Asking LLMs to Verify First is Almost Free Lunch

Anonymous ACL submission

Abstract

To enhance the reasoning capabilities of Large Language Models (LLMs) without high costs of training, nor extensive test-time sampling, we introduce Verification-First (VF), a strategy that prompts models to verify a provided candidate answer, even a trivial or random one, before generating a solution. This approach triggers a "reverse reasoning" process complementary to standard forward Chain-of-Thought (CoT), which restricts the logical search space of the answer by pruning the LLM's output distribution. We further generalize VF prompting to Iter-VF, a sequential test-time scaling (TTS) method that iteratively cycles the verification-generation process using the model's previous answer. Extensive experiments across various benchmarks and various LLMs confirm that VF prompting with random answer consistently outperforms standard CoT with minimal computational overhead, and Iter-VF outperforms existing TTS strategies. VF is also effective on SOTA thinking models. For example, by using the simple VF prompting, we obtain a new SOTA 94.9% accuracy on GPQA-Diamond with Gemini-3-Pro-Preview where VF reduces its errors by $\sim 30\%$ relatively. Code is provided at `https://anonymous.4open.science/r/VF-Code-DDD2`.

1 Introduction

To make LLMs adept at complex reasoning tasks, it is common to convert a complex problem into multi-step, modular and primary reasoning steps within their capacity. A fundamental technique is to ask the LLM to "think step by step", forming CoT (Wei et al., 2022). Though generating such a reasoning path to the final solution would be much more simpler than directly output the solution, their reliability is still undermined by their tendency to generate plausible but incorrect solutions, including hallucination and error propagation. This fallibility stems from their pre-trained nature to generate coherent natural language auto-regressively word by word, but the lack of logical reasoning, which can prioritize fluency over factual or logical rigor.

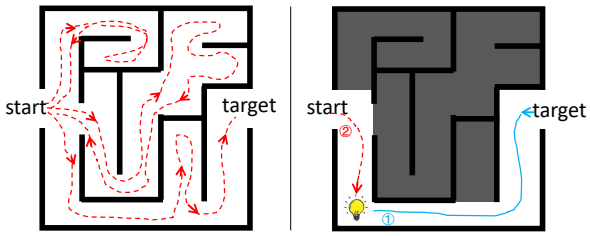


Figure 1: The verification process (blue) restricts the logical search space of forward reasoning process (red).

To enhance LLM reasoning, existing methods take CoT process as primary components, to develop strategies beyond, with significant costs at three dimensions: prior knowledge, test-time computation, and training. Strategies typically involve crafting problem-specific prompts (Wei et al., 2022; Chia et al., 2023; Alazraki et al., 2025), increasing inference budgets through expensive parallel sampling (Wang et al., 2022) or sequential reflection (Madaan et al., 2023; Shinn et al., 2023), fine-tuning models (Cobbe et al., 2021; Kumar et al., 2025), or involving multiple above perspectives (Yao et al., 2023; Lightman et al., 2023; Besta et al., 2024; Snell et al., 2024; Muennighoff et al., 2025; Setlur et al., 2025). This suggests a prevailing understanding: *better reasoning can only be attained at a significant cost.*

In this paper, we propose a method that is extremely cheap to improve LLM reasoning, by complementing the CoT process from the start. The core is Verification-First (VF) strategy, providing an answer (regard its correctness or not) along with the problem and ask LLM to first verify/evaluate the provided answer then give correct answer, in contrast to ordinary reasoning that starting from the problem only. The key insight is that verifying an

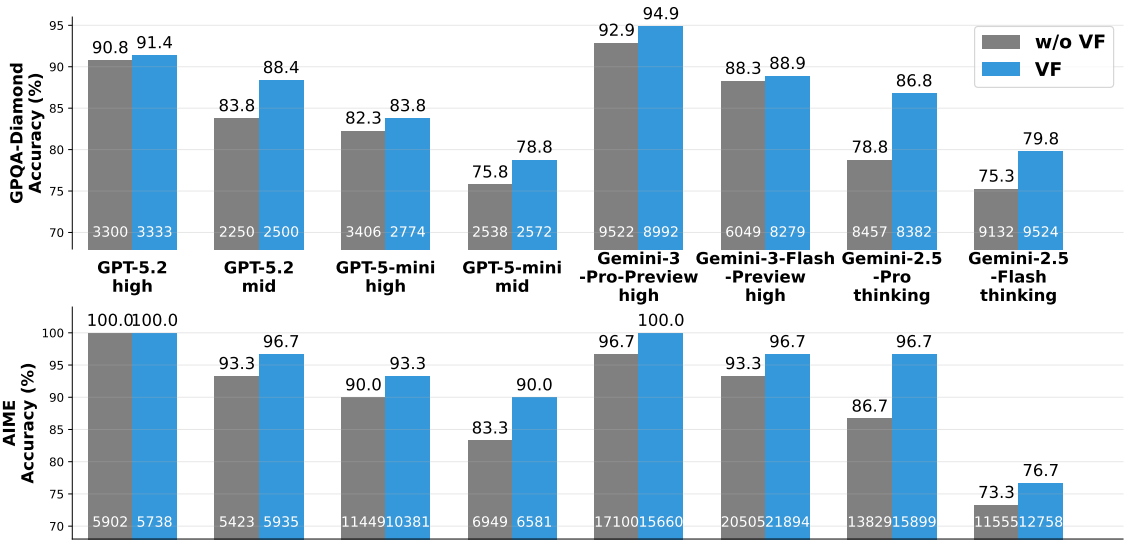


Figure 2: Performance comparison of thinking models between VF (prompting with random/trivial answer) and without (w/o) VF. The white number at bottom of each bar represent average token consumption.

answer restricts the logical search space of the final answer by pruning the LLM's output distribution autoregressively, with theoretical analysis provided. The VF strategy can be implemented by providing a random/trivial answer with minimal prior knowledge, or previously generated answer, and thus can be generalized as a TTS method. It turns out to improve reasoning with costing **minimal/zero prior knowledge, no training, and minimal test-time computation** (comparing with existing TTS methods). For example shown in Figure 2, adding one simple sentence in the prompt can effectively improve the performance of SOTA thinking models, with almost no additional cost. Our contributions are:

- We propose Verification-First strategy toimprove LLM reasoning with low cost, by triggering reverse reasoning to restrict logical search space.
- We show VF strategy is easy to implement with existing LLMs, and is applicable in usages from zero-shot prompting to TTS.
- Extensive experiments show that the proposed algorithms outperforms standard CoT and existing TTS methods across various tasks and models.

2 Related Works

To address the fallibility of LLMs generating coherent natural language, which can prioritize fluency over factual or logical rigor, many methods tries to improve LLM reasoning ability beyond CoT. Existing methods typically impose additional costs from three distinct perspectives: prior knowledge, test-time computation, and training.

