Exploring Extrinsic and Intrinsic Properties for Effective Reasoning with Code Interpreter

Anonymous authors
Paper under double-blind review

Abstract

Reasoning with a Code Interpreter (CI) has emerged as an effective paradigm for enhancing the reasoning capabilities of large language models (LLMs) through executable computation and iterative verification. Despite its growing adoption, the behavioral properties underlying effective code reasoning remain largely underexplored. In this work, we investigate code reasoning from two distinct perspectives inspired by prior studies of natural language reasoning: extrinsic properties, represented by *crucial tokens*, and intrinsic properties, represented by *code-specific cognitive behaviors*. Across multiple LLMs, we find that stronger CI reasoning models consistently exhibit a higher prevalence of crucial tokens and cognitive behaviors, particularly verification, backtracking, and backward chaining. Building on these observations, we examine how these properties can be leveraged during both inference and training. At inference time, appending code-specific *crucial tokens* improves performance on several reasoning capabilities, including mathematical, ordering, and optimization, while yielding limited benefits elsewhere. At training time, augmenting a state-of-the-art framework with code-specific cognitive behaviors improves supervised fine-tuning and reinforcement learning performance in two of three evaluated models. Further analysis shows that these behaviors reduce overthinking in incorrect responses and improve token efficiency, while also revealing factors that limit gains in a certain model. Our findings provide the first systematic characterization of effective reasoning with CI and demonstrate both the potential and limitations of leveraging key properties to improve CI-based reasoning.

1 Introduction

Reasoning with a Code Interpreter (CI) (Chen et al., 2023; Gao et al., 2023) has emerged as a promising paradigm to enhance the problem-solving capabilities of Large Language Models (LLMs). By delegating computation to an external execution environment while focusing on high-level reasoning (Gou et al., 2024; Chen et al., 2026), CI-based reasoning mitigates common failures of natural language (NL) reasoning (Wei et al., 2022; Wang et al., 2023), including arithmetic errors and brittle multi-step recursion. Prior work showed that reasoning with CI improves downstream performance, with gains of up to 13.5% on grade-school mathematics benchmarks (Cobbe et al., 2021). These advances underscore the importance of external tools for improving LLM reasoning and planning (Yao et al., 2023; Wang et al., 2024b). They are further reflected in coding agents, where models must execute, test, and iteratively debug code within feedback loops (Yang et al., 2024; Zhang et al., 2024). In this setting, CI-based reasoning grounds model outputs from external evidence, reducing hallucinations and improving downstream performance (Bouzenia et al., 2025).

Concurrently, a growing body of research has sought to understand the mechanisms underlying NL reasoning. We organize this paper into two distinct perspectives: extrinsic and intrinsic. Extrinsic approaches examine how inference-time interventions, such as *crucial token* (Kim et al., 2026; Qian et al., 2025) injection, influence reasoning behavior. For example, Muennighoff et al. (2025) showed that appending a *wait* token during inference encourages additional reasoning computation, improving performance in certain settings (Wu, 2025; Zhao et al., 2026). For the intrinsic perspective, we define intrinsic properties as cognitive behaviors (CoBe) (Gandhi et al., 2025; Shen et al., 2025; Wei et al., 2026), such as planning, verification, and backward chaining, that emerge during training and correlate with higher task performance.

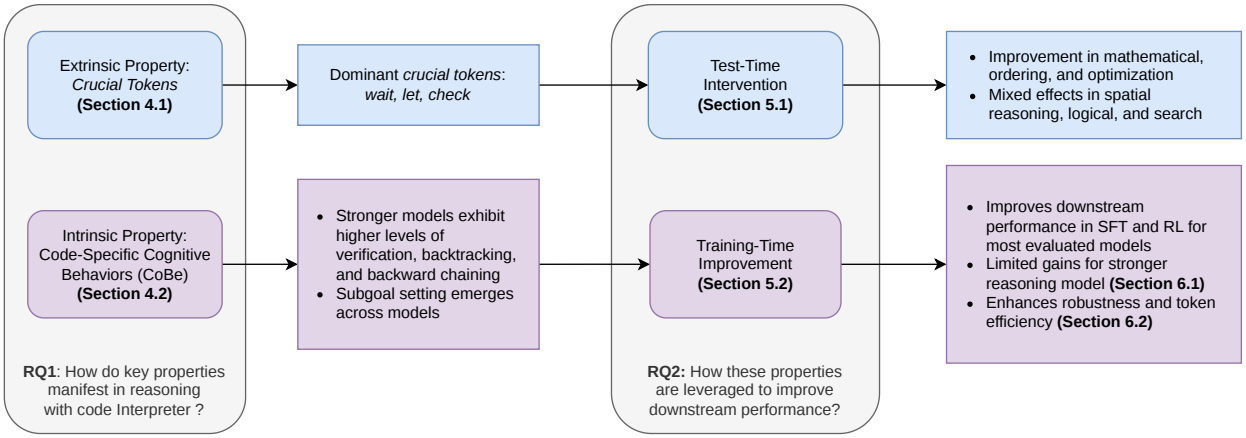


Figure 1: Overview of how extrinsic and intrinsic properties characterize and improve reasoning with CI.

Despite these extensive studies on NL reasoning, the key properties underlying effective CI-based reasoning remain underexplored. Our central question is: *How do crucial tokens and CoBe manifest and leveraged in CI-based reasoning?* As shown in Figure 1, this central question is decomposed into two research questions:

- **RQ1:** How do *crucial tokens* (extrinsic) and CoBe (intrinsic cognitive behaviors) manifest in effective reasoning with the CI?
- **RQ2:** How can *crucial tokens* and CoBe be leveraged, at test time and training time, to improve downstream performance?

To answer RQ1, we investigate CI-based reasoning across both properties. First, we examine **extrinsic properties** by identifying frequently occurring tokens across multiple LLMs, termed *crucial tokens* (Section 4.1). We find that the strongest model consistently exhibits higher frequencies of these tokens, notably *wait*, *let*, and *check*. Second, we analyze **intrinsic properties** by extending the cognitive behavior from (Gandhi et al., 2025) to CI-based reasoning (Section 4.2). We observe that a high-performing model exhibits more code-specific behaviors, particularly checking intermediate results, revising unsuccessful attempts, and working backward from the target outcome. In contrast, the behavior of decomposing a problem into smaller, manageable subgoals was consistently observed across all models, regardless of their performance level.

Building on these findings, we next investigate RQ2: how the identified properties can be leveraged to improve downstream performance at inference and training time. At inference, we evaluate whether appending *crucial tokens* improves performance on reasoning and planning tasks (Section 5.1). We observe gains in mathematical, ordering, and optimization tasks, but mixed results on spatial, logical, and search reasoning, consistent with prior work (Wu, 2025; Zhao et al., 2026) showing that test-time scaling is task-dependent. At training time, we examine whether inducing code-specific cognitive behaviors improves CI-based reasoning (Section 5.2). Incorporating CoBe into the recent CI-based training framework of Feng et al. (2026) improves accuracy in both supervised fine-tuning and reinforcement learning stages in two of three models across both training stages. In particular, CoBe yields gains of 6.3% during supervised fine-tuning and 10.7% during reinforcement learning on Qwen2.5-7B-Instruct. In contrast, for Qwen3-8B, the reduction of *crucial token* frequency leads to lower performance relative to standard training, indicating that extrinsic properties also contribute to effective CI-based reasoning (Section 6.1). Further analyses in Section 6.2 indicate that CoBe mitigates overthinking and enhances token efficiency throughout the reinforcement learning process.

Our main contributions are summarized as follows:

- We conduct a systematic study of effective reasoning with CI, identifying *crucial tokens* and *cognitive behaviors* and characterizing how these properties manifest in CI-based reasoning.
- We show that these properties can be leveraged at both test time and training time, improving downstream performance through test-time scaling and training-time augmentation.
- We analyze the benefits and limitations of these properties across models, revealing factors that improve efficiency and factors that limit their performance gains in a certain model.

2 Related Works

2.1 Reasoning with Code-Interpreter

Chen et al. (2023) and Gao et al. (2023) introduced tool use through Code Interpreters, in which reasoning is conducted via program generation. By producing executable code and offloading computation to external environments, this paradigm allows LLMs to solve problems that require complex reasoning and precise calculations. By grounding intermediate reasoning in execution, these approaches improve reliability on mathematical and algorithmic tasks (Wang et al., 2024a; Gou et al., 2024; Li et al., 2025c). Subsequent work extended this paradigm to broader tool use. Yao et al. (2023) interleaved reasoning and tool invocation for grounded problem solving, while Schick et al. (2023); Jin et al. (2025); Li et al. (2025b); Wei et al. (2025); Zheng et al. (2025) enabled models to determine by themselves when and how to use external tools. Related studies have further explored tool-augmented agents operating in external environments, including coding agents that iteratively generate, execute, test, and refine code (Wang et al., 2024b; Yang et al., 2024; Zhang et al., 2024; Bouzenia et al., 2025; Li et al., 2025a). More recently, reinforcement learning with verifiable rewards (RLVR) (Guo et al., 2025; Olmo et al., 2026) has been applied to Code Interpreter reasoning. Frameworks such as Feng et al. (2026) and Chen et al. (2026) showed that outcome-based rewards improve tool interaction and mathematical reasoning through iterative execution and verification. While prior work primarily focuses on improving downstream performance, our work investigates the properties underlying effective Code Interpreter reasoning and how they can be leveraged at both inference and training time.

2.2 Key Properties in Natural Language Reasoning

To systematically investigate the properties of reasoning in CI-based systems, building on prior studies of natural language reasoning, we define the extrinsic property as interactions with *crucial tokens* (Kim et al., 2026) and the intrinsic property as the dynamics of *cognitive behaviors* (Gandhi et al., 2025).

