
GPT-4 Doesn't Know It's Wrong: An Analysis of Iterative Prompting for Reasoning Problems

Kaya Stechly
Arizona State University
kstechl@asu.edu

Matthew Marquez
Arizona State University
mmarqu22@asu.edu

Subbarao Kambhampati
Arizona State University
rao@asu.edu

Abstract

There has been considerable divergence of opinion on the reasoning abilities of Large Language Models (LLMs). While the initial optimism that reasoning might emerge automatically with scale has been tempered thanks to a slew of counterexamples—ranging from multiplication to simple planning, there is still the wide spread belief that LLMs can self-critique and improve their own solutions in an iterative fashion. This belief seemingly rests on the assumption that verification of correctness should be easier than generation—a rather classical argument from computational complexity, that should be irrelevant to LLMs to the extent what they are doing is approximate retrieval. In this paper, we set out to systematically investigate the effectiveness of iterative prompting of LLMs in the context of *Graph Coloring*, a canonical NP-complete reasoning problem that is related to propositional satisfiability as well as practical problems like scheduling and allocation. We present a principled empirical study of the performance of GPT4 in solving graph coloring instances or verifying the correctness of candidate colorings—both in direct and iterative modes. In iterative modes, we experiment both with the model critiquing its own answers and an external correct reasoner verifying proposed solutions. In both cases, we analyze whether the content of the criticisms actually affects bottom line performance. The study seems to indicate that (i) LLMs are bad at solving graph coloring instances (ii) they are no better at verifying a solution—and thus are not effective in iterative modes with LLMs critiquing LLM-generated solutions (iii) the correctness and content of the criticisms—whether by LLMs or external solvers—seems largely irrelevant to the performance of iterative prompting. We show that the observed effectiveness of LLMs in iterative settings is largely due to the correct solution being fortuitously present in the top-k completions of the prompt (and being recognized as such by an external verifier). Our results thus call into question claims about the self-critiquing capabilities of state of the art LLMs.

1 Introduction

Large Language Models (LLMs), essentially n-gram models on steroids which have been trained on web-scale language corpus, have caught the imagination of the AI research community with linguistic behaviors that no one expected text completion systems to possess. Their seeming versatility has lead many researchers to wonder whether they can also do well on reasoning tasks typically associated with system 2 competency. Initial excitement based on anecdotal performance of LLMs on reasoning tasks has dissipated to some extent by the recent spate of studies questioning the robustness of such behaviors—be it planning [17, 8], simple arithmetic and logic [5], or general mathematical and abstract benchmark[14, 6]. There still exists considerable optimism that even if LLMs can't generate correct solutions in one go, their accuracy improves in a iterative prompting regime, where LLMs will be able to "self-critique" their candidate solutions and refine them to the point of correctness [20, 19, 15, 18, 7]. This belief seem to rest largely on the assumption that verification of correctness

should be easier than generation for many reasoning problems—a rather classical argument from computational complexity. There are grounds to be skeptical of this assumption as complexity of the reasoning task should be irrelevant to LLM performance if what they are doing is approximate retrieval.

In this paper, we set out to systematically investigate effectiveness of iterative prompting in the context of *Graph Coloring*, a canonical NP-complete reasoning problem. We chose graph coloring as it is representative both of standard classes of reasoning problems studied in AI—propositional satisfiability and constraint satisfaction—and practical problems like scheduling and allocation. Our methodology involves a principled empirical study of the performance of GPT4 on two tasks: solving a large suite of random graph coloring instances and, separately, verifying the correctness of the candidate colorings—both in direct and iterative modes. In iterative modes, we experiment both with an LLM critiquing LLM-produced solutions and an external, guaranteed correct reasoner verifying solutions. In both cases, we analyze whether the content of criticisms actually affects bottom line performance.

Our results indicate that in direct mode, LLMs are, perhaps not surprisingly, pretty bad at solving graph coloring instances. More interestingly, as we suspected, they are no better at verifying solutions. In iterative modes, given the inability of LLMs to verify solutions, it should come as no surprise that our experiments show that the strategy of LLMs self-critiquing their solutions does not improve over the baseline. It is *actually worse* because the system can't recognize a correct coloring and thus merrily passes over fortuitously correct colorings it has generated, ending up with a wrong one!

We next experimented with an iterative strategy where an external coloring verifier does the back-prompting. Here we looked at three different types of back prompting: (1) the verifier just asks the LLM to try again when the coloring is incorrect, (2) the verifier gives a backprompt showing the first violated constraint in the current candidate coloring and (3) the verifier sends a backprompt showing all violated coloring constraints. We note that these three strategies do lead to modest improvements in the bottom-line performance—improving from about 16% to nearly 40%. The surprising finding however is that the minimal information "try again" feedback is nearly as effective as the ones with meaningful backprompts. This lead us to consider whether the improvement is due to the type of backprompting (as authors who advocate these types of iterative approaches [20, 19, 15, 10, 4, 11] seem to assume) or because the answer just happens to be in the top-K completions (even if the LLM is itself not cognizant of it). To check this, we experiment with a version of the direct mode where we query the LLM so that it generates more than one potential solution, and have the external verifier pick out any correct solution in the list. The results show that top-k correctness with an external, guaranteed correct verifier is pretty competitive with any iterative backprompting.

Our investigation thus raises significant grounds to be skeptical about the effectiveness of iterative prompting techniques in general, and those relying on the self-critiquing capabilities of LLMs in particular. In the remainder of the paper, we discuss related work, present our experimental methodology, and then detail the results of our experiments.