Some approaches depend heavily on task-specific customization. These methods require humans to provide extensive prior knowledge to craft prompts with more few-shot examples or delicate, task-specific instructions (Wei et al., 2022; Chia et al., 2023; Alazraki et al., 2025). This limits generalization as the prompt must be tailored to the specific problem. Another prominent line of work increases inference costs to make reasoning more deliberate. This is often achieved through parallel strategies, such as generating multiple candidates and voting (Wang et al., 2022) or selecting the best via a reward model (Lightman et al., 2023). Alternatively, sequential strategies iteratively reflect on and refine previous steps (Madaan et al., 2023; Shinn et al., 2023). More complex strategies combine these by decomposing steps into trees or graphs (Yao et al., 2023; Besta et al., 2024). Recent studies on TTS suggest that significant performance gains in this paradigm generally require a proportional increase in token generation, meanwhile the evaluator or LLM itself should be correspondingly trained to be capable with the scaling strategy (Snell et al., 2024; Muennighoff et al., 2025; Setlur et al., 2025). Finally, it has been widely accepted that model performance can be improved by fine-tuning on high-quality reasoning data (Cobbe et al., 2021; Kumar et al., 2025). However, this incurs substantial training costs and is restrictive, as it requires access to open-source parameters and curated datasets, which is not feasible for all applications or closed-source models.

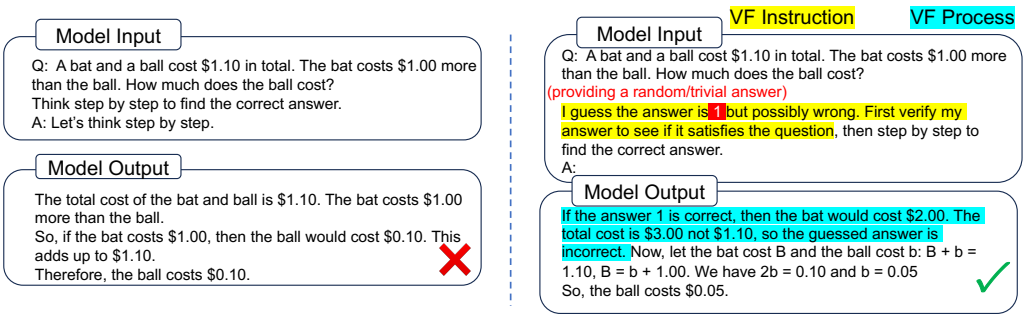


Figure 3: VF prompting with random/trivial answer (Right), comparing with standard CoT prompting (Left)

3 Proposed Method

The main idea of VF strategy is simple, complementing CoT reasoning. Considering we have a problem $\mathbf{Q}$ for LLM to answer, while the standard CoT instructs the LLM by $\texttt{CoT}(\mathbf{Q}) :\simeq$ "*Think step by step to find the answer of* $\mathbf{Q}$", VF instructs the LLM by $\texttt{VF}(\mathbf{Q}, \mathbf{A}') :\simeq$ which is defined as

> $\texttt{VF}(\mathbf{Q}, \mathbf{A}') :\simeq$ "*A possible answer of* $\mathbf{Q}$ *is* $\mathbf{A}'$. *First verify if* $\mathbf{A}'$ *is correct, then think step by step to find the answer.*"

It is expected that the LLM would output to first verify the provided answer as a critic, like bringing back to the question to see if it satisfies conditions, and then reaches the final answer. We discuss two different ways to provide $\mathbf{A}'$, corresponding to two scenarios to apply the method: zero-shot prompting (Section 3.1) and test-time scaling (Section 3.2).

3.1 VF Prompting: Zero-Shot Prompting

With minimal cost, $\mathbf{A}'$ can be random (or trivial). As we expect VF inherently takes advantage from the verification process, rather than the information gain in $\mathbf{A}'$. For problems with simple answer space, where a random answer can be determined with minimal prior knowledge, while being nontrivial to verify, user can provide such an answer as $\mathbf{A}'$. For example, For instance, a general heuristic is to $\mathbf{A}' = 1$ for math word problems and $\mathbf{A}' =$"*Option B*" for choice problems. The algorithm is described in Algorithm 1., and an illustrative case is provided in Figure 3.

We now establish a formal framework grounded in **constraint satisfaction** and **search-space restriction**. To ensure this formalization is self-consistent, we first explicitly state the assumptions governing the underlying capacities and behaviors of the LLM.

3.1.1 Theoretical Analysis

A problem $\mathbf{Q}$ can be formalized as a set of constraints $C^* = \{c_1, c_2, \ldots, c_k\}$. A valid solution must satisfy all constraints simultaneously. Thus, the true correct answer space S^* is the intersection of the sets of answers satisfying each individual constraint: $S^* = \bigcap_{c_i \in C^*}^{k} S(c_i)$ where $S(c_i)$ represents the space of all possible answers that satisfy constraint c_i.

Consider the following three core behavioral properties of LLMs, which are inherent to human-like intelligence: **Assumption 1 (Limited Capacity):** The LLM's attention is inherently bounded. It naturally prioritizes explicit, lexically salient, or frequent constraints while failing to attend to implicit or logically complex constraints. **Assumption 2 (Verification Asymmetry):** Verifying an candidate answer against the problem is easier for an LLM than generating the correct answer from scratch (the majority of people agree P≠NP and LLMs are aligned with majority people). If the candidate is incorrect, this verification acts as a "collision test" that explicitly surfaces constraints violated by the candidate. **Assumption 3 (Contextual Adherence):** Once a constraint is explicitly verbalized in the model's immediate context window, the model's subsequent autoregressive decoding probability mass is strictly conditioned upon it. Now given problem $\mathbf{Q}$, denote all answers possibly generated by CoT as $\mathbf{A}_{\text{CoT}} \in S_{\text{CoT}}$, and all possible answers possibly generated by VF as $\mathbf{A}_{\text{VF}} \in S_{\text{VF}}$. We have the following theorem:

Theorem 1 (Search-Space Restriction via VF) *$S^* \subseteq S_{VF} \subseteq S_{CoT}$, and $P(\mathbf{A}_{VF} \in S^*) \geq P(\mathbf{A}_{CoT} \in S^*)$.*

This means VF improves reasoning by restricting the search space of possible answers, proved and illustrated as following.

3.1.2 Search Space Restriction via Verification

In standard CoT, the LLM generates a reasoning path autoregressively. Due to Assumption 1, its inherently bounded attention during open-ended generation, the model often fails to instantiate all necessary constraints, actively operating on only a subset of constraints $C_{\text{CoT}} \subseteq C^*$. Consequently, standard CoT operates within an effectively larger, under-constrained search space: $S_{\text{CoT}} = \bigcap_{c_i \in C_{\text{CoT}}} S(c_i)$ Because $C_{\text{CoT}} \subseteq C^*$, it follows that $S_{\text{CoT}} \supseteq S^*$.

