DOI: 10.1111/nyas.15339

COMMENTARY

ANNALS OF THE NEW YORK ACADEMY OF SCIENCES

THE SCIENCES

(How) Do reasoning models reason?

Subbarao Kambhampati | **Kaya Stechly** | **Karthik Valmeekam**

School of Computing & Augmented Intelligence, Arizona State University, Tempe, Arizona, USA

Correspondence
Subbarao Kambhampati, School of Computing, and Augmented Intelligence, Arizona State University, Tempe, AZ 85287, USA.
Email: rao@asu.edu

Funding information
Office of Naval Research, Grant/Award Number: N0001423-12409; Qualcomm; Defense Advanced Research Projects Agency, Grant/Award Number: HR00112520016

Abstract

We provide a broad unifying perspective on the recent breed of large reasoning models such as OpenAI o1 and DeepSeek R1, including their promise, sources of power, misconceptions, and limitations.

KEYWORDS
large language models, reasoning, planning, reinforcement learning post-training, test-time inference

INTRODUCTION

Large language models (LLMs), which have been autoregressively trained on humanity's digital footprint, have shown the ability to generate coherent text responses to a vast variety of prompts. Although they show impressive System 1 capabilities and excel in producing completions that mimic appropriate styles, System 2 capabilities like factuality, reasoning, and planning have remained elusive aims, if not Achilles heels. [1]

In response, researchers have developed a new breed of models—sometimes called *large reasoning models* (LRMs)—which build on vanilla LLM architectures and training recipes. The best-known of these are OpenAI's o1 and DeepSeek's R1, which have shown significant performance improvements on reasoning and planning tasks previously outside the range of older LLM capabilities.

These models have been built on insights from two broad but largely orthogonal classes of ideas: (i) *test-time inference* scaling techniques, which involve getting LLMs to do more work than simply providing the most likely direct answer; and (ii) *post-training methods*, which complement simple autoregressive training on web corpora, with additional training on data containing intermediate tokens. In this article, we will use the term "derivational traces" as a neutral stand-in for these intermediate tokens, rather than the more popular anthropomorphized phrases "chains of thought" and "reasoning traces."

Although these ideas are leading to performance leaps on benchmarks, there is little consensus yet on when and why they work. Test-time scaling approaches that leverage sound verification can be made sense of because they enforce guarantees on the output; but other popular methods are only justified empirically, working in some domains but not in others. There is even more confusion about the success of new post-training techniques. As we will argue, the prevalent narrative—that novel generalization capabilities arise from training LLMs on chains of thought so that they produce intermediate tokens dubbed "reasoning traces" before solutions—is questionable.

Our aim in this article is to provide a broad unifying perspective on these approaches, including their promise, sources of power misconceptions, and limitations.

BUILDING REASONING MODELS

Test-time inference

Not all problems require an equal amount of effort or time. A two digit by two digit addition problem can be solved with just three one-digit additions, while a four by four digit problem may require seven. There is a rich history of approaches that use scalable online computation to improve upon faster initial guesses, including limited depth min-max, real-time A* search and dynamic programming, and Monte Carlo tree search.[2] Test-time inference approaches (see Figure 1) mirror these ideas and are inspired by the anthropomorphic observation that people

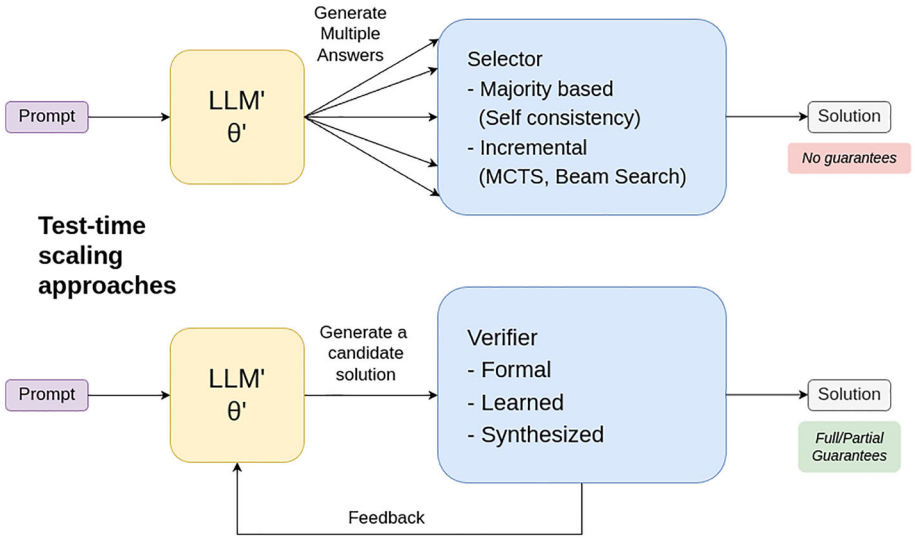

FIGURE 1 Test-time scaling approaches for teasing out reasoning. (MCTS: Monte Carlo tree search)

also seem to do better on certain problems when they can think longer about them.

