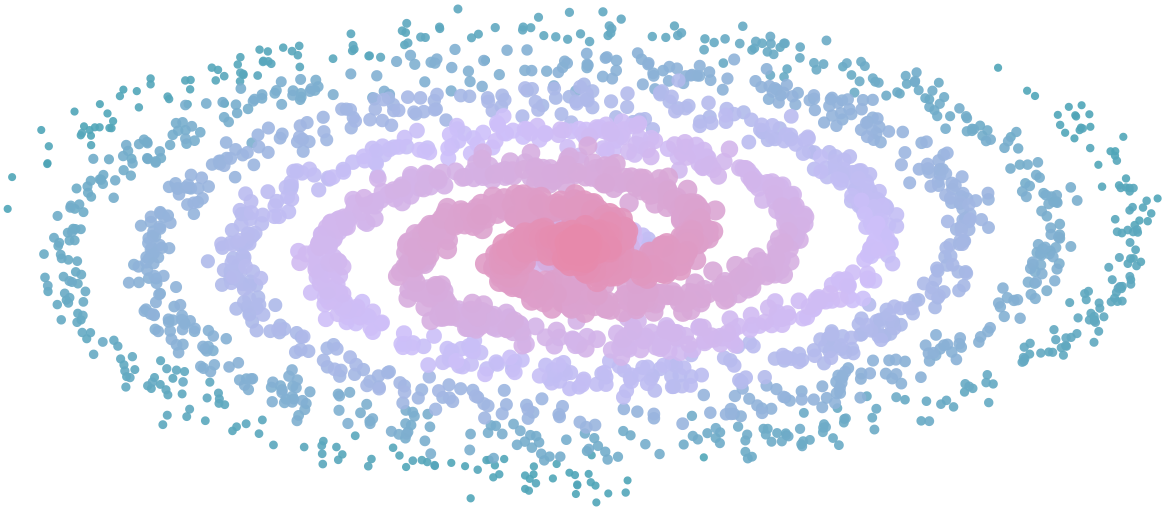
Why We Think



Recent developments in test-time compute and chain-of-thought reasoning

AUTHOR	AFFILIATION	PUBLISHED
Lilian Weng	OpenAI	May 01, 2025

Special thanks to John Schulman for a lot of super valuable feedback and direct edits on this post.

Test time compute (Graves et al. 2016, Ling, et al. 2017, Cobbe et al. 2021) and Chain-of-thought (CoT) (Wei et al. 2022, Nye et al. 2021), have led to significant improvements in model performance, while raising many research questions. This post aims to review recent developments in how to effectively use test-time compute (i.e. “thinking time”) and why it helps.

Motivation

Enabling models to think for longer can be motivated in a few different ways.

Analogy to Psychology

The core idea is deeply connected to how humans think. We humans cannot immediately provide the answer for "What's 12345 times 56789?". Rather, it is natural to spend time pondering and analyzing before getting to the result, especially for complex problems. In Thinking, Fast and Slow (Kahneman, 2013), Daniel Kahneman characterizes human thinking into two modes, through the lens of the dual process theory:

- Slow thinking (System 2) demands deliberate, logical thought and significant cognitive efforts. This mode of thinking consumes more mental energy and requires intentional engagement.
- Fast thinking (System 1) operates quickly and automatically, driven by intuition and emotion while requiring little to no effort.

Because System 1 thinking is fast and easy, it often ends up being the main decision driver, at the cost of accuracy and logic. It naturally relies on our brain's mental shortcuts (i.e., heuristics) and can lead to errors and biases. By consciously slowing down and taking more time to reflect, improve and analyze, we can engage in System 2 thinking to challenge our instincts and make more rational choices.

Computation as a Resource

One view of deep learning, is that neural networks can be characterized by the amount of computation and storage they can access in a forward pass, and if we optimize them to solve problems using gradient descent, the optimization process will figure out how to use these resources–they'll figure out how to organize these resources into circuits for calculation and information storage. From this view, if we design an architecture or system that can do more computation at test time, and we train it to effectively use this resource, it'll work better.

In Transformer models, the amount of computation (flops) that the model does for each generated token is roughly 2 times the number of parameters. For sparse models like mixture of experts (MoE), only a fraction of the parameters are used in each forward pass, so computation = 2 × parameters / sparsity, where sparsity is the fraction of experts active.