The core mechanism of VF is to introduce a candidate answer $\mathbf{A}'$ and instruct the model to verify it before the CoT process. When instructed to verify A', the model acts as a critic. Consider Assumption 2, there are two circumstances: If $\mathbf{A}'$ happens to be correct, then LLM acknowledge it is correct with higher chance than generating the correct answer from scratch. If $\mathbf{A}'$ is incorrect, $A' \notin S^*$, which is the usual case with random/trivial $\mathbf{A}'$. It will violate at least one constraint $c_j \in C^*$. Following Assumption 2, verifying $\mathbf{A}'$ acts as collision test, explicitly triggering the instantiation of at least one previously ignored constraint $c_j \in C^*$. Meanwhile the generated verification trace explicitly articulates c_j and adds it to the context window (e.g., "*If the answer is A', then condition c_v would not hold...*") before CoT process. Following Assumption 3, for the subsequent CoT generation phase, the model's active constraint set is expanded: $C_{\text{VF}} = C_{\text{CoT}} \cup \{c_j\}$ Consequently, the effective search space for the final answer is restricted: $S_{\text{VF}} = S_{\text{CoT}} \cap S(c_j)$ So we have $S^* \subseteq S_{\text{VF}} \subseteq S_{\text{CoT}}$, i.e., the space of plausible-but-incorrect reasoning paths is pruned. Thus for a answer $\mathbf{A}_{\text{VF}} \in S_{\text{VF}}$, and a answer $\mathbf{A}_{\text{CoT}} \in S_{\text{CoT}}$, we have $P(\mathbf{A}_{\text{VF}} \in S^*) \geq P(\mathbf{A}_{\text{CoT}} \in S^*)$, i.e., the answer generated by VF prompting has greater chance to be correct than CoT.

3.1.3 Example Illustration

To make this formalization concrete, consider the case shown in Figure 3.

The Constraints (C): c_1: Total cost is \$1.10 (Bat + ball = 1.10). c_2: Bat is \$1.00 more than the ball (Bat = ball + 1.00).

CoT: The model focuses on the numbers (\$1.10 and \$1.00) and the operation of subtraction (c_1), while failing to rigorously instantiate the relative difference (c_2). It searches in an under-constrained space ($C_{\text{CoT}} = \{c_1\}$) and easily outputs ball = 0.10.

Algorithm 1 VF Prompting (zero-shot prompting)

Require: LLM M, problem $\mathbf{Q}$, a random/trivial answer $\mathbf{A}'$.
Get final answer $\mathbf{A}$ from $M(\texttt{VF}(\mathbf{Q}, \mathbf{A}'))$;
return answer $\mathbf{A}$.

Algorithm 2 Iter-VF (TTS)

Require: LLM M, problem $\mathbf{Q}$, computation budget B, (optional: initial answer $\mathbf{A}_0$).
for $i = 1, \cdots, , B$ **do**
  Get final answer $\mathbf{A}_i$ from $M(\texttt{VF}(\mathbf{Q}, \mathbf{A}_{i-1}))$;
  Early stop if $\mathbf{A}_i == \mathbf{A}_{i-1}$;
end for
return answer $\mathbf{A}_B$.

VF: We supply a trivial answer $A' = 1$. To check this, the model substitutes ball = 1 into the constraints. It computes Bat = 2.00 (c_2) and checks Total = 3.00 $\neq$ 1.10 (c_1). This "collision" forces the model to explicitly write out the simultaneous relationship of both c_1 and c_2 in the context. The search space is now restricted by both constraints explicitly. The naive heuristic path (0.10) is pruned from the valid continuation space, virtually guaranteeing the model finds the correct algebraic intersection (0.05).

3.2 Iter-VF: Test-Time Scaling

While zero-shot VF prompting is effective, it faces two limitations: defining a meaningful initial answer for complex, open-ended tasks is challenging, and it lacks a mechanism for controllable test-time scaling (TTS). To address these issues, we propose Iter-VF.

Generating the Initial Answer. For complex, open-ended tasks like coding, a trivial guess (e.g., $\mathbf{A}'$ ="*print('Hello World')*") provides little verification signal. To supply a non-trivial candidate without relying on prior knowledge, we simply ask the LLM to generate it. We first prompt the LLM using standard CoT to obtain an initial answer $\mathbf{A}_1$, and then apply VF prompting, i.e., $\texttt{VF}(\mathbf{Q}, \mathbf{A}_1)$ to yield a final answer $\mathbf{A}_2$. This two-call approach can be applied to any complex problem (Section 4.3.1).

Iterative Test-Time Scaling. To achieve controllable TTS, we generalize this process into a sequential iteration. In each step i, a VF process takes an old answer $\mathbf{A}_{i-1}$ as input and generates an improved answer $\mathbf{A}_i$. We repeat this cycle up to a defined computation budget B, as outlined in Algorithm 2. The initial candidate $\mathbf{A}_0$ can be either a user-provided trivial answer or one generated via standard CoT.

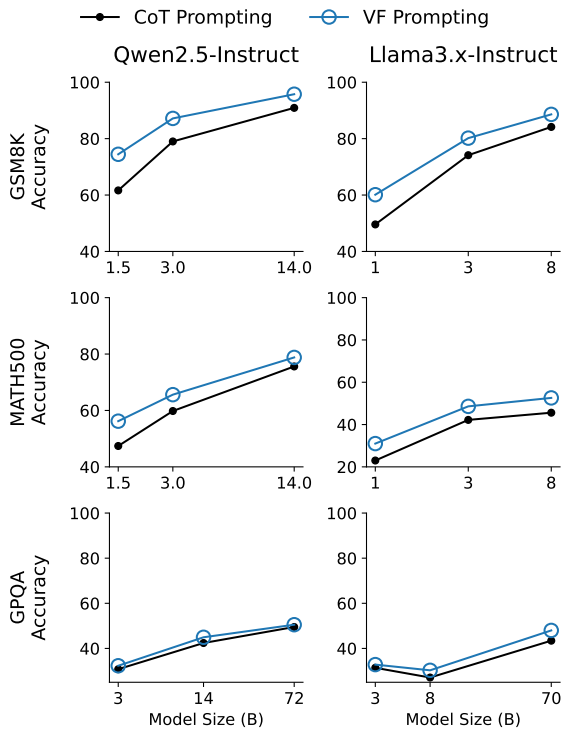


Figure 4: VF prompting consistently outperforms standard CoT prompting.

As a sequential TTS strategy, Iter-VF seems similar with existing strategies Self-Correction / Refine / Reflexion in the iterations. In fact they are very different, as discussed in Appendix A. Iter-VF distinguishes itself by (i) maintaining a Markovian process across iterations, that avoids context overflow and error accumulation; and (ii) instructing to do reasoning from scratch and benefit from the verification process in every iteration.