2 Related Work

As mentioned in the introduction, there has been a large recent body of work investigating the reasoning capabilities of LLMs [15, 19, 9]. The studies span different types of reasoning problems—planning [17], logic and arithmetic [5], or 24 puzzle [19]. The conclusions have also been divergent—with some studies highlighting the limitations of LLMs in reasoning [12, 2], and others arguing that iterative prompting of LLMs can improve their ability to reason. For example, [15] states *we explore this emergent property of self-reflection in LLMs and empirically show that self-reflection is extremely useful to learn complex tasks over a handful of trials*. This paper focuses on understanding these sorts of claims—and especially of the effectiveness of iterative prompting. The problem we chose—graph coloring—is a canonical NP-complete reasoning problem well studied in AI and computer science [13]. It has rich connections to propositional logical reasoning—specifically satisfiability, constraint satisfaction problems, and is also related to practical problems including resource allocation and scheduling.

3 Methodology

3.1 The Graph Coloring Problem

Because we are interested in LLMs' self-critique capabilities, we chose *Graph Coloring*, a reasoning domain which is human readable, provides relatively short description and critique lengths, and, most importantly, is very easy to verify and provide feedback for. Though it is difficult to be certain, we also believe that this domain is diverse enough even at low node and edge counts that the instances we examine are very unlikely to be found in the LLM's training data, thus minimizing the risk of model contamination and memorization.

Graph coloring is a canonical NP-complete reasoning problem that is related to both propositional satisfiability as well as practical problems like scheduling and allocation. It is broad enough to give insights into reasoning more generally, and simple enough to be specified and evaluated by a human or basic pattern matching.

Common graph coloring benchmark sets consist of the sorts of problems that exact solvers struggle on, boasting triple or quadruple digit numbers of nodes and edges[16]. Current language models don't have sufficiently large context windows to process these, and—as we'll see later—are unlikely to do well on graphs with over twenty nodes.

Therefore, we built our own dataset. We use `GrinPy`¹ to handle common graph operations. Each graph is constructed using the Erdős–Rényi method ($p = 0.4$), modified so that any generation that fails to be planar or happens to be isomorphic to a previously generated one is retried. Once a successful candidate is found, it is compiled into the standard DIMACS format[1], appended with a comment containing its precalculated chromatic number.

For the following experiments, we generated 100 instances with an average of 24 edges each spread across node counts from 10 to 17—a distribution chosen because empirical probing revealed it to be an area with volatile enough performance to be interesting. An example of one of the graphs we used is shown in Figure 1, together with the LLM's first response, the backprompt on that response, and the final correct coloring.

3.2 Architecture for Iterative Backprompting

All code and results will be made public.

Prompt Generator:

The generator takes a DIMACS instance and constructs a natural language prompt by translating each edge into a sentence and then wrapping the whole in a common set of instructions. We deliberately minimize differences between instances' prompts to reduce how much problem-specific information we leak to the LLM. Examples of each prompt type can be found in the appendix.

Large Language Model:

Off the shelf, this system allows for the use of any LLM accessible through the OpenAI API: the user need only pass the model name through the appropriate flag at runtime. The present work focuses on GPT-4, the current state of the art, because of recent claims about its "emergent" reasoning capabilities[3].

We provide a system role of "You are a constraint satisfaction solver that solves various CSP problems." and set the temperature to 0, thus ensuring output is mostly deterministic.

Extensibility:

This architecture easily extends to other domains of constraint satisfaction problem solving. In the public repository, we provide a way to add a new domain description by adding just one file to the project in plug-and-play fashion.

¹<https://pypi.org/project/grinpy/>

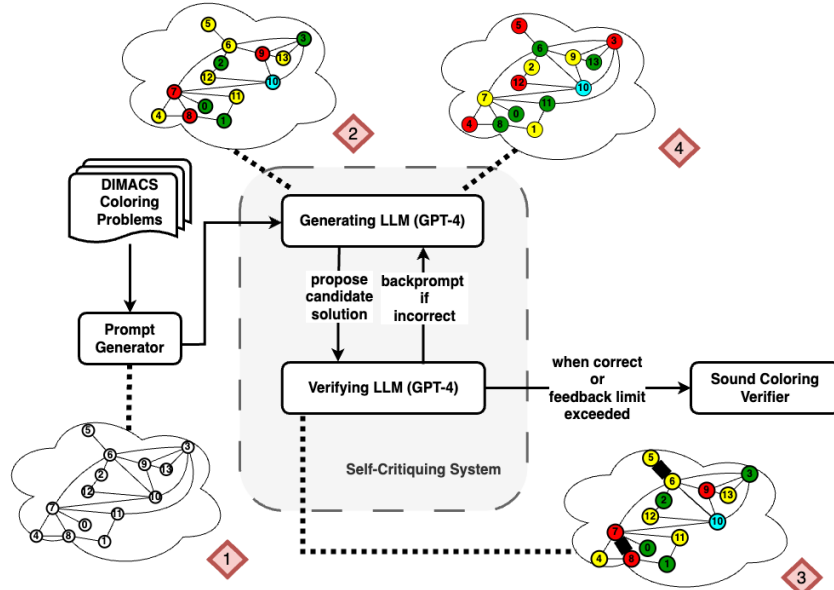


Figure 1: Overview of backprompt architecture for a single instance. Clouds provide an illustrated interpretation of the current state of the problem at different points in the system. Red diamonds indicate progression of a single problem: a planar graph is first passed to GPT-4 acting as a generator (1), which returns a proposed coloring (2). GPT-4 will then be used as a verifier to determine whether the coloring is correct. When not correct, GPT-4 provides feedback, along with previous history, through a backprompt (3) that will be used in the next generation request (4). Each new coloring will be evaluated by the GPT-4 working as a verifier. If GPT-4 determines the coloring to be correct or 15 iterations have passed, it approves the final answer, where it is then evaluated against a sound verifier.

3.3 Backprompt Generation

In verification mode, the LLM receives a different sort of prompt. Apart from standard instructions, it contains only the graph description and the proposed coloring. It is tasked with verifying correctness, optimality, and whether every vertex has been given an assignment. If the coloring is incorrect, it must reply with a set of contradicting edges.