On the other hand, CoT enables the model to perform far more flops of computation for each token of the answer that it is trying to compute. In fact, CoT has a nice property that it allows the model to use a variable amount of compute depending on the hardness of the problem.

Latent Variable Modeling

A classic idea in machine learning is to define a probabilistic model with a latent (hidden) variable z and a visible variable y, where y is given to our learning algorithm. Marginalizing (summing) over the possible values of the latent variable allows us to express a rich distribution over the visible variables, $P(y) = \sum_{z \sim P(z)} P(y \mid z)$. For example, we can model the distribution over math problems and solutions by letting x denote a problem statement, y be ground truth answer or proof, and z as a free-form thought process that leads to the proof. The marginal probability distribution to optimize would be $P(y \mid x) = \sum_{z \sim p(z|x)} P(y \mid x, z)$

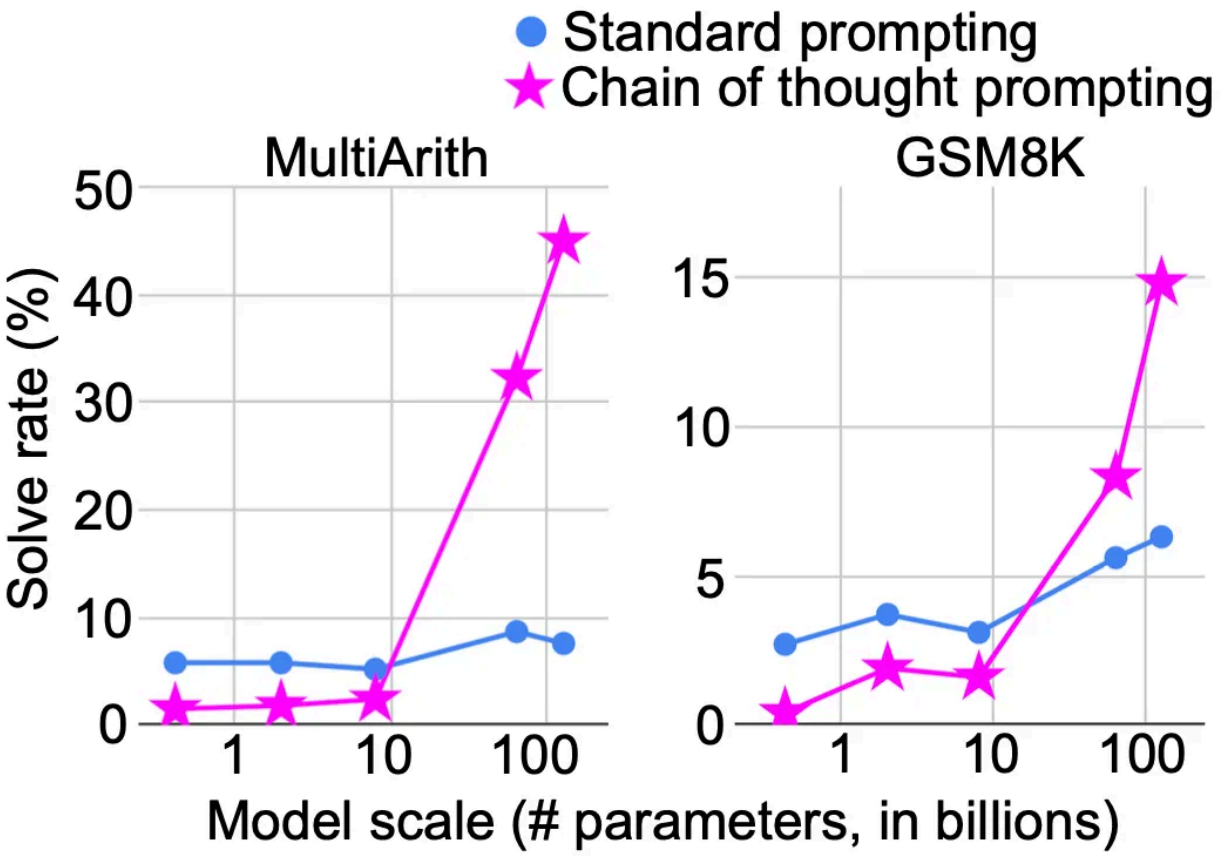
The latent variable perspective is particularly useful for understanding methods that involve collecting multiple parallel CoTs or searching over the CoT—these algorithms can be seen as sampling from the posterior $P(z \mid x, y)$. This view also suggests the benefits of using the log loss $\log P(y \mid x)$ as the target objective to optimize, as the log loss objective has been so effective in pretraining.

