Article

Falsification-Augmented Bidirectional Self-Consistency for Robust Aggregation under Correlated Reasoning Errors

Firstname Lastname [1], Firstname Lastname [2] and Firstname Lastname [2,*]

[1] Affiliation 1; e-mail@e-mail.com
[2] Affiliation 2; e-mail@e-mail.com
* Correspondence: e-mail@e-mail.com; Tel.: (optional; include country code; if there are multiple corresponding authors, add author initials) +xx-xxxx-xxx-xxxx (F.L.)

Abstract

Self-Consistency (SC) improves chain-of-thought (CoT) reasoning by sampling multiple solutions and selecting the most frequent answer, implicitly treating answer agreement as evidence of correctness. This premise breaks when a single model's samples share correlated mistakes, yielding a high-agreement but wrong majority. We study whether purely prompt-time interventions can make SC more robust without training an external verifier. We propose Falsification-Augmented Bidirectional Self-Consistency (FAB-SC), which attaches to each sampled solution two complementary reliability signals produced by the same base model: a backsolve "plug-back" verification that checks whether a proposed answer satisfies the problem's explicit constraints, and a small stochastic refuter ensemble that attempts to falsify the solution and outputs an estimated probability of error. We additionally apply a length-based penalty to reduce verbosity-driven scoring artifacts, and aggregate answers via importance-weighted voting. In inference-only pilots with GPT-4o and a matched forward-sampling budget ($m = 8$), FAB-SC yields mixed outcomes: on SVAMP (first 300 test items) it slightly improves accuracy (0.853 vs 0.843 for SC), while on GSM8K (first 200 test items) it reduces accuracy (0.785 vs 0.810) and increases wall-clock runtime by about 18×. These results clarify both the promise and the pitfalls of prompt-only verification when the model "grades its own work" and motivate more calibrated verification signals and compute-aware variants.

Keywords: keyword 1; keyword 2; keyword 3 (List three to ten pertinent keywords specific to the article; yet reasonably common within the subject discipline.)

1. Introduction

Large language models (LLMs) can often be induced to perform multi-step reasoning by producing intermediate natural-language explanations, commonly termed chain-of-thought (CoT) [**?**]. In arithmetic word problems, CoT can make implicit intermediate computations explicit, improving accuracy relative to direct answer-only prompting. Yet CoT does not eliminate brittleness: a single decoding pass can be derailed by small but consequential mistakes such as an arithmetic slip, a unit mismatch, or an incorrect mapping from text to equations. Worse, the generated rationale may remain fluent and coherent despite being wrong, which complicates downstream trust.

Self-Consistency (SC) addresses this brittleness at inference time by sampling multiple diverse CoT solutions from the same model and aggregating the resulting final answers, typically by majority vote [**?**]. SC can be viewed as a self-ensemble that marginalizes over

Received:
Revised:
Accepted:
Published:

stochastic reasoning paths: instead of trusting a single trajectory, it trusts the most recurrent answer across sampled trajectories.

However, SC relies on an informal but consequential assumption: sampled rationales behave like approximately independent evidence for an answer, so that agreement implies correctness. In practice, this assumption can fail in a particularly damaging way. Samples from a single model share parameters, training data priors, and common high-probability heuristics, so their errors are often correlated. If the model systematically misreads a quantity, repeats an incorrect conversion, or applies the same flawed heuristic across samples, then SC can amplify the shared error into a confident wrong majority. This "high-agreement wrong answer" failure mode is especially risky in settings where the model's consensus is used as a reliability signal.

Human problem solvers rarely treat repeated reasoning in the same direction as sufficient verification. Instead, they often use complementary checks that are intentionally different from the original reasoning direction: attempting to falsify the solution by searching for a counterargument, and verifying the proposed answer by plugging it back into the explicit constraints of the problem statement. This paper asks whether these habits can be approximated at prompt time using only a single base LLM, without training a separate verifier or using external tools.