As a comparison point, we also construct a guaranteed correct verifier, with the ability to list every single contradicting edge. Since LLM responses are also in natural language, we first translate them into a format amenable to analysis. To make this process more consistent, we design our initial prompt to describe an exact output format to which the model conforms. Then, the response is evaluated for correctness.

In both cases, if the verifier says the answer is correct, we end there. If it has been more than 15 rounds (16 total queries), we give up. Otherwise, a backprompt is created, wrapped in standard instructions, appended to the previous message history, and sent back to the model as a new prompt.

In this domain a valid piece of error feedback consists of a pair of vertices which were given the same color but share an edge. To construct a backprompt, we have to decide exactly how much feedback to give. We examine five cases:

1. **None:** A single iteration baseline. No backprompting.
2. **Pass/Fail:** The only feedback given is that the answer was incorrect.
3. **First:** Only the first error encountered is returned.
4. **Full:** A comprehensive list of errors.
5. **LLM:** Feedback is provided by the language model through a separate prompt, given in the appendix. We pass any and all response back to the generator, regardless of its validity or correctness.

By comparing results under these regimes, we can deduce how much of the given information the LLM is actually using, versus how much of the performance increase stems from merely getting

more tries. We also compare these cases to four further cases: higher temperature, single iteration queries which ask for multiple answers. These do not involve any backprompting, reprompting, or giving any information past the original prompt to the LLM.

6-8. **Top 5:** With temperatures 0.5, 1, and 1.5, query the LLM for $n = 5$ responses.

9. **Top 15:** With a temperature of 1, query the LLM for $n = 15$ responses.

3.4 Verification

In order to gain more insight into their LLM verification, we examine how well they find errors in proposed colorings. Intuitively, these should be very easy to identify: if the two vertices making up an edge share a color, immediately return that edge. Algorithmically, all this requires is looping over edges and comparing each vertex’s color to that of its partner.

We use the same pipeline for this analysis, but construct a new domain we call `color_verification`. The LLM is prompted to check correctness, optimality, and if every vertex has been assigned in the coloring. If the coloring is incorrect, it is instructed to list errors in the coloring, that is, if two connected nodes share a color, it is to return the edge to represent the error. No backprompts are given. We use the same graph instances from before, but generate four kinds of colorings to test the model on:

1. **Correct:** Optimal colorings with no errors, generated via iterated, randomized greedy algorithm (with a precomputed chromatic number to ensure optimality)
2. **Ablated:** The previous set of colorings, each with a random node changed to one of its neighbor’s colors
3. **Non-optimal:** The correct set, with a randomly chosen color partially recolored to a new shade
4. **Random:** Completely randomly assigned colors, with the number of different colors equal to the graph’s chromatic number
5. **LLM:** Colorings randomly selected from the LLM-generated outputs of the previous experiment

4 Results

4.1 Backprompting as Self-Critique

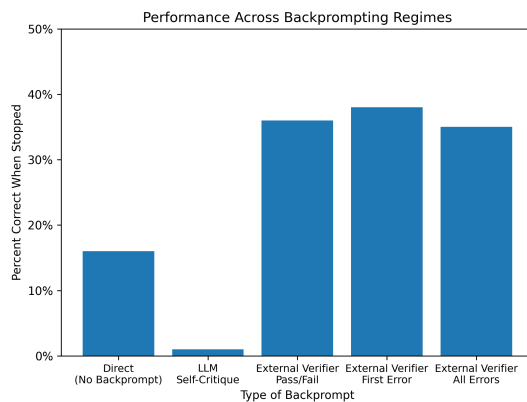


Figure 2: Performance versus backprompting technique. Correctness is evaluated for the response the verifier claims as correct, or after 15 iterations.

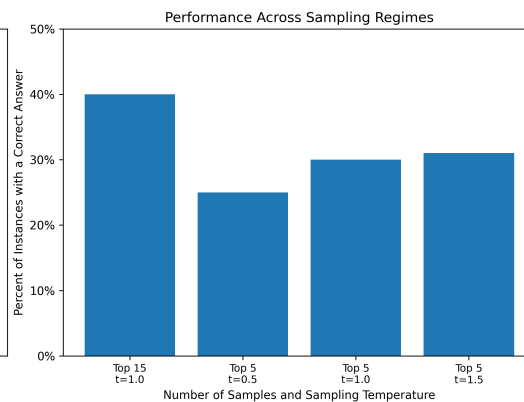


Figure 3: Performance versus sampling technique. An instance is marked correct if *any* answer in the top n was correct.

Prompting the LLM, evaluating the answer, and moving on to the next instance without any backprompts whatsoever gives a baseline score of 16%. When we run the same instances, but this time backprompt the LLM with feedback generated by the same language model acting as a verifier, performance plummets—only a single instance of the 100 was answered correctly.

Table 1: Summary of Backprompt Techniques

Strategy	Example Prompt
Direct LLM	Color the following graph, described as a set of edges, such that no two vertices on the same edge share a color. You may use at most 3 colors. Vertex 0 is connected to vertex 2...
Iterative: LLM Self-Critique	This is incorrect. Feedback: Vertices 0 and 11 share an edge and are both colored with Color 1. Vertices 5 and 11 [...] Using this feedback...
Iterative (with external Verifier): Pass/Fail	This is not correct. Using the previously provided graph...
Iterative (with external Verifier): First error	This is not correct. Vertex 1 and vertex 7 were both colored Color 1 despite being connected by an edge...
Iterative (with external Verifier): All errors	This is not correct. Vertex 1 and vertex 7 were both colored Color 1 despite being connected by an edge. Vertex 2 and vertex 4 were both colored Color 0 despite...

The problem is caused by the lack of an accurate stopping condition. If the system ever outputs a correct coloring during a backprompting session, we expect a verifier to stop it. However, in the self-verification case, the LLM doing the verification can fail to notice success and instead produce spurious feedback. This is exactly what happens.