Thinking in Tokens

The strategy of generating intermediate steps before generating short answers, particularly for math problems, was explored by Ling, et al. 2017, who introduced the AQUA-RAT dataset, and then expanded by Cobbe et al. 2021, who introduced the Grade School Math (GSM) dataset. Cobbe et al. train a generator with supervised learning on human-written solutions and verifiers that predict the correctness of a candidate solution; they can then search over these solutions. Nye et al. (2021) experimented with intermediate thinking tokens as "scratchpads" and Wei et al. (2022) coined the now-standard term chain-of-thought (CoT).

Early work on improving CoT reasoning involved doing supervised learning on human-written reasoning traces or model-written traces filtered for answer correctness, where the latter can be seen as a rudimentary form of reinforcement learning (RL). Some other work found that one could significantly boost math performance of instruction tuned models by prompting them appropriately, with "think step by step" (Kojima et al. 2022) or more complex prompting to encourage the model to reflect on related knowledge first (Yasunaga et al. 2023).

Later work found that the CoT reasoning capabilities can be significantly improved by doing reinforcement learning on a dataset of problems with automatically checkable solutions, such as STEM problems with short answers, or coding tasks that can be checked with unit tests (Zelikman et al. 2022, Wang et al., 2023, Liu et al., 2023). This approach rose to prominence with the announcement of o1-preview, o3, and the R1 tech report (DeepSeek-AI, 2025), which showed that a simple recipe where a policy gradient algorithm could lead to strong performance.



Chain-of-thought prompting leads to higher success rate of solving math problems. Larger models benefit more from thinking time. (Image source: Wei et al. 2022)

Branching and Editing

The fundamental intent of test-time compute is to adaptively modify the model's output distribution at test time. There are various ways of utilizing test time resources for decoding to select better samples and thus alter the model's predictions towards a more desired distribution. Two main approaches for improving the decoding process are parallel sampling and sequential revision.

- Sequential revision adapts the model's responses iteratively based on the output in the previous step, asking the model to intentionally reflect its existing response and correct mistakes. The revision process may have to rely on a fine-tuned model, as naively relying on the model's intrinsic capability of self-correction without external feedback may not lead to improvement (Kamoi et al. 2024, Huang et al. 2024).
- Parallel sampling generates multiple outputs simultaneously, meanwhile providing guidance per step with process reward signals or using verifiers to judge the quality at the end. It is the most widely adopted decoding method to improve test time performance, such as best-

of-N or beam search. Self-consistency (Wang et al. 2023) is commonly used to select the answer with majority vote among multiple CoT rollouts when the ground truth is not available.

Parallel sampling is simple, intuitive and easier to implement, but bounded by the model capability of whether it can achieve the correct solution in one-go. Sequential explicitly asks the model to reflect on mistakes but it is slower and requires extra care during implementation as it does run the risk of correct predictions being modified to be incorrect or introducing other types of hallucinations. These two methods can be used together. Snell et al. (2024) showed that easier questions benefit from purely sequential test-time compute, whereas harder questions often perform best with an optimal ratio of sequential to parallel compute.