Perhaps the most popular and enduring class of test-time inference ideas involves generating many candidate solutions from an LLM and using some selection procedure to choose the final output. The simplest implementation is known as *self-consistency*:[3] choose the most common answer. Total time spent is proportional to the number of solutions generated, but this method provides no theoretical guarantees that its answers will be more correct.[a]

More sophisticated selection procedures attempt to verify that an LLM's output is correct. When paired with an LLM in this manner, the combined system can be seen as a *generate-test* framework, and naturally raises questions about the verification process: *Who does it*, and *with what guarantees?* A variety of approaches have been tried—including using LLMs themselves as verifiers[4] (although this is known to be problematic[5]), learning verifiers,[6,7] and using external sound verifiers that come with either full or partial guarantees. In cases where verifiers provide explanations or feedback when a guess is incorrect, these can be passed back to the LLM so it generates better subsequent guesses. Our LLM-Modulo framework[1,8] provides a thorough overview of these types of verification-based approaches, along with their guarantees.

Post-training on derivational traces

LLMs are trained using a very simple objective: given a chunk of text, predict the most likely next token. This procedure, when employed with sufficiently high-capacity models on web-scale corpora, has been surprisingly successful at inducing the ability to capture the style of many different classes of text. The linguistic medium they operate over and the sheer amount of varied data they have ingested opens up the possibility of applying them to nearly any domain, including reasoning and planning. However, while sufficiently accurate mimicry on peta scale corpora might be enough to succeed at these tasks in theory, vanilla LLMs have not yet managed to do so. Their completions almost always look reasonable despite often being incorrect,[1] seemingly relying on statistical features and stylistic quirks rather than robust procedures, even when this fails.

One intuition driving today's research is that this is partly because the training data are incomplete. LLMs have soaked up every article, post, and book on the Internet but not what it took to produce them—whether internal verbalizations, scratch paper outlines, or typed-up but discarded drafts. Perhaps, the hope here goes, if more of these *reasoning traces* were included, this would help LLMs replay versions of the same processes.

While promising, it is far from immediately clear how to source data like this at a sufficient scale. There are few if any large collections of generic derivational traces. Not only is it burdensome for people to produce granular step-by-step representations of their own thoughts, but they are unlikely to have direct and explicit access to those processes in the first place. And in those cases where they do, they may deliberately efface their tracks. As Gauss famously remarked when asked to give step-wise intuitions for his proofs: no self-respecting architect leaves the support structure in place once the edifice is complete!

Nevertheless, a variety of approaches have tried to make up for this shortfall, ranging from paying annotators for step-by-step derivations to generating and selecting them with LLMs. We classify these in terms of (i) how candidate traces are generated and filtered and (ii) how they are used to improve the underlying LLM; see Figure 2.

Before diving into the details, we should point out that the gap between this anthropomorphic motivation in terms of internal thoughts and actual LLM operation is quite stark. Often, the "derivational traces" used in practice do not have any systematic relation to

[a] The hope here is that generated answers will be distributed around the correct answer, rather than skewed elsewhere by some systematic source of error, and that thus selecting from larger sets of noisy trajectories will converge to the desired response.