Extrinsic Properties. Early work on exploiting the extrinsic properties of extended reasoning in LLMs explored inference-time interventions that encourage models to continue reasoning after reaching a completion signal, such as repeatedly appending a *wait* token (Muennighoff et al., 2025). Although this simple strategy shows performance improvements on mathematical benchmarks, subsequent studies questioned its robustness and generalization. Wu (2025) showed that gains from repeated *wait* injections are highly inconsistent, with models often oscillating between answers or producing nearly identical reasoning traces despite additional computation. Extending the analysis to knowledge-intensive tasks, Zhao et al. (2026) found no consistent benefit from budget forcing and reported increased hallucination rates for stronger reasoning models, while Ghosal et al. (2025) observed a non-monotonic scaling effect in which excessive deliberation can degrade performance through overthinking. More recent work argues that test-time scaling gains depend less on reasoning length and more on the informational content of reasoning trajectories. The 80:20 hypothesis suggests that only a small fraction of reasoning tokens contribute disproportionately to performance improvements (Wang et al., 2025), while Qian et al. (2025) identifies "critical tokens" that coincide with key reasoning transitions and carry outsized influence on final predictions. Similarly, Kim et al. (2026) argues that improvements stem from epistemic verbalization, the explicit expression of uncertainty, doubt, and self-correction, rather than from specific trigger tokens or longer chains of thought alone. However, these studies focus predominantly on natural language reasoning, leaving the extrinsic properties of reasoning with external tools, particularly code interpreters, largely unexplored.

Intrinsic Properties. A growing line of research examines the intrinsic cognitive behaviors and emergent exploration tendencies of LLMs that dictate their capacity for reinforcement learning (RL) and test-time scaling. An influential study in this area, Gandhi et al. (2025) empirically demonstrated that foundational cognitive habits, such as verification, subgoal setting, backtracking, and backward chaining, serve as important priors for RL-driven self-improvement, while models exhibiting these behaviors scale effectively under RL, behavior-deficient models quickly plateau unless primed with behavior-rich trajectories. Building on this premise, recent works focus on how to systematically instill, condense, or manipulate these internal mental mechanics. Sprague et al. (2026) introduces a self-distillation framework to isolate and transmit these pure

structural reasoning behaviors to smaller base models to prevent early RL stagnation, while Aytes et al. (2025) draws from human cognitive psychology to compress verbose reasoning traces into efficient, dense internal intermediaries via adaptive sketching. To maximize the utility of these habits during inference, Zhang et al. (2025b) identifies specialized internal attention heads dedicated to non-linear reasoning steps and introduces test-time steering mechanisms to balance overthinking and underthinking. Furthermore, models like Satori (Shen et al., 2025) manifest these intrinsic behaviors explicitly by framing verification and backtracking as autoregressive "meta-actions" within a unified chain-of-action-thought. Extending this paradigm beyond pure text, Sarch et al. (2025); Wei et al. (2026) demonstrate that these linguistic cognitive habits structurally generalize to visual reasoning through iterative grounding and verification under RLVR. Collectively, these studies showed that a model's intrinsic cognitive architecture governs its learning efficiency. However, these dynamics remain underexplored in tool-integrated environments like reasoning with CI, where external compiler feedback introduces distinct, deterministic feedback loops.

3 Background

In this section, we discuss reasoning in LLMs, covering NL reasoning and reasoning with CI interactions, including multi-turn settings (Section 3.1). We then examine how test-time scaling can be integrated into the CI-based reasoning framework (Section 3.2). Finally, Section 3.3 presents the training pipeline, including supervised fine-tuning for cold-start initialization and reinforcement learning for LLMs' reasoning with CI.

3.1 Reasoning with Code Interpreter

Natural language reasoning considers an input problem x and a generated response $\hat{y}$, where the response is sampled from a model policy π_θ parameterized by θ, i.e., $\hat{y} \sim \pi_\theta(\cdot \mid x)$. The generated response typically consists of a reasoning trajectory (rationale) $\hat{r}$ and a final prediction $\hat{a}$, where the answer is commonly formatted within a predefined pattern, such as `\boxed`{*answer*}.

Unlike traditional natural language reasoning, where a model has to tackle both high-level problem-solving and actual computation, reasoning with CI lets the model generate code, commonly Python, to leverage the computational capability from CI, letting the model focus solely on problem-solving. Conventionally, for the first turn, the model has to generate both reasoning trajectory $\hat{y}$ and code $\hat{c}$ from a given problem, represented as $(\hat{y}_0, \hat{c}_0) \sim \pi_\theta(\cdot|x)$. After that, the code will be parsed by CI, resulting in an output $\hat{o}_0 \sim \text{CI}(\hat{c}_0)$. The model then takes code execution output and generate the next turn $(\hat{y}_1, \hat{c}_1) \sim \pi_\theta(\cdot|\hat{o}_0, \hat{y}_0, \hat{c}_0, x)$, typically called multi-turn interaction with CI until reaching the final turn (T), generally represented as

$$(\hat{y}_T, \hat{c}_T) \sim \pi_\theta(\cdot|\hat{\mathbf{o}}_{t<T}\hat{\mathbf{y}}_{t<T}, \hat{\mathbf{c}}_{t<T}, x) \tag{1}$$

Similar to typical reasoning where $\hat{y}_T$ contains rationale $\hat{r}$ and final answer $\hat{a}$. A representative trace of this structure is illustrated in Figure 7.

3.2 Test-Time Scaling

Test-time scaling is an emerging paradigm that improves downstream performance by allocating additional computational resources during inference (Muennighoff et al., 2025; Wang et al., 2023). Existing approaches can be broadly categorized into parallel and sequential scaling (Zhang et al., 2025a). Parallel scaling generates multiple independent reasoning trajectories and aggregates or selects among them, whereas sequential scaling devotes computation to extending a single trajectory, enabling iterative refinement and error correction. To study how models interact with extrinsic factors, we focus on sequential test-time scaling in the context of CI-based reasoning and investigate its scaling behavior across different capabilities and model variants.

To investigate these dynamics, we manipulate the test-time scaling budget by overriding the model's self-termination signal. Typically, reasoning models signify the completion of a reasoning by generating a specific token, such as </THINK>. We intervene by replacing this token with a *crucial token* (w), such as *wait*, which compels the model to extend its reasoning. Formally, given an initial reasoning $\hat{r}_0 \sim \pi_\theta(\cdot|x)$, we iteratively inject N such tokens to produce an extended trajectory:

$$\hat{r}_N \sim \pi_\theta(\cdot \mid w, \hat{r}_{N-1}, \ldots, w, \hat{r}_0, x) \tag{2}$$

The final code, $\hat{c}$, is then sampled conditioned on the full extended reasoning:

$$\hat{c} \sim \pi_\theta(\cdot \mid \hat{r}_N, w, \hat{r}_{N-1}, \ldots, w, \hat{r}_0, x) \tag{3}$$

This methodology allows us to systematically analyze the scaling behavior of model accuracy as a function of the total tokens allocated to the reasoning phase, forcing the model to further explore the problem space or verify its logic before committing to a computational solution.

3.3 Supervised Fine-Tuning and Reinforcement Learning

Following prior work that studied CI-based reasoning with LLMs (Feng et al., 2026; Chen et al., 2026), the training process is divided into two main phases. The first phase uses supervised fine-tuning (SFT) as a cold start to help the model understand the basic structure of multi-turn CI tasks. In the second phase, reinforcement learning (RL) is used to further optimize the model through multi-turn interaction with the CI. During this stage, the model learns by writing code and interacting with a real CI environment over several turns. This enables the model to delegate compute-intensive operations, such as arithmetic and recursive computations, to the CI while focusing on higher-level reasoning before producing a final solution.

Supervised Fine-Tuning. In cold-start SFT for multi-turn CI interactions, the model is typically trained on a curated dataset denoted as $\mathcal{D}^{\text{SFT}} = \{(x_i, \mathbf{y}_i, \mathbf{c}_i, \mathbf{o}_i)\}_{i=1}^{N}$, where N represents the total number of training samples. Each individual sample X_i is a tuple consisting of a natural language instruction x_i, a gold-standard response $\mathbf{y}_i$, the corresponding code snippets $\mathbf{c}_i$, and the resulting execution outputs $\mathbf{o}_i$. To ensure the model focuses its learning capacity on generative tasks, specifically, the text responses and executable code. The training objective utilizes a masked loss function. For a given training sample i, the SFT loss over a sequence of J tokens is formulated as:

$$\mathcal{L}_i^{\text{SFT}} = -\frac{1}{\sum_j \delta_{i,j}} \sum_{j=1}^{J} \delta_{i,j} \log P_\theta(X_{i,j} | X_{i,<j}) \tag{4}$$

In this objective, a delta function $\delta_{i,j}$ serves as a binary mask, where $\delta_{i,j} = 1$ if the token $X_{i,j}$ belongs to the set of target outputs $\{\mathbf{y}_i, \mathbf{c}_i\}$, and $\delta_{i,j} = 0$ otherwise. This approach effectively isolates the gradient updates to the model's primary generative tasks.

Reinforcement Learning. Following the ReTool framework (Feng et al., 2026), the RL pipeline employs the Proximal Policy Optimization (PPO) paradigm (Schulman et al., 2017). In this specific implementation, Group Relative Policy Optimization (GRPO) (Shao et al., 2024) is utilized to refine the policy. The RL training dataset, denoted as $\mathcal{D}^{\text{RL}}$, comprises pairs of problem (x) and their corresponding ground-truth answers (a). The optimization objective for the policy update is defined as:

$$\mathcal{J}^{\text{GRPO}}(\theta) = \frac{1}{G} \sum_{g=1}^{G} \frac{1}{|\hat{y}_g|} \sum_{j=1}^{|\hat{y}_g|} \left\{ \min \left[\frac{\pi_\theta(\hat{y}_{g,j} | x, \hat{y}_{g,<j}; \text{CI})}{\pi_{\theta_{\text{old}}}(\hat{y}_{g,j} | x, \hat{y}_{g,<j}; \text{CI})} \hat{A}_{g,j}, \text{clip}\left(\frac{\pi_\theta(\hat{y}_{g,j} | x, \hat{y}_{g,<j}; \text{CI})}{\pi_{\theta_{\text{old}}}(\hat{y}_{g,j} | x, \hat{y}_{g,<j}; \text{CI})}, 1-\varepsilon, 1+\varepsilon \right) \hat{A}_{g,j} \right] \right\} \tag{5}$$

In this formulation, the objective maximizes the lower bound of the clipped surrogate advantage. The probability ratio between the current policy π_θ and the behavior policy $\pi_{\theta_{\text{old}}}$ is constrained by the hyperparameter ε to ensure stable updates, while the advantage estimator $\hat{A}_{g,j}$ guides the policy toward sequences that yield higher relative rewards within the CI environment.