We propose Falsification-Augmented Bidirectional Self-Consistency (FAB-SC), a prompt-only decoding and aggregation procedure that extends SC with two structured checks per forward sample. First, a backsolve or plug-back check conditions on the proposed final answer and forces the model to verify constraint satisfaction rather than re-solve freely. Second, an adversarial falsification ensemble runs several short "refuter" prompts that explicitly try to find concrete flaws (misread quantities, invalid algebra, arithmetic slips, or missing constraints) and output an estimated probability that the solution is wrong. FAB-SC combines these two verification signals with an external, token-count-based length penalty designed to reduce the known confound between verbosity and self-scoring. The final answer is selected by an importance-weighted vote rather than an unweighted majority.

We evaluate FAB-SC as a drop-in replacement for SC under a matched forward-sampling budget using GPT-4o. The experiments are inference-only pilot runs on two standard arithmetic word-problem benchmarks used in prior CoT and SC work: GSM8K and SVAMP [**? ?**]. To control cost, we follow the run configurations and evaluate on prefixes of the official test splits: the first 200 GSM8K test items and the first 300 SVAMP test items. The results show mixed outcomes. On GSM8K ($N = 200$), FAB-SC underperforms SC (0.785 vs 0.810). On SVAMP ($N = 300$), FAB-SC slightly improves over SC (0.853 vs 0.843). Across both datasets, FAB-SC incurs substantial additional wall-clock runtime, about $18\times$ in these runs, reflecting the additional verification calls per forward sample.

These findings sharpen the empirical question underlying robust inference-time aggregation: prompt-only verification using the same base model may provide useful signals on some distributions, but it can also mis-rank candidates and negate the benefits of self-ensembling on others. Equally importantly, the compute overhead is large, making compute-aware evaluation essential.

Our contributions are:

- **FAB-SC procedure.** We propose FAB-SC, a prompt-only augmentation to self-consistency that combines (i) a backsolve, constraint-based verification signal and (ii) a stochastic falsification ensemble into per-sample reliability weights for importance-weighted aggregation.
- **Reproducible inference implementation.** We provide an end-to-end inference implementation (Hydra-configured) that records forward rationales, verification outputs,

derived scores, and aggregated predictions, enabling reproducible analyses under matched forward-sampling budgets.

- **Controlled pilot evaluation.** We report controlled pilot results on GSM8K (first 200 test items) and SVAMP (first 300 test items) comparing FAB-SC to standard SC with the same forward-sampling budget, and we quantify the associated wall-clock runtime overhead.

Several follow-up directions are suggested by these pilots. First, evaluating the hypothesized mechanism directly requires metrics beyond aggregate accuracy, such as the rate at which FAB-SC corrects cases where SC yields a wrong majority. Second, the verification prompts and parsing format may need better calibration to avoid systematic self-verification biases. Third, given the substantial overhead, compute-reduction strategies such as early stopping for SC [**?**] become even more relevant when verification is introduced, motivating hybrids that adaptively allocate either forward samples or verification calls.

2. Related Work

Chain-of-thought prompting established that providing intermediate reasoning steps in the prompt can substantially improve multi-step performance on arithmetic reasoning benchmarks [**?**]. In our setting, CoT is not the endpoint but the substrate: FAB-SC assumes CoT-style forward reasoning traces exist and uses them as candidates for aggregation and verification. A key limitation of CoT prompting highlighted by subsequent practice is that fluent rationales may still contain subtle mistakes, so additional reliability mechanisms are needed when outputs are used in high-stakes settings.

Self-consistency is a prominent inference-time method that explicitly uses redundancy to improve reliability: it samples multiple diverse CoT completions and aggregates the final answers, most commonly by majority vote [**?**]. SC's implicit statistical intuition is that agreement across diverse reasoning paths is evidence of correctness. FAB-SC builds directly on SC but focuses on the specific regime where SC's agreement heuristic can fail: when the sampled set is not diverse in the relevant error dimensions and instead contains correlated mistakes. Under correlated errors, majority vote can be overconfident and systematically wrong.

