

# Prior Prompt Engineering for Reinforcement Fine-Tuning

Pittawat Taveekitworachai<sup>1</sup>, Potsawee Manakul<sup>1</sup>,  
Sarana Nutanong<sup>2</sup>, Kunat Pipatanakul<sup>1</sup>

<sup>1</sup>SCB 10X R&D,

SCB 10X, SCBX Group, Thailand

<sup>2</sup>School of Information Science and Technology,  
Vidyasirimedhi Institute of Science and Technology, Thailand

[pittawat@scb10x.com](mailto:pittawat@scb10x.com), [potsawee@scb10x.com](mailto:potsawee@scb10x.com), [snutanon@vistec.ac.th](mailto:snutanon@vistec.ac.th), [kunat@scb10x.com](mailto:kunat@scb10x.com)

## Abstract

This paper investigates prior prompt engineering (pPE) in the context of reinforcement fine-tuning (RFT), where language models (LMs) are incentivized to exhibit behaviors that maximize performance through reward signals. While existing RFT research has primarily focused on algorithms, reward shaping, and data curation, the design of the prior prompt—the instructions prepended to queries during training to elicit behaviors such as step-by-step reasoning—remains underexplored. We investigate whether different pPE approaches can guide LMs to internalize distinct behaviors after RFT. Inspired by inference-time prompt engineering (iPE), we translate five representative iPE strategies—reasoning, planning, code-based reasoning, knowledge recall, and null-example utilization—into corresponding pPE approaches. We experiment with Qwen2.5-7B using each of the pPE approaches, then evaluate performance on in-domain and out-of-domain benchmarks (e.g., AIME2024, HumanEval+, and GPQA-Diamond). Our results show that all pPE-trained models surpass their iPE-prompted counterparts, with the null-example pPE approach achieving the largest average performance gain and the highest improvement on AIME2024 and GPQA-Diamond, surpassing the commonly used reasoning approach. Furthermore, by adapting a behavior-classification framework, we demonstrate that different pPE strategies instill distinct behavioral styles in the resulting models. These findings position pPE as a powerful yet understudied axis for RFT.

## 1 Introduction

Recent advancements in reasoning models mark a significant step forward in improving language model (LM) performance by allocating additional compute budget at test time. A common approach to developing such models is reinforcement fine-tuning (RFT), which incentivizes an LM to perform extended reasoning during inference by using re-

ward signals—based on the correctness of generated answers—during training. Current studies have explored various components of the RFT pipeline, including objective functions and training algorithms (Liu et al., 2025; Yu et al., 2025; Yeo et al., 2025; Yue et al., 2025), data domains and curricula (Xie et al., 2025; Wei et al., 2025; Su et al., 2025; Hu et al., 2025), reward functions and shaping (Yeo et al., 2025; Su et al., 2025; Hu et al., 2025), and the influence of inherent behaviors across different LM families and model sizes (Liu et al., 2025; Zeng et al., 2025a; Gandhi et al., 2025). However, despite these improvements for various components of the RFT pipeline, one critical aspect remains understudied: *the design of the prompt*.

To scope our study, we separate a prompt used during RFT into two main components: the instruction and the task content (see Figure 2). The instruction guides the model to exhibit desired behaviors (e.g., step-by-step reasoning). We refer to this section as the **prior prompt**, which is the main focus of this study. Examples of prior prompts from existing work are provided in Appendix C. While some studies briefly note the role of prior prompts in training stability and performance (Xie et al., 2025; Zeng et al., 2025a), there has been little systematic investigation into how different prior prompting approaches during RFT shape model behaviors. This study therefore centers on the following question: *Can different prior prompt engineering approaches guide language models to internalize distinct behaviors during RFT?*

The breadth of prompt engineering, which we define in this paper as **inference-time prompt engineering** (iPE) to distinguish it from prompt engineering during training, demonstrates its effectiveness in eliciting diverse behaviors (i.e., generation patterns) from LMs (Kojima et al., 2022), ultimately leading to varying performance outcomes. For instance, chain-of-thought prompting (CoT) (Wei et al., 2022b) elicits step-by-step reasoning

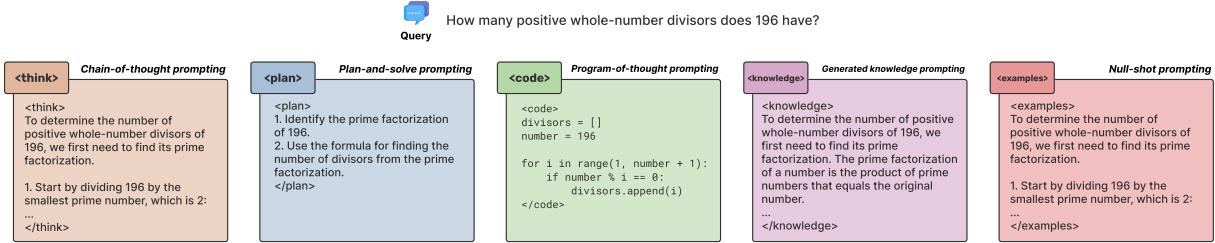


Figure 1: Five generated responses from five distinct models post-RFT with different pPE approaches—`<think>`, `<plan>`, `<code>`, `<examples>`, and `<knowledge>`. Each pPE approach is inspired by a corresponding iPE paradigm: chain-of-thought, plan-and-solve, program-of-thought, null-shot, and generated knowledge prompting, respectively.

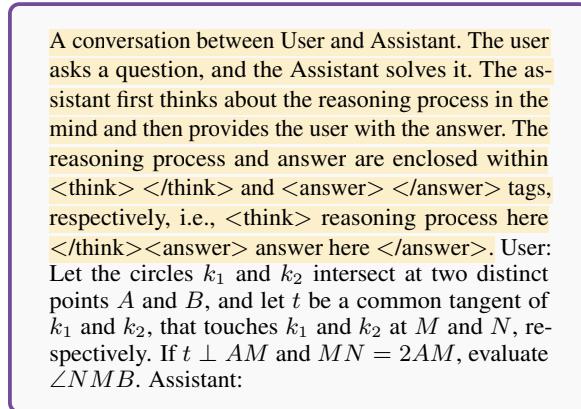


Figure 2: The prompt used during RFT by DeepSeek-AI et al. (2025). The prior prompt is highlighted in yellow. Non-highlighted content is task content.

before producing a final answer; plan-and-solve prompting (PS) (Wang et al., 2023) first generates a high-level plan before problem solving; and program-of-thought prompting (PoT) (Chen et al., 2023) induces code-based reasoning. These examples illustrate that different iPE approaches not only elicit distinct behaviors (e.g., reasoning, planning, coding) but also lead to varied performance results.

Inspired by iPE, we introduce the term **prior prompt engineering** (pPE) to denote approaches for modifying the prior prompt in RFT. Just as iPE guides behavior during *inference*, we conjecture that pPE can shape model behavior during *training*. By combining the varied elicitation induced by pPE with RFT’s incentivization mechanism, the resulting models may exhibit diverse behaviors and achieve different levels of performance impact.

In this paper, we study the effects of various pPE approaches on model behaviors and performance impact after RFT. We select five representative iPE approaches based on their distinct elicited behaviors—reasoning, planning, coding, knowledge recall, and null example utilization—and translate them into corresponding pPE approaches. We em-

ploy these five pPE approaches to train Qwen2.5 7B into five distinct models with RFT using math-only training data. We then compare each RFT-trained model to its corresponding iPE-only baseline.

We evaluate our models using both quantitative and qualitative methods. Quantitatively, we measure performance on mathematical reasoning, coding, and question-answering benchmarks (e.g., AIME2024, GPQA Diamond, and HumanEval+) to assess impact on in-domain and out-of-domain tasks. Qualitatively, we employ a modified behavior-classification framework from Gandhi et al. (2025) to quantify differences in model behaviors. To test generalization, we replicate our experiments at smaller scales on Qwen2.5 3B, Qwen2.5 Coder 7B, and Llama 3.1 8B.

We find that all pPE-trained models surpass their corresponding iPE-only baselines. Among pPE approaches, the null-example utilization approach—which exhibits behavioral similarities to the reasoning approach—achieves the largest improvement on GPQA Diamond and the highest average performance gain across tasks. Figure 1 illustrates the five models trained with different pPE strategies, each demonstrating distinct behavioral styles and indicating that pPE can incentivize diverse behaviors. Our contributions are as follows:

- We introduce the concepts of **prior prompt** and **prior prompt engineering** (pPE) as critical yet previously understudied aspects of RFT.
- We demonstrate that different pPE strategies elicit distinct behaviors, including variations in performance impact, response structure, verbosity, and behavior types.
- We propose an updated systematic behavior classification approach to quantify both cognitive and elicited behaviors, revealing how different pPE approaches shape model behavior.

## 2 Prior Prompt Engineering for Reinforcement Fine-Tuning

Our main question in this study is whether different pPE approaches can lead an LM to internalize distinct behavioral styles after RFT. If different pPE approaches indeed yield different behaviors, this could provide a simple means—by only changing the prior prompt—to train models for specialized behaviors beyond reasoning (e.g., plan generation, code-based reasoning, or knowledge generation). To answer this question, we select five representative iPE approaches and translate them into pPE approaches. We then apply a standard RFT setup to train five distinct models, differing only in their pPE approach and format reward (see Section 3). The overall process and distinctions between iPE and pPE are depicted in Figure 3.

We evaluate each model quantitatively and qualitatively to assess performance changes and behavioral differences. Quantitative evaluation uses established benchmarks for mathematical reasoning, coding, and question answering. For qualitative evaluation, we adapt the framework of Gandhi et al. (2025) to classify each post-RFT model’s behavior into one of four cognitive categories and five pPE-specific categories. We also apply these evaluations to the base model at inference time with different iPE approaches, to further compare iPE and pPE.

To explore the impact of pPE approaches on prior prompts, we select five representative iPE approaches based on their differences in behavioral elicitation when used to prompt an LM:

1. **Reasoning:** Chain-of-thought prompting (CoT) (Wei et al., 2022b) elicits an LM to generate step-by-step reasoning before producing a final answer. This iPE approach is mapped to `<think>` in pPE and is the most commonly used in RFT studies, resulting in reasoning models (DeepSeek-AI et al., 2025; Xie et al., 2025). This serves as our baseline for comparison.
2. **Planning:** Plan-and-solve prompting (PS) (Wang et al., 2023) elicits the model to first generate a plan (e.g., numbered steps) and then execute that plan, yielding improvements over standard CoT. The planning approach is mapped to `<plan>` in pPE. We expect the post-RFT model to generate a plan before providing an answer.
3. **Code-based reasoning:** Program-of-thought prompting (PoT) (Chen et al., 2023) elicits

structured reasoning through code by asking a model to generate relevant code for problem solving. PoT has shown strong performance on math and logic tasks, especially with code-pretrained models such as CodeLlama (Rozière et al., 2024), Qwen2.5-Coder (Hui et al., 2024), and StarCoder 2 (Lozhkov et al., 2024). This iPE approach is mapped to `<code>` in pPE. We expect the post-RFT model to generate code and comments that solve the given task.

4. **Knowledge recall:** Generated knowledge prompting (Liu et al., 2022) asks the model to recall or synthesize relevant knowledge before answering, simulating a form of self-retrieval and improving performance on commonsense benchmarks. This approach is mapped to `<knowledge>` in pPE. We expect the post-RFT model to recall definitions, theorems, or formulas before proceeding to a final answer.
5. **Null-example utilization:** Null-shot prompting (Taveekitworachai et al., 2024) prompts the model to utilize non-existent in-context examples relevant to the question, exploiting inductive biases without providing real demonstrations. It maps to `<examples>` in pPE, and we expect the post-RFT model to generate or reference illustrative examples relevant to a query.

With these five distinct pPE approaches for eliciting different behaviors in LMs during RFT, we expect not only differences in performance impact and post-RFT behaviors but also in training dynamics, such as average response length or per-step reward trajectories.

## 3 Experimental Setup

### 3.1 Prior Prompts

To construct our prior prompts, we adapt the template of Xie et al. (2025). For each iPE approach, we modify the instruction in the template (e.g., “plan,” “recall relevant knowledge,” “write required code”) and update the corresponding tag `<x></x>` as described in Section 2. The `<think></think>` pPE approach thus is the standard RFT setup. These same templates are also used when evaluating iPE-prompted models. The complete prior prompt templates are provided in Appendix D.1.