In alignment with the ReTool framework, the reward signal is formulated as a binary outcome contingent on the accuracy of the final derived answer. This sparse reward mechanism incentivizes the model to converge on the correct solution via valid reasoning and code execution. Formally, the reward function $R(a, \hat{a})$ is defined such that $R(a, \hat{a}) = 1$ if the predicted final answer $\hat{a}$ matches the ground-truth label a, and -1 otherwise. Within the GRPO framework, the advantage $\hat{A}_{g,j}$ for a given output sequence is determined by evaluating its reward relative to the mean performance of a group of G sampled outputs for the same query. Specifically, for each trajectory o_g within a sampled group $\{o_1, o_2, \ldots, o_G\}$, the advantage is computed as the z-score of the group's reward distribution $R(x, \hat{y}_g)$. This relative advantage estimation utilizes the group's collective performance as a dynamic baseline, eliminating the need for a separate value function (critic) and thereby streamlining the RL process for CI-based tasks.

4 Key Properties of Effective Reasoning with Code Interpreter (RQ1)

In this section, we decompose RQ1 into two orthogonal lines of investigation. First, we investigate the extrinsic properties of effective reasoning with CI by analyzing the frequency of specific *crucial tokens* generated by strong models compared to weaker baselines in code-generation settings (Section 4.1). Second, we explore the intrinsic properties by quantifying cognitive behaviors specific to reasoning with CI (Section 4.2), offering empirical insights into the models' reasoning behavioral patterns.

4.1 Extrinsic Property

Motivation. Test-time scaling via token injection has been shown to improve NL reasoning in certain settings (Muennighoff et al., 2025), although its effectiveness varies across models (Wu, 2025; Zhao et al., 2026). At the same time, code interpreters have demonstrated strong benefits for complex reasoning tasks (Chen et al., 2023; Feng et al., 2026), yet their surface-level reasoning patterns remain largely unexplored. We therefore investigate *crucial tokens* in CI-based reasoning and evaluate their utility for test-time scaling.

Setup. Similar to Kim et al. (2026), we evaluate three distinct model families: Qwen3-8B (Yang et al., 2025) as a representative reasoning model, Qwen2.5-7B-Instruct (Qwen et al., 2025) as a general-purpose baseline, and include Llama3.1-8B-Instruct (Grattafiori et al., 2024) as a standard instruction-tuned model. Experiments are conducted on SymBench (Chen et al., 2025), a comprehensive benchmark designed for symbolic reasoning. This dataset encompasses a diverse set of tasks, including mathematics, spatial reasoning, logic, sequential ordering, optimization, and search-based problems. Building on the concept of epistemic tokens[1] (Kim et al., 2026) originally identified in NL reasoning, we empirically investigate their surface patterns for reasoning with CI. Furthermore, we include *let*, an auxiliary definition in programming language (Landin, 1966). Collectively, we refer to these as *crucial tokens*. To quantify their prevalence, we measure the frequency of each *crucial token* normalized by the total number of generated tokens.

Result and Discussion. Regarding model performance, a high-capacity model such as Qwen3 achieves an accuracy of 66.5%, whereas Qwen2.5 and Llama3.1 attain 36.2% and 10.3%, respectively. Focusing on Qwen3-8B, which is the best-performing model, Figure 2 shows that the four most frequent *crucial tokens* are *so, let, check, and wait*, in this order, which is consistent with prior studies Wang et al. (2025); Qian et al. (2025); Kim et al. (2026). Notably, *let* appears as the second most dominant token for reasoning with CI. In contrast, the frequency of *crucial tokens* is substantially lower in Qwen2.5 and Llama3.1. Nevertheless, *let* and *check* remain the two most frequent tokens, suggesting that they may play a significant role in inference-time reasoning with CI. Building on these observations, Section 5.1 evaluates the utility of *crucial tokens* for inference-time scaling. We focus on *let* and *check* due to their prevalence across top-performing models, and *wait* for comparing with prior works, excluding *so* because it is largely absent from the second-best model.

4.2 Intrinsic Property

Motivation. Prior work demonstrates that high performing models exhibit key cognitive behaviors, including verification, backtracking, subgoal setting, and backward chaining, which are largely absent in weaker models (Gandhi et al., 2025). These behaviors enable models to validate intermediate results, revise unsuccessful reasoning paths, decompose complex problems into manageable subproblems, and reason backward from desired outcomes. Motivated by these findings, we investigate whether such behaviors also emerge in CI-based reasoning and the extent to which they contribute to solving challenging tasks.

Setup. Following Section 4.1, we employ the same set of models and benchmark for our empirical analysis. To investigate internal reasoning processes, we adopt an LLM-as-a-Judge framework, extending the approach of Gandhi et al. (2025). However, preliminary experiments reveal that the original prompts, designed to evaluate cognitive behavior in natural language reasoning, do not consistently yield reliable outcomes in code-oriented reasoning contexts. We therefore refine the probing prompt to better suit scenarios involving

[1] *wait, hmm, perhaps, maybe, alternatively, guess, correct, check, so, now, actually, likely, might,* and *seems.*

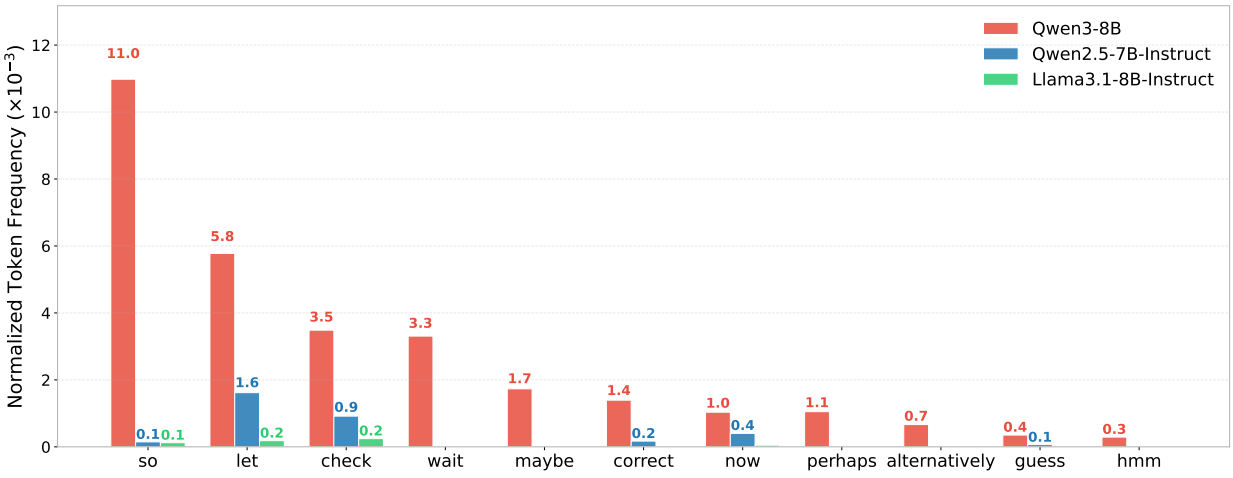


Figure 2: Distribution of *crucial token* frequencies across various models on SymBench. We present the ten most frequent epistemic tokens identified by Kim et al. (2026), along with the inclusion of the *let* token.

reasoning with a code interpreter (see Appendix B for further details). For the Oracle LLM, we use Qwen3-Next-80B-A3B-Instruct (Yang et al., 2025), which has demonstrated strong performance on mathematical problem-solving and coding tasks.

Result and Discussion. Figure 3 shows that Qwen3 exhibits the highest frequency across all cognitive behavior types. In particular, verification, subgoal setting, and backward chaining appear in at least 96% of its responses. Backtracking is less frequent in Qwen3 (44%), though still markedly higher than in Qwen2.5 and Llama3.1 (9% and 7%, respectively). This pattern contrasts with typical natural language reasoning, where backtracking is most prevalent. It is the least frequent behavior in reasoning with CI. In contrast, subgoal setting dominates in our settings, where planning is implicitly encouraged, thereby increasing its prevalence. The overall trend is consistent with Gandhi et al. (2025). These findings suggest that such cognitive behaviors are also foundational for solving challenging problems in reasoning with CI, as reflected in downstream performance. Building on this insight, we incorporate these behaviors into an existing state-of-the-art training framework, designed for reasoning with CI, as detailed in Section 5.2.

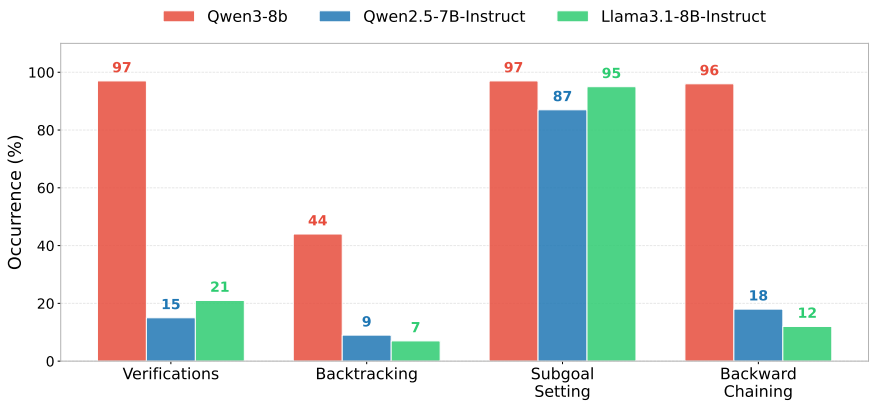


Figure 3: Analysis of reasoning behaviors across base models. The figure shows the percentage appearance of reasoning responses that exhibit key cognitive behaviors across comparable model sizes.