4 Experiments

The experiment evaluation includes the following main results: (i) VF prompting with random/trivial answer shows consistent improvement over standard CoT prompting. (ii) Iter-VF outperforms existing TTS strategies under limited test-time computation budget on models without additional training. (iii) On open-end tasks in real-world scenarios, VF prompting with previous generated answer performs the best comparing with existing strategies using similar computation budget. (iv) With thinking models and thought-hidden LLM service where other strategies might be inapplicable or duplicate, VF strategy is still effective. Code will be provided to public.

Dataset	GSM8K	MATH500	GPQA
CoT	365.6	808.3	739.3
VF	533.6	1109.6	901.8

Table 1: Numbers of output tokens.

4.1 VF Prompting with Random/Trivial Answer

Adapting VF prompting with random/trivial answer requires the random/trivial answer can be easily defined. Most reasoning benchmark satisfies such requirement. We use reasoning benchmarks of math problems GSM8K (Cobbe et al., 2021) and MATH500 (Hendrycks et al., 2021; Lightman et al., 2023), and a graduate-level science Q&A benchmark GPQA-Diamond (Rein et al., 2024). We compare with standard CoT prompting (0-shot), as they are generally applicable costing comparable minimal prior knowledge. We provide trivial answer "1" for all problems in GSM8K and MATH500, and random choice of shuffled options for all problems in GPQA-Diamond.

We evaluate LLMs from the Qwen2.5 (Yang et al., 2024) and Llama3 (Grattafiori et al., 2024) families, covering size from 1B to 72B: Qwen2.5-1.5B / 3B / 14B / 72B-Instruct, Llama3.2-1B / 3B-Instruct, LLama3.1-8B-Instruct, Llama3.3-70B-Instruct.

Performance and Cost Comparison. Figure 4 shows the results, and Appendix D provides details. We can find that VF consistently outperforms CoT, with performance advantage being stable across different model sizes. We notice the advantage on GSM8K and MATH500 is much more significant than GPQA-Diamond. This could be interpreted by the intuition and fact (Zhao et al., 2025) that facilitating reasoning improve LLMs' performance on knowledge-intensive problems much more hardly than computation/logic-intensive problems.

For test-time computation, Table 1 summarizes the numbers of output tokens to reach the final answer, on the three benchmarks respectively averaged over the tested models. Comparing with standard CoT prompting, VF prompting outputs about 20%∼50% more tokens to make verification first. Such additional test-time computation cost could be considered minimal, comparing with other methods to achieve similar improvement, which would be shown in Section 4.2.

Appendix B provides Empirical results to supportLLMs verifying an answer is easier than gen-

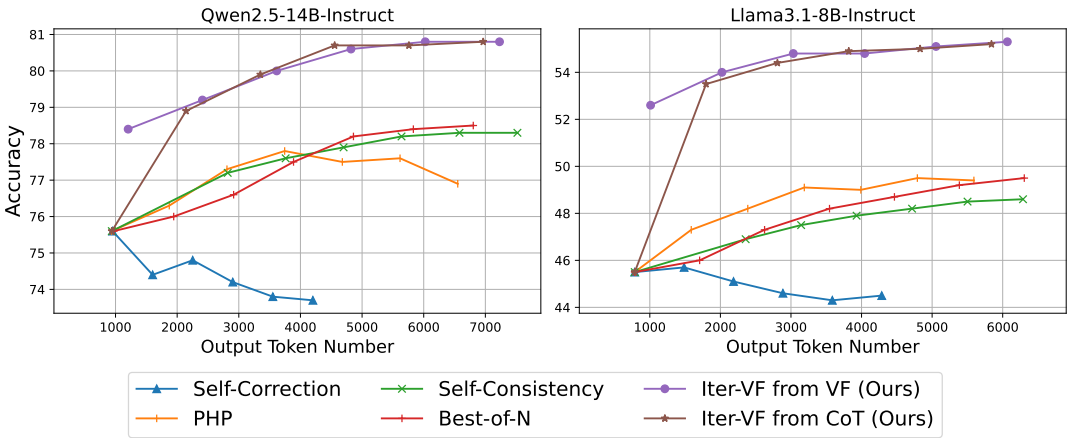


Figure 5: Comparison of different TTS methods on MATH500.

erating the correct answer and providing different answers with minimal prior knowledge would not effect the final performance of VF prompting. Appendix E provides error case analysis where CoT gets correct but VF gets wrong.

4.2 Iter-VF for Test-Time Scaling

Baselines. Since Iter-VF does not require additional training, external supervision, nor task-specific prior knowledge to decompose the problem, we compare Iter-VF with TTS strategies including sequential ones: *Self-Correction* (Madaan et al., 2023; Shinn et al., 2023; Muennighoff et al., 2025), *PHP* (Zheng et al., 2023); and parallel ones: *Self-Consistency* (Wang et al., 2022), *Best-of-N* (Lightman et al., 2023; Yao et al., 2023). The detail implementation and discussion are provided in Appendix F. We implemented two variants of Iter-VF: *Iter-VF from VF* which starts VF prompting with the trivial answer "1" in the initial iteration; and *Iter-VF from CoT* which uses standard CoT prompting in the initial iteration and start verification from the second iteration.

Performance Comparison and Analysis. Figure 5 shows the results. It can be observed that Iter-VF significantly outperforms other baselines with limited test-time computation budget.

Comparing between Iter-VF from VF and from CoT, the initial difference but fast catching up indicates what matters is the verification process rather than the initially provided answer. This observation can also explain the advantage of Iter-VF over PHP, which only provides previous answers as "near" hints, that LLM could not necessarily make effective use of them. Another important difference between Iter-VF and the other two sequential TTS strategies (PHP, Self-Correction), is that they consider accumulated information through the iterations, while Iter-VF is Markovian as it only considers the answer of last one output. The accumulation of information can have negative effect for LLMs, as they have limited capacity for long context, and hallucinate a lot. Especially, Self-Correction considers not only previous answers, but also entire thinking process. It's performance become catastrophic with the accumulation of history. This observation is in line with previous study (Huang et al., 2023). Though such method can be effective after specifically fine-tuning LLM with curated data (Snell et al., 2024; Muennighoff et al., 2025; Setlur et al., 2025), it would be far from our general setting.

As for parallel strategies (Self-Consistency, Best-of-N), they are conservative that do not utilize history information before determining final answer. Their performances improve stably with the growth of computation, though not significant. Iter-VF outperforms them, and can be integrated with them as the following discussion.

