme, and Daniel Khashabi. 2024. Self-[in]correct: Llms struggle with discriminating self-generated responses. *Preprint*, arXiv:2404.04298.

Mehran Kazemi, Quan Yuan, Deepti Bhatia, Najoung Kim, Xin Xu, Vaiva Imbrasaite, and Deepak Ramachandran. 2024. Boardgameqa: A dataset for natural language reasoning with contradictory information. *Advances in Neural Information Processing Systems*, 36.

Xiaonan Li, Changtai Zhu, Linyang Li, Zhangyue Yin, Tianxiang Sun, and Xipeng Qiu. 2024. Llatrieval: Llm-verified retrieval for verifiable generation. *Preprint*, arXiv:2311.07838.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

Rohin Manvi, Anikait Singh, and Stefano Ermon. 2024. Adaptive inference-time compute: Llms can predict if they can do better, even mid-generation. *arXiv preprint arXiv:2410.02725*.

David A. Noever and Ryerson Burdick. 2021. Puzzle solving without search or human knowledge: An unnatural language approach. *ArXiv*, abs/2109.02797.

Theo Olausson, Alex Gu, Ben Lipkin, Cedegao Zhang, Armando Solar-Lezama, Joshua Tenenbaum, and Roger Levy. 2023. LINC: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5153–5176, Singapore. Association for Computational Linguistics.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. 2023. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3806–3824, Singapore. Association for Computational Linguistics.

Tal Schuster, Ashwin Kalyan, Oleksandr Polozov, and Adam Tauman Kalai. 2021. Programming puzzles. *arXiv preprint arXiv:2106.05784*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Nemika Tyagi, Mihir Parmar, Mohith Kulkarni, Aswin RRV, Nisarg Patel, Mutsumi Nakamura, Arindam Mitra, and Chitta Baral. 2024. Step-by-step reasoning to solve grid puzzles: Where do llms falter? *EMNLP 2024*.

Evan Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, Will Song, Vaskar Nath, Ziwen Han, Sean Hendryx, Summer Yue, and Hugh Zhang. 2024a. Planning in natural language improves llm search for code generation. *arXiv preprint arXiv:2409.03733*.

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

Xiyao Wang, Linfeng Song, Ye Tian, Dian Yu, Baolin Peng, Haitao Mi, Furong Huang, and Dong Yu. 2024b. Towards self-improvement of llms via mcts: Leveraging stepwise knowledge with curriculum preference learning. *arXiv preprint arXiv:2410.06508*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

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

Sean Welleck, Amanda Bertsch, Matthew Finlayson, Hailey Schoelkopf, Alex Xie, Graham Neubig, Ilia Kulikov, and Zaid Harchaoui. 2024. From decoding to meta-generation: Inference-time algorithms for large language models. *arXiv preprint arXiv:2406.16838*.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2024. An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Dan Zhang, Sining Zhoubian, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024a. Rest-mcts*: Llm self-training via process reward guided tree search. *arXiv preprint arXiv:2406.03816*.

Di Zhang, Jiatong Li, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. 2024b. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b. *arXiv preprint arXiv:2406.07394*.

Yunxiang Zhang and Xiaojun Wan. 2022. Birdqa: A bilingual dataset for question answering on tricky riddles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 11748–11756.

A Extended Related Work

Many inference-time algorithms have been developed, such as greedy search, majority voting, best of n, weighted voting, and tree search algorithms, demonstrating promising improvements in the reasoning and decision-making capabilities of LLMs in various domains (Welleck et al., 2024; Snell et al., 2024; Wu et al., 2024). For instance, Zhang et al. (2024b) introduced the MCT Self-Refine (MCTSr) algorithm, which combines LLMs with Monte Carlo Tree Search (MCTS) to improve mathematical reasoning. Additionally, several studies have evaluated the effectiveness of inference-time algorithms on challenging benchmarks such as SWE-Bench, GSM8k, MATH, and GPQA (Brown et al., 2024; Manvi et al., 2024; Zhang et al., 2024a; Wang et al., 2024b). Recently, Wang et al. (2024a) proposed a method that enhances LLMs' performance in code generation by incorporating natural language planning to guide the search process.

B Step-Search Agent Modules

The step search agent implements the iterative greedy search algorithm for TRACE. It begins by prompting the LLM as shown in Figure 3 to generate a set of steps. Often the model fails to follow the strict instructions to generate only one step at a time. In such cases, we clean up the generated response to extract only a singular step. The step counter is monitored so that in the next iteration the Step [n+1] is extracted. All these extracted steps are sent to the verifier, which returns the best step. This step is appended to the solution and also attached with the initial prompt before the next iteration. This process goes on until the model produces the Final Answer table or fails to generate any new steps.

C Verifier Agent Modules

There are three main modules in the constraint verification agent. These modules are responsible for scoring and verifying each step before adding it to the solution.

C.1 Constraint Generator

This module is responsible for converting the clues and the best step chosen at each iteration into semi-formal logical constraints. The prompt structure for this module is shown in Figure 4. To describe the process in detail a sample set of clues is provided in Table 3. The corresponding logical constraints for these clues are provided in Table 4.

Clues
1. Cal Chandler was from Arkansas.
2. Hilda Hunt won somewhat less than Cal Chandler.
3. The person from Hawaii won 5 million more than the winner from Iowa.
4. The person who won $15 million was either the winner from Hawaii or the person from Arkansas.
5. Ferris Fry was from Iowa.
6. The winner from Tennessee won somewhat more than Ferris Fry.