### 3.2 Training

We follow a standard RFT setup similar to DeepSeek-AI et al. (2025). Specifically, we use

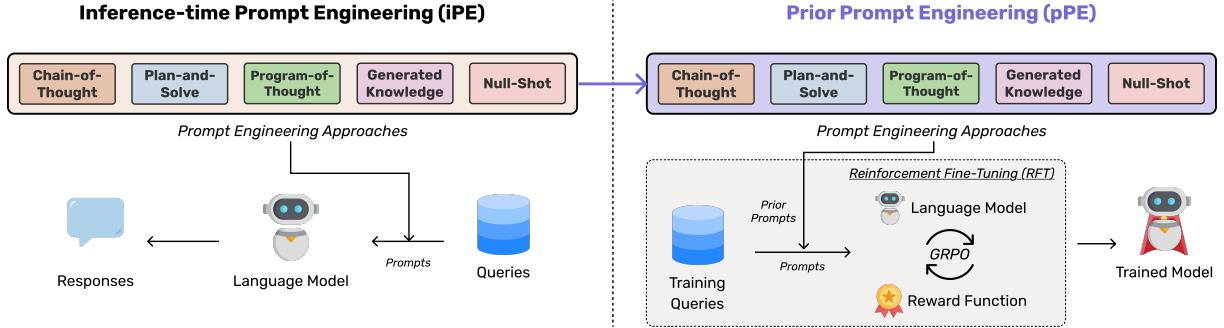


Figure 3: *Left*: iPE approaches are applied to a prompt during inference, before inputting it into an LM, to elicit desired behaviors in the response. *Right*: pPE approaches are translated from iPE approaches and applied to the prior prompt to elicit desired behaviors during training.

Group Relative Policy Optimization (GRPO) with a pretrained base LM. Our training stack is Open-RLHF v0.6.4 (Hu et al., 2024) for policy optimization and vLLM v0.8.2 (Kwon et al., 2023) for rollout generation. We train using prompts from the STILLv3 dataset (Chen et al., 2025), which contains approximately 30K mathematical problems and is used to train a reasoning model. We note that the use of math-only training datasets is common in the existing literature (Liu et al., 2025; Yeo et al., 2025; Yu et al., 2025). In addition, math-only training datasets provides a simplicity in verifiable reward design, i.e., value equivalent checking between a generated answer and the ground truth, unlike other domains, which inconclusive in implementation standards of the reward function. Additional details and hyperparameters are listed in Appendix D.3.

Our reward function comprises two equally weighted components (summing to 1.0): (1) *accuracy*, which assesses whether the model produces the correct final answer; and (2) *format*, which assesses whether the model’s output follows the expected format- $\langle x \rangle \langle /x \rangle$  followed by  $\langle \text{answer} \rangle \langle / \text{answer} \rangle$ , where  $x$  is one of {think, plan, code, knowledge, examples}. The expected format is updated dynamically to match the pPE approach. Additional details on the reward function are available in Appendix D.2.

For our main experiments, we use Qwen2.5-7B (Qwen et al., 2025) as the base model. All five pPE variants are trained with the same settings, differing only in the pPE approaches. We also use Qwen2.5-7B, prompted at inference with each iPE approach, as our comparison baseline. To isolate the effects of inherent performance changes from those of the dataset, we also train Qwen2.5-7B on the dataset without any prior prompts—thus without

format instructions and without a format reward; accuracy reward maxed at 1.0. This setup serves as our No PP baseline to distinguish dataset effects from those of pPE approaches.

We select Qwen2.5-7B, a base model, to follow the R1-Zero (DeepSeek-AI et al., 2025) approach and mitigate confounding factors from instruction tuning of instruct models; evaluation of the instruct variant is left for future work. Although prompting base models with iPE approaches has become less common in recent years due to the prevalence of instruct models, prior studies introducing the iPE methods (Wei et al., 2022b; Wang et al., 2023; Chen et al., 2023; Liu et al., 2022; Taveekitworachai et al., 2024) considered here have shown it to be effective with base models.

To assess generalization under budget constraints, we conduct scaled-down experiments along two dimensions: model size and model family. For model size, we train Qwen2.5-3B with  $\langle \text{think} \rangle$  and  $\langle \text{plan} \rangle$ , as it belongs to the same family as Qwen2.5-7B from the main experiment and allows direct size comparison. For model family, we evaluate Llama 3.1-8B (Grattafiori et al., 2024) with  $\langle \text{think} \rangle$  and  $\langle \text{plan} \rangle$ , chosen for its comparable size to Qwen2.5-7B, and Qwen2.5-Coder-7B (Hui et al., 2024) with  $\langle \text{think} \rangle$  and  $\langle \text{code} \rangle$ , included to examine differences between a code-specialized model and its base counterpart. We prioritize  $\langle \text{plan} \rangle$  as the main comparator due to its distinct behaviors during and after RFT, while  $\langle \text{code} \rangle$  probes domain-specific specialization.

### 3.3 Evaluation

We evaluate all models and prompting methods via quantitative and qualitative analyses.

**Quantitative benchmarks** Although our training set is math-only, we also evaluate all models on

non-mathematical benchmarks to assess generalization. We report average accuracy across the following: *Mathematical reasoning*: AIME2024 (AIME) (Li et al., 2024), AMC12 ’22–’23 (AMC) (Li et al., 2024), and MATH-500 (MATH) (Hendrycks et al., 2021); *Coding*: HumanEval+ (HE+) base and extra sets (Liu et al., 2023); *Question answering*: GPQA-Diamond (GPQA) (Rein et al., 2024). Additional details are provided in Appendix D.4.1.

**Qualitative analysis** We analyze differences across: (1) *Training dynamics*, (2) *Average response length*, (3) *Ratio of four fundamental cognitive behaviors* (Gandhi et al., 2025), and (4) *Ratio of behavior patterns specific to each of the five pPE categories*. Four fundamental cognitive behaviors are (i) Verification: identifying errors; (ii) Backtracking: proposing an alternative approach; (iii) Subgoal setting: generating intermediate steps; and (iv) Backward chaining: reasoning from the result to inputs. For (3) and (4), we employ the LM-based classification framework of Gandhi et al. (2025) to automatically classify model responses. Further details are in Appendix D.4.2.

## 4 Results and Findings

In this section, we present and discuss results from our experiments, as described the setup in Section 3. Our objective is to answer the core question posed earlier: *whether and how different pPE approaches can guide LMs to internalize distinct behaviors during RFT*. To address this question, we examine three key aspects:

1. **Performance impact:** Do different pPE approaches lead to measurable improvements over the baseline and their iPE counterparts? Do they result in distinct performance gains across tasks, or do they converge to similar outcomes?
2. **Behavioral differences:** Do different pPE approaches induce differences in fundamental cognitive behaviors and elicited generation patterns? Do the behavioral profiles of pPE-trained models align with those observed under iPE?
3. **Generalization:** How well do pPE approaches generalize across model sizes and families?

The following subsections address each of these aspects in detail. Additional and detailed results, including results from the generalization study, are presented in Appendix E.

### 4.1 Performance Impact

Model	AIME	AMC	GPQA	MATH	HE+	Avg.
<b>Qwen2.5-7B</b>	13.33	37.35	24.24	55.60	72.60	40.62
<b>iPE</b>						
Think	10.00	31.33	24.24	56.00	75.00	39.31
Plan	10.00	30.12	24.24	51.20	73.80	37.87
Code	13.33	26.51	24.24	51.40	72.00	37.50
Knowledge	20.00	25.30	24.24	59.60	72.00	40.23
Examples	16.67	32.53	24.24	56.80	0.00	26.05
<b>RFT</b>						
No PP	<b>26.67</b>	37.35	21.21	70.40	73.80	45.41
<b>pPE</b>						
Think	20.00	43.37	28.28	<b>73.20</b>	70.10	46.99
Plan	20.00	<u>44.58</u>	24.75	69.60	68.90	45.57
Code	16.67	<b>46.99</b>	25.25	66.20	<b>78.00</b>	46.62
Knowledge	16.67	37.35	21.72	71.00	73.20	43.99
Examples	20.00	43.37	<b>30.81</b>	<u>71.20</u>	72.60	<b>47.60</b>

Table 1: Benchmark accuracy (%) of Qwen2.5-7B when prompted with different iPE or RFT with different pPE approaches across five benchmarks. **No PP** represents a baseline trained with RFT without any prior prompts. **Bold** indicates the best performance per column; underlined indicates the second best per column.

Table 1 presents the performance of Qwen2.5-7B when prompted with different iPE approaches or fine-tuned using RFT with different pPE approaches across benchmarks. Notably, *all iPE approaches result in lower average performance compared to the base model*. For instance, under the null-example utilization approach, iPE fails to generate parsable code during HE+ evaluation, yielding 0.00.

In contrast, *all post-RFT models—regardless of the pPE approach—achieve performance improvements over the base model*. Part of these gains can be attributed to the dataset itself: training without a prior prompt (No PP) raises AIME from 13.33 to 26.67 and MATH from 55.60 to 70.40, outperforming every iPE variant and even all pPE variants on these two math-heavy benchmarks. This indicates that the dataset and RFT alone drive substantial improvements in mathematical reasoning.

The influence of prior prompts becomes more apparent on other benchmarks. On AMC, No PP provides no improvement (37.35 to 37.35), yet every pPE variant surpasses it, with code-based reasoning reaching 46.99. Several pPE approaches also improve GPQA, and the code-based approach raises HE+ from 73.80 to 78.00. Importantly, all pPE variants except knowledge recall outperform No PP in average performance, demonstrating that prior prompts exert an independent effect beyond dataset-driven gains.

The widely used reasoning approach, `<think>`, serves as a strong pPE baseline and delivers substantial gains (+6.37 points). Surprisingly, the

null-example utilization approach, which performs worst under iPE, achieves the highest average improvement (+6.98 points) after RFT—surpassing `<think>`. Notably, while the null-example iPE approach fails entirely on HE+, its pPE counterpart maintains strong performance on that benchmark. Conversely, the knowledge recall approach, which yields the best iPE performance, produces the weakest results in the pPE setting. These contrasts underscore that *performance trends in iPE do not directly translate to pPE*, highlighting the fundamentally different mechanisms underlying inference-time prompting and RFT.

Finally, as with iPE, pPE methods exhibit diverse benchmark-specific effects. The code-based reasoning approach, while expectedly excelling on HE+, also delivers the strongest AMC score. In contrast, knowledge recall fails to provide meaningful gains on GPQA and even underperforms relative to the base model. Together, these results suggest that *the impact of pPE is more nuanced than simply aligning a domain-specific prompt with a domain-specific task*. We leave further investigation of these dynamics to future work.

## 4.2 Behavioral Differences

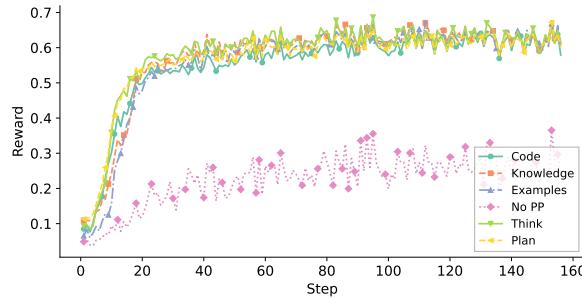


Figure 4: The reward progression of Qwen2.5-7B during RFT exhibits similar trends—an initial climb followed by fluctuations—across all pPE approaches, except for No PP, which yields lower rewards as it focuses only on accuracy without the format component.

**Training dynamics** Figures 4 and 5 show the reward and average response length dynamics during RFT for each pPE approach, respectively. For all pPE variants, the training curves are highly consistent: reward increases sharply in the first 20 steps, likely reflecting the model learning to follow format constraints, and then enters a steadier phase with minor fluctuations. This phase coincides with a gradual recovery of response length after an initial drop, suggesting that the model begins to exploit a larger token budget in pursuit of higher rewards.

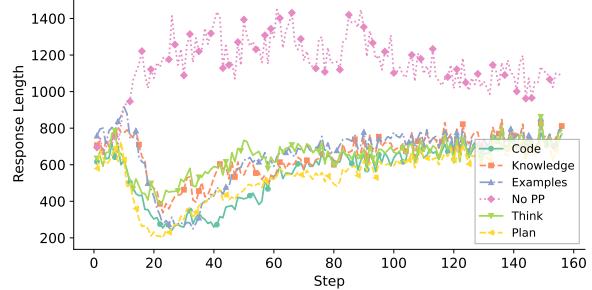


Figure 5: Evolution of the average response length for Qwen2.5-7B during RFT shows an initial drop followed by gradual recovery across pPE approaches, whereas the No PP baseline maintains a higher and more stable response length throughout.

The No PP baseline, however, exhibits markedly different dynamics. In reward progression, it fails to reach the same level as pPE approaches, as it optimizes only for accuracy without benefiting from a format reward. In response length, it shows consistently longer outputs and a relatively steady trend rather than the sharp dip-and-recovery pattern observed in pPE approaches.

These results show that *pPE affects not only final model performance but also the training process itself*. Models trained with pPE share consistent dynamics—rapid reward gains followed by stabilization, along with a dip and recovery in response length—whereas the No PP baseline follows a completely different trajectory, with lower rewards and longer responses. This contrast highlights that the inclusion of a prior prompt fundamentally shapes how the model learns during RFT.

At the same time, other factors such as the training algorithm (Yu et al., 2025; Liu et al., 2025), the base model family (Zeng et al., 2025a), and hyperparameter choices remain important determinants of training behavior. Finally, the divergence between training dynamics and final benchmark results suggests that *metrics like reward progression and response length should not be relied upon as predictors of final performance*.