Discussion. Note that the current implementation Iter-VF is consistent with Algorithm 2, generating one sequence, and evaluating final $\mathbf{A}_B$ only (keeping consistent with all sequential strategies). However, considering its unique fully Markov property, the simple early-stopping rule could be applied to Iter-VF: as soon as $\mathbf{A}_i$ is identical with $\mathbf{A}_{i-1}$, Iter-VF stops and returns the current answer. However, we disable this adaptive rule and use a fixed budget B for all methods, because adaptive stopping changes the actual compute consumed by each method and introduces an additional design dimension beyond the update rule itself. With more computation budget, we suggest to simply modify the implementation of Iter-VF by combining with

some parallel strategies: one can first generate the Iter-VF sequence and then determining the final answer by majority-voting among all answers occurred in the iterations $(\mathbf{A}_1, \cdots, \mathbf{A}_B)$ to make full usage of them; or parallelize multiple Iter-VF paths independently.

4.3 Applying in Real-Wold Scenarios

In this section, we show the effect of proposed method in real-world scenarios beyond choice problem or math Q&A. We discuss two perspectives which users are likely to face: (i) facing open-ended applications, where it is difficult to define a random/trivial answer without prior knowledge; (ii) using thinking models and even thought-hidden LLM service, where the exact input prompt and output about the LLM thinking process are not accessible.

4.3.1 On Open-Ended Applications

We consider the following coding and API tasks where VF with random/trivial answer is no longer applicable. We evaluate VF prompting with previously generated answer (calling LLM twice, as the procedure in Figure 6(a)).

Benchmarks. For coding tasks, we evaluate on HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021). In such benchmarks, a coding problem is like "check if in given list of numbers, are any two numbers closer to each other than given threshold" or "write a function to find the volume of a sphere". For API tasks, we evaluate on API-Bank (Li et al., 2023) Level-1 and Level-2, which simulates a agentic scenario where LLM need to understand user's intention from dialogue, and refer to documentary containing API description of tool functions, to output a API call with correct API function and arguments.

Baselines To make fair comparison with similar cost, we consider baselines calling the LLM twice:

- CoT×2 (pass@2): Call the LLM twice independently. The final answer would be considered correct if at least one answer passes all test cases.
- Self-Correction: In the second call, the thinking process and the answer output by the first call is provided along with the problem, and the LLM is asked to reflect and refine the answer.

No other parallel methods would be considered as such "pass@k" implementation is necessarily the best parallel methods can do with budget k. No other sequential methods would be considered, as under this setting without external feed-back, Reflexion (Shinn et al., 2023) is reduced to such implementation; PHP has only been designed for math problems.

Performance. Table 2 shows the results. For sequential methods Self-Correction and Iter-VF, "pass@1" evaluates the success rate using the second output, and "pass@2" evaluates if at least one of the first and the second output is success, to be distinguished from evaluating two independent trials.

Comparing the final performance under the "pass@2" metric, Iter-VF performs the best. Note that on API-Bank tasks, especially Level-1, Self-Correction has very similar performance with Iter-VF. This should be considered with the fact that on such problems, the reasoning processes only take a small proportion of the context, much shorter than the problem input and output answer. In such cases, the factual difference between them is very little. Comparing the first three rows, the advantage of Self-Correction and Iter-VF (calling the LLM twice) over CoT (calling the LLM once) indicates additional test-time computation does benefit in most cases. Comparing the last three rows with the first three rows, twice the evolution chances makes considerable improvement, while Iter-VF (pass@1) still outperforms CoT×2 (pass@2) in most cases.

4.3.2 With Thinking Models and Thought-Hidden LLM Service

Recent advanced LLMs are increasingly equipped with built-in "thinking modes," utilizing inherent reasoning and TTS strategies. Moreover, cutting-edge commercial models (e.g., GPT-5, Gemini 3) present a unique challenge: they are not only closed-source but also conceal their internal reasoning traces. Users face a thought-hidden problem where they can only provide input instructions and receive post-processed final answers. The exact prompts guiding the model and the intermediate reasoning tokens are completely obscured.

Failure of Existing Prompting and TTS Methods. This hidden thinking process is often extensive. For example, our evaluations on GPT-5 Nano/Mini reveal that internal reasoning consumes roughly 10× more tokens than the visible output on benchmarks like MATH and GPQA. Because this process is opaque and the model's internal strate-

Model	Qwen2.5-14B-Instruct				Llama3.1-8B-Instruct			
Task	Coding		API-Bank		Coding		API-Bank	
Dataset	HEval	MBPP	Level-1	Level-2	HEval	MBPP	Level-1	Level-2
CoT (pass@1)	91.5	70.0	73.2	49.8	81.1	56.8	55.1	38.6
Self-Correction ("pass@1")	95.1	71.8	85.0	64.4	85.9	56.2	63.9	55.6
Iter-VF ("pass@1")	96.9	74.8	85.4	72.6	90.2	61.5	64.0	57.1
CoT×2 (pass@2)	94.5	75.9	76.7	55.2	84.8	62.6	56.6	41.2
Self-Correction ("pass@2")	97.6	77.4	**86.8**	68.9	89.0	62.2	**65.5**	**59.5**
Iter-VF ("pass@2")	**99.4**	**80.6**	**87.7**	**77.9**	**93.3**	**69.3**	**65.2**	**60.3**

Table 2: Performance comparison of different TTS methods (budget: 2 calls) on coding and API tasks. Rows of comparable results are not split by horizontal line. “HEval” is short for “HumanEval”.

gies are unknown, traditional prompting and TTS methods become either redundant or entirely invalid. For instance, appending a standard "think step by step" prompt is unlikely to yield further improvements if the model already does so natively. We do not know if GPT-5 is already using trace-dependent methods like Self-Correction. Even if it has not, users cannot ask the LLM to reflect on or refine its reasoning process because they are denied access to the reasoning trace itself.

Feasibility and Advantages of VF. In contrast, the proposed VF strategy remains highly feasible and advantageous in this paradigm. Because VF only requires modifying the initial input instruction—asking the model to evaluate a provided candidate answer before proceeding—it bypasses the need to access, monitor, or explicitly edit the model’s internal trace. The "first-verify-then-generate" formulation gives a clear, distinct directive that triggers a reverse-reasoning process. This is fundamentally distinguishable from the standard forward-reasoning strategies these models likely employ internally. Consequently, VF successfully steers and restricts the hidden reasoning space from the outset. As shown in Figure 2, VF effectively enhances the performance of powerful, thought-hidden models, bringing tangible accuracy improvement, and surprisingly, with hardly no additional cost. This could be interpreted as that by explicitly restricting the logical search space from the outset, VF prevents thinking models from wasting their naturally extensive internal reasoning tokens on dead-end trajectories, making the overall generation process more efficient and sometimes reducing total token consumption.

5 Conclusion

In this paper, we introduced Verification-First (VF), a simple but effective strategy to enhance LLM reasoning capabilities. By prompting models to verify a candidate answer, even a random or trivial one, before generating a solution, we effectively trigger a "reverse reasoning" process which restricts the logical search space of the answer by pruning the LLM’s output distribution. We further extended this concept to Iter-VF, a sequential test-time scaling method that iteratively verifies and improves answers. Our experiments demonstrate that VF improves reasoning ability with minimal computational overhead, without requiring any model training nor task-specific prior knowledge. These findings suggest that the cognitive gap between verification and generation offers an "almost free lunch" for improving LLM reliability, establishing VF as a general-purpose, plug-and-play enhancement for modern reasoning models.