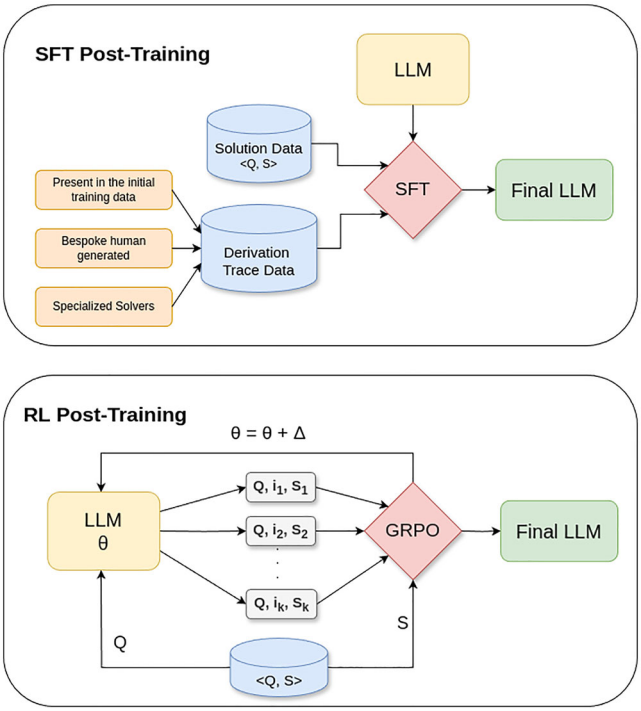

FIGURE 2 Post-training approaches for teasing out reasoning. LLM, large language model; RL, reinforcement learning. (SFT: supervised fine tuning; GRPO: group relative policy optimization)

robust reasoning processes, despite resulting in empirical performance improvements. We shall come back to this point later.

Generating candidate derivational traces

An obvious way to obtain additional derivational data is to have humans create it. OpenAI paid contractors to write questions and step-by-step solutions to grade school math problems to create GSM8k.[9] While companies have continued to source data like this, it is infeasibly expensive, especially at the data scales necessary for large-scale model training.

A much more scalable approach is to use formal solvers to automatically generate both solutions and rationales derived from solver-specific intermediate representations. Searchformer[10] and Stream of Search[11] use standard search algorithms to produce datasets containing not just answers but also the execution traces generated along the way. For instance, when using A* search to solve a problem, SearchFormer's data generation pipeline will provide a representation of each manipulation of the open and closed lists as a derivational trace. Unfortunately, domain-specific solvers cannot be used to generate traces for arbitrary problems, limiting the generality of this technique.

Rather than creating high-quality traces from the start, an increasingly popular approach is to generate them from an LLM and filter afterwards. This sort of generation is feasible because modern LLMs are pretrained on data that already contains some derivational traces (e.g., educational web pages, grade school math explanations, and other sources that try to show their work)[b] and outputs that match these styles can be reliably induced, often by merely appending "Let's think step-by-step" to the prompt.[12]

Filtering traces

These traces are often not useful until they are filtered, with some discarded in favor of others. Researchers have varied in how they approach this trace selection process, ranging from selecting only those that are correct at each step (according to human labelers), training process reward models that attempt to automate human verification,[9] to selecting traces by formally verifying whether they lead to correct final solutions without considering the trace content.[13,14]

Improving LLMs using derivational traces

Once derivational traces have been selected, they can be used to further train an LLM. The hope is that, by outputting useful intermediate tokens, the LLM will be more likely to output correct solutions across a wider variety of problems. Early approaches fine-tuned LLMs directly on such traces,[10,11,13] but more recent advances have pivoted towards using reinforcement learning (RL) instead.

The first major successful and publicly understood models trained this way were DeepSeek's R1-Zero and R1 models.[14] After completing normal LLM pretraining, they begin an RL post-training phase on a new dataset—consisting of questions whose answers can be automatically verified. During this phase, the LLM generates multiple possible completions for each question; these completions take the form of traces culminating in separately marked final answers and are scored according to the correctness of that final answer. The best completions are then rewarded, adjusting the model to be more likely to output them rather than those completions that did not lead to a correct final answer. In essence, this RL process views the LLM as a token-choosing policy and uses a policy gradient algorithm to iteratively improve its parameters.[c]

Conceptually, this RL phase can be considered a two-step process repeated many times: first, generate potential trajectories from the LLM and weight them using an automatically computed success criterion; second, selectively fine-tune the same LLM on its own output. This reframing makes it clear that pure fine-tuning and RL approaches are not as different as might be initially assumed.