A practical limitation of SC is its linear scaling of inference cost with the number of samples. Recent work therefore explores compute-aware variants, including early-stopping schemes that terminate sampling once agreement is deemed sufficient [**?**]. This line of work is complementary to FAB-SC in two ways. First, our method increases per-sample compute by adding verification calls, so any deployment-level viability depends even more strongly on compute management. Second, early-stopping ideas provide a concrete template for future FAB-SC variants: one can imagine stopping either forward sampling or verification when additional evidence is unlikely to change the weighted vote.

Finally, FAB-SC relates to prompt-only self-evaluation and critique methods in which the same model is asked to score or critique its own outputs. FAB-SC differs from an undifferentiated self-score by (i) enforcing a constrained backsolve procedure that is explicitly framed as constraint checking rather than free re-solving, and (ii) using multiple stochastic refuters that aim to produce negative evidence ("find a flaw") rather than a single confidence statement. Nevertheless, because FAB-SC still uses the same base model for both generation and verification, its scores can inherit shared biases, and our pilot results illustrate that such prompt-only verification is not uniformly beneficial even when designed to be complementary to forward solving.

3. Background

This work studies inference-time aggregation for short arithmetic word problems, following the evaluation paradigm commonly used in CoT and self-consistency studies [**?** **?**]. Each example consists of a natural-language question and a single numeric gold answer, and performance is measured by answer accuracy after deterministic extraction of a numeric prediction.

3.1. Problem setting and notation

Let x denote a word-problem question and let y^* denote its gold numeric answer. A base LLM is queried with a prompt that requests step-by-step reasoning and a final numeric answer. A single model completion contains a rationale r and an answer string from which we extract a numeric answer a via a deterministic parser. For a fixed question x, standard self-consistency draws m stochastic generations, yielding pairs (r_i, a_i) for $i \in \{1, \ldots, m\}$, and predicts the aggregated answer $\hat{a}$ as the most frequent value among $\{a_i\}$ (majority vote) [**?**].

3.2. Correlated errors and the limits of agreement

SC's empirical success is often attributed to diversity across sampled reasoning paths: if errors are uncorrelated, wrong answers are less likely to recur, while correct answers recur more frequently. In a single-model setting, however, the distribution over sampled rationales can exhibit structured correlations. Even with stochastic decoding, the model's highest-probability interpretations of an ambiguous statement, its preferred solution templates, and its characteristic arithmetic slips can recur across samples. Under such correlations, agreement becomes an unreliable proxy for correctness, and SC can converge to a wrong majority.

3.3. Prompt-time verification as additional evidence

A natural response is to attach to each candidate (r_i, a_i) an estimate of reliability computed at prompt time. FAB-SC uses two bounded scalar scores in $[0, 1]$. The backsolve score B_i is intended to measure whether the proposed answer a_i satisfies constraints implied by x when checked explicitly. The falsification score F_i is intended to measure how well the solution withstands targeted attempts to find a concrete error. Both signals are produced by the same base model used for forward sampling.

3.4. Length confounding and external regularization

Prompt-based self-evaluation can be influenced by superficial correlates of apparent quality, including fluency and verbosity. Because FAB-SC relies on model-generated scores, it adds an explicit length penalty L_i computed outside the model using token counts. This penalty is not a correctness guarantee; rather, it is a simple regularizer intended to reduce the influence of "verbosity-as-confidence" artifacts in the final weighted vote.

Taken together, these components shift aggregation from unweighted counting to importance weighting: rather than treating each forward sample as an equal vote, FAB-SC treats each sample as a vote weighted by an estimated reliability signal derived from constraint checking, survival under refutation, and a length-based penalty. The next section specifies the procedure precisely as implemented in our inference code.

4. Method

FAB-SC is an inference-time procedure that augments self-consistency with two verification channels and an external length-based regularizer, and then aggregates candidate answers by an importance-weighted vote. The method is prompt-only in the sense that it

uses the same base LLM for forward generation, backsolve verification, and falsification attempts; no fine-tuning or external verifier is introduced.

For each question x, FAB-SC performs the following steps.