At some point in the backprompts of 40 instances, the generating model returned an optimal coloring. In none of those instances did the verifying GPT realize this. In 39 cases, it hallucinated pairs of vertices that it claimed were adjacent and same-colored. In the one case marked correct, the coloring was provided after the final backprompt, and so became the model’s final answer by virtue of timeout. This also points to the model’s hesitancy to agree that a coloring is correct. In fact, only 4 out of 100 cases were stopped by the LLM-as-verifier, and not one of those was correct. Whether bad feedback itself is worsening the results, or it’s merely the case that correct responses tend to be earlier in the backprompt sequence—optimistically viewed as a result of being higher probability completions which are ruined by a self-destructive thinking process—is unclear. Our results here and in the next few subsections are so far conflicting.

The results when backprompted with a sound verifier seem, at first, a lot more promising. The number of instances correctly answered nears 40%, but if this is supposed to indicate that GPT-4 is listening to, improving with, and reasoning from feedback, then we should expect more informative and accurate backprompts to yield better results. However, in this domain, the raw scores (see Figure 2) don’t bear this out. When run with a sound verifier, the differences between binary feedback, a single error, or the full suite of mistakes are insignificant.

We can relax our analysis of the LLM self-critique case by labeling an instance as correct if at *any* point during the backprompt chain, the LLM generated a correct coloring. This is equivalent to rerunning the experiment with a combined feedback system: the sound verifier is in charge of stopping while allowing the LLM to write all the (still potentially spurious) feedback. Given this modification, it scores a comparable 40%. Using this charitable number, all four types of backprompting give roughly similar results.

It seems then that feedback or lack thereof is thus less important to the improvement of the score than number of iterations: if the model has fifteen chances to generate a correct answer, it is much more likely to succeed. We test this idea by querying the same set of 100 instances, but now allowing for higher temperatures and receiving multiple, separate, non-interacting responses. The results make up

Table 2: Distribution of hallucinations during verification task. This table counts the number of instances that featured each type of hallucination and compares it to the total number of erroneous edges encountered across all coloring instances in each subset.

	Hallucinations				Coloring	
	Vertex	Edge	Both	None	Errors	Correct
Correct	29	72	7	2	0	100
Ablated	24	52	5	24	187	0
Non-optimal	18	65	3	10	0	0
Random	10	26	5	66	736	0
LLM	26	41	6	27	240	18
Total	107	256	26	129	282	118

the rest of Figure 3. With $n=5$, it’s close, not quite there, but with $n=15$ ($\tau=1.0$), the performance is comparable to backprompting, achieving a score of 40%.

In other words: blindfolded guessing does just as well as careful, crafted feedback.

The rest of our analysis examines where the system is going wrong. We will attempt to answer two questions: to what extent is the LLM capable of determining if a solution is right or wrong? How, if at all, does the LLM respond to feedback?

4.2 Verification by LLM

We test GPT-4’s ability to verify colorings on the same instances, but we generate five different kinds of colorings for each. What is immediately obvious is a result that exactly agrees with the LLM self-verification results above: the model is unwilling to mark almost *any* answer as correct. Out of 100 optimal colorings, it only agreed that 2 were correct. Expanding to the entire set of 500 colorings, of which 118 of them are correct, it only claimed 30 of them as correct. Of those, it was right 5 times. This isn’t because of any special property of correctness—the same holds true in the non-optimal coloring set, in which it only marked 10% of instances as non-optimal.

Overall, this pattern holds. Fewer than ten percent of cases resulted in a "correct", "non-optimal", or "missing assignment" response from the LLM. Among those, the behavior looks somewhat random. In around a quarter of instances, it responds with a "this is incorrect" verification where the explanation matches reality, and it only manages this by naming no more than a single edge, which minimizes the chance of misstating something.

Table 2 summarizes the results. Note that, proportionally, hallucinations decrease when the error rate of the domain increases. That is to say, when there are more incorrect edges, the model is more likely to point to one of them. Intuitively, this makes sense: it’s easier to guess one edge which is wrong when half of all the edges are miscolored, as is the case on average among randomly colored graphs.

Edge hallucinations are more common than vertex. Essentially, typical behavior is to pick two vertices that are the same color in the coloring, but which aren’t associated by an edge in the graph description, and claim that they are connected and thus illegally colored. Vertex color hallucination is when the reverse happens: instead of ascribing an edge to same-color nodes, the colorings of two connected vertices are misstated. The overlap between the two cases, where a non-existent edge is declared to be violated by non-existent colorings is much rarer than either. Note that it never hallucinates new vertex *names*, only that vertices which are in graph have colors differing from reality.

Even rarer cases did spring up in the response data. At times the model lost track of the question being asked and reversed it, explicitly claiming that two same-colored vertices violate the conditions because they *aren’t* connected; or it began to contradict itself mid-deduction, making multiple claims about a vertex’s color. We relegate these examples to the appendix.

Our overall conclusion is that, despite the common-sense nature of this domain, the LLM’s verification powers are surprisingly weak.

4.3 Inside the Backprompt Chain

To figure out what information it is or isn't using, we examine the evolution of GPT-4's responses within a backprompt chain. We compare three types of informative backprompting: providing the first wrong edge, listing all wrong edges, and choosing a random correct edge to claim is incorrect. The first two cases were described in more detail above. The final one, the so-called "evil" case is new, and provided as a way to check how blindly the system follows corrective advice.

Given a backprompt, we examine the response to it. We only look at the rates of local error correction. Given a backprompt, we consider it "listened to" if the edges it listed as incorrect were changed in the response so that each vertex is a different color from the other. We summarize the results by averaging over backprompts. The results are summarized in Table 3.