It is worth noting that whether SFT or RL is used to modify the parameters of the base LLM, the resulting model's architecture is still the same as that of any other LLM. The only difference is in the probability distribution the model captures: one that favors outputting

[b] There is also some speculation that the popularity of chain of thought prompting techniques has led to a greater availability of diverse step-by-step trace data in the massive web crawls that make up much of pretraining data.
[c] The "state" here is the context window; the next action is just the token emitted by the policy. See https://x.com/rao2z/status/1888699945232089262

Domain		Large Language Models					Large Reasoning Models		
		Claude-3.5 Sonnet	GPT-4o	GPT-4	LLaMA-3.1 405B	Gemini 1.5 Pro	o1-preview	o1-mini	Deepseek R1
Blocksworld	Zero shot	329/600 (54.8%)	213/600 (35.5%)	210/600 (34.6%)	376/600 (62.6%)	143/600 (23.8%)	**587/600 (97.8%)**	340/600 (56.6%)	582/600 (97%)
Mystery Blocksworld	Zero shot	0/600 (0%)	0/600 (0%)	1/600 (0.16%)	5/600 (0.8%)	-	**317/600 (52.8%)**	115/600 (19.1%)	256/600 (42.6%)
Avg. API Cost per 100 instances	-	$0.44	$0.65	$1.80	-	$0.33	**$42.12**	$3.69	$2.07

FIGURE 3 LLMs versus LRMs on PlanBench.

intermediate tokens (which mimic the derivational traces it was trained on) followed by the LLM's guess at the solution.

In fact, it has been empirically observed that fine-tuning a second LLM on verified outputs of an RL-trained model can close the performance gap, and in some cases may even result in an LLM that outperforms the original.[d] These distillation approaches have generated so much excitement that there are now public training datasets of filtered traces extracted from DeepSeek's R1,[e] sidestepping the need for an RL phase.

UNDERSTANDING REASONING MODELS

We have provided an overview of two broad but orthogonal approaches–post-training and test-time scaling—that have been largely responsible for the excitement about the planning and reasoning abilities of LRMs. Now, it is worth setting the record straight on some prevalent misconceptions.

How good are LRMs?

One of the reasons for the excitement surrounding LRMs is their noticeably higher reasoning performance compared to vanilla LLMs—as measured by popular benchmarks. Vendors, most notably OpenAI and DeepSeek, market their models by putting improvements on these metrics front and center in announcements, explanatory blog posts, and technical reports.

Our independent analysis on PlanBench,[15,16] a test set of simple planning problems that earlier models struggled on, has corroborated some of their results. While the new generation still suffers from generalization failures, it is a clear step up on these kinds of problems. Figure 3 provides some illustrative results. o1 and R1 saturate our static Blocksworld benchmark, and—arguably even more impressively—are the first models to make nontrivial progress on our Mystery Blocksworld domain. Problems in this domain are exact copies of Blocksworld problems, just with the names of objects and actions obfuscated.

While impressive when compared to older LLMs, the ability to solve some (rather than none!) of the most basic Mystery problems is still a long way away from the sort of robustness and generality real-world applications require. Furthermore, our extended results show that larger instances, even from the vanilla Blocks World, still trip up LRMs, and that they suffer from significant hallucination issues when prompted with unsolvable problems. In the case of unsolvable problems, the models not only confidently generate (impossible) plans but attempt to provide elaborate (and obviously false) justifications—running the risk of convincing (gas lighting?) naive users to trust them.

Due to these inaccuracy issues, we believe that these models are best viewed as *better generators* with a higher density of correct solution guesses than vanilla LLMs. Indeed, in our work analyzing o1,[16] we saw that having it play the same role as a standard LLM in the LLM-Modulo framework further improves accuracy while providing the guarantee that solutions are only output if they first pass formal verification.

Are LRMs cost-effective?

Compared to standard LLMs, LRMs involve significant additional costs both in training and inference phases, and vendors pass some of these costs on to the end user. Indeed, when OpenAI introduced o1, it started charging users for the intermediate tokens that o1 produced during inference time, despite the fact that the end user never got to see them!

While the post-training costs with synthetic data can be quite hefty, at least these are costs that are not directly seen by the end users. The situation is quite different with test-time inference costs. Regardless of the approach, all versions of test-time inference bring up the issue of ballooning costs. Whereas the time and money required for a vanilla LLM to generate a completion is entirely determined by the length of that completion, the cost per prompt for these systems can be arbitrarily high, and in the best case proportional to the computational complexity of the underlying reasoning problem. Test-time inference techniques thus throw a spanner into the current business model of

[d] See https://x.com/JJitsev/status/1886210147388711192
[e] See, e.g., Open Thoughts: https://github.com/open-thoughts/open-thoughts

LLM companies, as they can no longer front-load model costs (during training) and then sell completions at a pittance.