**Average response length** Table 2 reports the average response length, measured as the mean number of generated tokens during quantitative evaluation. We find that reasoning and null-example utilization iPE approaches already elicit longer responses compared to the base model. After RFT, average response length generally increases further, though the No PP baseline produces by far the longest responses across all benchmarks, despite not achieving the strongest performance. In con-

Model	AIME	AMC	GPQA	MATH	Avg.
<b>Qwen2.5-7B</b>	1416.80	1352.54	534.29	841.74	1036.34
<b>iPE</b>					
Think	2512.17	1367.69	534.29	804.85	1304.75
Plan	1662.57	644.90	534.29	540.98	845.69
Code	641.07	953.51	534.29	635.09	690.99
Knowledge	1406.30	1237.22	534.29	780.31	989.53
Examples	2274.17	1316.12	534.29	752.60	1219.30
<b>RFT</b>					
No PP	2902.97	1543.40	982.79	850.33	1569.87
<b>pPE</b>					
Think	2042.70	1024.96	476.86	612.10	1039.16
Plan	1685.17	1085.47	476.47	601.18	962.07
Code	1657.47	836.28	492.44	690.42	919.15
Knowledge	2015.57	1082.96	587.10	626.45	1078.02
Examples	1136.20	831.98	442.48	685.79	774.11

Table 2: Average response length, i.e., number of tokens, of Qwen2.5-7B when prompted with different iPE or RFT with different pPE approaches.

trast, pPE variants yield shorter and more varied response lengths.

Interestingly, the null-example utilization pPE approach achieves the highest overall performance while producing some of the shortest responses on average, making it the most efficient in terms of test-time compute. By comparison, the reasoning pPE approach also reduces response length relative to its iPE counterpart while still delivering strong performance gains.

These results indicate that *different pPE approaches shape not only the performance but also the efficiency of post-RFT models*. In particular, these results highlight *pPE as a practical tool for influencing the trade-off between model accuracy and computational efficiency*.

**Four fundamental behaviors** Figure 6 shows the ratio of four fundamental cognitive behaviors in responses from the quantitative evaluation, both when prompted with iPE approaches and after RFT with pPE approaches. We observe that *backward chaining is the most prominent behavior across all models*—regardless of whether iPE or pPE is used—and is already present in the base model. Interestingly, the base model with RFT without any prior prompts displays even higher levels of backward chaining, backtracking, and verifications than the raw base, indicating that RFT with the math-only dataset alone encourages more cognitive behaviors.

In general, iPE approaches increase the frequency of backward chaining, while pPE approaches tend to reduce it, with the exception of the planning approach. More broadly, pPE approaches tend to decrease the overall presence of all four fundamental cognitive behaviors compared to iPE. Importantly, the ratio of these behaviors does not cor-

relate well with final model performance. However, these ratios remain useful for highlighting how fundamental cognitive behaviors shift post-RFT, and for differentiating between pPE approaches based on their behavioral profiles.

We speculate that this behavior classification framework—originally developed to analyze reasoning models (Gandhi et al., 2025)—may not generalize well to models trained with different pPE paradigms, which may incentivize different forms of fundamental behavior beyond those captured by the current classification framework.

**Five elicited behaviors** Figure 7 shows changes in the frequency of five elicited behaviors when models are prompted or trained using iPE or pPE approaches, relative to the base model under zero-shot prompting. We observe that most iPE approaches—with the exception of reasoning and planning—elicit high levels of reasoning, planning, and knowledge recall behaviors. In contrast, post-RFT behavior patterns are more targeted: *post-RFT models tend to show their largest gains in the behavior aligned with the specific pPE approach they were trained on, with the notable exception of the null-example utilization approach*. For instance, the planning pPE yields the strongest increases in planning, while the code pPE uniquely boosts code-related behaviors—consistent with expectations.

The No PP baseline shows that RFT alone increases reasoning, planning, and knowledge recall behaviors relative to the zero-shot baseline. This indicates that RFT with math-only datasets already incentivizes models to exhibit these behaviors.

Finally, each pPE still induces a distinct distribution across the five behaviors. Notably, the null-example utilization pPE yields the fewest knowledge recall instances, yet achieves the highest performance gains on GPQA. Furthermore, it also exhibits the lowest number of null-example behavior instances—in contrast to its iPE counterpart and to our expectations. This suggests that *a pPE approach may not always result in the model exhibiting the anticipated behavior*. Instead, the model may discover more effective behavior patterns during RFT, independent of the specific pPE approach.

**Qualitative behaviors** We present qualitative examples of generated responses in Appendix F. We observe that *post-RFT models are generally able to produce behaviors aligned with the pPE approaches*. For example, the planning pPE approach results in models that generate a numbered list

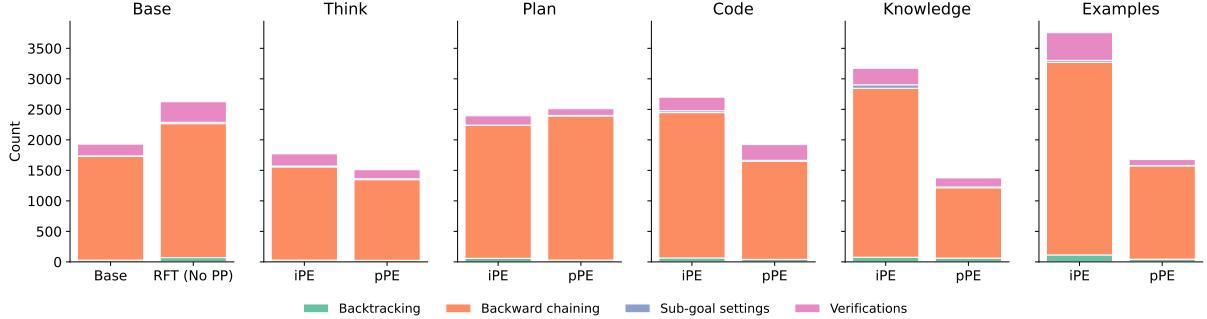


Figure 6: Ratio of the four fundamental cognitive behaviors—backtracking, backward chaining, subgoal setting, and verification—across different prompting (iPE) and RFT (pPE) approaches with Qwen2.5-7B. Backward chaining dominates across setups, especially under iPE.

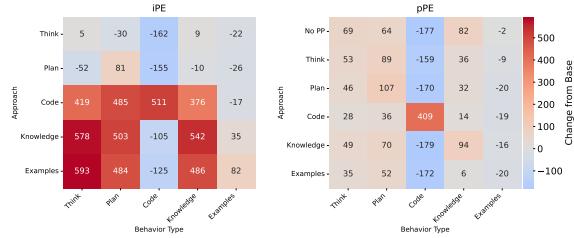


Figure 7: Ratio of five elicited behavior categories—reasoning, planning, code-based reasoning, knowledge recall, and null-example utilization—as observed when behaviors are elicited through prompting with different iPE approaches (left) and after RFT with different pPE approaches (right).

of steps to solve the problem and then execute them. The reasoning pPE approach leads to step-by-step reasoning, while the knowledge recall pPE approach elicits definitions and formulas relevant to solving the task. Interestingly, the null-example utilization pPE approach somewhat resembles the behavior of the reasoning pPE model, despite its differences in performance trends.

We also observe that Qwen2.5-7B tends to prefer natural language reasoning over code-based reasoning during RFT. Specifically, under the code-based reasoning pPE approach, the model frequently generates natural language reasoning, followed by a statement such as:

```
<code>
# We don't need to write any Python code
since the problem is solved analytically.
</code>
```

This stands in contrast to the code-specialized model, which, as shown in the qualitative examples, relies more heavily on code generation as part of its problem-solving process. *These behavioral differences among post-RFT models suggest that*

*RFT with different pPE approaches can be used to steer models toward exhibiting distinct, desired behaviors—similar to RLHF (Ouyang et al., 2022).* For instance, it is possible to train a plan-generating model by applying RFT with a prior prompt that elicits plan generation. However, for such a model to be effective, the plan must not only be valid but also executable in a way that achieves a high reward. In this context, the reward signal serves as a proxy for plan quality. We note that all qualitative analyses presented here are preliminary and focus on observed differences in behaviors. A deeper analysis of the mechanisms by which pPE or RFT influence the trained model’s behaviors is beyond the scope of this work. We further discuss the implications, extensions, and applications of pPE for RFT in Appendix B, as well as limitations and possible extensions in Limitations.

### 4.3 Generalization

**Performance impact** Table 3 shows the performance of Qwen2.5-3B, LLaMA 3.1-8B, and Qwen2.5-Coder-7B, which serve as representative models for our generalization studies. Additional accompanying results—including training dynamics and behavior classification—are available in Appendix E. We observe that the reasoning pPE approach, i.e., `<think>`, is consistently robust across model families and sizes. This is likely due to its alignment with behavioral patterns already familiar to models from prior fine-tuning on CoT-like data (Chung et al., 2024). In contrast, *smaller or weaker model families show limited success with non-reasoning pPE approaches*, aligning with findings from (Zeng et al., 2025a) that such models benefit less from reasoning RFT.

Model	AIME	AMC	GPQA	MATH	HE+	Avg.
<b>Qwen2.5-3B</b>	<b>13.33</b>	<b>24.10</b>	<u>9.60</u>	<b>49.40</b>	<b>62.80</b>	<b>31.85</b>
<b>iPE</b>						
Think	<u>13.33</u>	<u>22.89</u>	<u>9.60</u>	<u>37.20</u>	<u>61.00</u>	<u>28.80</u>
Plan	<u>13.33</u>	<u>15.66</u>	<u>9.60</u>	<u>35.20</u>	<u>60.40</u>	<u>26.84</u>
<b>pPE</b>						
Think	10.00	12.05	<b>11.11</b>	28.40	<b>61.00</b>	24.51
Plan	0.00	0.00	7.07	0.00	59.10	13.23
<b>Llama 3.1-8B</b>	0.00	1.20	0.00	5.00	31.70	7.58
<b>iPE</b>						
Think	<u>3.33</u>	3.61	0.00	8.00	31.70	9.33
Plan	<b>6.67</b>	<u>4.82</u>	0.00	6.80	31.70	<b>10.00</b>
<b>pPE</b>						
Think	<u>3.33</u>	<b>6.02</b>	0.00	<b>9.60</b>	<u>32.30</u>	<b>10.25</b>
Plan	<u>3.33</u>	2.41	0.00	8.40	<u>32.90</u>	9.41
<b>Qwen2.5-Coder-7B</b>	<b>6.67</b>	15.66	25.25	<u>27.60</u>	<b>81.10</b>	31.26
<b>iPE</b>						
Think	0.00	10.84	25.25	22.20	75.60	26.78
Code	3.33	10.84	25.25	12.00	78.00	25.88
<b>pPE</b>						
Think	<b>13.33</b>	<b>37.35</b>	<u>26.26</u>	<b>68.80</b>	<u>78.70</u>	<b>44.89</b>
Code	<b>13.33</b>	<u>18.07</u>	<b>34.85</b>	26.80	<u>75.00</u>	<u>33.61</u>

Table 3: Benchmark accuracy (%) of Qwen2.5-3B, Llama 3.1-8B, and Qwen2.5-Coder-7B when prompted with different iPE or RFT with different pPE approaches across five benchmarks. **Bold** indicates the best performance per column under the same base model; underlined indicates the second best per column under the same base model.

**Behavioral differences** We also observe instances of reward hacking when RFT is applied with the planning pPE approach in Qwen2.5-3B and LLaMA 3.1-8B. In these cases, the models output only correctly formatted responses in order to maximize the format reward, while neglecting further exploration of the accuracy reward (see Figures 25 to 28). Another notable observation is that the code-specialized Qwen2.5-Coder-7B model is more effective at exhibiting code-based reasoning behaviors—under both iPE and pPE—compared to Qwen2.5-7B. This illustrates how different model families can influence the behaviors exhibited after RFT (Zeng et al., 2025a). Nevertheless, both reasoning and code-based reasoning pPE approaches improve performance over the baseline, although the reasoning pPE approach achieves the highest performance. Still, pPE demonstrates measurable success over iPE in steering model behavior post-RFT, as illustrated in Figures 34 to 36. These findings suggest that *pPE generalizes reliably in stronger, behaviorally aligned models, while less capable models are more prone to reward hacking or fail to internalize the intended behaviors*.

## 5 Conclusions

This paper investigates the impact of pPE in the context of RFT by evaluating five pPE approaches

inspired by iPE: reasoning, planning, code-based reasoning, knowledge recall, and null-example utilization. While these approaches often degrade performance when applied only at inference time (iPE), incorporating them during RFT (pPE) consistently improves performance relative to the base model. In particular, null-example utilization proves more effective than the reasoning approach for enhancing downstream task performance.