Table 3: Local error correction rates per backprompt information type. Full (any) gives credit if any edge mentioned in the backprompt was corrected. Full (all) gives each backprompt response a percentage score calculated from the number of mentioned edges which were corrected divided by the total number of edges mentioned in the backprompt. Evil backprompting claims a random correct edge is incorrect.

	# Backprompts	# Incorrect Edges Fixed	% Incorrect Edges Fixed
First	1066	1004	94%
Full (any)	1102	1077	98%
Full (all)	1102	2870	84%
Evil	1083	1017	94%

Even though performance was unaffected, GPT *did* correct most errors that were pointed out. However, it didn't discriminate between real errors or the evil case's false ones, blindly applying local "fixes" without regard for overall correctness.

5 Conclusion

In this work, we have set out to investigate the effectiveness of iterative prompting strategies in improving the accuracy of LLMs on reasoning problems. We were motivated, in particular, by claims in prior work that even when LLMs produce incorrect answers at first, they are good at self-critiquing and improving their answers. Our results on graph coloring call these claims into question. They show that LLMs are in fact very poor at verifying solutions (in our case, colorings), something that is critical for self-critiquing. Not surprisingly, iterative framework with LLMs self-critiquing does even worse than LLMs directly generating a single answer. We do show that iterative prompting can help when there is an external provably correct verifier in the loop. Even here, we found that the actual content of iterative back prompts is not important, and that the improvements seen can also be obtained by just having the LLM produce multiple answers, and letting verifier check and pick any correct answer that was fortuitously generated. Our results thus raise legitimate questions about claims of the effectiveness of iterative prompting, adding further fuel to the skepticism surrounding the reasoning capabilities of LLMs.

Acknowledgments and Disclosure of Funding

This research was supported by ONR grants N00014-18-1-2442, N00014-18-1-2840, N00014-19-1-2119 and N00014-23-1-2409, and a JP Morgan AI Faculty Research Grant to Kambhampati.

References

- [1] DIMACS Implementation Challenges. Archive available at <http://archive.dimacs.rutgers.edu/Challenges/>.
- [2] Konstantine Arkoudas. Gpt-4 can't reason. *Preprints*, August 2023.
- [3] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. Sparks of artificial general intelligence: Early experiments with gpt-4, 2023.

- [4] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug, 2023.
- [5] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and Fate: Limits of Transformers on Compositionality. 2023. Publisher: arXiv Version Number: 2.
- [6] Gaël Gendron, Qiming Bao, Michael Witbrock, and Gillian Dobbie. Large language models are not abstract reasoners, 2023.
- [7] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models, 2022.
- [8] Subbarao Kambhampati. Can LLMs Really Reason and Plan?, 2023. Available at <https://cacm.acm.org/blogs/blog-cacm/276268-can-llms-really-reason-and-plan/fulltext>.
- [9] Takeshi Kojima, S. Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large Language Models are Zero-Shot Reasoners. *ArXiv*, May 2022.
- [10] Zhan Ling, Yunhao Fang, Xuanlin Li, Zhiao Huang, Mingu Lee, Roland Memisevic, and Hao Su. Deductive verification of chain-of-thought reasoning. *arXiv preprint arXiv:2306.03872*, 2023.
- [11] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhunoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023.
- [12] R. Thomas McCoy, Shunyu Yao, Dan Friedman, Matthew Hardy, and Thomas L. Griffiths. Embers of autoregression: Understanding large language models through the problem they are trained to solve, 2023.
- [13] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.
- [14] Tomohiro Sawada, Daniel Paleka, Alexander Havrilla, Pranav Tadepalli, Paula Vidas, Alexander Kranias, John J. Nay, Kshitij Gupta, and Aran Komatsuzaki. ARB: Advanced Reasoning Benchmark for Large Language Models. 2023. Publisher: arXiv Version Number: 2.
- [15] Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning, June 2023. arXiv:2303.11366 [cs].
- [16] Michael Trick. Graph Coloring Instances. Available at <https://mat.tepper.cmu.edu/COLOR/instances.html>.
- [17] Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large Language Models Still Can’t Plan (A Benchmark for LLMs on Planning and Reasoning about Change), apr 2023. arXiv:2206.10498 [cs].
- [18] Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Kang Liu, and Jun Zhao. Large language models are better reasoners with self-verification, 2023.
- [19] 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, 2023.
- [20] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023.

A Appendix

A.1 Prompts

All of following examples are built on the same graph instance.

DIMACS Format For Graphs

```
e 0 7
e 0 8
e 0 9
e 0 11
e 1 13
e 2 9
e 3 8
e 3 11
e 3 12
e 4 12
e 5 11
e 6 9
e 7 10
e 7 13
e 9 11
e 10 13
e 11 13
c OPTIMAL CHROMATIC NUMBER === 3
```

Baseline, Direct Prompt

Color the following graph, described as a set of edges, such that no two vertices on the same edge
↪ share a color.
You may use at most 3 colors.
Vertex 0 is connected to vertex 7.
Vertex 0 is connected to vertex 8.
Vertex 0 is connected to vertex 9.
Vertex 0 is connected to vertex 11.
Vertex 1 is connected to vertex 13.
Vertex 2 is connected to vertex 9.
Vertex 3 is connected to vertex 8.
Vertex 3 is connected to vertex 11.
Vertex 3 is connected to vertex 12.
Vertex 4 is connected to vertex 12.
Vertex 5 is connected to vertex 11.
Vertex 6 is connected to vertex 9.
Vertex 7 is connected to vertex 10.
Vertex 7 is connected to vertex 13.
Vertex 9 is connected to vertex 11.
Vertex 10 is connected to vertex 13.
Vertex 11 is connected to vertex 13.
There are a total of 14 vertices. Please label every vertex, even if it is disconnected from the
↪ rest of the graph. Please provide each vertex's color. Do not skip any vertices. Each color
↪ must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}:
↪ {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