Where before, in comparisons between LLMs and alternatives, we could brush off training costs as essentially amortized over the lifetime of the model, now we must take this new source of increased cost into account. It is unclear to what extent this will come out in favor of LRMs over other, more specialized or domain-specific systems like classical solvers.

How different are LRMs from LLMs?

The term LLM, though originally a reference to the number of parameters and the size of the training corpus, has over time specialized in common use to refer to one particular kind of large model that processes language: enormous neural networks built on the transformer architecture and pretrained on massive text corpora before being instruction-tuned and user-aligned using a technique called reinforcement learning from human feedback. Given that LRMs are organized around the same underlying neural architecture and are trained using a superset of these techniques, this raises the question of why we separate them. Are the relevant training and architectural changes sufficient to label them a new class of models? Or can they legitimately be called LLMs?

It is clear that test-time inference techniques fundamentally alter the nature of the final system. A standard LLM, when prompted with a chunk of text, will autoregressively output a response to that text.[f] This is an essential part of the central conceit and utility of LLMs: as soon as the user presses enter, the answer begins to stream to them, token by token. Systems that use test-time inference techniques, especially those that explore multiple possibilities in parallel, do not retain this important ability and may only begin to output a final answer towards the very end of their processing—early tokens in the response may only be determined during later stages of execution.

What about LRMs (e.g., R1) that only differ from their base models because they were post-trained on derivational traces? We have noted that changing training procedures does not change the fundamental nature of the model at the inference stage—it will still take a prompt as input and output a completion. What it does change is what *kinds of completions* the model is more likely to output. R1 is illustrative as it is trained to respond to all queries using a very specific format: first, a sequence of intermediate "reasoning" tokens, then a reserved piece of text to mark the transition, and finally followed by a sequence of "answer" tokens. While this segmentation, when interpreted by us, is reminiscent of humans thinking before speaking, it does not carry any special execution-level semantics at inference time—every token is generated just like it would be in a standard LLM.

Unlike in the case of test-time scaling, there is no problem-dependent *adaptive computation* here. While these models do in practice produce *variable length* intermediate token sequences, we find the view that this is a novel test-time scaling paradigm rather strange—as this is a feature of all previous LLMs! Even the original release of ChatGPT responded with different numbers of tokens in response to different queries, and—with the right prompts—many of these length differences could easily be interpreted as first producing scratch work before a final answer.

We have argued before that LLMs are primarily defined by their pretraining data and should be analyzed as doing "approximate retrieval",[1] Especially in the case of models that differ only in their post-training, it is thus important to ask: Are LRMs sufficiently independent of the content effects of their pretraining data to move beyond approximate retrieval? Certainly, there is some evidence that performance improvements may stem from significant pretraining on the benchmark tasks—whether AIME (American Invitational Mathematics Examination) or Math Olympiad—and that this performance can be brittle when faced with simple prompt variations,[17] but the jury is still out.

Are LRMs reasoning or retrieving?

Most documented advances of LRMs on reasoning problems have been on tasks for which there are formal verifiers from traditional AI and computer science. The modus operandi of current LRMs is leveraging these verifiers in a *generate-test* loop at test time, training time, or distillation time in order to partially compile the verification signal into generation. In other words, post-training LRMs can be seen as iteratively compiling reasoning into retrieval. This iteration is needed because, for reasoning problems which can be arbitrarily scaled in complexity (e.g., multidigit multiplication with increasing digit numbers), an LLM trained on instances of a certain size quickly loses its ability to provide good guesses at larger sizes.[18] As we have seen, post-training approaches depend on the ability of the base LLM to have high enough top-k accuracy (i.e., be capable of generating at least one correct solution given k guesses) so that the verifier has something to select (otherwise, there is no signal either for fine-tuning or the RL phase!).

This general idea is consistent with the dictum (attributed to Marvin Minsky) that *intelligence is shifting the test part of the generate-test into the generate part*. In particular, using verifiers at test time has already been advocated by the LLM-Modulo framework,[8] As we have discussed, LRM post-training approaches crucially depend on the signal from the verifier to separate trajectories supplied by the base LLM into those that reach correct solutions versus those that do not (and thus, this can be seen as a form of "train time LLM-Modulo"). Once this is done, these traces are used to refine the base LLM ("generator") via either fine-tuning or RL. This part can thus be interpreted as partially compiling the verifier signal into the generator. Finally, while Deepseek R1 just deploys the refined LLM at the inference stage, without resorting to any test time verification, they do wind up using verifiers when they develop additional synthetic data with the help of R1 to distill other models.