5 Leveraging Key Properties To Enhance Downstream Performance (RQ2)

Building upon the findings in Section 4, we empirically examine the potential benefits and limitations of leveraging these properties in downstream applications. For test-time scaling, Section 5.1 investigates the impact of three selected *crucial tokens* identified in Section 4.1. We assess their scalability and effectiveness across six capabilities on challenging tasks using a diverse range of models. For training-time enhancement, inspired by the observations in Section 4.2 that cognitive behaviors are important for code reasoning, Section 5.2 integrates four cognitive reasoning behaviors into the training process. We further show that integrating these behaviors leads to further gains during both supervised fine-tuning and reinforcement learning.

5.1 Test-Time Scaling with Crucial Tokens.

Motivation. Sequential test-time scaling for NL reasoning has been extensively studied across diverse settings (Muennighoff et al., 2025; Wu, 2025; Zhao et al., 2026). However, its effectiveness and limitations in CI-based reasoning remain under-investigated. To address this, we conduct an empirical study of scaling behavior under *crucial token* intervention and analyze performance variations across models.

Setup. For test-time scaling in reasoning with CI, we follow the implementation introduced in Section 3.2. Based on the analysis in Section 4.1, we select the tokens *wait*, *let*, and *check* as the *crucial tokens* for study in this section. We adopt SymBench as the evaluation benchmark to analyze downstream scaling behavior across six reasoning capabilities. Our primary model is Qwen3-8B, and we additionally include its smaller counterpart, Qwen3-4B, to investigate the effect of model scale. To further examine the variance across model families, we incorporate DeepSeek-R1 (Guo et al., 2025) distilled models of comparable size, namely DeepSeek-R1-Distill-Qwen-7B and DeepSeek-R1-Distill-Llama-8B.

Result and Discussion. Figure 4 shows that test-time scaling via the forced appending of *crucial tokens* generally leads to substantial performance improvements over the non-extrapolated baseline across all evaluated reasoning domains. The scaling effect is particularly pronounced and consistent in mathematical reasoning, order reasoning, and optimization tasks, where the *let* token demonstrates near-monotonic gains and significantly outperforms alternative tokens as the number of appended tokens increases. By contrast, although some tasks benefit from test-time scaling, their performance trajectories exhibit greater volatility and non-monotonic behavior, especially under the *wait* and *check* strategies. For example, in the search and logical capabilities, the *wait* token produces sharp performance drops at intermediate token lengths before partially recovering, indicating sensitivity to the extrapolated sequence length. Despite these task-dependent variations, the overall results suggest that the *let* token provides the most reliable outcome for leveraging additional inference-time computation, yielding consistent improvements across most capabilities. These findings are partially consistent with Muennighoff et al. (2025), which reported that selecting an appropriate *crucial token* under specific configurations can systematically enhance model performance, as observed for the *let* token in mathematical reasoning. Furthermore, our results indicate that, in certain capabilities such as spatial reasoning, logical reasoning, and search, simply appending *crucial tokens* does not necessarily guarantee superior performance. This observation aligns with prior studies (Wu, 2025; Zhao et al., 2026), which similarly highlight the challenges when adopting test-time scaling in more complex reasoning settings.

Following the ablation setup in Muennighoff et al. (2025), we inject individual *crucial tokens* to evaluate their effects across different reasoning model families and token variants. As shown in Table 1, the effectiveness of *crucial token* extrapolation varies substantially across model families and scales. For Qwen3-8B, the *let* and *check* tokens generally improve downstream performance across multiple reasoning domains, with *let* achieving the best gains in mathematics and optimization, while *check* performs best in spatial, logical, and search tasks. In contrast, the *wait* token consistently underperforms the baseline. For the smaller Qwen3-4B model, *crucial token* intervention yields limited gains, with degradation on most tasks and only marginal improvement in optimization, consistent with Ghosal et al. (2025), where sub-7B models struggle with test-time scaling on challenging problems. The DeepSeek distilled models exhibit stronger scaling behavior. DeepSeek-R1-Distill-Llama-8B shows improvements primarily in mathematics, spatial, and logical reasoning, whereas DeepSeek-R1-Distill-Qwen-7B demonstrates consistent gains across all evaluated competencies, with substantial improvements in mathematics, spatial reasoning, and optimization under all token variants.

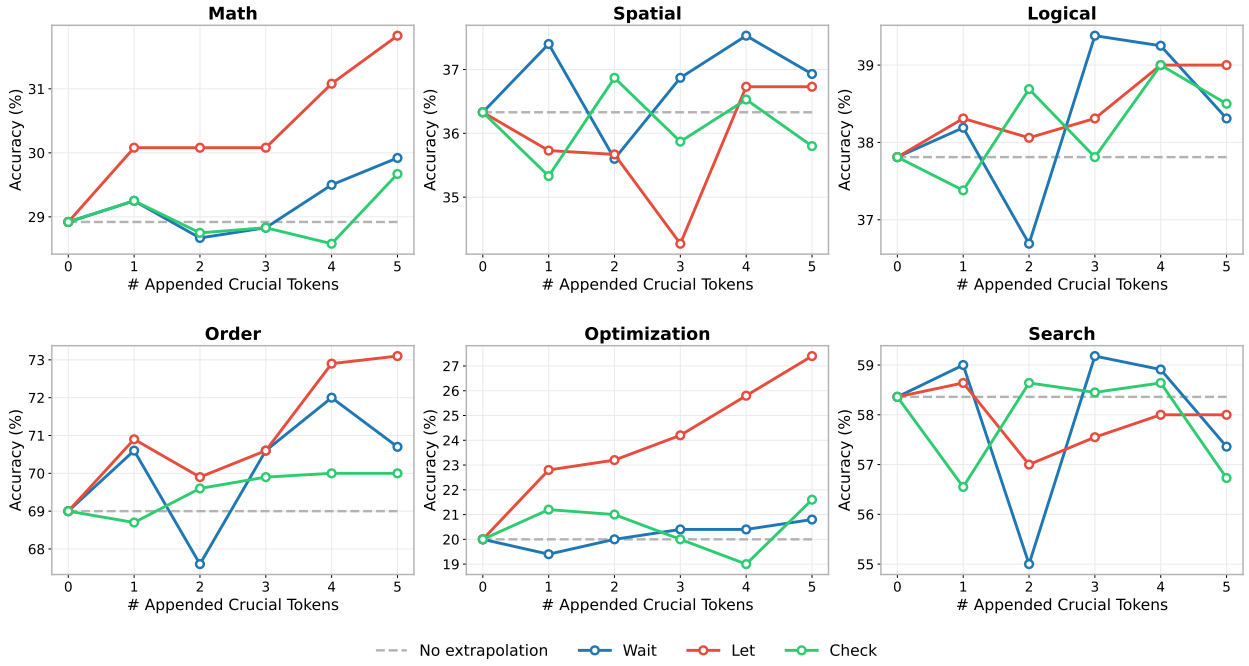


Figure 4: Accuracy progression of Qwen3-8B under test-time scaling, achieved by iteratively appending the *crucial tokens*. Each marker denotes an iteration where the individual token is injected, corresponding to the generated token count shown on the x-axis.

Table 1: Test-time scaling performance under different *crucial token* injection strategies, evaluated reasoning models, Qwen3 in two sizes, and two Deepseek's distilled reasoning models.

Method	Math	Spatial	Logical	Order	Optimization	Search
Qwen3-8B						
No extrapolation	28.92	36.33	37.81	69.00	20.00	58.36
2x *wait*	28.67	35.60	36.69	67.70	20.00	55.00
2x *let*	**30.08**	35.67	38.06	**69.90**	**23.20**	57.00
2x *check*	28.75	**36.87**	**38.69**	**69.90**	21.00	**58.64**
Qwen3-4B						
No extrapolation	**32.92**	**37.33**	**35.56**	**68.40**	15.60	**55.64**
2x *wait*	28.92	35.67	33.19	65.30	10.60	53.36
2x *let*	30.00	34.33	33.75	64.00	**16.60**	52.00
2x *check*	29.17	33.80	33.88	64.90	16.20	51.64
DeepSeek-R1-Distill-Llama-8B						
No extrapolation	34.83	23.23	31.88	**37.50**	11.40	**33.73**
2x *wait*	**35.25**	22.47	**34.94**	35.50	11.60	33.36
2x *let*	33.92	21.33	34.19	36.30	**13.00**	31.36
2x *check*	35.17	**24.73**	32.19	37.00	11.40	32.91
DeepSeek-R1-Distill-Qwen-7B						
No extrapolation	18.58	19.07	23.81	36.00	3.60	27.64
2x *wait*	23.25	23.13	25.19	**42.60**	9.00	**31.18**
2x *let*	**24.58**	**23.67**	**25.31**	42.30	8.60	30.45
2x *check*	23.92	23.47	24.38	40.80	**9.60**	29.36

Overall, these results suggest that the effectiveness of *crucial token* intervention depends strongly on model architecture and scale. The most consistent gains are observed in mathematics and optimization, where

Qwen3-8B and both DeepSeek distilled models benefit from at least one token variant, whereas improvements in spatial, logical, and search reasoning are more model-dependent.

5.2 Incorporating Cognitive Behaviors at Training Time

Motivation. Inspired by Gandhi et al. (2025), prior work has shown that cognitive behaviors such as verification, backtracking, subgoal setting, and backward chaining can substantially improve the reasoning capabilities of weaker models. In addition, our empirical findings in Section 4.2 suggest that these behaviors constitute important intrinsic properties for effective reasoning. Motivated by these observations, we investigate whether equipping models with code-specific cognitive behaviors (CoBe) can further enhance downstream performance on reasoning tasks involving multi-turn interactions with code interpreter feedback.