Example LLM Response

```
0: Color1
1: Color1
2: Color1
3: Color2
4: Color1
5: Color2
6: Color2
7: Color2
8: Color3
9: Color3
10: Color3
11: Color3
12: Color3
13: Color1
```

Prompt To Elicit Verification

The following graph, described as a set of edges, has an optimal coloring number of 3:
Vertex 0 is connected to vertex 7.
Vertex 0 is connected to vertex 8.
Vertex 0 is connected to vertex 9.
Vertex 0 is connected to vertex 11.
Vertex 1 is connected to vertex 13.
Vertex 2 is connected to vertex 9.
Vertex 3 is connected to vertex 8.
Vertex 3 is connected to vertex 11.
Vertex 3 is connected to vertex 12.
Vertex 4 is connected to vertex 12.
Vertex 5 is connected to vertex 11.
Vertex 6 is connected to vertex 9.
Vertex 7 is connected to vertex 10.
Vertex 7 is connected to vertex 13.
Vertex 9 is connected to vertex 11.
Vertex 10 is connected to vertex 13.
Vertex 11 is connected to vertex 13.
Please check if this coloring is correct:0: Color1
1: Color1
2: Color1
3: Color2
4: Color1
5: Color2
6: Color2
7: Color2
8: Color3
9: Color3
10: Color3
11: Color3
12: Color3
13: Color1
If it is, say 'Verifier confirmed success.' Do not provide anything else in your response. If it
↪ is incorrect, please point out which same-color vertices share an edge.

A.2 Backprompts

Backprompt Generated From Self-Critique

This is incorrect. Feedback:
Vertices 0 and 3 share an edge and are both colored with Color1. Vertices 9 and 11 share an edge
↪ and are both colored with Color3.
Using this feedback, please try again. Please provide each vertex's color. Do not skip any
↪ vertices. Each color must be provided on a new line in the response and should be formatted as
↪ "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your
↪ response.

Backprompt Generated From Sound Verifier (Pass/Fail)

This is not correct. Using the previously provided graph, please provide a correct coloring.
↪ Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a
↪ new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR
↪ ASSIGNMENT}". Please do not provide anything else in your response.

Backprompt Generated From Sound Verifier (First)

Vertex 9 and vertex 11 were both colored Color3 despite being connected by an edge.
This is wrong. Please recolor. Please provide each vertex's color. Do not skip any vertices. Each
↪ color must be provided on a new line in the response and should be formatted as "{VERTEX
↪ NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

Backprompt Generated From Sound Verifier (Full)

Vertex 9 and vertex 11 were both colored Color3 despite being connected by an edge. Vertex 10 and
↪ vertex 13 were both colored Color3 despite being connected by an edge. Vertex 11 and vertex 13
↪ were both colored Color3 despite being connected by an edge.
This is wrong. Please recolor. Please provide each vertex's color. Do not skip any vertices. Each
↪ color must be provided on a new line in the response and should be formatted as "{VERTEX
↪ NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

A.3 Complete Example of Iterative Prompt Sequence

Full Sequence of 15 (LLM-Generated) Backprompts

Color the following graph, described as a set of edges, such that no two vertices on the same edge
↪ share a color.
You may use at most 3 colors.
Vertex 0 is connected to vertex 7.
Vertex 0 is connected to vertex 8.
Vertex 0 is connected to vertex 9.
Vertex 0 is connected to vertex 11.
Vertex 1 is connected to vertex 13.
Vertex 2 is connected to vertex 9.
Vertex 3 is connected to vertex 8.
Vertex 3 is connected to vertex 11.
Vertex 3 is connected to vertex 12.
Vertex 4 is connected to vertex 12.
Vertex 5 is connected to vertex 11.
Vertex 6 is connected to vertex 9.
Vertex 7 is connected to vertex 10.
Vertex 7 is connected to vertex 13.
Vertex 9 is connected to vertex 11.
Vertex 10 is connected to vertex 13.
Vertex 11 is connected to vertex 13.
There are a total of 14 vertices. Please label every vertex, even if it is disconnected from the
↪ rest of the graph. Please provide each vertex's color. Do not skip any vertices. Each color
↪ must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}:
↪ {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

```
0: Color1
1: Color1
2: Color1
3: Color1
4: Color1
5: Color2
6: Color2
7: Color2
8: Color2
9: Color3
10: Color3
11: Color3
12: Color2
13: Color3
```

This is incorrect. Feedback:
Vertices 0 and 3 share an edge and are both colored with Color1. Vertices 9 and 11 share an edge
↪ and are both colored with Color3.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any
↪ vertices. Each color must be provided on a new line in the response and should be formatted as
↪ "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your
↪ response.

```
0: Color1
1: Color1
2: Color2
3: Color2
4: Color2
5: Color1
6: Color1
7: Color2
8: Color3
9: Color3
10: Color1
11: Color2
12: Color3
```

13: Color3

This is incorrect. Feedback:

Vertices 2 and 3 are both colored with Color2 and share an edge. Vertices 7 and 11 are both colored with Color2 and share an edge. Vertices 9 and 13 are both colored with Color3 and share an edge.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

0: Color1
1: Color1
2: Color2
3: Color3
4: Color3
5: Color3
6: Color3
7: Color3
8: Color2
9: Color1
10: Color2
11: Color2
12: Color1
13: Color1

This is incorrect. Feedback:

Vertices 0 and 9 share an edge and are both colored with Color1. Vertices 0 and 11 share an edge and are both colored with Color2. Vertices 1 and 13 share an edge and are both colored with Color1. Vertices 7 and 13 share an edge and are both colored with Color3.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

0: Color1
1: Color1
2: Color2
3: Color2
4: Color3
5: Color1
6: Color1
7: Color2
8: Color3
9: Color3
10: Color1
11: Color3
12: Color1
13: Color3

This is incorrect. Feedback:

Vertices 0 and 5 both have Color1 and share an edge. Vertices 0 and 11 both have Color3 and share an edge. Vertices 7 and 10 both have Color2 and share an edge. Vertices 11 and 13 both have Color3 and share an edge.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

0: Color1
1: Color1
2: Color2
3: Color2
4: Color1
5: Color1
6: Color1
7: Color2
8: Color3
9: Color3
10: Color3
11: Color2
12: Color3
13: Color1

This is incorrect. Feedback:

Vertices 0 and 1 both have Color1 and share an edge. Vertices 0 and 5 also both have Color1 and
↪ share an edge. Vertices 0 and 6 also both have Color1 and share an edge. Vertices 7 and 11
↪ both have Color2 and share an edge. Vertices 9 and 11 both have Color2 and share an edge.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any
↪ vertices. Each color must be provided on a new line in the response and should be formatted as
↪ "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your
↪ response.