[f] That is, a forward pass of the network will be performed and result in a single token being output. Then the model will be automatically and recursively reprompted with the resultant augmented chunk, with this process of forward pass and reprompt continuing in a loop until an end of string token.

One way of seeing this training-, test-, and distillation-time verification is as a staged approach to compile the verification signal into an underlying LLM. In particular, as we discussed, the base LLM used for R1 already has the capability of generating plausible solution trajectories (potentially from the derivational trace data that was already present in the pretraining data). Post-training can be seen as further refining it to come up with accurate solutions for longer/harder problems in fewer tries. Distillation can be seen as propagating this even further. At each stage, the verification signal is being compiled into the underlying LLM for longer and longer "inference horizons." This understanding is consistent with studies on the effectiveness of chain of thought,[18] use of internal versus external planning approaches for games,[19] as well as self-improvement in transformers.[20] In the last case, we would qualify any "self-improvement" claims by saying that it is more the case of incrementally compiling the verifier signal into the base LLM.

Are intermediate tokens actually traces of LLM reasoning?

As we discussed, post-training can induce a model to first generate long strings of intermediate tokens before outputting its final answer. There has been a tendency in the field to view these intermediate tokens as the human-like "thoughts" of the model or to see them as *reasoning traces* which recapitulate internal processes. This is by far the most contentious issue about LRMs.

The fact that intermediate token sequences often reasonably look like better-formatted and spelled human scratch work—mumbling everything from *"hmm…," "aha!," "wait a minute"* to "interesting." along the way—does not tell us much about whether they are used for anywhere near the same purposes that humans use them for, let alone about whether they can be used as an interpretable window into what the LLM is "thinking".[g] After all, LLMs already imitate the style of everything under the sun, so why would not they also imitate the style of any human musings present in the pretraining data? (An uncharitable observer might even say that we went from LLMs to LMMs—*large mumbling models*).

One reason that this anthropomorphization continues unabated because it is hard to either prove or disprove the correctness of these generated traces. DeepSeek's R1, even on very small and simple problems, will babble over 30 pages worth of text in response to each and every query, and it is far from clear how to check if these monologues constitute sound reasoning.[h] It is no wonder then that few if any LRM evaluations even try to check their pre-answer traces and focus only on evaluating the correctness of their final answers.

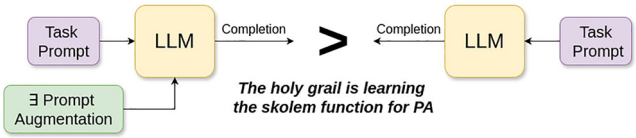

FIGURE 4 Augmenting a task prompt with additional tokens often seems to improve the accuracy of LLM completion even if the tokens do not have human-parseable meaning. LLM, large language model. (PA: prompt augmentation)

However, while evaluating the intermediate tokens produced by general LRMs may be out of direct reach, we *can* formally verify the status of traces generated by format-constrained models trained to imitate the derivational traces of domain-specific solvers. Our experiments[i] (run on the SearchFormer[10] and Stream of Search[11] models) show that, while these traces have the right *form*, a significant fraction of them are judged as invalid by the original generating algorithm[j]—even though these wrong traces may still stumble their way to the right answer! All of this makes one wonder if the correctness of training traces even matters in the first place.[21]

Yet, much research in this direction unquestioningly assumes trace correctness and legibility are necessary optimization targets. Even DeepSeek, despite dabbling in training an RL-only model (R1-Zero), released a final version (R1) that was trained with additional data and filtering steps specifically to reduce the model's default tendencies to produce illegible intermediate token sequences—mixing languages, formats, and so forth.

Given that these traces may not have any semantic import, deliberately making them *appear* more human-like is dangerous, potentially exploiting the cognitive flaws of users to convince them of the validity of incorrect answers. In the end, LRMs are supposed to provide solutions that users do not already know (and which they may not even be capable of directly verifying). Engendering false confidence and trust by generating stylistically plausible ersatz reasoning traces seems ill-advised!