Algorithm 1 Falsification-Augmented Bidirectional Self-Consistency (FAB-SC)

Input: question x, forward sample count m, refuter count k, exponents α, β, length parameters $\lambda_{\text{length}}, t_0$
Sample m forward CoT solutions; parse numeric answers to obtain candidates $\{(r_i, a_i)\}_{i=1}^{m}$
Discard candidates with unparsable a_i
for $i = 1$ to m **do**
  Backsolve-check prompt with (x, r_i, a_i); parse $B_i \in [0, 1]$
  Run k refuter prompts with (x, r_i, a_i); parse $\{p_{i,j}\}_{j=1}^{k}$ with $p_{i,j} \in [0, 1]$
  Compute falsification survival $F_i \leftarrow 1 - \frac{1}{k}\sum_{j=1}^{k} p_{i,j}$
  Compute length penalty $L_i \leftarrow \exp\big(-\lambda_{\text{length}}\, t_i / t_0\big)$
  Compute weight $w_i \leftarrow (B_i^{\alpha})(F_i^{\beta})L_i$
end for
Aggregate by weighted vote: $\hat{a} \leftarrow \arg\max_a \sum_{i:a_i=a} w_i$
Output: aggregated answer $\hat{a}$

4.1. Forward solve sampling

We construct a CoT prompt that asks the model to solve the problem step by step. We sample m forward completions using stochastic decoding, obtaining texts that contain a rationale r_i and (after parsing) a numeric answer a_i. This matches the standard SC setup except that FAB-SC will later weight samples rather than count them equally [**?**]. Samples whose completions do not yield a parsable numeric answer are discarded.

4.2. Backsolve (plug-back) verification

For each remaining pair (r_i, a_i), we create a verification prompt that includes the original question x, the proposed numeric answer a_i, and the proposed rationale r_i as context. The prompt instructs the model not to re-solve freely, but to (i) identify key constraints from the problem statement, (ii) plug the proposed answer into those constraints, and (iii) recompute derived quantities to ensure consistency. The model is required to return a numeric field "CONFIDENCE" between 0 and 1 and a "VERDICT" field (PASS or FAIL). We parse the CONFIDENCE value as B_i, clipping to $[0, 1]$ if needed. In the current implementation, backsolve decoding uses a lower temperature (0.3) than forward sampling to reduce variability.

4.3. Adversarial falsification ensemble

For each (r_i, a_i), we run k short refuter generations. Each refuter prompt puts the model in the role of a critical reviewer and asks it to find a flaw, focusing on common error categories such as misread quantities, unit mismatches, arithmetic slips, invalid assumptions, or missing constraints. To encourage diversity across refuters without changing the underlying task, refuters cycle through three emphasis instructions: checking for unit/quantity misreads, checking algebra or arithmetic, and checking whether all constraints were considered. Each refuter output must include an "ERROR_PROB" between 0 and 1 and a one-sentence "REASON". We parse the error probabilities $p_{i,1}, \ldots, p_{i,k}$ and define the falsification survival score as

$$F_i = 1 - \frac{1}{k}\sum_{j=1}^{k} p_{i,j}.$$

Thus F_i is higher when refuters assign low error probability and lower when refuters find convincing reasons the solution may be wrong. Refuter decoding uses stochastic sampling (temperature 0.7) to promote diverse critiques.

4.4. Anti-verbosity length penalty

Let t_i be the token count of the forward rationale text r_i as computed by the model's tokenizer. FAB-SC computes an external penalty

$$L_i = \exp\big(-\lambda_{\text{length}}\, t_i / t_0\big).$$

This term monotonically decreases with rationale length. In the executed configuration, $t_0 = 200$ tokens and $\lambda_{\text{length}} = 0.001$.

4.5. Importance-weighted aggregation

FAB-SC combines the three signals into a nonnegative reliability weight

$$w_i = (B_i^{\alpha})(F_i^{\beta})L_i,$$

where α and β are nonnegative exponents controlling the influence of B-check and F-check signals. FAB-SC then aggregates answers by summing weights for each distinct candidate answer value a and selecting the answer with maximal total weight:

$$\hat{a} = \arg\max_{a} \sum_{i:a_i=a} w_i.$$

When backsolve and falsification are disabled and L_i is constant across samples, this reduces to unweighted majority vote. The intended benefit is robustness to correlated wrong-majority behavior: if a high-frequency wrong answer tends to fail constraint checks (low B_i) or is easily refuted (low F_i), it should receive lower total weight than a lower-frequency but more reliable answer.