Setup. To evaluate the effectiveness of CoBe, we adopt the training framework described in Section 3.3, which consists of SFT for cold-start initialization in a CI environment, followed by multi-turn reinforcement learning. SFT To incorporate CoBe in the SFT pipeline, we develop a data synthesis pipeline extending the method of Gandhi et al. (2025). Given an existing reasoning trajectory $\hat{\mathbf{y}}$ and a cognitive behavior prompt p_{CoBe}, we generate an augmented trajectory $\mathbf{y}_{\text{CoBe}} \sim \text{LLM}(\mathbf{x}, \mathbf{y}; p_{\text{CoBe}}, \text{CI})$. Following Gandhi et al. (2025), we employ Claude Sonnet 4.5 as the oracle language model for trajectory synthesis, motivated by its near-perfect performance on AIME 2025.[2] For consistency with ReTool, we primarily evaluate the Qwen2.5 family, including Qwen2.5-7B-Instruct-Coder and its instruction-tuned variants, and additionally consider Qwen3-8B, a reasoning model. We evaluate on AIME24, AIME25, and MATH500 (Hendrycks et al., 2021). Additional details are provided in Appendix C.

Table 2: Main results on mathematical problems.

Method	AIME24	AIME25	MATH500	Average
Qwen2.5-Coder-7B-Instruct	7.1	2.5	58.6	22.7
+ ReTool-SFT	15.8	12.1	**73.9**	33.9
+ ReTool-SFT w/ CoBe	**23.3**	**13.8**	73.2	**36.8**
+ ReTool-RL	40.0	29.6	84.4	51.6
+ ReTool-RL w/ CoBe	**42.1**	**31.7**	**84.5**	**52.8**
Qwen2.5-7B-Instruct	12.1	3.3	69.0	28.2
+ ReTool-SFT	13.3	10.4	75.7	33.1
+ ReTool-SFT w/ CoBe	**17.1**	**14.2**	**74.5**	**35.2**
+ ReTool-RL	32.5	24.6	77.1	44.7
+ ReTool-RL w/ CoBe	**40.0**	**26.7**	**81.9**	**49.5**
Qwen3-8B	**43.8**	**37.9**	73.7	51.8
+ ReTool-SFT	40.8	27.3	**88.2**	**52.1**
+ ReTool-SFT w/ CoBe	32.5	21.3	81.5	45.1
+ ReTool-RL	**67.5**	**59.2**	**95.4**	**74.0**
+ ReTool-RL w/ CoBe	58.8	46.3	91.9	65.6

Result and Discussion. As shown in Table 2, ReTool improves mathematical reasoning performance on both Qwen2.5-7B-Instruct and Qwen2.5-Coder-7B-Instruct across the SFT and RL stages. Building on this foundation, incorporating CoBe yields additional gains in two of the three models, suggesting that CoBe can effectively complement the ReTool training framework. In contrast, Qwen3-8B does not benefit from CoBe and instead exhibits performance degradation relative to the corresponding ReTool baseline. This discrepancy reveals that the effectiveness of CoBe may depend on model-specific characteristics. We investigate this phenomenon further in Section 6.1. Examining the training stages separately, augmenting ReTool with CoBe during SFT improves the average performance of Qwen2.5-Coder-7B-Instruct from 33.9 to 36.8 and Qwen2.5-7B-Instruct from 33.1 to 35.2. The gains become more pronounced during RL, where ReTool-RL with CoBe further increases the average score from 51.6 to 52.8 on Qwen2.5-Coder-7B-Instruct and from 44.7 to 49.5 on Qwen2.5-7B-Instruct. These results suggest that cognitive behaviors can

[2] https://www-cdn.anthropic.com/963373e433e489a87a10c823c52a0a013e9172dd.pdf

be effectively leveraged at training time through CoBe augmentation, providing additional supervision that improves downstream mathematical reasoning, particularly for Qwen2.5-based models. Further analysis of the observed improvements beyond aggregate performance is provided in Section 6.2.

6 Analysis

6.1 Investigating the Lack of Performance Improvements in Qwen3-8B

Although Qwen3-8B was explicitly primed with the desired intrinsic property, we verify that CoBe-trained models consistently exhibit the intended behaviors during downstream evaluation (see Appendix F). We observe no downstream performance improvement. We hypothesize that this phenomenon may be confounded by two factors: (1) discrepancies in response-length distributions introduced during SFT, and (2) the insufficient preservation of the extrinsic property. Regarding response length, the left panel of Figure 5 shows that outputs produced by both ReTool-SFT and ReTool-SFT with CoBe are substantially shorter than the corresponding zero-shot generations. While response length itself may not directly determine downstream accuracy, such a substantial distributional shift may negatively affect model behavior. This observation is consistent with the degraded AIME performance of both SFT variants reported in Table 2.

We further assess the relationship between performance and the extrinsic property through an analysis of *crucial token* frequency. As illustrated in the right panel of Figure 5, ReTool-RL exhibits the higher length-normalized frequency of *crucial tokens* on both AIME24 and AIME25, aligning with its superior performance. By comparison, ReTool-RL with CoBe yields lower *crucial token* frequencies and correspondingly weaker results. Overall, this finding highlights that maintaining both intrinsic and extrinsic properties is also important for preserving strong reasoning performance.

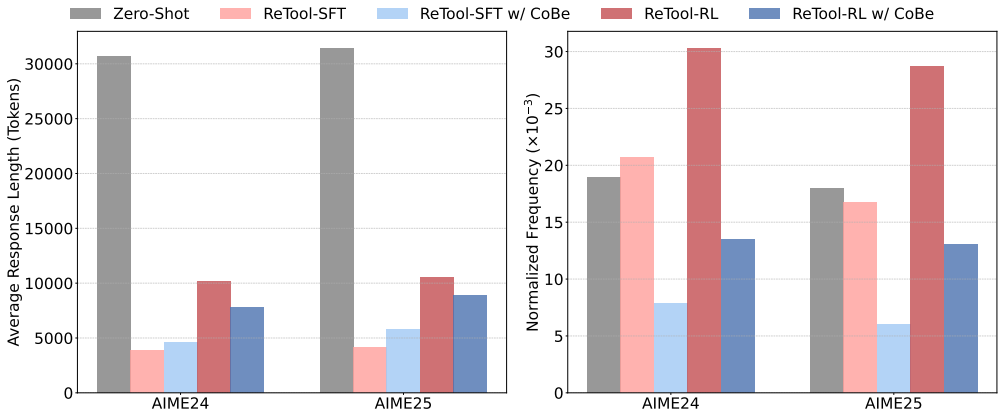


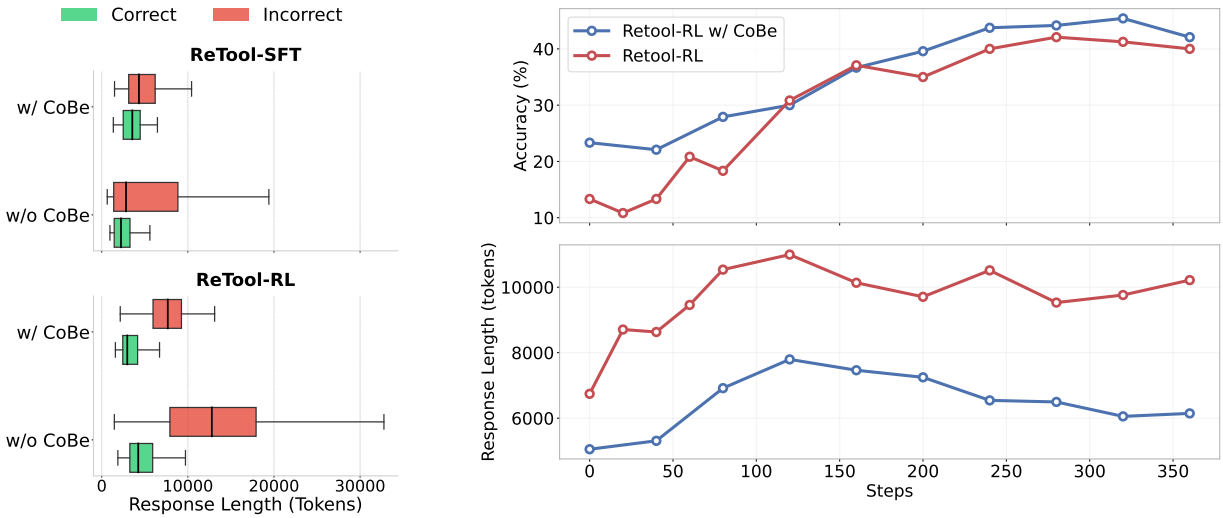
Figure 5: Comparative analysis of Qwen3-8B outputs across mathematical benchmarks. (Left) Comparison of average response length between the zero-shot, standard ReTool, and ReTool integrated with CoBe. (Right) Length-normalized frequency of total *crucial tokens*, referenced from Section 4.1.

To further disentangle these factors, we conduct an additional empirical study using Qwen3-8B-Base instead of the post-trained Qwen3-8B model. We apply both ReTool-SFT and ReTool-SFT with CoBe to the base model in order to isolate the effects. As demonstrated in Appendix E, incorporating CoBe into the base model setting improves the ReTool-SFT baseline by approximately 7% on average. In particular, performance increases by 15% on AIME24 and 35.1% on AIME25, while performance on MATH500 decreases by only 1.1%. These results suggest that CoBe can be beneficial when applied to the base model, but its effectiveness may vary for models that have already undergone extensive post-training. Further investigation is therefore required to understand how cognitive-behavior augmentation interacts with existing model capabilities.

6.2 The Impact of Cognitive Behaviors on Reasoning With CI Beyond Task Performance

Prior work (Wang et al., 2026) has observed that incorrect reasoning trajectories are often associated with excessively long generations, suggesting that LMs may overthink when faced with difficult problems. Motivated by this insight, we investigate whether incorporating CoBe into the training pipeline can encourage more structured reasoning and thereby mitigate inefficient generation behaviors. Figure 6 shows that integrating CoBe consistently reduces response lengths for incorrect samples across both the SFT and RL stages. This trend indicates that CoBe enables the model to reason more efficiently, particularly in cases where the model ultimately fails to reach the correct solution.