Furthermore, human legibility may be counterproductive if the goal is to increase performance. RL can potentially train LLMs to output any old intermediate token sequences—all that matters is that the bottom line improves. Indeed, we believe[k] that de-anthropomorphization of intermediate tokens starts by acknowledging the common assumption across most "chain of thought" approaches: that an LLM will generate more accurate completions when provided with an appropriate *prompt augmentation* rather than just the base task prompt (see Figure 4). The big question then is how to get the right prompt augmentation.[l] Zero-shot and *k*-shot chain of thought prompting, as well as the variety

[g] Another confusion arising from "reasoning trace" terminology is that it conflates derivational traces—unfiltered intermediate tokens produced by the LLM—with post facto explanations or rationalizations of the process or the product of the said "thinking." For example, o1 often provides a sanitized summary/rationalization in lieu of intermediate tokens. It is well known that for humans at least, such post facto exercises rarely shed any meaningful light on the thinking that went in.

[h] Before DeepSeek, the entire question was moot. OpenAI's o1 model deliberately hides its intermediate tokens from end users, despite charging based on how many were produced!

[i] https://x.com/rao2z/status/1891260345165263128

[j] To illustrate: SearchFormer is trained on derivational traces produced using A*. During test time, the model's semantic failures include trying to remove nodes from the open list that are not actually in the open list, adding non-neighbor nodes to the open set, closing nodes that were already closed, or deriving a plan through the algorithm but outputting an unrelated final answer. Worse, these issues do not stem from generalization failures, as the test questions are identically distributed to the training set.

[k] see https://x.com/rao2z/status/1865985285034553478

[l] That is, given a task prompt T, $\exists PA s.t. Pr(Sol(LLM(T + PA), T)) > Pr(Sol(LLM(T), T))$, where *PA* is some appropriate prompt augmentation, *LLM*(*x*) is the completion output by LLM given *x* as the

of approaches for getting derivational traces for post-training, can all be seen as ways of answering this question. It is worth investigating additional approaches, including those where prompt augmentations are proposed and refined by a second, separate LLM. This LLM could use any token vocabulary without any restriction to anthropomorphic sounding mumbles! (Indeed, we can understand work on LLM adversarial attacks,[22] which generate universal token strings that push LLMs into giving undesirable responses, from this perspective!) In this framework, more powerful forms of RL such as those used in AlphaZero and MuZero can be employed to learn an intermediate token language focused only on improving solution accuracy. Ironically, this was our original speculation as to how o1 might be working![m]

SUMMARY

The past year has seen the rise of "LRMs," which seems to bring with them significant improvements in planning and reasoning problems that LLMs previously struggled with. We surveyed two broad classes of techniques driving the shift from the System 1 capabilities of LLMs to System 1+2 capabilities of LRMs and took a critical look at some of the popular anthropomorphic misconceptions about their success. Despite setting new accuracy records on benchmarks, these systems are still brittle to prompt variation, overconfident in their assertions, and do not generalize robustly. We argue that they are perhaps best suited to playing the role of more accurate generators within LLM-Modulo-like hybrid frameworks that provide external guarantees over their completions.

COMPETING INTERESTS

The authors declare no competing interests.

AUTHOR CONTRIBUTIONS

S.K. conceptualized and wrote the first draft of the article, which was then refined together with K.S. and K.V.

ACKNOWLEDGMENTS

Our research is supported in part by DARPA Grant HR00112520016, ONR grant N0001423-12409, and a gift from Qualcomm. We thank the entire Yochan group for spirited discussions, and Vardhan Palod and Atharva Gundawar for preliminary studies on reasoning traces.