Table 3: Set of clues from a Logic-grid puzzle.

Constraints
1. CalChandler_State = Arkansas
2. HildaHunt < CalChandler
3. Hawaii_winner = Iowa_winner + 5 million
4. Winner_$15M ∈ {Winner_Hawaii, Winner_Arkansas}
5. FerrisFry_State = Iowa
6. Winner(Tennessee) = Ferris_Fry + 1

Table 4: Set of constraints derived from the puzzle clues.

C.2 Relation Extractor

This module extracts the pair-wise relations from the generated steps. The pairs are more easier to compare with the logical constraints than the full sentences. The prompt structure for this module is described in Figure 5. For step, S1 = "Ferris Fry is from Iowa." the pairwise relation would be "Ferris Fry - Iowa."

C.3 Step Selection Module

The generated steps are scored in this module. The prompt structure for the step selector is given in Figure 6. There can be one or more relations present in each step. Therefore, we iterate through the list of all relations in a step and compare them with the given list of constraints. Every instance of comparison is strictly scored 1, 0, or -1 (satisfies, neutral, or violates.) Once the score for each step is summed up, the best step is chosen and then added to the solution. The best step is also converted into a logical constraint and the list of constraints is updated.

stepSearch Module

System Prompt: "You are a Logic Puzzle solving agent and you solve all the puzzles one step at a time."

User Prompt =

<Puzzle>

While answering use the following format:
Step-by-step solution:
(Your steps showing how you are solving the puzzle. Make sure the steps to solve the puzzle are of the forem Step 1: <the first step only>. Only genrate a single sentence to solve the puzzle 1 step at a time. Do not generate heading, sub-headings for the steps or multiple sentences.)
Step 1:
Step 2:
.
.
.
Step n:

Final Answer format:
(Example: Fill the following table to show your final answer.
March | correct option from witnesses | correct option from locations | correct option from times
April | correct option from witnesses | correct option from locations | correct option from times
May | correct option from witnesses | correct option from locations | correct option from times
June | correct option from witnesses | correct option from locations | correct option from times)

Solve the puzzle 1 step at a time unless you are at Final Answer, then you can generate the full table. But only generate one sentence for each step at a time.

Step-by-step solution:

Output: S1, S2, and S3

Figure 3: The prompt structure for the Step Search Agent. The same prompt is repeated after a new step is appended to the solution. The solution is added to the end of the User Prompt.

addConstraint Module

System Prompt: "You are a logical constraint generator. You take in the clues in the form of sentences and convert them into logic constraints such as:
Clue: Abby was the person at Central Park.
Constraint: Abby - Central Park
Clue: The club from Manchester could not have won in 1962.
Constraint: Manchester - not 1962"

for clue in Clues:

User Prompt = "Convert this clue into a logical constraint: {clue} Constraint: "

constraint = call_gem(System prompt, User prompt, 0)
constraints.append(constraint)

Figure 4: This is the prompt for the Constraint Generator module. This module takes in the clues given in the puzzle and converts them into corresponding logical constraints.

extractRelation Module

System Prompt: "You are a Pair-wise relation extractor."

User Prompt = "From each step identify the pairwise conclusion relations from each step. Every conclusion might have 1 or more pairwise relations. For example:

Step 1: Sam is assigned to the year 2013 or 2015.
Pair-wise relation:
Sam - 2013
Sam - 2015

Step 1: The club from Manchester can not have won in the year 1962.
Pair-wise relation:
Manchester - not 1962

Remember: Do not include any additional sentences or formatting. Only use the values from the clues given and return the pairs. Extract pair-wise relations from the following sentence:"

constraint = call_gem(prompt_text, 0)
constraints.append(constraint)

<Step>

relations = call_gem(System prompt, User prompt, 0)

Figure 5: This is the prompt for the Relation Extractor module. It takes the steps generated at each iteration as inputs and returns their respective relation pairs as a list.

scoreStep Module

System Prompt: "You are a constraint verifier and check whether the given relations satisfy the constraints or not. You only respond with 'satisfies' or 'violates'"

Exemplar:
1. Relation: Max - 2020
   Constraint: Max<Hanna
   Result: neutral
2. Relation: Abby - Central Park
   Constraint: Abby - not Central Park
   Result: violates
3. Relation: Mary - less than 1994
   Constraint: Mary - 1992
   Result: satisfies
4. Relation: Person with Roses - 9:30 pm
   Constraint: Person with roses = Person with jasmine + 30 minutes
   Result: neutral
5. Relation: Jim - Football winner
   Constraint: Baseball winner < Football winner
   Result: neutral
6. Relation: Spartan code - Angela
   Constraint: Angela - Code of Barstool
   Result: violates

User Prompt = "If the relation is similar to the constraint then output 'satisfies', if the relation violates the constraint by being a negation of the constraint then output 'violates', if the relation and constraint are unrelated output 'neutral'. Use the examples given below to score the constraint against the relation:
{Exemplar}
Does the following relation satisfy or violate this constraint?
Relation: **{relation}**
Constraint: **{constraint}**
Result:"

if "satisfies" in result: score += 1; elif "violates" in result: score -= 1; elif "neutral" in result: score +=0

Figure 6: This is the prompt structure for the Step Selection Module. This module takes all the constraints and the relation pairs of each step as input. It compares the constraints and relations and scores each step.