6 Limitations

Despite the effectiveness of VF and Iter-VF, our work has several limitations. For open-ended or creative tasks where defining a "trivial" or "random" initial answer is semantically difficult, the zero-shot VF approach requires a preceding CoT step to generate a seed answer, increasing the initial cost. While Iter-VF dominates in low-to-medium compute budgets, its sequential Markovian nature could means it may reach a performance ceiling earlier than the other non-Markovian TTS strategies, when the computation budget is virtually unlimited. Future work may explore optimal combinations of VF with parallel search algorithms to address this upper bound.

References

Lisa Alazraki, Maximilian Mozes, Jon Ander Campos, Tan Yi-Chern, Marek Rei, and Max Bartolo. 2025. No need for explanations: Llms can implicitly learn from mistakes in-context. *arXiv preprint arXiv:2502.08550*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Theodore Baker, John Gill, and Robert Solovay. 1975. Relativizations of the p=?np question. *SIAM Journal on computing*, 4(4):431–442.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and 1 others. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 17682–17690.

Stephen D Brookfield. 1987. *Developing critical thinkers: Challenging adults to explore alternative ways of thinking and acting*. Jossey-Bass.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Yew Ken Chia, Guizhen Chen, Luu Anh Tuan, Soujanya Poria, and Lidong Bing. 2023. Contrastive chain-of-thought prompting. *arXiv preprint arXiv:2311.09277*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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. *arXiv preprint arXiv:2103.03874*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*.

Komal Kumar, Tajamul Ashraf, Omkar Thawakar, Rao Muhammad Anwer, Hisham Cholakkal, Mubarak Shah, Ming-Hsuan Yang, Phillip HS Torr, Fahad Shahbaz Khan, and Salman Khan. 2025. Llm post-training: A deep dive into reasoning large language models. *arXiv preprint arXiv:2502.21321*.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*.

Jean Piaget. 1976. Piaget's theory. In *Piaget and his school: A reader in developmental psychology*, pages 11–23. Springer.

George Polya. 1957. How to solve it.

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*.

Amrith Setlur, Nived Rajaraman, Sergey Levine, and Aviral Kumar. 2025. Scaling test-time compute without verification or rl is suboptimal. *arXiv preprint arXiv:2502.12118*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain

of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 40 others. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

James Xu Zhao, Bryan Hooi, and See-Kiong Ng. 2025. Test-time scaling in reasoning models is not effective for knowledge-intensive tasks yet. *arXiv preprint arXiv:2509.06861*.

Chuanyang Zheng, Zhengying Liu, Enze Xie, Zhenguo Li, and Yu Li. 2023. Progressive-hint prompting improves reasoning in large language models. *arXiv preprint arXiv:2304.09797*.

A Distinguishing Iter-VF via Test-Time Scaling Update Dynamics

As a sequential TTS strategy, Iter-VF seems similar with existing strategies Self-Correction / Refine / Reflexion in the iterations. In fact they are very different. Iter-VF distinguishes itself by (i) maintaining a Markovian process across iterations, that avoids context overflow and error accumulation; and (ii) instructing to do reasoning from scratch in every iteration.

Sequential TTS strategies can be expressed as an iterative procedure. Let $\mathbf{Q}$ be the problem formulation and let $\mathbf{S}_t$ denote the method's carried state at iteration t. A generic test-time method can be written as an update operator $\mathcal{T}$:

$$\mathbf{S}_{t+1} = \mathcal{T}(\mathbf{Q}, \mathbf{S}_t). \tag{1}$$

Different methods primarily differ in (i) what information is stored in the state $\mathbf{S}_t$, and (ii) how the operator $\mathcal{T}$ uses that state. This perspective allows us to categorize sequential TTS methods into two fundamental algorithmic families, summarized in Table 4: Trace-Editing Refinement (TER) and Answer-Hypothesis Testing with ReSolve (AHTRS).

While TER explicitly encourages the model to reuse and patch its own prior reasoning R_t, structural errors and spurious rationales can be easily reinforced. Furthermore, the context window grows continuously with the iteration budget, leading to potential context overflow and degraded attention allocation. We attribute the observed empirical results (Section 4.2) to such reason (Huang et al., 2023). As AHTR, by decoupling the Update step $\mathcal{T}$ from the historical trace $\mathbf{R}_t$, Iter-VF maintains a lower, Markovian dependence. It inherits the search-space restriction benefits described in Section 3.1, without the anchor of the possibly flawed reasoning path.

B Sensitivity on Provided Answer to Verify

It is important to understand if the provided answer $\mathbf{A}'$ in VF prompt makes influence. We show the effect of providing different answers to VF prompting in Figure 7. For GSM8K and MATH500, we evaluate the effect of providing trivial answers "1" and "2025", and the true answer respectively. For GPQA-Diamond problems which are single choice problems, we evaluate the effect of providing random option, false option, and the true option respectively.

Obviously, though it is not a valid method in application, providing the true answer in VF prompting (without telling the correctness, asking the LLM to verify first and then give correct answer) significantly improves performance. Such result indicates LLMs' capacity on verification and the fact that verifying an answer is easier than generating the correct answer.

Oracle true answers can provide extra information, but it is not a practically applicable method, only studied for understanding the effect. Among minimal-prior-knowledge candidates the dominant factor is whether verification elicits useful constraints rather than the exact candidate value. Most importantly, it can be observed that providing different answers with minimal prior knowledge would not effect the final performance of VF prompting a lot.