ORCID

Subbarao Kambhampati https://orcid.org/0000-0002-9069-0265

REFERENCES

1. Kambhampati, S. (2024). Can large language models reason and plan? *Annals of the New York Academy of Sciences*, *1534*(1), 15–18.
2. Russell, S. J., & Norvig, P. (2010). Artificial intelligence: A modern approach. Pearson.
3. Wang, X., Wei, J., Schuurmans, D., Le, Q. V., Chi, E. H., Narang, S., Chowdhery, A., & Zhou, D. (2023). Self-consistency improves chain of thought reasoning in language models. [Paper presentation]. The 11th International Conference on Learning Representations, Kigali, Rwanda.
4. Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., & Narasimhan, K. R. (2023). Tree of thoughts: Deliberate problem solving with large language models. In Proceedings of the thirty-seventh conference on neural information processing systems (pp. 11809–11822). Curran Associates.
5. Stechly, K., Valmeekam, K., & Kambhampati, S. (2025). On the self-verification limitations of large language models on reasoning and planning tasks. In Proceedings of ICLR.
6. Arora, D., & Kambhampati, S. (2023). Learning and leveraging verifiers to improve planning capabilities of pre-trained language models [Paper presentation]. ICML Workshop on Knowledge and Logical Reasoning in the Era of Data-driven Learning, Honolulu, HI, USA.
7. Zhang, L., Hosseini, A., Bansal, H., Kazemi, M., Kumar, A., & Agarwal, R. (2024). Generative verifiers: Reward modeling as next-token prediction. arXiv:2408.15240.
8. Kambhampati, S., Valmeekam, K., Guan, L., Verma, M., Stechly, K., Bhambri, S., Saldyt, L. P., & Murthy, A. B. (2024). Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. Forty-First International Conference on Machine Learning, Vienna, Austria.
9. Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., & Cobbe, K. (2023). Let's verify step by step.arXiv:2305.20050.
10. Lehnert, L., Sukhbaatar, S., Su, D., Zheng, Q., Mcvay, P., Rabbat, M., & Tian, Y. (2024). *Beyond A*: Better planning with transformers via search dynamics bootstrapping*. First Conference on Language Models (COLM), Philadelphia, PA, USA.
11. Gandhi, K., Lee, D., Grand, G., Liu, M., Cheng, W., Sharma, A., & Goodman, N. D. (2024). *Stream of search (SoS): Learning to search in language* First Conference on Language Modeling (COLM), Philadelphia, PA, USA.
12. Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., & Iwasawa, Y. (2022). Large language models are zero-shot reasoners. *Advances in Neural Information Processing Systems*, *35*, 22199–22213.
13. Zelikman, E., Wu, Y., Mu, J., & Goodman, N. (2022). Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, *35*, 15476–15488.
14. DeepSeek-AI. (2025). DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. arXiv:2501.12948.
15. Valmeekam, K., Stechly, K., & Kambhampati, S. (2024). LLMs still can't plan: Can LRMs? A preliminary evaluation of OpenAI's o1 on PlanBench. arXiv:2409.13373.
16. Valmeekam, K., Stechly, K., Gundawar, A., & Kambhampati, S. (2024). Planning in strawberry fields: Evaluating and improving the planning and scheduling capabilities of LRM o1. arXiv:2410.02162, To appear in *Transactions on Machine Learning Research*..
17. McCoy, R. T., Yao, S., Friedman, D., Hardy, M. D., & Griffiths, T. L. (2024). When a language model is optimized for reasoning, does it still show embers of autoregression? An analysis of OpenAI o1. arXiv:2410.01792.
18. Stechly, K., Valmeekam, K., & Kambhampati, S. (2024). Chain of thoughtlessness: An analysis of CoT in planning . In Thirty-Eighth Annual Conference on Neural Information Processing Systems (NeurIPS 2024), Vancouver, Canada.
19. Schultz, J., Adamek, J., Jusup, M., Lanctot, M., Kaisers, M., Perrin, S., Hennes, D., Shar, J., Lewis, C., Ruoss, A., Zahavy, T., Veličković, P., Prince, L., Singh, S., Malmi, E., & Tomašev, N. (2024). Mastering board games by external and internal planning with language models. arXiv:2412.12119.

prompt, and *Sol*(*y*, *T*) checks, with the aid of a verifier, if *y* contains a solution for *T*. The holy grail then is learning the skolem function for prompt augmentation.

[m] See https://x.com/rao2z/status/1834354533931385203

20. Lee, N., Cai, Z., Schwarzschild, A., Lee, K., & Papailiopoulos, D. (2025). Self-improving transformers overcome easy-to-hard and lengthy generalization challenges. arXiv:2502.01612.
21. Li, D., Cao, S., Griggs, T., Liu, S., Mo, X., Patil, S. G., Zaharia, M., Gonzalez, J. E., & Stoica, I. (2025). LLMs can easily learn to reason from demonstrations structure, not content, is what matters!. arXiv:2502.07374.
22. Zou, A., Wang, Z., Carlini, N., Nasr, M., Kolter, J. Z., & Fredrikson, M. (2023). Universal and transferable adversarial attacks on aligned language models. arXiv:2307.15043.

How to cite this article: Kambhampati, S., Stechly, K., & Valmeekam, K. (2025). (How) do reasoning models reason? *Ann NY Acad Sci.*, 1–8. https://doi.org/10.1111/nyas.15339