Beyond these performance impact, different pPE approaches induce distinct behavioral patterns in the fine-tuned models. For example, models trained with the planning pPE approach tend to exhibit a “plan-and-solve” behavior, i.e., generating a list of steps before execution. Finally, we explore the generalization of pPE approaches across model sizes and families. We hope this study will inspire further research into the role of pPE in RFT, especially given the extensive literature on iPE.

## Limitations

Due to computational resource constraints, we were unable to conduct experiments with larger model sizes, larger datasets, or a higher number of steps. As a result, the behavioral trends observed in this study remain inconclusive for larger models where emergent abilities (Wei et al., 2022a)—known to appear only at scale—may lead to different outcomes. While we believe many of our findings will generalize across model sizes (as is often the case in iPE studies), this assumption remains to be validated. In contrast, smaller models may not capture the complexity or expressiveness of their larger counterparts due to their lower capacity and the limited potential of pPE, similar to iPE.

Additionally, we fixed the training data domain (mathematics), the reinforcement learning algorithm (GRPO), and other experimental configurations to isolate the effect of pPE approaches. Future work should investigate how different domains, RL algorithms, and reward schemes interact with pPE. We conjecture that, as with iPE, once a model demonstrates the ability to exhibit structured behaviors, those behaviors will generalize across architectures and settings. However, further studies are necessary to confirm this generalization in a broader context of RFT.

## Ethical Considerations

Prompting language models to elicit specific behaviors is inherently unpredictable due to their stochas-

tic nature (Bengio et al., 2000). RFT, which aims to amplify specific generation patterns for improved performance, may also unintentionally reinforce undesirable or unsafe behaviors—especially those that were already latent in the pretrained model.

We strongly recommend integrating established alignment techniques (Grattafiori et al., 2024; Bai et al., 2022; Dai et al., 2024) and safety measures (Zeng et al., 2025b; Inan et al., 2023), as prior studies (DeepSeek-AI et al., 2025; Seed et al., 2025) have shown that such safeguards remain effective even after RFT. As with iPE, models in dynamic or open-ended environments are vulnerable to misuse. For example, they may be exposed to malicious prompts (Liu et al., 2024) or poisoned data (Zhao et al., 2025) during RFT, leading to unexpected or concerning behaviors.

Furthermore, the pPE framework proposed in this study can be extended to alignment and safety-focused training, similar to recent efforts in *deliberative alignment* (Guan et al., 2025). To mitigate risks, we recommend safeguards such as prompt auditing, robust reward design, safe rollout filtering, and post-training alignment steps—especially when applying RFT in safety-critical or user-facing applications.

## References

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, and 32 others. 2022. **Constitutional AI: Harmlessness from AI Feedback**. *Preprint*, arXiv:2212.08073.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2000. **A Neural Probabilistic Language Model**. In *Advances in Neural Information Processing Systems*, volume 13. MIT Press.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. **Language Models are Few-Shot Learners**. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. **Program of Thoughts** Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *Transactions on Machine Learning Research*.

Zhipeng Chen, Yingqian Min, Beichen Zhang, Jie Chen, Jinhao Jiang, Daixuan Cheng, Wayne Xin Zhao, Zheng Liu, Xu Miao, Yang Lu, Lei Fang, Zhongyuan Wang, and Ji-Rong Wen. 2025. **An Empirical Study on Eliciting and Improving R1-like Reasoning Models**. *Preprint*, arXiv:2503.04548.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, and 16 others. 2024. **Scaling instruction-finetuned language models**. *Journal of Machine Learning Research*, 25(70):1–53.

Josef Dai, Xuehai Pan, Ruiyang Sun, Jiaming Ji, Xinbo Xu, Mickel Liu, Yizhou Wang, and Yaodong Yang. 2024. **Safe RLHF: Safe Reinforcement Learning from Human Feedback**. In *The Twelfth International Conference on Learning Representations*.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhi-hong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. **DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning**. *Preprint*, arXiv:2501.12948.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinjin Chi, and Wanjun Zhong. 2025. **ReTool: Reinforcement Learning for Strategic Tool Use in LLMs**. *Preprint*, arXiv:2504.11536.

Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D. Goodman. 2025. **Cognitive Behaviors that Enable Self-Improving Reasoners, or, Four Habits of Highly Effective STaRs**. *Preprint*, arXiv:2503.01307.

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, and 542 others. 2024. **The Llama 3 Herd of Models**. *Preprint*, arXiv:2407.21783.

Melody Y. Guan, Manas Joglekar, Eric Wallace, Saachi Jain, Boaz Barak, Alec Helyar, Rachel Dias, Andrea Vallone, Hongyu Ren, Jason Wei, Hyung Won Chung, Sam Toyer, Johannes Heidecke, Alex Beutel, and Amelia Glaese. 2025. **Deliberative Alignment: Reasoning Enables Safer Language Models**. *Preprint*, arXiv:2412.16339.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. [Measuring Mathematical Problem Solving With the MATH Dataset](#). In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Jian Hu, Xibin Wu, Zilin Zhu, Xianyu, Weixun Wang, Dehao Zhang, and Yu Cao. 2024. [OpenRLHF: An Easy-to-use, Scalable and High-performance RLHF Framework](#). *Preprint*, arXiv:2405.11143.

Jingcheng Hu, Yinmin Zhang, Qi Han, Dixin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. 2025. [OpenReasoner-Zero: An Open Source Approach to Scaling Up Reinforcement Learning on the Base Model](#). *Preprint*, arXiv:2503.24290.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, and 5 others. 2024. [Qwen2.5-Coder Technical Report](#). *Preprint*, arXiv:2409.12186.

Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, and Madian Khabsa. 2023. [Llama Guard: LLM-based Input-Output Safeguard for Human-AI Conversations](#). *Preprint*, arXiv:2312.06674.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. [Large Language Models are Zero-Shot Reasoners](#). In *Advances in Neural Information Processing Systems*.

Komal Kumar, Tajamul Ashraf, Omkar Thawakar, Rao Muhammad Anwer, Hisham Cholakkal, Mubarak Shah, Ming-Hsuan Yang, Phillip H. S. Torr, Fahad Shahbaz Khan, and Salman Khan. 2025. [LLM Post-Training: A Deep Dive into Reasoning Large Language Models](#). *Preprint*, arXiv:2502.21321.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. [Efficient Memory Management for Large Language Model Serving with PagedAttention](#). In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP '23*, page 611–626, New York, NY, USA. Association for Computing Machinery.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, and 1 others. 2024. [Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions](#). *Hugging Face repository*, 13:9.

Jiacheng Liu, Alisa Liu, Ximing Lu, Sean Welleck, Peter West, Ronan Le Bras, Yejin Choi, and Hannaneh Hajishirzi. 2022. [Generated knowledge prompting for commonsense reasoning](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3154–3169, Dublin, Ireland. Association for Computational Linguistics.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. [Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation](#). In *Thirty-seventh Conference on Neural Information Processing Systems*.

Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, and Yang Liu. 2024. [Prompt Injection attack against LLM-integrated Applications](#). *Preprint*, arXiv:2306.05499.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025. [Understanding R1-Zero-Like Training: A Critical Perspective](#). *Preprint*, arXiv:2503.20783.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, and 47 others. 2024. [StarCoder 2 and The Stack v2: The Next Generation](#). *Preprint*, arXiv:2402.19173.

Sewon Min, Xinxixi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. [Rethinking the role of demonstrations: What makes in-context learning work?](#) In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11048–11064, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneweld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, and 21 others. 2025. [2 OLMo 2 Furious](#). *Preprint*, arXiv:2501.00656.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. [Training language models to follow instructions with human feedback](#). In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.

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, and 25 others.

2025. *Qwen2.5 Technical Report. Preprint*, arXiv:2412.15115.

Kiran Ramnath, Kang Zhou, Sheng Guan, Soumya Smruti Mishra, Xuan Qi, Zhengyuan Shen, Shuai Wang, Sangmin Woo, Sullam Jeoung, Yawei Wang, Haozhu Wang, Han Ding, Yuzhe Lu, Zhichao Xu, Yun Zhou, Balasubramaniam Srinivasan, Qiaojing Yan, Yueyan Chen, Haibo Ding, and 2 others. 2025. *A Systematic Survey of Automatic Prompt Optimization Techniques. Preprint*, arXiv:2502.16923.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. *GPQA: A Graduate-Level Google-Proof Q&A Benchmark. In First Conference on Language Modeling*.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémie Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, and 7 others. 2024. *Code Llama: Open Foundation Models for Code. Preprint*, arXiv:2308.12950.

Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2025. *A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. Preprint*, arXiv:2402.07927.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. *Proximal Policy Optimization Algorithms. Preprint*, arXiv:1707.06347.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2024. *Quantifying Language Models’ Sensitivity to Spurious Features in Prompt Design or: How I learned to start worrying about prompt formatting. In The Twelfth International Conference on Learning Representations*.

ByteDance Seed, :, Jiaze Chen, Tiantian Fan, Xin Liu, Lingjun Liu, Zhiqi Lin, Mingxuan Wang, Chengyi Wang, Xiangpeng Wei, Wenyuan Xu, Yufeng Yuan, Yu Yue, Lin Yan, Qiying Yu, Xiaochen Zuo, Chi Zhang, Ruofei Zhu, Zhecheng An, and 255 others. 2025. *Seed-Thinking-v1.5: Advancing Superb Reasoning Models with Reinforcement Learning. Preprint*, arXiv:2504.13914.

Murray Shanahan, Kyle McDonell, and Laria Reynolds. 2023. *Role play with large language models. Nature*, 623(7987):493–498.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. Preprint*, arXiv:2402.03300.

Yi Su, Dian Yu, Linfeng Song, Juntao Li, Haitao Mi, Zhaopeng Tu, Min Zhang, and Dong Yu. 2025. *Crossing the Reward Bridge: Expanding RL with Verifiable Rewards Across Diverse Domains. Preprint*, arXiv:2503.23829.

Xiangru Tang, Yiming Zong, Jason Phang, Yilun Zhao, Wangchunshu Zhou, Arman Cohan, and Mark Gerstein. 2024. *Struc-bench: Are large language models good at generating complex structured tabular data? In Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pages 12–34, Mexico City, Mexico. Association for Computational Linguistics.

Pittawat Taveekitworachai, Febri Abdullah, and Ruck Thawonmas. 2024. *Null-shot prompting: Rethinking prompting large language models with hallucination. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 13321–13361, Miami, Florida, USA. Association for Computational Linguistics.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. *Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634, Toronto, Canada. Association for Computational Linguistics.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022a. *Emergent Abilities of Large Language Models. Transactions on Machine Learning Research*. Survey Certification.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022b. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. 2025. *SWE-RL: Advancing LLM Reasoning via Reinforcement Learning on Open Software Evolution. Preprint*, arXiv:2502.18449.

Tian Xie, Zitian Gao, Qingnan Ren, Haoming Luo, Yuqian Hong, Bryan Dai, Joey Zhou, Kai Qiu, Zhirong Wu, and Chong Luo. 2025. *Logic-RL: Unleashing LLM Reasoning with Rule-Based Reinforcement Learning. Preprint*, arXiv:2502.14768.

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. 2025. *Chain of Draft: Thinking Faster by Writing Less. Preprint*, arXiv:2502.18600.

Edward Yeo, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. 2025. *Demystifying Long Chain-of-Thought Reasoning in LLMs*. *Preprint*, arXiv:2502.03373.

Qiyi Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, and 16 others. 2025. *DAPO: An Open-Source LLM Reinforcement Learning System at Scale*. *Preprint*, arXiv:2503.14476.

Yu Yue, Yufeng Yuan, Qiyi Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyu Du, Xiangpeng Wei, Xiangyu Yu, Gaohong Liu, Juncai Liu, Lingjun Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Chi Zhang, and 8 others. 2025. *VAPO: Efficient and Reliable Reinforcement Learning for Advanced Reasoning Tasks*. *Preprint*, arXiv:2504.05118.

Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. 2025a. *SimpleRL-Zoo: Investigating and Taming Zero Reinforcement Learning for Open Base Models in the Wild*. *Preprint*, arXiv:2503.18892.

Wenjun Zeng, Dana Kurniawan, Ryan Mullins, Yuchi Liu, Tamoghna Saha, Dirichi Ike-Njoku, Jindong Gu, Yiwen Song, Cai Xu, Jingjing Zhou, Aparna Joshi, Shravan Dheep, Mani Malek, Hamid Palangi, Joon Baek, Rick Pereira, and Karthik Narasimhan. 2025b. *ShieldGemma 2: Robust and Tractable Image Content Moderation*. *Preprint*, arXiv:2504.01081.

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. 2025. *A Survey on Test-Time Scaling in Large Language Models: What, How, Where, and How Well?* *Preprint*, arXiv:2503.24235.

Pinlong Zhao, Weiyao Zhu, Pengfei Jiao, Di Gao, and Ou Wu. 2025. *Data Poisoning in Deep Learning: A Survey*. *Preprint*, arXiv:2503.22759.