0: Color1
1: Color2
2: Color2
3: Color2
4: Color2
5: Color3
6: Color3
7: Color3
8: Color2
9: Color3
10: Color1
11: Color2
12: Color3
13: Color1

This is incorrect. Feedback:

Vertices 0 and 8 both have Color1 and share an edge. Vertices 0 and 11 both have Color1 and share
↪ an edge. Vertices 3 and 8 both have Color2 and share an edge. Vertices 3 and 11 both have
↪ Color2 and share an edge. Vertices 7 and 13 both have Color3 and share an edge. Vertices 9 and
↪ 11 both have Color3 and share an edge. Vertices 11 and 13 both have Color2 and share an edge.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any
↪ vertices. Each color must be provided on a new line in the response and should be formatted as
↪ "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your
↪ response.

0: Color1
1: Color2
2: Color2
3: Color3
4: Color1
5: Color1
6: Color1
7: Color3
8: Color2
9: Color3
10: Color2
11: Color2
12: Color3
13: Color1

This is incorrect. Feedback:

Vertices 0 and 11 share an edge and are both colored with Color1. Vertices 2 and 9 share an edge
↪ and are both colored with Color2. Vertices 3 and 12 share an edge and are both colored with
↪ Color3.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any
↪ vertices. Each color must be provided on a new line in the response and should be formatted as
↪ "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your
↪ response.

0: Color1
1: Color2
2: Color2
3: Color2
4: Color3
5: Color3
6: Color3
7: Color2
8: Color3
9: Color1
10: Color3
11: Color3
12: Color1
13: Color1

This is incorrect. Feedback:

Vertices 11 and 13 both have Color3 and share an edge.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any
↪ vertices. Each color must be provided on a new line in the response and should be formatted as
↪ "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your
↪ response.

```
0: Color1
1: Color2
2: Color2
3: Color2
4: Color3
5: Color3
6: Color3
7: Color2
8: Color3
9: Color1
10: Color3
11: Color1
12: Color1
13: Color2
```

This is incorrect. Feedback:
Vertices 0 and 11 share an edge and both are colored with Color1.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any
↪ vertices. Each color must be provided on a new line in the response and should be formatted as
↪ "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your
↪ response.

```
0: Color1
1: Color2
2: Color2
3: Color3
4: Color1
5: Color2
6: Color3
7: Color3
8: Color2
9: Color3
10: Color1
11: Color2
12: Color2
13: Color1
```

This is incorrect. Feedback:
Vertices 2 and 9 share an edge and are both colored with Color2. Similarly, vertices 11 and 13
↪ share an edge and are both colored with Color2.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any
↪ vertices. Each color must be provided on a new line in the response and should be formatted as
↪ "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your
↪ response.

```
0: Color1
1: Color2
2: Color2
3: Color3
4: Color1
5: Color2
6: Color3
7: Color3
8: Color2
9: Color1
10: Color2
11: Color3
12: Color2
13: Color1
```

This is incorrect. Feedback:
Vertices 2 and 9 share an edge and are both colored with Color2. Vertices 3 and 11 share an edge
↪ and are both colored with Color3. Vertices 7 and 13 share an edge and are both colored with
↪ Color3.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any
↪ vertices. Each color must be provided on a new line in the response and should be formatted as
↪ "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your
↪ response.

```
0: Color1
```

1: Color2
2: Color2
3: Color3
4: Color1
5: Color2
6: Color3
7: Color2
8: Color3
9: Color3
10: Color1
11: Color2
12: Color3
13: Color1

This is incorrect. Feedback:

Vertices 0 and 11 share an edge and are both colored with Color1. Vertices 3 and 8 share an edge
↪ and are both colored with Color3. Vertices 7 and 13 share an edge and are both colored with
↪ Color2.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any
↪ vertices. Each color must be provided on a new line in the response and should be formatted as
↪ "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your
↪ response.

0: Color1
1: Color2
2: Color2
3: Color3
4: Color1
5: Color2
6: Color3
7: Color2
8: Color3
9: Color1
10: Color3
11: Color2
12: Color1
13: Color3

This is incorrect. Feedback:

The coloring is incorrect. Vertices 0 and 11 share an edge and are both colored with Color1.
↪ Vertices 7 and 13 share an edge and are both colored with Color2.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any
↪ vertices. Each color must be provided on a new line in the response and should be formatted as
↪ "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your
↪ response.

0: Color1
1: Color2
2: Color2
3: Color3
4: Color1
5: Color2
6: Color3
7: Color3
8: Color2
9: Color1
10: Color2
11: Color3
12: Color1
13: Color1

This is incorrect. Feedback:

Vertices 0 and 9 share an edge and are both colored with Color1. Vertices 7 and 11 share an edge
↪ and are both colored with Color3. Vertices 1 and 13 share an edge and are both colored with
↪ Color2.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any
↪ vertices. Each color must be provided on a new line in the response and should be formatted as
↪ "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your
↪ response.