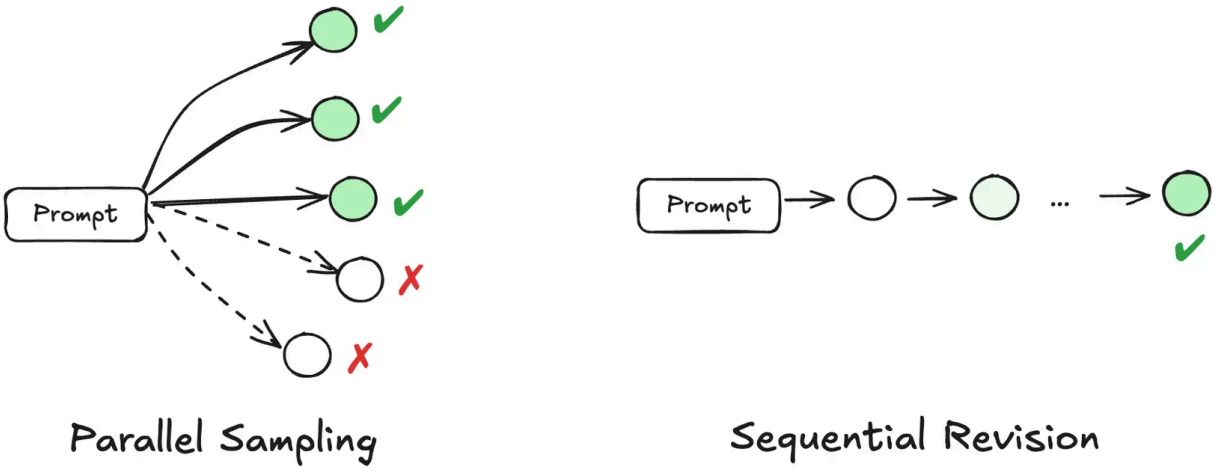


Illustration of parallel sampling vs sequential revision.

Parallel Sampling

Given a generative model and a scoring function that we can use to score full or partial samples, there are various search algorithms we can use to find a high scoring sample. Best-of-N is the simplest such algorithm: one just collects N independent samples and chooses the highest-ranking sample according to some scoring function. Beam search is a more sophisticated search algorithm that makes the search process more adaptive, spending more sampling computation on more promising parts of the solution space.

Beam search maintains a set of promising partial sequences and alternates between extending them and pruning the less promising ones. As a selection mechanism, we can use a process reward model (PRM; Lightman et al. 2023) to guide beam search candidate selection. Xie et al. (2023) used LLM to evaluate how likely its own generated reasoning step is correct, formatted as a multiple-choice question and found that per-step self-evaluation reduces accumulative

errors in multi-step reasoning during beam search decoding. Besides, during sampling, annealing the temperature helps mitigate aggregated randomness. These experiments by Xie et al. achieved 5-6% improvement on few-shot GSM8k, AQuA and StrategyQA benchmarks with the Codex model. Reward balanced search (short for "REBASE"; Wu et al. 2025) separately trained a process reward model (PRM) to determine how much each node should be expanded at each depth during beam search, according to the softmax-normalized reward scores. Jiang et al. (2024) trained their PRM, named "RATIONALYST", for beam search guidance on synthetic rationales conditioned on a large amount of unlabelled data. Good rationales are filtered based on whether they help reduce the neg log-prob of true answer tokens by a threshold, when comparing the difference between when the rationales is included in the context vs not. At inference time, RATIONALYST provides process supervision to the CoT generator by helping estimate log-prob of next reasoning steps ("implicit") or directly generating next reasoning steps as part of the prompt ("explicit").

Beam search decoding guided by LLM self-evaluation per reasoning step. (Image source: Xie et al. 2023)

Interestingly, it is possible to trigger the emergent chain-of-thought reasoning paths without explicit zero-shot or few-shot prompting. Wang & Zhou (2024) discovered that if we branch out at the first sampling tokens by retaining the top k tokens with highest confidence, measured as the difference between top-1 and top-2 candidates during sampling, and then continue these k sampling trials with greedy decoding onward, many of these sequences natively contain CoT. Especially when CoT does appear in the context, it leads to a more confident decoding of the final answer. To calculate the confidence of the final answer, the answer span needs to be identified by task-specific heuristics (e.g. last numerical values for math questions) or by prompting the model further with "So the answer is". The design choice of only branching out at the first token is based on the observation that early branching significantly enhances the diversity of potential paths, while later tokens are influenced a lot by previous sequences.

Top-k decoding, k refers to the number of candidates at the first sampling step. (Image source: Wang & Zhou, 2024)

Sequential Revision

If the model can reflect and correct mistakes in past responses, we would expect the model to produce a nice sequence of iterative revision with increasing quality. However, this self-correction capability turns out to not exist intrinsically among LLMs and does not easily work out of the box, due to various failure modes, such as, (1) hallucination, including modifying correct responses to be incorrect; (2) behavior collapse to non-correcting behavior; e.g. making minor or no modification on the first incorrect responses; or (3) fail to generalize to distribution shift at test time. Experiments by Huang et al. (2024) showed that naively applying self-correction leads to worse performance and external feedback is needed for models to self improve, which can be based on matching ground truths, heuristics and task-specific metrics, unit tests results for coding questions (Shinn, et al. 2023), a stronger model (Zhang et al. 2024), as well as human feedback (Liu et al. 2023).