## A Related Work

### A.1 Reinforcement Fine-Tuning (RFT)

Reinforcement learning (RL) has become a common post-training method for large language models (LLMs) (Kumar et al., 2025). One prominent RL approach is reinforcement learning from human feedback (RLHF) (Ouyang et al., 2022), where a reward model—trained on human preference comparisons—predicts scalar scores for model outputs. This enables optimization of model behavior toward human-aligned responses, typically using Proximal Policy Optimization (PPO) (Schulman et al., 2017).

A recent shift in RL for LLM post-training is the introduction of *reinforcement learning with verifiable rewards* (RLVR), also known as *reinforcement fine-tuning* (RFT). First introduced by OLMo et al. (2025), RFT replaces the reward model with task-specific, rule-based reward functions for domains with verifiable answers such as mathematics, logic, and code. This not only improves performance but also eliminates the need to train a separate reward model and maintain it during training.

RFT gained widespread attention following the release of DeepSeek-R1-Zero (DeepSeek-AI et al., 2025), which extends the RLVR paradigm by incorporating two key modifications: (1) replacing PPO with Group Relative Policy Optimization (GRPO) (Shao et al., 2024) to eliminate the need for a separate value model, reducing compute cost; and (2) introducing a **prior prompt** to elicit reasoning behavior during training. While the former has received significant attention, the latter—prior prompt—remains largely understudied.

The core components of RFT include: (1) the RL algorithm, (2) base language model, (3) training dataset, (4) reward function, and (5) prior prompt. Recent studies have explored improvements in RL algorithms (e.g., Dr. GRPO (Liu et al., 2025), DAPO (Yu et al., 2025), VAPO (Yue et al., 2025)), base model effects (Zeng et al., 2025a; Gandhi et al., 2025), and expanding verifiable tasks such as logic (Xie et al., 2025), coding (Wei et al., 2025), and function calling (Feng et al., 2025). However, the *role* of prior prompts has received little attention. Aside from one study noting their effect on training stability (Xie et al., 2025; Hu et al., 2025), prompt design in RFT remains significantly underexplored. Given the importance of prompting in inference-time settings (iPE), we argue that pPE deserves focused study as a core axis of RFT.

### A.2 Inference-Time Prompt Engineering (iPE)

iPE has seen rapid development since the introduction of ChatGPT (Sahoo et al., 2025). iPE refers to techniques for prompting LLMs to produce desirable outcomes. A prominent direction in iPE is reasoning-centric prompting, with the seminal work being chain-of-thought (CoT) prompting (Wei et al., 2022b), which uses in-context examples to demonstrate multi-step reasoning. This was later extended by zero-shot CoT (Kojima et al., 2022) prompting, where a simple phrase like “Let’s think step by step.” is sufficient to elicit similar behavior

from capable LLMs.

Since then, many variants have emerged to elicit a range of intermediate reasoning patterns—beyond just step-by-step reasoning. These include prompting for planning (Wang et al., 2023), code generation (Chen et al., 2023), knowledge recall (Liu et al., 2022), and hallucination induction (Taveekitworachai et al., 2024).

We note a conceptual parallel between iPE and pPE: both *initially* focused on reasoning, but iPE has since broadened to include diverse useful behaviors (Sahoo et al., 2025). Motivated by this, our study extends RFT by incorporating a range of iPE-inspired prompting strategies as prior prompts. We aim to investigate whether these paradigms, when moved from inference to training time, yield corresponding behavioral changes during RFT.

## B Additional Discussions

### B.1 Domain generalization

We observe that, although only mathematical problems is used during training, performance improvements often extend to other domains—as seen in GPQA and HE+ in Tables 1 and 3. This demonstrates the robustness of the RFT approach in general and suggests that RFT may function more as a mechanism for discovering useful generation patterns than for infusing the model with new knowledge. We expect broader performance generalization to emerge with more diverse training data, such as by incorporating code or logic problems.

### B.2 pPE for RFT

As demonstrated in this study, the importance of pPE in RFT is analogous to the role of iPE for LMs. Prompts play a critical role in conditioning the base model’s generation, which in turn affects the trajectories sampled during RFT—ultimately leading to distinct post-training behaviors.

This also implies that properties known to affect LMs during inference, such as sensitivity to prompt wording (Kojima et al., 2022; Shanahan et al., 2023), formatting (Sclar et al., 2024; Tang et al., 2024), and prompt order (Taveekitworachai et al., 2024; Min et al., 2022), can similarly influence RFT outcomes. However, our results in Section 4.1 suggest that insights from iPE do not directly translate to RFT—reinforcing the need for targeted study of pPE. That said, the model’s instruction-following capabilities can still be leveraged to incentivize distinct behavioral patterns through care-

fully designed prior prompts.

### B.3 Beyond Reasoning Models

We discuss here several promising directions enabled by pPE, inspired by advances in iPE:

**Not only think, plan, code, and recall knowledge** The reasoning trace itself can be an important vehicle for interpretability and user trust (Wei et al., 2022b). Given that we can elicit distinct reasoning styles, we may tailor them to user preferences or application requirements. For instance, an LLM could reason in a self-talk style using <dialogues> tags, or imitate a specific style using few-shot demonstrations (Brown et al., 2020).

Recent work on chain-of-draft prompting (Xu et al., 2025) shows that natural language constraints can guide the model to produce shorter but still effective reasoning traces. Such behavior can likely be transferred into pPE settings, especially with models that possess strong instruction-following capabilities. The breadth of iPE research suggests many additional styles—beyond those we studied here—could be explored and reinforced via pPE.

**Dynamic pPE** As shown in Table 1, different pPE approaches excel in different domains. Dynamically selecting the prior prompt based on the task or question difficulty could further enhance performance. This idea aligns with the test-time scaling paradigm (Zhang et al., 2025), which advocates allocating more resources to harder inputs.

Furthermore, prior prompts could become part of the RL optimization process, akin to automatic prompt search (Ramnath et al., 2025). While such approaches increase system complexity, they offer a path toward more adaptive and robust behaviors.

**Structured thinking** Instead of using a single behavior tag (e.g., <think>), we may extend to multi-tag structures (e.g., combining <plan> and <code>) to guide the model through more structured multi-phase reasoning processes. This may be especially beneficial in tasks requiring distinct reasoning modes at different stages (e.g., planning followed by execution).

**Incentivizing behaviors through verifiable rewards** Consider a model trained with the <plan> prompt. During training, it learns to produce a useful plan inside the <plan> tag before solving the problem in <answer>. Because final task accuracy is used as a reward, this implicitly incentivizes the model to generate effective intermediate content.

Thus, verifiable rewards can act as a *surrogate signal* for training behaviors—such as a planning or coding—without *direct* supervised signals.

This logic extends to other prompts: if we stop generation after `</plan>` or `</code>`, we can repurpose these models to act as plan generators or code synthesizers. This strategy opens up a broader class of behavioral specialization, where useful intermediate behaviors can be extracted and repurposed for downstream applications—all trained indirectly via RFT.

## C Prior Prompt Examples

In this section, we provide additional two examples of prior prompts used in existing RFT studies to elicit reasoning behavior during RFT. These are Figures 8 and 9.

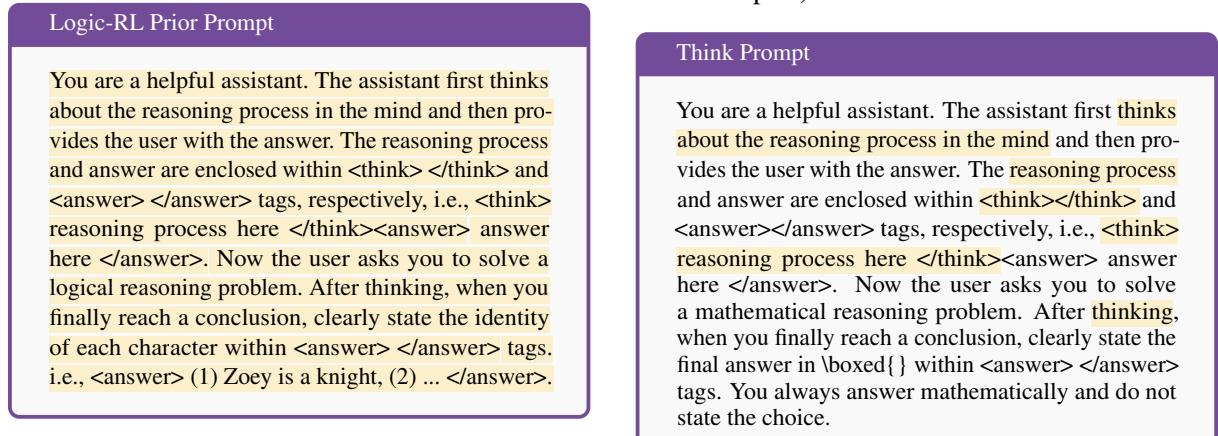


Figure 8: The prompt used during RFT by Logic-RL (Xie et al., 2025).

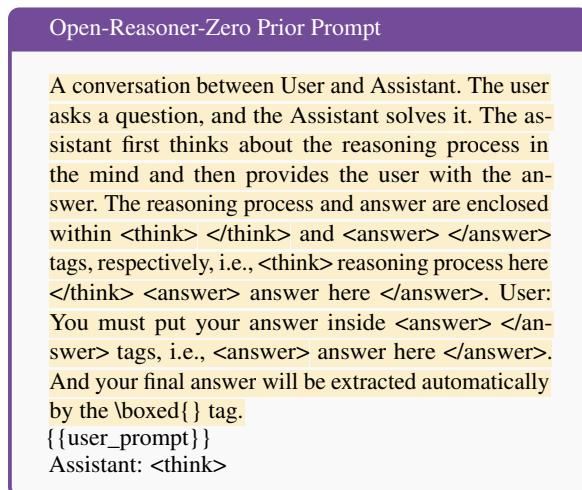


Figure 9: The prompt used during RFT by Open-Reasoner-Zero (Hu et al., 2025).

## D Additional Experimental Setup Details

This section provides additional implementation details of our experimental setup, including prior prompt templates in Appendix D.1, training scripts in Appendix D.3, reward function design in Appendix D.2, and evaluation details in Appendix D.4.

### D.1 Prior Prompts

This section presents the full set of prior prompts used in our experiments, as described in Section 3. Each prompt was designed to elicit different behavioral styles from the model. These prompts are: `<think>` (Figure 10, for step-by-step reasoning), `<plan>` (Figure 11, for planning), `<code>` (Figure 12, for reasoning through code), `<knowledge>` (Figure 13, for recalling relevant facts), and `<examples>` (Figure 14, for utilizing null-examples).

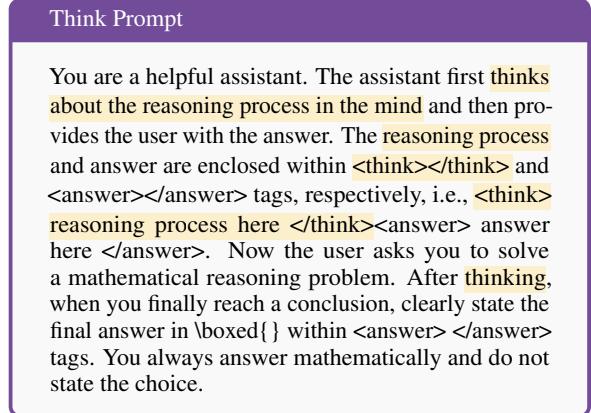


Figure 10: The `<think>` prior prompt, inspired by chain-of-thought (CoT) prompting (Wei et al., 2022b), encourages the model to reason step by step before concluding with an answer.

### D.2 Reward Design

We design our reward function with two equally weighted components: (1) an **accuracy reward** and (2) a **format reward**. This setup follows the approach introduced by Xie et al. (2025). While some studies suggest the format reward may not be necessary (Zeng et al., 2025a), we find that it is crucial in our setting to ensure the model outputs are well-structured.

- **Accuracy:** The accuracy reward is based on the model’s predicted answer, extracted from the content enclosed in `\boxed{}`. We use the `math-verify`<sup>1</sup> package (Apache License 2.0)

<sup>1</sup><https://github.com/huggingface/Math-Verify>

#### Plan Prompt

You are a helpful assistant. The assistant first plans about the reasoning process in the mind and then provides the user with the answer. The plan and answer are enclosed within `<plan></plan>` and `<answer></answer>` tags, respectively, i.e., `<plan>` detailed plan here `</plan><answer>` answer here `</answer>`. Now the user asks you to solve a mathematical reasoning problem. After planning, when you finally reach a conclusion, clearly state the final answer in `\boxed{}` within `<answer></answer>` tags. You always answer mathematically and do not state the choice.

Figure 11: The `<plan>` prior prompt, based on plan-and-solve prompting (Wang et al., 2023), asks the model to explicitly lay out a plan before solving the problem.

#### Code Prompt

You are a helpful assistant. The assistant first writes required code used to solve the problem and then provides the user with the answer. The code and answer are enclosed within `<code></code>` and `<answer></answer>` tags, respectively, i.e., `<code>` detailed code here `</code><answer>` answer here `</answer>`. Now the user asks you to solve a mathematical reasoning problem. After coding, when you finally reach a conclusion, clearly state the final answer in `\boxed{}` within `<answer></answer>` tags. You always answer mathematically and do not state the choice.