We further compare the training dynamics of ReTool-RL with and without CoBe under identical training settings. The CoBe-enhanced variant outperforms the standard ReTool-RL baseline in 8 out of 10 checkpoints on AIME24, while consistently generating shorter responses. In contrast, standard ReTool-RL exhibits a bias towards longer responses with limited accuracy improvements, reflecting the length bias identified by Liu et al. (2025). These results suggest that extending length alone is insufficient for reasoning with a code interpreter; instead, the quality and structure of intermediate reasoning behaviors are also critical.



(a) Response lengths.

(b) Evolution of accuracy and average response lengths.

Figure 6: Response length distributions and training dynamics of Qwen2.5-7B-Coder-Instruct on AIME24. CoBe reduces overthinking on incorrect predictions, enhancing token efficiency and accuracy.

7 Conclusion

Although reasoning with code interpreter is increasingly adopted, the underlying extrinsic *crucial tokens* and intrinsic cognitive behaviors that facilitate effective reasoning remain largely underexplored. To address this, we conduct a systematic study of how these properties manifest across multiple LLMs, focusing on *crucial tokens* and cognitive behaviors such as verification, backtracking, subgoal setting, and backward chaining. Our results show that both properties can be leveraged to improve downstream performance in certain conditions. In particular, *crucial token* intervention, especially with *let*, enables effective test-time scaling, while incorporating code-specific cognitive behaviors into training improves performance during both supervised fine-tuning and reinforcement learning in most cases. Beyond accuracy gains, these behaviors also improve token efficiency and reduce overthinking in incorrect responses, highlighting the importance of intrinsic behavior rather than simply extending generation length. At the same time, the performance degradation observed in a certain specialized model suggests that the effectiveness of behavioral augmentation depends on its interaction with existing model capabilities, motivating further investigation into factors that constrain gains in advanced reasoning models.

Limitations and Potential Risks

Although this work provides a systematic characterization of extrinsic and intrinsic properties in code-interpreter-based reasoning, several limitations should be acknowledged. First, both the evaluation process and data synthesis pipeline rely on high-capacity Oracle LLMs, which may introduce model-specific biases and variability in generated outputs. Moreover, our interventions investigate these properties independently: training primarily targets intrinsic behaviors (CoBe), whereas test-time scaling focuses on *crucial tokens* associated with extrinsic properties. As demonstrated in our analysis of Qwen3-8B, reductions in *crucial token* frequency can adversely affect task performance even when intrinsic reasoning behaviors are maintained. Consequently, this study does not provide a unified framework that jointly leverages both intrinsic and extrinsic properties. Second, the observed findings may not readily generalize to broader tool-augmented agent settings, such as web browsing, multi-tool orchestration, or open-ended API interactions. In addition, our conclusions are derived from a specific set of publicly available datasets and evaluation frameworks, which may limit their applicability to other domains and tasks. Finally, this study involves the execution of LLM-generated code as part of the evaluation process. While all executions are performed within controlled sandbox environments and are unlikely to produce harmful effects, the possibility of unintended or unanticipated behaviors arising from model-generated code cannot be entirely eliminated.

References

Simon A. Aytes, Jinheon Baek, and Sung Ju Hwang. Sketch-of-thought: Efficient LLM reasoning with adaptive cognitive-inspired sketching. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 24296–24320, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1236. URL https://aclanthology.org/2025.emnlp-main.1236/.

Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. Repairagent: An autonomous, llm-based agent for program repair. In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering*, ICSE '25, pp. 2188–2200. IEEE Press, 2025. ISBN 9798331505691. doi: 10.1109/ICSE55347.2025.00157. URL https://doi.org/10.1109/ICSE55347.2025.00157.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=YfZ4ZPt8zd.

Yongchao Chen, Yilun Hao, Yueying Liu, Yang Zhang, and Chuchu Fan. Codesteer: Symbolic-augmented language models via code/text guidance. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=ezna4V4zHs.

Yongchao Chen, Yueying Liu, Junwei Zhou, Yilun Hao, Jingquan Wang, Yang Zhang, Na Li, and Chuchu Fan. R1-code-interpreter: LLMs reason with code via supervised and multi-stage reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=FNlNHOiFOx.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=tRk1nofSmz.

Kanishk Gandhi, Ayush K Chakravarthy, Anikait Singh, Nathan Lile, and Noah Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective STars. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=QGJ9ttXLTy.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models, 2023. URL https://arxiv.org/abs/2211.10435.

Soumya Suvra Ghosal, Souradip Chakraborty, Avinash Reddy, Yifu Lu, Mengdi Wang, Dinesh Manocha, Furong Huang, Mohammad Ghavamzadeh, and Amrit Singh Bedi. Does thinking more always help? mirage of test-time scaling in reasoning models, 2025. URL https://arxiv.org/abs/2506.04210.

Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. ToRA: A tool-integrated reasoning agent for mathematical problem solving. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Ep0TtjVoap.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing

Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL `https://arxiv.org/abs/2407.21783`.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong

Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Rwhi91ideu.

Jeonghye Kim, Xufang Luo, Minbeom Kim, Sangmook Lee, Dongsheng Li, and Yuqing Yang. Understanding reasoning in llms through strategic information allocation under uncertainty, 2026. URL https://arxiv.org/abs/2603.15500.

P. J. Landin. The next 700 programming languages. *Commun. ACM*, 9(3):157–166, March 1966. ISSN 0001-0782. doi: 10.1145/365230.365257. URL https://doi.org/10.1145/365230.365257.

Dacheng Li, Shiyi Cao, Chengkun Cao, Xiuyu Li, Shangyin Tan, Kurt Keutzer, Jiarong Xing, Joseph E. Gonzalez, and Ion Stoica. S*: Test time scaling for code generation. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 15964–15978, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.865. URL https://aclanthology.org/2025.findings-emnlp.865/.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 5420–5438, Suzhou, China, November 2025b. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.276. URL https://aclanthology.org/2025.emnlp-main.276/.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. Torl: Scaling tool-integrated rl, 2025c. URL https://arxiv.org/abs/2503.23383.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=5PAF7PAY2Y.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candes, and Tatsunori Hashimoto. s1: Simple test-time scaling. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321,

Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1025. URL https://aclanthology.org/2025.emnlp-main.1025/.

Team Olmo, :, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3, 2026. URL https://arxiv.org/abs/2512.13961.

Chen Qian, Dongrui Liu, Haochen Wen, Zhen Bai, Yong Liu, and Jing Shao. Demystifying reasoning dynamics with mutual information: Thinking tokens are information peaks in LLM reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=E1FrjgaG1J.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Gabriel Sarch, Snigdha Saha, Naitik Khandelwal, Ayush Jain, Michael J. Tarr, Aviral Kumar, and Katerina Fragkiadaki. Grounded reinforcement learning for visual reasoning, 2025. URL https://arxiv.org/abs/2505.23678.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023. URL https://arxiv.org/abs/2302.04761.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y.K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Maohao Shen, Guangtao Zeng, Zhenting Qi, Zhang-Wei Hong, Zhenfang Chen, Wei Lu, Gregory W. Wornell, Subhro Das, David Daniel Cox, and Chuang Gan. Satori: Reinforcement learning with chain-of-action-thought enhances LLM reasoning via autoregressive search. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=j4FXxMiDjL.

Zayne Rea Sprague, Jack Lu, Manya Wadhwa, Sedrick Keh, Mengye Ren, and Greg Durrett. Skillfactory: Self-distillation for learning cognitive behaviors. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ttMLNXBWKY.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. Mathcoder: Seamless code integration in LLMs for enhanced mathematical reasoning. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=z8TW0ttBPp.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xiong-Hui Chen, Jianxin Yang, Zhenru Zhang, Yuqiong Liu, An Yang, Andrew Zhao, Yang Yue, Shiji Song, Bowen Yu, Gao Huang,

and Junyang Lin. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for LLM reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=yfcpdY4gMP.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024b.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Yue Wang, Qiuzhi Liu, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Linfeng Song, Dian Yu, Juntao Li, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Thoughts are all over the place: On the underthinking of long reasoning models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=WcUo7Z2Jnh.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

Yana Wei, Liang Zhao, Jianjian Sun, Kangheng Lin, jisheng yin, Jingcheng Hu, Yinmin Zhang, En Yu, Haoran Lv, Zejia Weng, Jia Wang, Qi Han, Zheng Ge, Xiangyu Zhang, Daxin Jiang, and Vishal M. Patel. Open vision reasoner: Transferring linguistic cognitive behavior for visual reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=oEgybA04dY.

Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, Hyokun Yun, and Lihong Li. WebAgent-r1: Training web agents via end-to-end multi-turn reinforcement learning. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 7909–7928, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.401. URL https://aclanthology.org/2025.emnlp-main.401/.

Guojun Wu. It's not that simple. an analysis of simple test-time scaling, 2025. URL https://arxiv.org/abs/2507.14419.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: agent-computer interfaces enable automated software engineering. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Wenyue Hua, Haolun Wu, Zhihan Guo, Yufei Wang, Niklas Muennighoff, Irwin King, Xue Liu, and Chen Ma. A survey on test-time scaling in large language models: What, how, where, and how well?, 2025a. URL https://arxiv.org/abs/2503.24235.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2024, pp. 1592–1604, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706127. doi: 10.1145/3650212.3680384. URL https://doi.org/10.1145/3650212.3680384.

Zhenyu Zhang, Xiaoxia Wu, Zhongzhu Zhou, Qingyang Wu, Yineng Zhang, Pragaash Ponnusamy, Harikaran Subbaraj, Jue WANG, Shuaiwen Leon Song, and Ben Athiwaratkun. Understanding and steering the cognitive behaviors of reasoning models at test-time. In *NeurIPS 2025 Workshop on Efficient Reasoning*, 2025b. URL https://openreview.net/forum?id=yKAEasJpdr.