0: Color1
1: Color2
2: Color3
3: Color2
4: Color2


```
5: Color1
6: Color2
7: Color3
8: Color3
9: Color2
10: Color1
11: Color3
12: Color1
13: Color1
```

This is incorrect. Feedback:
Vertices 3 and 9 share an edge and are both colored with Color2.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any
↪ vertices. Each color must be provided on a new line in the response and should be formatted as
↪ "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your
↪ response.

```
0: Color1
1: Color2
2: Color3
3: Color2
4: Color3
5: Color1
6: Color2
7: Color3
8: Color2
9: Color1
10: Color2
11: Color3
12: Color1
13: Color1
```

A.4 LLM as Verifier

In this section, multiple equals signs in a row ("====") are dividers between separate examples, not parts of the examples.

Examples of LLM Output on the Verification Task

The vertices 1 and 4 are both color 0 and share an edge.
=====

The vertices 9 and 12 are both colored with Color 1 and they share an edge.
=====

Vertex 1: Color 2 is connected to Vertex 2: Color 2.
=====

Vertex 5 and vertex 10 are both Color 2 and share an edge.
=====

Same-color vertices 0 and 1 share an edge.
=====

Vertex 0 (Color 0) and vertex 3 (Color 0) share an edge. Vertex 1 (Color 1) and vertex 2 (Color 1)
↪ share an edge. Vertex 4 (Color 1) and vertex 8 (Color 1) share an edge. Vertex 2 (Color 1) and
↪ vertex 4 (Color 1) share an edge. Vertex 4 (Color 1) and vertex 11 (Color 0) share an edge.
↪ Vertex 12 (Color 2) and vertex 15 (Color 2) share an edge.

The vertices 3 and 0 both have Color 2 and they are adjacent. The vertices 1 and 11 both have
↪ Color 1 and they are adjacent. The vertices 2 and 5 both have Color 0 and they are adjacent.
↪ Therefore, the given coloring is incorrect.
=====

0: Color 0 and 9: Color 0 share an edge.

0: Color 0 and 10: Color 0 share an edge.

0: Color 0 and 14: Color 0 share an edge.

9: Color 0 and 13: Color 0 share an edge.
=====

Vertex 3 is colored with Color 4, which exceeds the optimal coloring number of 4. Therefore, this
↪ coloring is non-optimal.
=====

Vertex 0 is missing in the coloring. Missing assignment.
=====

Vertex 0 (color 3) is connected to vertex 3 (color 2), sharing an edge. Vertex 0 (color 3) is also
↪ connected to vertex 4 (color 0), and to vertex 5 (color 0), and to vertex 9 (color 0), and to
↪ vertex 12 (color 2), sharing an edge with them. Therefore, this assignment is correctly
↪ colored with color 0, color 2, and color 3, making use of 3 different colors.

Vertex 1 (color 2) is connected to vertex 4 (color 0), and to vertex 5 (color 0), and to vertex 6 (color 0), and to vertex 7 (color 3), and to vertex 8 (color 0), and to vertex 10 (color 1), and to vertex 11 (color 1), sharing an edge with them. Therefore, this assignment is also correctly colored.

Vertex 2 (color 0) is connected to vertex 12 (color 2), sharing an edge. Therefore, this assignment is correctly colored.

Vertex 3 (color 2) is connected to vertex 5 (color 0), and to vertex 8 (color 0), and to vertex 10 (color 1), sharing an edge with them. Therefore, this assignment is correctly colored.

Vertex 4 (color 0) is connected to vertex 7 (color 3), and to vertex 11 (color 1), sharing an edge. Therefore, this assignment is correctly colored.

Vertex 5 (color 0) is connected to vertex 7 (color 3), sharing an edge. Therefore, this assignment is correctly colored.

Vertex 6 (color 0) is connected to vertex 10 (color 1), and to vertex 12 (color 2), sharing an edge. Therefore, this assignment is correctly colored.

Vertex 8 (color 0) is connected to vertex 10 (color 1), sharing an edge. Therefore, this assignment is correctly colored.

Vertex 9 (color 0) is connected to vertex 11 (color 1), sharing an edge. Therefore, this assignment is correctly colored.

Vertex 10 (color 1) is connected to vertex 12 (color 2), sharing an edge. Therefore, this assignment is correctly colored.

Vertex 11 (color 1) isn't connected to vertex 10 (color 1), hence, this assignment is correctly colored.

Vertex 12 (color 2) isn't connected to vertex 2 (color 0), hence, this assignment is correctly colored.

No adjacent vertices share a same color, and no colors exceed the optimal coloring number. Hence, the coloring is correct.
 Verifier confirmed success.

Examples of (Rare) Mangled, Illogical, and Otherwise Strange Output on the Verification Task

Vertex 10, colored 4, is connected to vertex 11, also colored 1. However, it is also colored 4, which is greater than the optimal coloring of 4. So, this coloring is non-optimal.
 =====

Vertex 9 (Color 2) and Vertex 10 (Color 1) are connected and share the same color. Vertex 9 (Color 2) and Vertex 11 (Color 1) are also connected and share the same color.
 =====

Vertex 2 (Color 0) is connected to Vertex 0 (Color 0).
 Vertex 5 (Color 0) is connected to Vertex 1 (Color 2) and Vertex 14 (Color 1), but also to Vertex 7 (Color 1).
 Vertex 6 (Color 0) is connected to Vertex 3 (Color 2), Vertex 10 (Color 1) and 13 (Color 2), but also to Vertex 8 (Color 2).
 =====

The coloring is incorrect. Vertex 1 (Color 2) and Vertex 3 (Color 3) are connected but have different colors. Vertex 3 (Color 3) is connected to Vertex 6 (Color 3).
 =====

Vertex 10, colored 4, is connected to vertex 11, also colored 1. However, it is also colored 4, which is greater than the optimal coloring of 4. So, this coloring is non-optimal.