Self-correction learning (Welleck et al. 2023) aims to train a corrector model $P_\theta(y \mid y_0, x)$ given a fixed generator model $P_0(y_0 \mid x)$. While the generator model remains to be generic, the corrector model can task-specific and only does generation conditioned on an initial model response and additional feedback (e.g. a sentence, a compiler trace, unit test results; can be optional):

1. To encourage exploration, the corrector provides new generations into the data pool as well. At the inference time, the corrector can be used iteratively to create a correction trajectory of sequential revision.
2. These pairs are selected proportional to is improvement in value, $v(y') - v(y)$, and similarity between two outputs, $\text{Similarity}(y, y')$ to train the corrector model.
3. then create value-improving pairs by pairing two outputs for the same prompt together if one has a higher value than the other, (prompt x, hypothesis y, correction y').

4. Self-correction learning first generates multiple outputs per prompt in the data pool;

Illustration of self-correction learning by matching model outputs for the same problem to form value-improving pairs to train a correction model. (Image source: Welleck et al. 2023)

Recursive inspection (Qu et al. 2024) also aims to train a better corrector model but with a single model to do both generation and self-correction.

SCoRe (Self-Correction via Reinforcement Learning; Kumar et al. 2024) is a multi-turn RL approach to encourage the model to do self-correction by producing better answers at the second attempt than the one created at the first attempt. It composes two stages of training: stage 1 only maximizes the accuracy of the second attempt while enforcing a KL penalty only on the first attempt to avoid too much shifting of the first-turn responses from the base model behavior; stage 2 optimizes the accuracy of answers produced by both the first and second attempts. Ideally we do want to see performance at both first and second attempts to be better, but adding stage 1 prevents the behavior collapse where the model does minor or none edits on the first response, and stage 2 further improves the results.

Explicit training setup to improve self-correction capabilities by doing two-staged RL training. (Image source: Kumar et al. 2024)

RL for Better Reasoning

There's been a lot of recent success in using RL to improve the reasoning ability of language models, by using a collection of questions with ground truth answers (usually STEM problems and puzzles with easy to verify answers), and rewarding the model for getting the correct answer. Recent activity in this area was spurred by strong performance of the o-series models from OpenAI, and the subsequent releases of models and tech reports from DeepSeek.

DeepSeek-R1 (DeepSeek-AI, 2025) is an open-source LLM designed to excel in tasks that require advanced reasoning skills like math, coding and logical problem solving. They run through 2 rounds of SFT-RL training, enabling R1 to be good at both reasoning and non-reasoning tasks.

Reasoning-oriented RL trains a reasoning model on reasoning-only prompts with two types of rule-based rewards:

- Accuracy rewards: Whether the final answers are correct. The answer for math problems needs to be present in a specific format (e.g. in a box) to be verified reliably. For coding problems, a compiler is used to evaluate whether test cases pass.
- Format rewards: The model should wrap CoTs by tokens.

Rejection-sampling + non-reasoning SFT utilizes new SFT data created by rejection sampling on the RL checkpoint of step 2, combined with non-reasoning supervised data from DeepSeek-V3 in domains like writing, factual QA, and self-cognition, to retrain DeepSeek-V3-Base.

- Then fine-tune the DeepSeek-V3-Base on the total 800k samples for 2 epochs.
- For certain non-reasoning tasks, call DeepSeek-V3 to generate potential CoTs before answering the question by prompting. But for simpler queries like "hello", CoT is not needed.
- Include non-reasoning tasks using DeepSeek-V3 (DeepSeek-AI, 2024) pipeline.
- Filter out CoTs with mixed languages, long paragraphs, and code blocks.