Figure 12: The `<code>` prior prompt encourages the model to reason through code, inspired by program-of-thought (PoT) prompting (Chen et al., 2023).

#### Knowledge Prompt

You are a helpful assistant. The assistant first recalls relevant knowledge used to solve the problem and then provides the user with the answer. The knowledge and answer are enclosed within `<knowledge></knowledge>` and `<answer></answer>` tags, respectively, i.e., `<knowledge>` comprehensive knowledge here `</knowledge><answer>` answer here `</answer>`. Now the user asks you to solve a mathematical reasoning problem. After recalling knowledge, when you finally reach a conclusion, clearly state the final answer in `\boxed{}` within `<answer></answer>` tags. You always answer mathematically and do not state the choice.

Figure 13: The `<knowledge>` prior prompt elicits factual recall relevant to the problem before beginning reasoning, inspired by generated knowledge prompting (Liu et al., 2022).

#### Examples Prompt

You are a helpful assistant. The assistant first lists relevant examples used to solve the problem and then provides the user with the answer. The examples and answer are enclosed within `<examples></examples>` and `<answer></answer>` tags, respectively, i.e., `<examples>` relevant examples here `</examples><answer>` answer here `</answer>`. Now the user asks you to solve a mathematical reasoning problem. After listing examples, when you finally reach a conclusion, clearly state the final answer in `\boxed{}` within `<answer></answer>` tags. You always answer mathematically and do not state the choice.

Figure 14: The `<examples>` prior prompt draws on null-shot prompting (Taveekitworachai et al., 2024) to encourage the model to provide illustrative examples before answering.

to check for mathematical equivalence with the ground-truth answer. If the answer is equivalent, the model receives a reward of 0.5; otherwise, it receives 0.0.

- **Format:** We adopt a relaxed version of the format reward from the open-r1<sup>2</sup> (Apache License 2.0) implementation. The reward is given if the response includes *exactly one pair* of the expected XML tags (e.g., `<think>...</think>` followed by `<answer>...</answer>`), and the content satisfies basic XML structure constraints, even if the tags are not the only elements in the string. This constraint discourages generation of multiple or malformed tag pairs. If the response satisfies these constraints, a reward of 0.5 is granted; otherwise, it receives 0.0.

The total reward is the sum of these two components, yielding a final reward in the range [0, 1]. This balanced reward design helps incentivize both correct and well-structured responses during RFT.

### D.3 Training Setup and Hyperparameters

We use the training script illustrated in Figure 15 for training the models as described in Section 3.2. All training runs use a single node equipped with 8xH100 GPUs. Across all experiments presented in this paper, we utilized a total of 78 GPU-hours.

We note that both OpenRLHF and vLLM are available under the Apache License 2.0. The Qwen2.5-7B model used in our main experiments is distributed under the Apache License 2.0. For our generalization studies, the Qwen2.5-3B model

<sup>2</sup><https://github.com/huggingface/open-r1>

is distributed under the Qwen Research License, the Llama 3.1-8B model under the Llama 3.1 Community License Agreement, and the Qwen2.5-Coder-7B model under the Apache License 2.0. All of these licenses permit use for research purposes.

```
OpenRLHF Training Script

python3 -m openrlhf.cli.train_ppo_ray \
--ref_num_nodes 1 \
--ref_num_gpus_per_node 8 \
--actor_num_nodes 1 \
--actor_num_gpus_per_node 8 \
--vllm_num_engines 8 \
--vllm_tensor_parallel_size 1 \
--colocate_all_models \
--vllm_enable_sleep \
--vllm_gpu_memory_utilization 0.5 \
--pretrain "Qwen/Qwen2.5-7B" \
--remote_rm_url "reward_function.py" \
--micro_train_batch_size 1 \
--train_batch_size 64 \
--micro_rollout_batch_size 8 \
--rollout_batch_size 64 \
--n_samples_per_prompt 8 \
--enable_prefix_caching \
--max_epochs 1 \
--prompt_max_len 1024 \
--max_samples 10000 \
--generate_max_len 4096 \
--init_kl_coef 1e-6 \
--gamma 1.0 \
--use_kl_loss \
--kl_estimator k3 \
--advantage_estimator group_norm \
--zero_stage 2 \
--bf16 \
--actor_learning_rate 5e-7 \
--prompt_data "user/stillv3" \
--prompt_split "train" \
--input_key "query" \
--label_key "answer" \
--apply_chat_template \
--normalize_reward \
--adam_offload \
--gradient_checkpointing \
--flash_attn \
--packing_samples
```

Figure 15: Training script using the OpenRLHF for RFT. This script specifies the model, dataset, GRPO algorithm, reward configuration, and other relevant hyperparameters.

## D.4 Evaluation

In this section, we provide additional details on quantitative and qualitative evaluation, mentioned in Section 3.3.

### D.4.1 Quantitative Analysis

We evaluate the performance of each trained model using both in- and out-of-domain benchmarks. Dur-

ing evaluation, we consistently prepend the same prior prompt used during training to elicit the trained behaviors. We evaluate once with fixed random seed using pass@1 accuracy. The benchmarks are:

- **Mathematical reasoning:** AIME24 (Li et al., 2024), AMC12 ’22–’23 (Li et al., 2024), and MATH-500 (Hendrycks et al., 2021) are benchmarks used for evaluating mathematical reasoning and serve as our primary in-domain evaluations.
- **Coding:** HumanEval+ (base and extra) (Liu et al., 2023) is used to evaluate general coding ability and serves as an out-of-domain probe.
- **Knowledge-based question answering:** GPQA-Diamond (Rein et al., 2024) evaluates factual knowledge and complex reasoning. We include it to assess whether math-centric training with different prior prompts can elicit behaviors associated with knowledge recall. While this ability may appear unrelated to solving math problems, it can be useful for recalling definitions, theorems, or formulas relevant to a given problem.

Additional metadata of these evaluation benchmarks, along with our training set, is available in Table 4.

### D.4.2 Qualitative Analysis

To investigate whether different pPE approaches lead to distinct behavioral patterns after RFT, we assess the following aspects:

**Training dynamics and response length** We analyze whether different pPE approaches result in distinct training dynamics across models. In addition, we compute the average number of tokens in generated responses.

**Four fundamental cognitive behaviors of reasoning models** Gandhi et al. (2025) identify four fundamental cognitive behaviors commonly exhibited by reasoning models. These behaviors are considered core components of what makes a model capable of reasoning: (1) **Verification**: Identifying errors in intermediate results, (2) **Backtracking**: Abandoning the current approach and trying alternatives, (3) **Subgoal setting**: Breaking problems down into smaller, more manageable steps, and (4) **Backward chaining**: Reasoning backward from the expected answer to the given inputs.

Dataset	Task	Split	Count	Answer Type	License
<b>Training dataset</b>					
STILLv3 (Chen et al., 2025)	Math	Train	29925	N/A	N/A
<b>Evaluation benchmark</b>					
AIME24 (Li et al., 2024)	Math	Test	30	Number	N/A
AMC12 '22-'23 (Li et al., 2024)	Math	Test	83	Number	N/A
MATH-500 (Hendrycks et al., 2021)	Math	Test	500	Number	MIT License
HumanEval+ (Liu et al., 2023)	Code	Test	164	Code	Apache 2.0
GPQA-Diamond (Rein et al., 2024)	QA	Test	198	MC	CC BY 4.0

Table 4: Overview of the training dataset and evaluation benchmarks. We note that all datasets are available for the purposes used in this study.

Following their methodology, we use an LLM-based classifier to detect the presence of each behavior in model outputs. While the original study used gpt-4o-mini, we employ a more recent model, gpt-4.1-mini-2025-04-14, for classification. Our goal is to compare the distribution of these behaviors across models trained or prompted using different iPE/pPE strategies. Prompts used for classification are provided in Figures 16 to 19.

**Verifications Classification Prompt**

You are a helpful assistant that analyzes mathematical reasoning.

Here is a response that a language model generated while trying to solve a the following problem the goal of accurately answer the problem.

```
# Problem
{problem}
```

The response the model used is the following:

```
# Response
{completion}
```

Evaluate whether the response contains any answer-verification steps. An example of an answer-verification step is: 'This sequence results in 1, which is not equal to 22' and 'Since 25 is not equal to 22' for explicit verification and 'Too high!' or 'This works!' for implicit verification. We want to mark instances where the response explicitly checks the current result.

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

Figure 16: Prompt used to identify verification behavior—explicit checking or validation of intermediate results—in the model’s reasoning.

**Five pPE-specific behaviors** We further adapt the same classification approach to evaluate

**Backtracking Classification Prompt**

You are a helpful assistant that analyzes mathematical reasoning.

Here is a response that a language model generated while trying to solve a the following problem the goal of accurately answer the problem.

```
# Problem
{problem}
```

The response the model used is the following:

```
# Response
{completion}
```

Evaluate whether the response contains any backtracking behavior, where the model realizes a path won’t work and explicitly goes back to try a different approach.

Count the number of distinct backtracking instances and provide the count between the tags <count> </count>. If the response does not contain any backtracking behavior, please provide a count of 0 as <count>0</count>.

Figure 17: Classification prompt used to detect instances of backtracking—when a model revises or abandons a previous approach—in its reasoning trace.

**Subgoal Settings Classification Prompt**

You are a helpful assistant that analyzes mathematical reasoning.

Here is a response that a language model generated while trying to solve a the following problem the goal of accurately answer the problem.

```
# Problem
{problem}
```

The response the model used is the following:

```
# Response
{completion}
```

Evaluate whether the response contains any backward-chaining behavior, where the model starts from the target and works backwards to the initial problem. An example of backward-chaining when the target is 24 and the numbers are 12 and 2 is: "Let's work backwards from the target.  $24/2 = 12$ . So,  $12*2=24$ ." and if the target is 22 and the numbers are 25 and 3 is: "Since the target is 22, and  $22 + 3 = 25$ , ...".

Count the number of distinct backward-chaining instances and provide the count between the tags `<count> </count>`. If the response does not contain any backward-chaining behavior, please provide a count of 0 as `<count>0</count>`.

Figure 18: Prompt used to detect subgoal setting behavior, where the model breaks a problem into smaller, intermediate steps.

**Backward Chaining Classification Prompt**

You are a helpful assistant that analyzes mathematical reasoning.

Here is a response that a language model generated while trying to solve a the following problem the goal of accurately answer the problem.

```
# Problem
{problem}
```

The response the model used is the following:

```
# Response
{completion}
```

Evaluate whether the response contains any explicit subgoal setting, where the model breaks down the problem into smaller, intermediate goals. An example of subgoal setting is: "First, I'll try to get close to target/2, then...".

Count the number of distinct subgoals set and provide the count between the tags `<count> </count>`. If the response does not contain any subgoal setting, please provide a count of 0 as `<count>0</count>`.

Figure 19: Classification prompt used to identify backward chaining—reasoning from the goal back to known facts—within a model’s response.

whether the target behavior elicited by each pPE (e.g., reasoning, planning, coding, knowledge recall, or example generation) is present in model responses. This is treated as a binary classification task, assessing the presence or absence of the expected behavior per response. Classification prompts are provided in Figures 20 to 24.

For all analyses, we exclude HumanEval+ due to missing raw responses from the evaluation program. We also omit a very small number of responses (less than 0.05% of the total) due to response parsing errors from gpt-4.1-mini-2025-04-14.

## E Additional Results

This section presents additional results from the generalization studies discussed in Section 4.3. Training dynamics for the three language models are provided in Appendix E.1. Results for the classification of the four fundamental behaviors and the five elicited behavior categories are available in Appendix E.2 and Appendix E.3, respectively. In addition, we provide an alternative visualization of performance across benchmarks from the main experiments on Qwen2.5-7B, showing changes over the baseline in Table 5.

### Think Classification Prompt

You are a helpful assistant that analyzes mathematical reasoning.

Here is a response that a language model generated while trying to solve a the following problem the goal of accurately answer the problem.

```
# Problem  
{problem}
```

The response the model used is the following:

```
# Response  
{completion}
```

Evaluate whether the response contains internal step-by-step logical reasoning to reach a conclusion. Look for clear sequences of deductive or mathematical steps that reflect the assistant "thinking through" the problem logically.

If you find any reasoning steps, please return `<result>YES</result>`. If the response does not contain any reasoning steps, please return `<result>NO</result>`.

Figure 20: Prompt used to classify whether the model is exhibiting reasoning aligned with the `<think>` prompt (step-by-step logical reasoning).

### Code Classification Prompt

You are a helpful assistant that analyzes mathematical reasoning.

Here is a response that a language model generated while trying to solve a the following problem the goal of accurately answer the problem.

```
# Problem  
{problem}
```

The response the model used is the following:

```
# Response  
{completion}
```

Evaluate whether the response includes written code (in any programming language) used to compute or check the solution. Code may be accompanied by brief comments or used directly to reason.