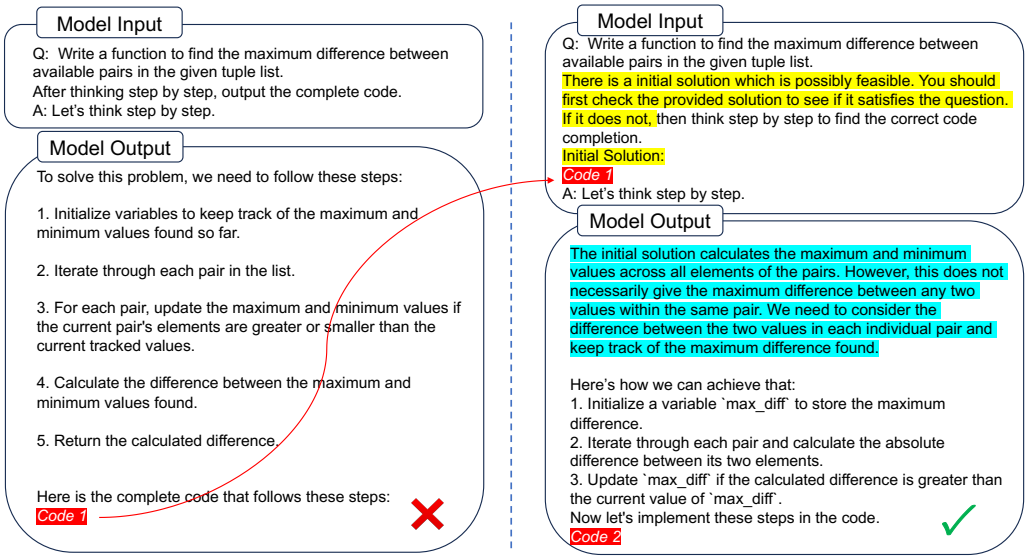
Note that for many problems, the answer space is not strict for LLMs to understand. As in MATH500 dataset, while many problems should not be answered with a scalar, the LLM can still interpret and verify the provided answer "1" with meaningful process. For example, the first test case of MATH500 is asking "Convert $(0, 3)$ in rectangular coordinates to polar coordinates. (r, θ)...", and the LLM can response the VF prompting by "If the answer 1 is correct, ... the point $(0, 3)$ in rectangular coordinates converts to polar coordinates (r, θ) with $r = 1$ ...". Complex problems where it is hard to provide a random answer nontrivial to verify, will be studied in Section 4.3.1.

C Prompts

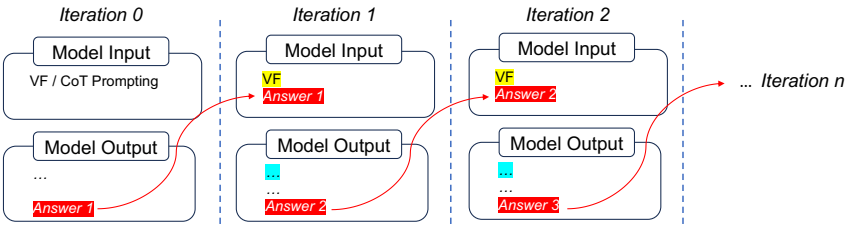
The table in Figure 8 provides the specific prompts we use for evaluation in our experiments. One can also use the provided code including specific prompts and evaluation protocols.

D Performance of CoT and VF with Random/Trivial Answer

Table 3 provides the results comparing CoT prompting and VF prompting. We evaluate LLMs from the Qwen2.5 (Yang et al., 2024) and Llama3 (Grattafiori et al., 2024) families, covering size from 1B to 72B: Qwen2.5-1.5B / 3B / 14B / 72B-Instruct, Llama3.2-1B / 3B-Instruct, LLama3.1-8B-Instruct, Llama3.3-70B-Instruct. We use different x in Llama3.x, because different xs provide mod-



(a) VF prompting with previously generated answer.



(b) Iter-VF for test-time scaling.

Figure 6: Illustration of (a) VF prompting with previously generated answer, and iterating such process as (b) Iter-VF for test-time scaling.

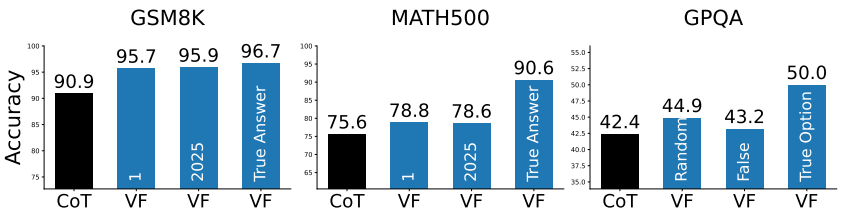


Figure 7: Providing different answers to VF for verification.

els in different sizes and some sizes only exist in specific x.

The performances of 1.5B/1B on GPQA-Diamond is not reported because they show performances almost indistinguishable with random guess (25%). This means the benchmark is too hard for 1.5B/1B models that they show zero knowledge/reasoning ability, thus lack of comparison value. So we additionally evaluate the larger 72B/70B on GPQA-Diamond instead.

E Error Case Analysis

Aggregate accuracy does not fully characterize how VF changes individual reasoning trajectories. We therefore inspect the overlap between CoT and VF outcomes by separating examples into four categories: both correct, VF-correct/CoT-wrong, VF-wrong/CoT-correct, and both wrong. The third category is especially important because it isolates cases where the verification-first instruction harms an otherwise correct CoT trajectory. Further inspection shows that such VF-wrong/CoT-correct cases do exist non-negligibly, especially for smaller models such as 3B or 8B models. Hence, VF should be understood as an average-case improvement mechanism rather than a per-instance monotonic guarantee over CoT.

Based on the theoretical framework in Section 3.1, we categorize VF failures into three main modes. The first two correspond to failures of the two key steps required by the search-space restriction argument: detecting a constraint collision and then faithfully using the surfaced constraint in subsequent generation. The third mode concerns tasks

Dataset		Inputs	Prompt
GSM8K, MATH500		question	"Question: {question}\nI guess the answer is 1 but is possibly wrong. Please first check my answer to see if it satisfies the question explain why, then think step by step to give the correct answer.\nAnswer: If the answer 1 is correct,"
GPQA-Diamond		question, choices_str, choices_shuffled	"Question: {question}\nOptions:\n{choices_str}\nI guess the answer is '{choices_shuffled[1]}' but possibly wrong. First check my answer to see if it satisfies the question and explain why, then think step by step to get the correct answer. Output your final answer with 'The final answer is A/B/C/D.'.\nAnswer:"
HumanEval	Call 1 (CoT)	question	"Please complete the following Python function:\n{question}\nAfter thinking step by step, output the complete code. Finally output the complete code between ``` and ```.\nAnswer:\n"
	Call 2 (VF)	question, generated_answer	"Please complete the following Python function:\n{question}\nThere is a initial solution which is possibly feasible. You should first check the provided solution to see if it satisfies the question. If it does not, then think step by step to find the correct code completion. Finally output the complete code between ``` and ```.\nInitial Solution:\n{generated_answer}\nAnswer:\n"
MBPP	Call 1 (CoT)	question, test_list[0]	"You are a coding assistant. {question} The code should satisfies ```{test_list[0]}```.\nAfter thinking step by step, output the complete code. Finally output the complete code between ``` and ```.\nAnswer:\n"
	Call 2 (VF)	question, test_list[0], generated_answer	"You are a coding assistant. {question} The code should satisfies ```{test_list[0]}```. There is a initial solution which is possibly feasible. You should first check the provided solution to see if it satisfies the question. If it does not, then think step by step to find the correct code completion. Finally output the complete code between ``` and ```.\nInitial Solution:\n{generated_answer}\nAnswer:\n"
API-Bank	Call 1 (CoT)	context, instruction	"You are a helpful assistant. You will be provided with a context dialogue between user and AI, and a formal instruction and API description to generate API request based on the user intention. You only can use the API provided with descriptions in Context and Instruction. After thinking step by step, output your final answer in the asked format finally.\n\nContext:\n{context}\n\nInstruction:\n{instruction}\n\nAnswer: Let's think step by step."
	Call 2 (VF)	context, instruction, generated_answer	"You are a helpful assistant. You will be provided with a context dialogue between user and AI, a formal instruction and API description to generate API request based on the user intention. Note that you only can use the API provided with descriptions in Context and Instruction. You will also be provided with an initial solution which might be feasible or not. You need to first verify the provided initial solution to see if it is the best solution. Then after thinking step by step, output the best solution in the asked format.\n\nContext:\n{context}\n\nInstruction:\n{instruction}\n\nInitial Solution:\nAPI-Request: {generated_answer}\n\nAnswer: First let's consider if API-Request: {generated_answer} satisfies the API description and the user's intention."