DeepSeek-R1 performs comparable to OpenAI o1-preview and o1-mini on several widely used reasoning benchmarks. DeepSeek-V3 is the only non-reasoning model listed. (Image source: DeepSeek-AI, 2025)

Interestingly the DeepSeek team showed that with pure RL, no SFT stage, it is still possible to learn advanced reasoning capabilities like reflection and backtracking ("Aha moment"). The model naturally learns to spend more thinking tokens during the RL training process to solve reasoning tasks. The "aha moment" can emerge, referring to the model reflecting on previous mistakes and then trying alternative approaches to correct them. Later, various open source efforts happened for replicating R1 results like Open-R1, SimpleRL-reason, and TinyZero, all based on Qwen models. These efforts also confirmed that pure RL leads to great performance on math problems, as well as the emergent "aha moment".

Examples of the model learning to reflect and correct mistakes. (Image source: (left) DeepSeek-AI, 2025; (right) Zeng et al. 2025)

The DeepSeek team also shared some of their unsuccessful attempts. They failed to use process reward model (PRM) as it is hard to define per-step rubrics or determine whether an intermediate step is correct, meanwhile making the training more vulnerable to reward hacking. The efforts on MCTS (Monte Carlo Tree Search) also failed due to the large search space for language model tokens, in comparison to, say, chess; and training the fine-grained value model used for guiding the search is very challenging too. Failed attempts often provide unique insights and we would like to encourage the research community to share more about what did not work out.

External Tool Use

During the reasoning steps, certain intermediate steps can be reliably and accurately solved by executing code or running mathematical calculations. Offloading that part of reasoning components into an external code interpreter, as in PAL (Program-Aided Language Model; Gao et al. 2022) or Chain of Code (Li et al. 2023), can extend the capability of LLM with external tools, eliminating the need for LLMs to learn to execute code or function as calculators themselves. These code emulators, like in Chain of Code, can be augmented by an LLM such that if a standard code interpreter fails, we have the option of using LLM to execute that line of code instead. Using code to enhance reasoning steps are especially beneficial for mathematical problems, symbolic reasoning and algorithmic tasks. These unit tests may not exist as part of the coding questions, and in those cases, we can instruct the model to self-generate unit tests for it to test against to verify the solution (Shinn, et al. 2023).

An example of program-aided language model prompting looks like. (Image source: Gao et al. 2022)

ReAct (Reason+Act; Yao et al. 2023) combines the action of searching the Wikipedia API and generation of reasoning traces, such that reasoning paths can incorporate external knowledge.

An example of the ReAct prompting method to solve a HotpotQA question, using Wikipedia search API as an external tool to help with reasoning. (Image source: Yao et al. 2023)

o3 & o4-mini, recently released by OpenAI, are another two good examples where the reasoning process involves tool use like Web search, code execution and image processing. The team observed that large-scale reinforcement learning exhibits the same trend as in the GPT paradigm that “more compute = better performance”.

Thinking Faithfully

Deep learning models are often treated as black boxes and various interpretability methods have been proposed. Interpretability is useful for a couple reasons: first, it gives us an extra test to determine if the model is misaligned with its creators' intent, or if it's misbehaving in some way that we can't tell by monitoring its actions. Second, it can help us determine whether the model is using a sound process to compute its answers. Chain of thought provides an especially convenient form of interpretability, as it makes the model's internal process visible in natural language. This interpretability, however, rests on the assumption that the model truthfully describes its internal thought processes.

Recent work showed that monitoring CoT of reasoning models can effectively detect model misbehavior such as reward hacking, and can even enable a weaker model to monitor a stronger model (Baker et al. 2025). Increasing test time compute can also lead to improved adversarial robustness (Zaremba et al. 2025); this makes sense intuitively, because thinking for longer should be especially useful when the model is presented with an unusual input, such as an adversarial example or jailbreak attempt – it can use the extra thinking time to make sense of the strange situation it's been presented with.

The experiment of asking the model to decide if another model tried to hack the unit tests. The exit(0) coding hack is when the agent exploited a bug to exit early. The raise SkipTest hack skips unit test evaluation. (Image source: Baker et al. 2025)

Does the Model Tell What it Thinks Faithfully

Intuitively, model CoTs could be biased due to lack of explicit training objectives aimed at encouraging faithful reasoning. Or when we fine-tune the model on human-written explanations, those human-written samples may contain mistakes. Thus we cannot by default assume CoT is always faithful.