If you find any code implementation, please return `<result>YES</result>`. If the response does not contain any code implementation, please return `<result>NO</result>`.

Figure 22: Prompt used to determine whether code-based reasoning patterns, encouraged by the `<code>` prompt, are present in the model output.

### Plan Classification Prompt

You are a helpful assistant that analyzes mathematical reasoning.

Here is a response that a language model generated while trying to solve a the following problem the goal of accurately answer the problem.

```
# Problem  
{problem}
```

The response the model used is the following:

```
# Response  
{completion}
```

Evaluate whether the response contains a structured plan or approach for solving the problem before executing calculations. Look for high-level steps, strategies, or intentions stated clearly (e.g., "First I will factor this... then I will check for...", "1. Conduct a hypothesis testing.").

If you find any solution planning, please return `<result>YES</result>`. If the response does not contain any solution planning, please return `<result>NO</result>`.

Figure 21: Prompt used to identify whether the model is engaging in explicit planning, consistent with the `<plan>` prompting style.

### Knowledge Classification Prompt

You are a helpful assistant that analyzes mathematical reasoning.

Here is a response that a language model generated while trying to solve a the following problem the goal of accurately answer the problem.

```
# Problem  
{problem}
```

The response the model used is the following:

```
# Response  
{completion}
```

Evaluate whether the response recalls relevant facts, theorems, or concepts before solving the problem. This includes definitions, properties, or known formulas used to justify or support the solution path.

If you find any background knowledge, please return `<result>YES</result>`. If the response does not contain any background knowledge, please return `<result>NO</result>`.

Figure 23: Prompt used to detect knowledge-recall behavior, as in `<knowledge>` approach.

## E.1 Training Dynamics and Average Response Length

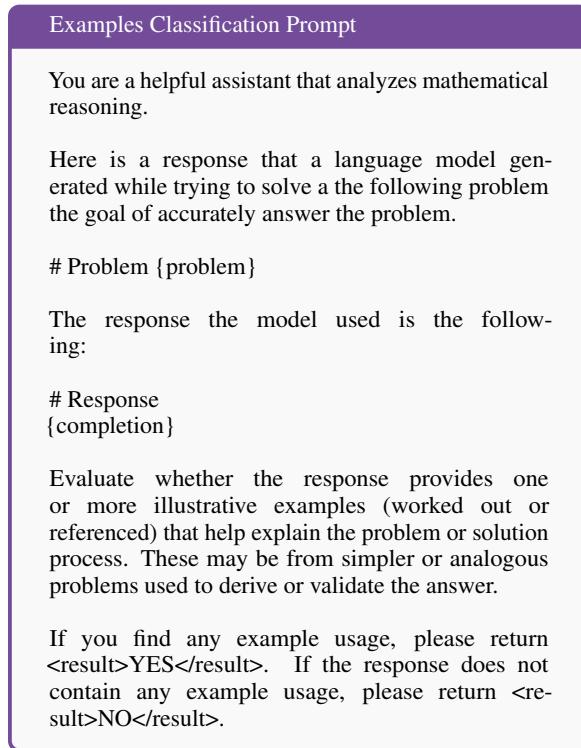


Figure 24: Prompt used to classify whether the model is generating illustrative examples, as intended by the <examples> prompting approach.

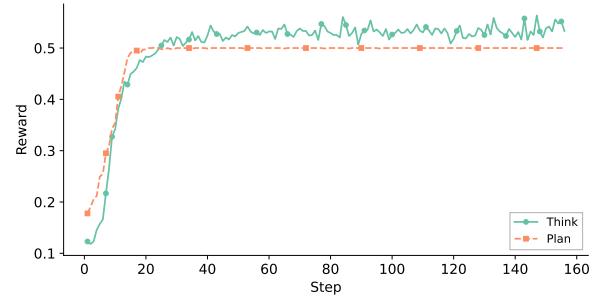


Figure 25: Reward progression for Qwen2.5-3B during RFT.

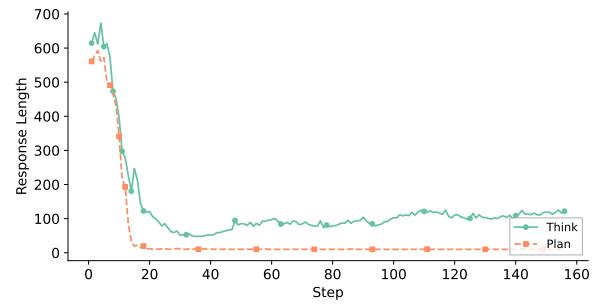


Figure 26: Evolution of the average response length for Qwen2.5-3B during RFT.

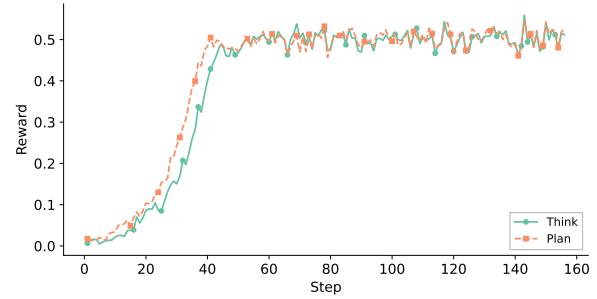


Figure 27: Reward progression for Llama 3.1-8B during RFT.

Figures 25 to 30 present the reward and response-length dynamics during RFT for Qwen2.5-3B, Llama 3.1-8B, and Qwen2.5-Coder-7B, respectively.

Finally, Table 6 reports the average response length, i.e., average number of tokens in responses for the generalization experiments.

## E.2 Four Fundamental Cognitive Behaviors

In this subsection, we present the results of four fundamental cognitive behavior classifications from the generalization studies for Qwen2.5-3B, Llama 3.1-8B, and Qwen2.5-Coder-7B, shown in Figures 31 to 33, respectively.

Table 5: Absolute change in accuracy (green = gain, red = drop) relative to the zero-shot Qwen2.5-7B base model.

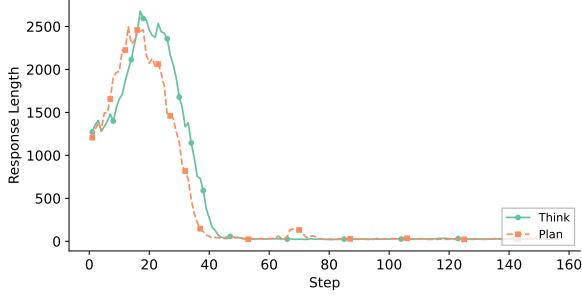


Figure 28: Evolution of the average response length for Llama 3.1-8B during RFT.

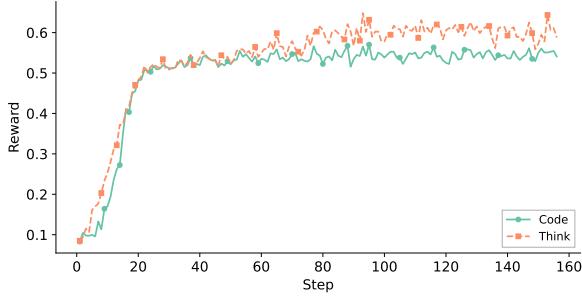


Figure 29: Reward progression for Qwen2.5-Coder-7B during RFT.

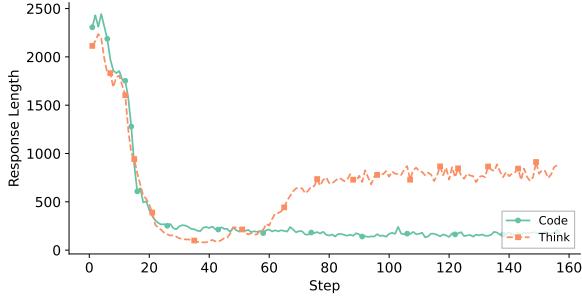


Figure 30: Evolution of the average response length for Qwen2.5-Coder-7B during RFT.

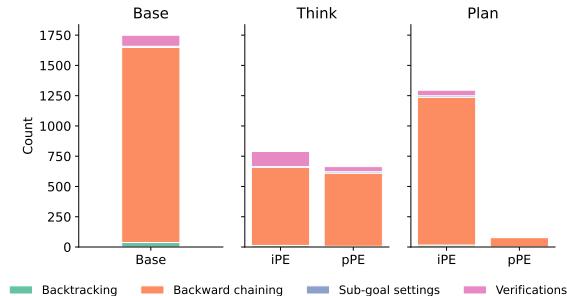


Figure 31: Ratio of the four fundamental cognitive behaviors—backtracking, backward chaining, subgoal setting, and verification—across different prompting (iPE) and RFT (pPE) approaches with Qwen2.5-3B.

Model	AIME	AMC	GPQA	MATH	Avg.
<b>Qwen2.5 3B</b>	1249.00	1297.70	455.85	981.93	996.12
<b>iPE</b>					
Think	1764.63	1156.81	455.85	754.45	1032.94
Plan	1349.13	1270.47	455.85	631.23	926.67
<b>pPE</b>					
Think	124.37	492.70	412.17	182.44	302.92
Plan	9.00	9.00	274.95	9.00	75.49
<b>Llama 3.1-8B</b>	8.03	11.25	N/A	128.93	49.40
<b>iPE</b>					
Think	6122.33	4347.72	N/A	4177.73	4882.59
Plan	4562.97	5150.96	N/A	4452.76	4722.23
<b>pPE</b>					
Think	22.70	23.23	N/A	25.84	23.92
Plan	21.10	20.37	N/A	53.94	31.80
<b>Qwen2.5-Coder-7B</b>	3856.83	2947.86	1510.95	3570.15	2971.45
<b>iPE</b>					
Think	2601.03	748.31	1510.95	917.51	1444.45
Code	779.67	337.59	1510.95	265.43	723.41
<b>pPE</b>					
Think	1850.67	1045.12	592.05	758.67	1061.63
Code	164.72	254.41	311.49	152.67	220.82

Table 6: Average response length, i.e., number of tokens, of Qwen2.5-3B, Llama 3.1-8B, and Qwen2.5-Coder-7B when prompted with different iPE or RFT with different pPE approaches across four benchmarks.

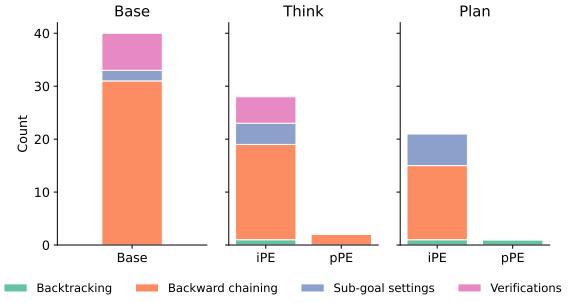


Figure 32: Ratio of the four fundamental cognitive behaviors—backtracking, backward chaining, subgoal setting, and verification—across different prompting (iPE) and RFT (pPE) approaches with Llama 3.1-8B.

In addition, we provide detailed tables of the exact behavior counts for the main experiment in Table 7, previously visualized in Figure 6, and for the generalization studies in Table 8, previously visualized in Figures 31 to 33.

### E.3 Five Elicited Behaviors

In this subsection, we present the results of five elicited cognitive behavior classifications from the generalization studies for Qwen2.5-3B, Llama 3.1-8B, and Qwen2.5-Coder-7B, shown in Figures 34 to 36, respectively.

In addition, we provide detailed tables of the exact behavior counts for the main experiment in Table 9, previously visualized in Figure 7, and for the generalization studies in Table 10, previously visualized in Figures 34 to 36.

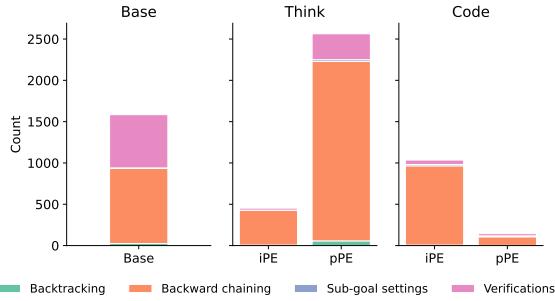


Figure 33: Ratio of the four fundamental cognitive behaviors—backtracking, backward chaining, subgoal setting, and verification—across different prompting (iPE) and RFT (pPE) approaches with Qwen2.5-Coder-7B.

Model	Backtrack.	Back.Chain.	Subgoal.Set.	Veri.
<b>Qwen2.5-7B</b>	30	1707	9	195
<b>iPE</b>				
Think	31	1542	15	201
Plan	60	2202	6	156
Code	66	2394	31	223
Knowledge	79	2794	<b>56</b>	274
Examples	<b>114</b>	<b>3159</b>	30	<b>453</b>
<b>RFT</b>				
No PP	70	2195	23	336
<b>pPE</b>				
Think	27	1354	16	150
Plan	32	2450	11	110
Code	42	1628	19	262
Knowledge	63	1180	14	148
Examples	46	1545	8	98

Table 7: Occurrence counts of the four fundamental cognitive behaviors—backtracking (Backtrack.), backward chaining (Back.Chain.), subgoal settings (Subgoal.Set.), and verifications (Veri.)—in Qwen2.5-7B’s responses under different iPE and pPE approaches. Counts are obtained via the LM-based classification framework of Gandhi et al. (2025). **Bold** marks the highest count and underlined marks the lowest count per column.