Figure 8: The specific prompts of VF for different datasets.

Model Family	(3.x-)Size (B)	GSM8K		MATH500		GPQA	
		CoT	VF	CoT	VF	CoT	VF
Qwen2.5-Instruct	1.5	61.64	**74.45**	47.40	**56.20**	–	–
	3	79.00	**87.17**	59.80	**65.60**	30.80	**32.32**
	14	90.91	**95.75**	75.60	**78.80**	42.40	**44.94**
	72	–	–	–	–	49.50	**50.51**
Llama3-Instruct	3.2-1	49.58	**60.12**	23.00	**31.00**	–	–
	3.2-3	74.13	**80.18**	42.20	**48.60**	31.41	**32.83**
	3.1-8	84.15	**88.60**	45.60	**52.60**	27.10	**30.30**
	3.3-70	–	–	–	–	43.43	**47.98**

Table 3: Comparison of accuracy (%) between CoT and VF Prompting across Qwen2.5 and Llama3.x models on GSM8K, MATH500, and GPQA datasets.

where the main bottleneck is missing knowledge rather than missing constraint instantiation.

These three modes share the same underlying source: limited model capability. VF introduces an additional verification-first requirement before the final answer is generated. When the model has sufficient verification, reasoning, and knowledge capacity, this extra requirement helps by surfacing constraints and restricting the logical search space. However, when the model is weaker, the same requirement can impose extra reasoning burden. Following the verification instruction may consume capacity or attention and may distract the model from a simpler forward reasoning path that CoT would have taken correctly. This explains why VF-wrong/CoT-correct cases can appear more frequently in smaller models, even though VF improves aggregate performance in our main experiments.

Thus, VF failures are not a separate contradiction of the proposed mechanism, but boundary cases where one of the required model abilities is insufficient: detecting the constraint collision, completing the verify-then-solve reasoning process under the surfaced constraints, or possessing the background knowledge needed for verification. This also suggests practical safeguards. For reliability-critical applications, one can compare CoT and VF outputs and accept the answer only when they agree, use an external checker when available, or apply a lightweight consistency vote. For Iter-VF, early stopping after answer stabilization can reduce unnecessary re-solving and mitigate over-correction.

Family	Trace-Editing Refinement (Existing)	Answer-Hypothesis Testing with Re-Solve (Ours)
Representative Methods	Self-Correction, Self-Refine, Reflexion	Iter-VF
State ($\mathbf{S}_t$)	$\mathbf{S}_t = (\mathbf{A}_t, \mathbf{R}_t)$ (Answer + prior trace)	$\mathbf{S}_t = \mathbf{A}_t$ (Candidate answer only)
Update Operator ($\mathcal{T}$)	Explicitly critiques and edits the earlier trace.	Verifies answer, then re-solves from scratch.
Context Window	$\lvert\mathbf{S}_t\rvert \sim \mathcal{O}(t)$. Grows continuously with the iteration budget.	$\lvert\mathbf{S}_t\rvert \sim \mathcal{O}(1)$. Remains compact and constant.

Table 4: Comparison of Sequential TTS Dynamics distinguishing Trace-Editing Refinement methods from our proposed Answer-Hypothesis Testing with Re-Solve approach.

Failure mode	**Mechanism**	**Representative symptom and implication**
Failure of Assumption 2: missed constraint collision	The candidate answer violates one or more problem constraints, but the model fails to detect the decisive collision, or rejects the candidate only superficially without surfacing the violated constraint.	The verification trace does not introduce useful new constraints into the context. The subsequent generation therefore remains close to the original under-constrained CoT trajectory, so VF provides little or no benefit.
Failure of Assumption 3: incomplete verification-to-generation transfer	The model detects that the candidate answer is wrong and may even verbalize relevant constraints, but it then misinstantiates, ignores, or incompletely reasons with these constraints during the final solving step.	In the house-flipping problem, VF rejects answer "1" but misinterprets "increased the value by 150%" as the final value being 150% of the purchase price, producing an incorrect profit instead of the correct \$70,000. In the omelet problem, VF rejects the trivial answer but double-counts the factor of three eggs, yielding 21 dozens instead of 7 dozens. These are genuine VF-induced errors, not merely failures to improve over CoT.
Knowledge-intensive rather than constraint-intensive tasks	The limiting factor is missing domain knowledge rather than failure to attend to explicit problem constraints. In such cases, verifying a random or trivial candidate does not reliably recover the needed facts.	VF can still help when verification exposes a useful inconsistency, but the expected gain is weaker because search-space restriction cannot compensate for absent knowledge. This helps explain why gains on knowledge-intensive tasks such as GPQA-Diamond are smaller than on more constraint-rich mathematical problems.

Table 5: Three representative VF failure modes aligned with the theoretical analysis. The first two correspond to failures of the verification-induced constraint restriction mechanism; the third occurs when the task is dominated by knowledge retrieval rather than explicit constraint satisfaction.

F Test-Time Scaling Baselines

we compare Iter-VF with following TTS strategies:

- Self-Correction (Madaan et al., 2023; Shinn et al., 2023; Muennighoff et al., 2025): A sequential strategy that ask LLM to reflect and refine output iteratively.
- PHP (Zheng et al., 2023): A sequential strategy that concatenates all previous answers after the input question by"(Hint: The answer is near to $a_1, \cdots, a_{t-1}$)".
- Self-Consistency (Wang et al., 2022): A parallel strategy that generates multiple reasoning paths leading to answers independently, and determines the final answer by majority-voting.
- Best-of-N (Lightman et al., 2023; Yao et al., 2023): A parallel strategy that multiple reasoning paths leading to answers independently, and asks the LLM to evaluate the quality of each one respectively, and determines the final answer with highest score.

Note that while some methods (Lightman et al., 2023; Yao et al., 2023; Shinn et al., 2023; Muennighoff et al., 2025) could require training an additional model as evaluator, and use it to progressively evaluate decomposed reasoning steps, or obtaining external feedback, they are reduced to such implementation for fair comparison under the no-training and no task-specific prior knowledge setting.