James Xu Zhao, Bryan Hooi, and See-Kiong Ng. Test-time scaling in reasoning models is not effective for knowledge-intensive tasks yet, 2026. URL https://arxiv.org/abs/2509.06861.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. DeepResearcher: Scaling deep research via reinforcement learning in real-world environments. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 414–431, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.22. URL https://aclanthology.org/2025.emnlp-main.22/.

A Sample of SFT dataset

Figure 7 illustrates the surface example of the SFT dataset from ReTool Feng et al. (2026), which consists of user prompts, assistant responses with Python scripts, and code interpreter results.

B Probing Code-Specific Cognitive Behaviors in Reasoning with CI

In preliminary experiments, we found that the cognitive-behavior prompts proposed by Gandhi et al. (2025) do not readily transfer to the CI-based reasoning setting. For instance, as illustrated in Figure 8, the original framework fails to identify any of the four cognitive behaviors in a representative CI-based reasoning trajectory. To better capture code-specific cognitive behaviors, we adapt the prompting framework for the CI-based reasoning regime. Consistent with prior work, each behavior is evaluated independently. Under the revised framework, the same trajectory is identified as exhibiting both subgoal setting and backtracking. The prompts used for backtracking and backward chaining are presented in Figure 9, while those for subgoal setting and verification are provided in Figure 10.

C Synthesize Code-Specific Cognitive Behaviors

To augment ReTool with CoBe, we follow the data synthesis procedure of Gandhi et al. (2025). Specifically, we construct a new dataset, $\mathcal{D}_{\text{CoBe}} = (\mathbf{x}, \mathbf{y}_{\text{CoBe}})$, from the original ReTool SFT dataset[3] $\mathcal{D} = (\mathbf{x}, \mathbf{y})$ by prompting Claude Sonnet 4.5 with both the original trajectory y and a cognitive-behavior prompt p_{CoBe} (Figure 11). The synthesized trajectory is generated as $\mathbf{y}_{\text{CoBe}} \sim \text{LLM}(\mathbf{x}, \mathbf{y}; p_{\text{CoBe}}, \text{CI})$. To ensure correctness, we retain only synthesized trajectories whose final answers match the corresponding gold labels, yielding 1,915 samples from the original 2,000 instances. We further compare the prevalence of cognitive behaviors between the original ReTool-SFT trajectories and the CoBe-augmented trajectories in Figure 12a. The results indicate substantial increases in the targeted behaviors, with verification, subgoal decomposition, and backward chaining appearing in all synthesized trajectories, while backtracking increases by three percentage points. The distribution of response lengths is shown in Figure 12b.

[3] https://huggingface.co/datasets/JoeYing/ReTool-SFT

User
There are 152 students at Dala High School. Assume the following:
- 100 students take a Math class
- 94 take a Science class
- 57 take an English class.
- 73 take Math and Science
- 24 take Math and English
- 27 take Science and English.
- 22 take Math, Science, and English.

How many students take neither a Math class nor a Science class nor an English class?
Remember to place the final answer in the format: `\boxed{'The final answer goes here.'}`

Assistant
Okay, I need to find the number of students taking none of the classes. The total is 152. I will use the principle of inclusion-exclusion (PIE) for three sets: $|M \cup S \cup E| = |M| + |S| + |E| - (|M \cap S| + |M \cap E| + |S \cap E|) + |M \cap S \cap E|$. Then, subtract that from the overall total.

```
M, S, E = 100, 94, 57
M_S, M_E, S_E = 73, 24, 27
M_S_E = 22
total_in_any = M + S + E - M_S - M_E - S_E + M_S_E
print(152 - total_in_any)
```

`<interpreter>3</interpreter>`

The code returns 3. To verify, I will break down each disjoint category (e.g., "Only Math") to ensure the total population sum is consistent.

```
only_M_S, only_M_E, only_S_E = 73-22, 24-22, 27-22
only_M = 100 - only_M_S - only_M_E - 22
only_S = 94 - only_M_S - only_S_E - 22
only_E = 57 - only_M_E - only_S_E - 22
total_verify = only_M + only_S + only_E + only_M_S + only_M_E + only_S_E + 22
print(total_verify)
```

`<interpreter>149</interpreter>`

The verification total is 149, so students not taking any classes are 152 - 149 = 3, confirming the initial result. `\boxed{3}`

Figure 7: Illustrative multi-turn reasoning trace featuring interleaved code execution. This sample is sourced from the ReTool cold-start supervised fine-tuning data (Feng et al., 2026).

User
Xiao Wang is a librarian responsible for shelving 6 books labeled A, B, C, D, E, F on a shelf with positions 1 to 6.

The constraints are:
1. Book D must be to the left of book F.
2. Book D must be adjacent to book A.
3. Book F must be to the left of book C.
4. Book C cannot be placed in position 1.
5. Book B cannot be placed in position 1.
6. Book C must be to the right of book B.
7. Book A must be to the left of book C.

Question:
Based on the above conditions, please arrange the 6 books from left to right on the bookshelf, i.e. positions 1 through 6. Return your answer with the format «< a list of strings »>, one solution is enough.for example:
«<["A", "B", "C"]»>
Your answer: `\boxed{'The final answer goes here.'}`

Assistant
To solve this problem, we need to find a valid arrangement of the books A, B, C, D, E, and F that satisfies all the given constraints. Let's break down the constraints step by step:

1. Book D must be to the left of book F.
2. Book D must be adjacent to book A.
3. Book F must be to the left of book C.
4. Book C cannot be placed in position 1.
5. Book B cannot be placed in position 1.
6. Book C must be to the right of book B.
7. Book A must be to the left of book C.

Given these constraints, we can use a brute-force approach to try all possible permutations of the books and check which one satisfies all the conditions. Let's implement this in Python.

```python
from itertools import permutations

# Define the books
books = ['A', 'B', 'C', 'D', 'E', 'F']

# Function to check if a permutation satisfies all constraints
def is_valid(arrangement):
    # Constraint 1: Book D must be to the left of book F
    if arrangement.index('D') > arrangement.index('F'):
        return False
    # Constraint 2: Book D must be adjacent to book A
    if abs(arrangement.index('D') - arrangement.index('A')) != 1:
        return False
    # Constraint 3: Book F must be to the left of book C
    if arrangement.index('F') > arrangement.index('C'):
        return False
    # Constraint 4: Book C cannot be placed in position 1
    if arrangement[0] == 'C':
        return False
    # Constraint 5: Book B cannot be placed in position 1
    if arrangement[0] == 'B':
        return False
    # Constraint 6: Book C must be to the right of book B
    if arrangement.index('C') < arrangement.index('B'):
        return False
    # Constraint 7: Book A must be to the left of book C
    if arrangement.index('A') > arrangement.index('C'):
        return False
    return True

# Generate all permutations and find a valid one
for arrangement in permutations(books):
    if is_valid(arrangement):
        print("<<<{}>>>".format(arrangement))
        break
```

Figure 8: Partial response generated by Qwen2.5-7B-Instruct on `permutations-and-combinations` task.

Backtracking

Task Description
You will be provided with code reasoning or explanation text. Your task is to detect whether the text includes algorithmic backtracking behavior — where the writer explicitly describes attempting one path or strategy in code, realizing it fails, and then reversing or undoing that choice to try another.

Focus on algorithmic backtracking, not surface-level rewordings or corrections.

Backtracking behavior may include:
- Describing recursive or trial-based algorithms that explore options and abandon invalid ones
- Reverting a previous decision in code (e.g., "remove", "undo", or "pop" a value)
- Trying all options in a loop or recursion, and skipping ones that violate constraints
- Returning early or continuing to next candidate when current state is invalid
- Mentioning backtracking explicitly as part of the approach

If you find any backtracking steps, count them and provide the count between the tags '<count>' '</count>'. If there is no backtracking behavior, provide a count of 0 as '<count>0</count>'.

Task Format
Format your response in markdown as follows:

Thoughts
[Brief description of what algorithmic backtracking behavior was noticed and where]

Does backtrack?
[yes/no]

Number of backtrack steps
<count> [number] </count>

Task to evaluate for backtracking
{response}

Response

Backward Chaining

Task Description
You will be provided with code reasoning or explanation text. Your task is to detect whether the text includes backward chaining behavior — where the writer starts from a desired goal, final output, or condition, and reasons *backward* to determine what inputs, conditions, or steps would achieve it.

This behavior is common in constraint-solving, logic programming, test-driven design, and debugging.

Backward chaining in code reasoning may include:
- Starting from the required output and asking what input would produce it
- Working backward from test results, outputs, or constraints to figure out the logic
- Describing a condition (e.g. "to return True. . . ") and identifying how to meet it
- Setting up reverse planning steps (e.g. "to reach this final state, we need. . . ")
- Using final state or assertions as anchors for inferring code paths

If you find any such backward-chaining steps, count them and provide the count between the tags '<count>' '</count>'. If there are no backward-chaining behaviors, provide a count of 0 as '<count>0</count>'.

Task Format
Format your response in markdown as follows:

Thoughts
[Brief description of what backward chaining behavior was noticed and where]

Does backward chaining?
[yes/no]

Number of backward chaining instances
<count> [1/2/...] </count>

Task to evaluate for backward chaining
{response}

Response

Figure 9: Prompt for analyzing backtracking and backward chaining.

Subgoal Setting

Task Description
You will be provided with an explanation or reasoning related to code problem-solving. Your task is to detect whether the writer engages in explicit subgoal setting — where they break down a complex coding task into smaller, intermediate objectives.

Subgoal setting in code reasoning may include:
- Listing or sequencing intermediate steps to reach the final solution
- Splitting a large function or algorithm into subtasks or helper routines
- Phrases indicating ordered steps (e.g., "first we. . . ", "then we. . . ", "finally. . . ")
- Planning ahead by structuring work into stages such as parsing input, preprocessing data, then computing output
- Describing recursive decomposition, e.g., "we solve the base case, then handle the recursive step"

If you find any subgoal-setting steps, count them and provide the count between the tags '<count>' '</count>'. If the text does not contain any subgoal setting, provide a count of 0 as '<count>0</count>'.