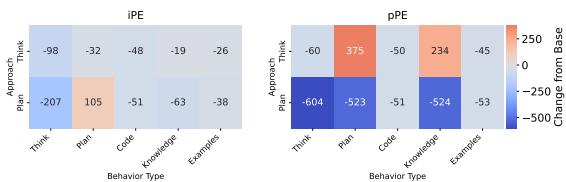


Figure 34: Behavior alignment heatmaps for Qwen2.5-3B: iPE on the left, RFT on the right.

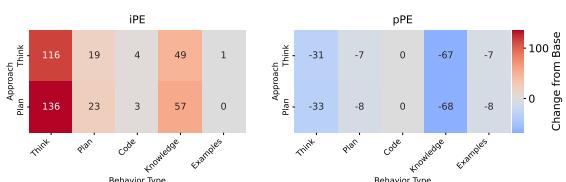


Figure 35: Behavior activation patterns for Llama 3.1-8B under iPE (left) and RFT (right).

Model	Backtrack.	Back.Chain.	Subgoal.Set.	Veri.
<b>Qwen2.5-3B</b>	<b>37</b>	<b>1611</b>	10	91
<b>iPE</b>				
Think	13	646	5	<b>125</b>
Plan	16	1220	<b>13</b>	46
<b>pPE</b>				
Think	6	604	12	42
Plan	0	78	0	6
<b>Llama 3.1-8B</b>	0	<b>31</b>	2	<b>7</b>
<b>iPE</b>				
Think	<b>1</b>	18	4	5
Plan	<b>1</b>	14	<b>6</b>	0
<b>pPE</b>				
Think	0	2	0	0
Plan	<b>1</b>	0	0	0
<b>Qwen2.5-Coder-7B</b>	25	908	7	<b>644</b>
<b>iPE</b>				
Think	6	420	<b>3</b>	<b>24</b>
Code	6	959	17	53
<b>pPE</b>				
Think	<b>57</b>	<b>2172</b>	<b>21</b>	313
Code	<b>2</b>	<b>104</b>	12	26

Table 8: Occurrence counts of the four fundamental cognitive behaviors—backtracking (Backtrack.), backward chaining (Back.Chain.), subgoal settings (Subgoal.Set.), and verifications (Veri.)—in Qwen2.5-3B, Llama 3.1-8B, and Qwen2.5-Coder-7B responses under different iPE and pPE approaches. Counts are obtained via the LM-based classification framework of Gandhi et al. (2025). **Bold** marks the highest count and underlined marks the lowest count per column under the same base model.

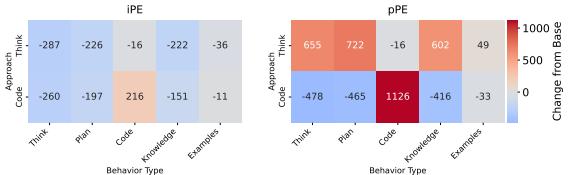


Figure 36: Behavior heatmaps for Qwen2.5-Coder 7B: iPE (left) vs. RFT (right).

Model	Code	Examples	Knowledge	Plan	Think
<b>Qwen2.5-7B</b>	193	77	626	652	711
<b>iPE</b>					
Think	31	55	635	<b>622</b>	716
Plan	38	<b>51</b>	<b>616</b>	<b>733</b>	659
Code	<b>704</b>	60	1002	1137	1130
Knowledge	88	112	<b>1168</b>	<b>1155</b>	1289
Examples	68	<b>159</b>	1112	1136	<b>1304</b>
<b>RFT</b>					
No PP	16	75	708	716	780
<b>pPE</b>					
Think	34	68	662	741	764
Plan	23	57	658	759	757
Code	602	58	640	688	739
Knowledge	14	61	720	722	760
Examples	21	57	632	704	746

Table 9: Occurrence counts of the five elicited behaviors—code, examples, knowledge, plan, and think—in Qwen2.5-7B’s responses under different iPE and pPE approaches. **Bold** marks the highest count and underlined marks the lowest count per column.

Model	Code	Examples	Knowledge	Plan	Think
<b>Qwen2.5-3B</b>	<b>51</b>	<b>53</b>	<b>554</b>	<b>545</b>	<b>630</b>
<b>iPE</b>					
Think	<u>3</u>	<u>27</u>	<u>535</u>	<u>513</u>	<u>532</u>
Plan	<u>0</u>	<u>15</u>	<u>491</u>	<u>650</u>	<u>423</u>
<b>pPE</b>					
Think	1	8	<b>788</b>	<b>920</b>	570
Plan	0	0	30	22	26
<b>Llama 3.1-8B</b>	0	<u>8</u>	71	9	36
<b>iPE</b>					
Think	<b>4</b>	<b>9</b>	<u>120</u>	<u>28</u>	<u>152</u>
Plan	<u>3</u>	<u>8</u>	<u>128</u>	<u>32</u>	<u>172</u>
<b>pPE</b>					
Think	0	1	4	2	5
Plan	0	0	3	1	3
<b>Qwen2.5-Coder-7B</b>	16	<u>54</u>	<u>577</u>	<u>525</u>	<u>669</u>
<b>iPE</b>					
Think	0	18	355	299	382
Code	<u>232</u>	<u>43</u>	<u>426</u>	<u>328</u>	<u>409</u>
<b>pPE</b>					
Think	0	103	<b>1179</b>	<b>1247</b>	<b>1324</b>
Code	<b>1142</b>	<u>21</u>	<u>161</u>	<u>60</u>	<u>191</u>

Table 10: Occurrence counts of the five elicited behaviors—code, examples, knowledge, plan, and think—in Qwen2.5-3B, Llama 3.1-8B, and Qwen2.5-Coder-7B responses under different iPE and pPE approaches. **Bold** indicates the best performance per column under the same base model; underlined indicates the second best per column under the same base model.

## F Qualitative Examples

To illustrate the behavioral differences induced by each pPE approach, we present qualitative outputs from models trained with different prior prompts in response to a shared math problem. Examples of model responses for each pPE approach are shown in Figures 38 to 42, using the shared prompt in Figure 37.

### Example Prompt

How many positive whole-number divisors does 196 have?

Figure 37: The shared prompt used to probe all models in this qualitative comparison. The question requires the model to determine the number of positive whole-number divisors of 196.

## G Declaration of AI Assistance

We utilized ChatGPT for grammatical checking and LaTeX support of the content presented in this study but did not use it for the initial draft of this study. Cursor was utilized for trivial and boilerplate code completion during data analysis. We declare that all content presented and code utilized in this study has been reviewed and edited by the authors.

### Qwen2.5 7B RFT with Think

<think>

To determine the number of positive whole-number divisors of 196, we first need to find its prime factorization.

1. Start by dividing 196 by the smallest prime number, which is 2:

$$196 \div 2 = 98$$

So,  $196 = 2 \times 98$ .

2. Next, divide 98 by 2:

$$98 \div 2 = 49$$

So,  $196 = 2 \times 2 \times 49$ .

3. Now, 49 is not divisible by 2, so we move to the next smallest prime number, which is 3. However, 49 is not divisible by 3 either. The next prime number is 5, and 49 is not divisible by 5. The next prime number is 7:

$$49 \div 7 = 7$$

So,  $196 = 2 \times 2 \times 7 \times 7$ .

4. We can write this in exponential form:

$$196 = 2^2 \times 7^2$$

To find the number of divisors, we use the formula for the number of divisors based on the prime factorization. If a number  $n$  has a prime factorization of  $p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_k^{e_k}$ , then the number of divisors  $D(n)$  is given by:

$$D(n) = (e_1 + 1)(e_2 + 1) \cdots (e_k + 1)$$

For 196, the exponents in the prime factorization are 2 and 2. Therefore, the number of divisors is:

$$D(196) = (2 + 1)(2 + 1) = 3 \times 3 = 9$$

So, the number of positive whole-number divisors of 196 is 9.

</think>

<answer>

9

</answer>

Figure 38: Response from Qwen2.5 7B trained with the <think> prior prompt. The model demonstrates step-by-step reasoning throughout its solution.

### Qwen2.5 7B RFT with Plan

<plan>

1. Identify the prime factorization of 196.
2. Use the formula for finding the number of divisors from the prime factorization.

</plan>

<answer>

To find the number of positive whole-number divisors of 196, we first need to determine its prime factorization.

1. Prime factorization of 196:

- 196 is even, so it is divisible by 2:  $196 \div 2 = 98$ .
- 98 is also even, so it is divisible by 2:  $98 \div 2 = 49$ .
- 49 is not even, but it is divisible by 7:  $49 \div 7 = 7$ .
- 7 is a prime number.

So, the prime factorization of 196 is  $2^2 \times 7^2$ .

2. Using the formula for finding the number of divisors from the prime factorization:

If a number  $n$  has a prime factorization of  $p_1^{e_1} \times p_2^{e_2} \times \dots \times p_k^{e_k}$ , then the number of divisors of  $n$  is given by  $(e_1 + 1)(e_2 + 1) \dots (e_k + 1)$ .

For 196, the exponents in the prime factorization are 2 and 2. Therefore, the number of divisors is  $(2 + 1)(2 + 1) = 3 \times 3 = 9$ .

Thus, the number of positive whole-number divisors of 196 is 9.

</answer>

Figure 39: Response from Qwen2.5 7B trained with the <plan> prior prompt. The model outlines a structured plan before proceeding to execution.

### Qwen2.5 7B RFT with Knowledge

<knowledge>

To determine the number of positive whole-number divisors of 196, we first need to find its prime factorization. The prime factorization of a number is the product of prime numbers that equals the original number.

Let's find the prime factorization of 196:

1. 196 is even, so it is divisible by 2:

$$196 \div 2 = 98$$

So, we have  $196 = 2 \times 98$ .

2. 98 is also even, so it is divisible by 2:

$$98 \div 2 = 49$$

So, we have  $196 = 2 \times 2 \times 49$ .

3. 49 is not even, but it is divisible by 7:

$$49 \div 7 = 7$$

So, we have  $196 = 2 \times 2 \times 7 \times 7$ .

4. 7 is a prime number, so we stop here.

The prime factorization of 196 is  $2^2 \times 7^2$ .

To find the number of positive divisors, we use the formula that involves adding 1 to each of the exponents in the prime factorization and then multiplying the results:

$$(2 + 1)(2 + 1) = 3 \times 3 = 9$$

Therefore, 196 has 9 positive whole-number divisors.

</knowledge>

<answer>

9

</answer>

### Qwen2.5 Coder 7B RFT with Code

<code>

```
divisors = []
number = 196

for i in range(1, number + 1):
    if number % i == 0:
        divisors.append(i)
```

</code>

<answer>The number 196 has 9 positive whole-number divisors.</answer>

Figure 40: Response from Qwen2.5-Coder 7B trained with the <code> prior prompt. The model uses Python code to assist in its reasoning process.

Figure 41: Response from Qwen2.5 7B trained with the <knowledge> prior prompt. The model first recalls definitions and relevant facts before solving the problem.

Qwen2.5 7B RFT with Examples

<examples>

To determine the number of positive whole-number divisors of 196, we first need to find its prime factorization.

1. Start by dividing 196 by the smallest prime number, which is 2:

$$196 \div 2 = 98$$

So, 2 is a prime factor.

2. Divide 98 by 2 again:

$$98 \div 2 = 49$$

So, 2 is a prime factor again.

3. Now, 49 is not divisible by 2, so we move to the next smallest prime number, which is 3. However, 49 is not divisible by 3 either. The next prime number is 5, but 49 is not divisible by 5. The next prime number is 7:

$$49 \div 7 = 7$$

So, 7 is a prime factor.

4. Finally, 7 is a prime number itself, so we divide it by 7:

$$7 \div 7 = 1$$

So, 7 is a prime factor again.

Thus, the prime factorization of 196 is:

$$196 = 2^2 \times 7^2$$

To find the number of positive whole-number divisors, we use the formula for finding the number of divisors from the prime factorization. If a number  $n$  has a prime factorization of  $p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_k^{e_k}$ , then the number of positive divisors of  $n$  is given by:

$$(e_1 + 1)(e_2 + 1) \cdots (e_k + 1)$$

For 196, the exponents in the prime factorization are 2 and 2. Therefore, the number of positive divisors is:

$$(2 + 1)(2 + 1) = 3 \times 3 = 9$$

So, 196 has 9 positive whole-number divisors.  
</examples>

<answer>  
9  
</answer>

Figure 42: Response from Qwen2.5 7B trained with the `<examples>` prior prompt. The model introduces illustrative examples to support its reasoning.