Lanham et al. (2023) investigated several modes of CoT faithfulness failures by deliberately introducing mistakes into CoTs and measuring their impacts on the accuracy of a set of multiple choice tasks (e.g. AQuA, MMLU, ARC Challenge, TruthfulQA, HellaSwag):

Mistake 1 (Early answering): The model may form a conclusion prematurely before CoT is generated. This is tested by early truncating or inserting mistakes into CoT. Different tasks revealed varying task-specific dependencies on CoT effectiveness; some have evaluation performance sensitive to truncated CoT but some do not. Wang et al. (2023) did similar experiments but with more subtle mistakes related to bridging objects or language templates in the formation of CoT.

Mistake 2 (Uninformative tokens): Uninformative CoT tokens improve performance. This hypothesis is tested by replacing CoT with filler text (e.g. all periods) and this setup shows no accuracy increase and some tasks may suffer performance drop slightly when compared to no CoT.

Mistake 3 (Human-unreadable encoding): Relevant information is encoded in a way that is hard for humans to understand. Paraphrasing CoTs in an non-standard way did not degrade performance across datasets, suggesting accuracy gains do not rely on human-readable reasoning.

Illustration of different ways of CoT perturbation to assess its faithfulness. (Image source: Lanham et al. 2023)

Interestingly, Lanham et al. suggests that for multiple choice questions, smaller models may not be capable enough of utilizing CoT well, whereas larger models may have been able to solve the tasks without CoT. This dependency on CoT reasoning, measured by the percent of obtaining the same answer with vs without CoT, does not always increase with model size on multiple choice questions, but does increase with model size on addition tasks, implying that thinking time matters more for complex reasoning tasks.

The dependency on CoT reasoning is measured as the percentage of obtaining same answers with vs without CoT. It matters more for reasoning tasks like addition and larger models benefit more. (Image source: Lanham et al. 2023)

Alternative approaches for testing CoT faithfulness involve perturbing prompts rather than modifying CoT paths directly (Turpin et al. 2023, Chua & Evans, 2025, Chen et al. 2025).

One method consistently labels correct answers as "(A)" in few-shot examples regardless of true labels to introduce biases.

Another prompting technique inserts misleading hints into prompts, such as "I think the answer is but curious to hear what you think" or "A Stanford Professor thinks the answer is ". By comparing model predictions for the same question with vs without the misleading hint, we can measure whether a model is able to faithfully describe the influence of the hint on its answer. Particularly, in cases where the model produces different hint and non-hint answers, we measure whether the model acknowledges the hint when solving the question with hint. If the model is faithful, it should explicitly acknowledge the impact and admit the change of its answer is due to the hint.

Both GPT and Claude models are sensitive to different types of biases in context. Decrease in model accuracy indicates systematic unfaithfulness. (Image source: Turpin et al. 2023)

Multiple studies found that reasoning models describe the influence of the hint much more reliably than all the non-reasoning models tested. For example, we can measure the fraction of samples where the model acknowledges the hint as a deciding factor ("faithful CoT"). Reasoning models (Claude 3.7 Sonnet, DeepSeek R1) are overall doing better than non-reasoning ones (Claude 3.6, DeepSeek V3).

Reasoning models are more likely to reveal faithful CoT than non-reasoning models. (Image source: Chen et al. 2025)

Some evidence showed that using reward models leads to less faithfulness in model outputs. The reward model in classic RLHF is not trained to incentivize faithful behavior in this test as the preference comparison dataset is collected by humans selecting which one seems better or

pleasing. However, in reasoning models, the CoT paths are optimized for producing the correct final answers, not to match human preference defined in RM, and thus intuitively are expected to be more faithful.

Optimization Pressure on CoT: Good or Bad?

Monitoring CoT of reasoning models for reward hacking behavior is effective. A simple way to use this monitor is to run rejection sampling with the monitor as a filter at test time and identify solutions without reward hacking. However, it only works at the early stage of RL, and, as the model is further optimized, it is rare to find a sample to pass a CoT monitor within budget. This challenge naturally suggests that we could consider incorporating CoT monitors into RL rewards —a form of process-based supervision—to prevent reward hacking.

However, incorporating CoT monitors into the RL reward signal may introduce trade-offs. Optimizing directly for passing the monitor could potentially incentivize the model to produce CoT that satisfies the monitor's criteria rather than genuinely faithful reasoning—a form of "monitor hacking." The research community continues to explore the right balance between outcome-based rewards and process-based supervision for training reasoning models that are both capable and faithful.