Task Format
Format your response in markdown as follows:

Thoughts
[Brief description of what subgoal-setting behavior was noticed and where]

Does subgoal setting?
[yes/no]

Number of subgoal setting steps
<count> [number] </count>

Task to evaluate for Subgoal Setting
{response}

Response

Verification

Task Description
You will be provided with a piece of code and an explanation or reasoning about how that code solves a problem. Your task is to identify whether the explanation contains any verification steps that check or validate the code's correctness. Since the code cannot actually be run here, look for logical checks, sample input walkthroughs, and sanity checks in the reasoning.

Verification steps may include:
- Running the code mentally with sample inputs to see if outputs match expectations.
- Comparing the code's output or behavior to the problem requirements or expected results.
- Checking edge cases (for example, empty input, very large values, negative values) in the reasoning.
- Inspecting function signatures, parameter types, or return types to ensure they match the specifications.
- Confirming that loops terminate or that recursion depth stays within safe limits given the constraints.
- Analyzing time or space complexity to ensure it is acceptable for the input size.
- Ensuring that all conditional branches and special cases in the code are considered.
- Verifying loop invariants or assumptions made in the code logic.

If you find any verification steps, count them and provide the count between the tags <count> </count>. If the text does not contain any verification steps, provide a count of 0 as <count>0</count>.

Task Format
Format your response in markdown as follows:

Thoughts
[Brief description describing what reasoning-based verification behaviors were detected.]

Does verification?
[yes/no]

Number of verification steps
<count> [number] </count>

Task to evaluate for Verification
{response}

Response

Figure 10: Prompt for analyzing subgoal setting and verification.

You are a helpful AI assistant specialized in solving complex mathematical problems with transparent, verifiable computation.

You will be given:
1. A complex math problem
2. A gold (reference) solution that includes:
- Natural-language reasoning
- Executable Python code enclosed in `<code>...</code>`
- Execution outputs wrapped in `<interpreter>...</interpreter>`
- A final boxed answer in `<answer> \boxed{...} </answer>` format

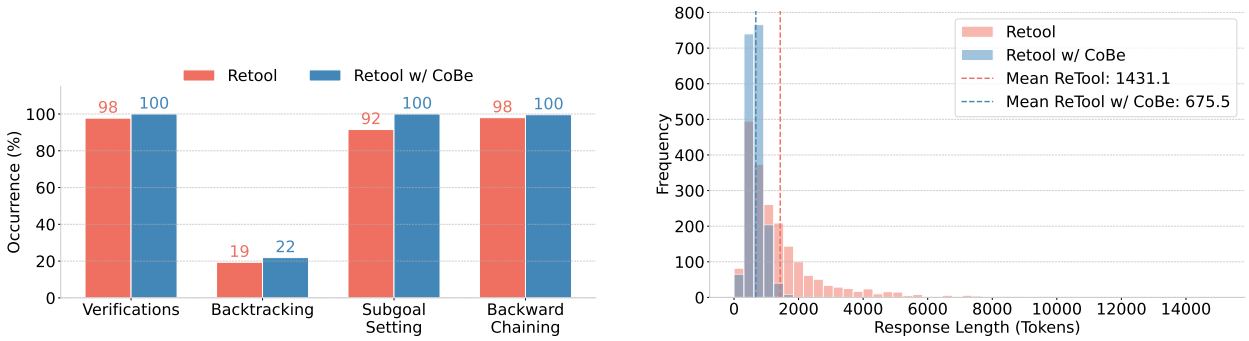
Task: Generate a new solution trajectory that correctly solves the problem and reaches the same final answer, using structured problem-solving and executable code.

You must demonstrate goal-directed problem solving by:
- Decomposing the problem into sub-goals.
- Solving sub-goals to reach the final objective.
- Verifying intermediate results.
- Backtracking and revising earlier steps if needed.
- Working backward from the final goal (of the given problem) if it makes things easier.

You should follow the same format as the gold solution, including:
- Natural-language reasoning in `<think>...</think>` tags
- Executable Python code enclosed in `<code>...</code>` tags
- Execution outputs from interpreter wrapped in `<interpreter>...</interpreter>`
- A final answer in `<answer> \boxed{...} </answer>`

Original Conversation
`<user> {original_prompt} </user>`
`<bot> {original_trajectory} </bot>`

Figure 11: Prompt for synthesizing a new response with CoBe.



(a) Code-specific cognitive behaviors.

(b) Response length comparisons.

Figure 12: Quantitative comparison between standard ReTool SFT data and ReTool augmented with CoBe across two characteristics: the prevalence of cognitive behaviors and response length distributions.

D Training Configurations and Computational Cost

All experiments were conducted using the training scripts provided in the official repository[4]. For supervised fine-tuning, models were trained for six epochs with a batch size of 32 and a learning rate of 10^{-6}. The maximum sequence length was set to 16,384 tokens, and Fully Sharded Data Parallel (FSDP) was employed to enable efficient distributed training. For reinforcement learning, we adopted the GRPO algorithm with an actor learning rate of 10^{-6}. The training was performed using a batch size of 64. To support long-chain-of-thought reasoning, the maximum prompt length and response length were set to 2,048 and 16,384 tokens, respectively. During rollout, 16 responses were sampled for each prompt. No KL-divergence regularization term was applied throughout training. The total computational expenditure, including both training and evaluation, amounted to approximately 1,608 GPU hours on NVIDIA A100 (80GB) GPUs.

E Additional Results

The results are consistent with our main findings. CoBe improves performance on Qwen2.5-3B-Instruct and Qwen3-8B-Base, suggesting that cognitive-behavior augmentation is generally beneficial. In contrast to Qwen3-8B-Instruct, the gains observed on Qwen3-8B-Base indicate that the degradation is likely related to post-training characteristics rather than the cognitive behaviors themselves.

Table 3: Additional evaluation results of CoBe on a smaller model and a base model.

Method	AIME 24	AIME25	MATH500	Average
Qwen2.5-3B-Instruct	12.1	3.3	69.0	28.2
+ ReTool-SFT	13.3	10.4	75.7	33.1
+ ReTool-SFT w/ CoBe	**17.1**	**14.2**	**74.5**	**35.2**
Qwen3-8B-Base	-	-	-	-
+ ReTool-SFT	25.4	17.1	**85.0**	42.5
+ ReTool-SFT w/ CoBe	**29.2**	**23.0**	84.1	**45.4**

F Additional Analysis

Figure 13 and Figure 14 show that ReTool with CoBe consistently induces the desired behaviors during evaluation, except in the SFT variant for verification on AIME 2024, where CoBe is 3% lower.

[4] https://github.com/verl-project/verl-recipe/tree/main/retool

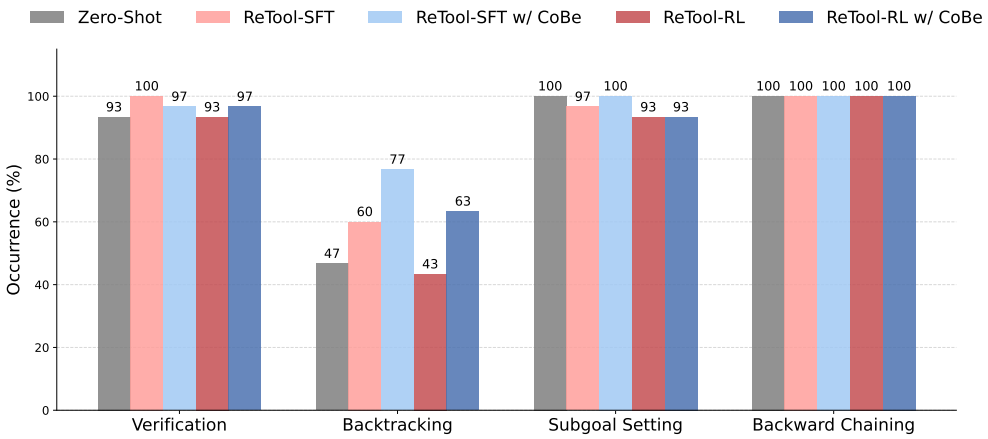


Figure 13: Cognitive behaviors of Qwen3-8B under ReTool training with and without CoBe on AIME24.

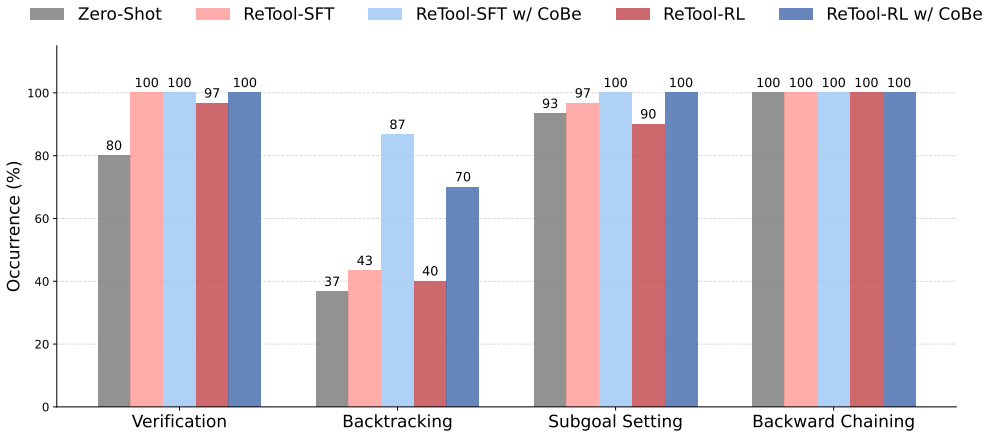


Figure 14: Cognitive behaviors of Qwen3-8B under ReTool training with and without CoBe on AIME25.