4.6. Implementation and logging

Our Hydra-configured inference script executes these steps per question via API calls, saves per-question results in JSONL (including forward rationales, verification responses, scores, and weights), and logs cumulative accuracy to Weights and Biases (WandB). This design makes it possible to compute additional diagnostics post hoc from saved artifacts, even though the present pilot reports only accuracy and runtime summary statistics.

5. Experimental Setup

We evaluate FAB-SC as an inference-only replacement for standard self-consistency on grade-school arithmetic word problems. The experiments are designed to isolate the effect of adding prompt-time verification and importance weighting, holding constant the base model and the forward-sampling budget.

5.1. Datasets and evaluation slices

We use GSM8K (main configuration) and SVAMP, both widely used for arithmetic word-problem evaluation in the CoT and SC literature [**? ?**]. Datasets are loaded using the HuggingFace datasets library. For GSM8K, gold answers are extracted from the dataset's canonical answer format by matching the "`#### X`" pattern; for SVAMP, gold answers are read as numeric values. To keep the verification-heavy method computationally feasible, we follow the run configurations and evaluate dataset prefixes: the first 200 questions of the GSM8K test split and the first 300 questions of the SVAMP test split.

5.2. Model and prompting

All runs use GPT-4o (API model identifier `gpt-4o-2024-08-06`) as the base LLM for both generation and verification. Forward prompts ask for step-by-step solutions. Backsolve prompts ask for constraint-based verification of a proposed answer, returning CONFIDENCE and PASS/FAIL. Falsification prompts ask a critical reviewer to identify flaws and return ERROR_PROB and a one-sentence reason.

5.3. Methods compared

We compare two inference-time aggregation methods.

5.4. Self-Consistency baseline

For each question, sample m forward CoT solutions and return the most frequent extracted answer (majority vote), following the standard SC procedure [**?**].

5.5. FAB-SC

Sample the same m forward solutions, compute per-sample backsolve and falsification signals, apply a length penalty, and aggregate answers by summing importance weights.

5.6. Decoding budgets and hyperparameters

Both methods use the same forward-sampling budget $m = 8$ completions per question, with temperature 0.7 and `max_tokens` $= 512$ per forward completion. For FAB-SC, backsolve verification is enabled with one call per forward sample (temperature 0.3, `max_tokens` $= 512$). Falsification is enabled with $k = 3$ refuter calls per forward sample (temperature 0.7, `max_tokens` $= 256$). Aggregation hyperparameters are fixed to $\alpha = 1.0$, $\beta = 1.0$, $\lambda_{\text{length}} = 0.001$, and $t_0 = 200$.

5.7. Answer extraction and correctness

Model outputs are mapped to numeric answers with a deterministic extractor that searches for common answer patterns ("The answer is X", "Therefore X", "`#### X`", "Final answer: X"), and otherwise falls back to the last number in the completion. Extracted answers are stored as floats. Inference correctness in the executed script is computed via an absolute tolerance check: a prediction is marked correct if $|\text{predicted} - \text{gold}| < 10^{-6}$. This criterion is applied identically to FAB-SC and SC.

5.8. Logging and artifacts

The inference script logs per-question correctness and cumulative accuracy to WandB, and writes all per-question data to a `results.jsonl` file. A separate evaluation script fetches WandB run summaries (or local metrics as fallback) and generates comparison plots. Reported runtimes are the wall-clock runtimes recorded in the WandB run summaries for the full end-to-end evaluation under each method and dataset slice.

6. Results

This section reports the pilot results logged by the executed runs for FAB-SC and the SC baseline on GSM8K (first 200 test examples) and SVAMP (first 300 test examples). All results use GPT-4o with a matched forward-sampling budget of $m = 8$. We report accuracy, counts of correct predictions, and end-to-end wall-clock runtime as recorded in the run summaries.

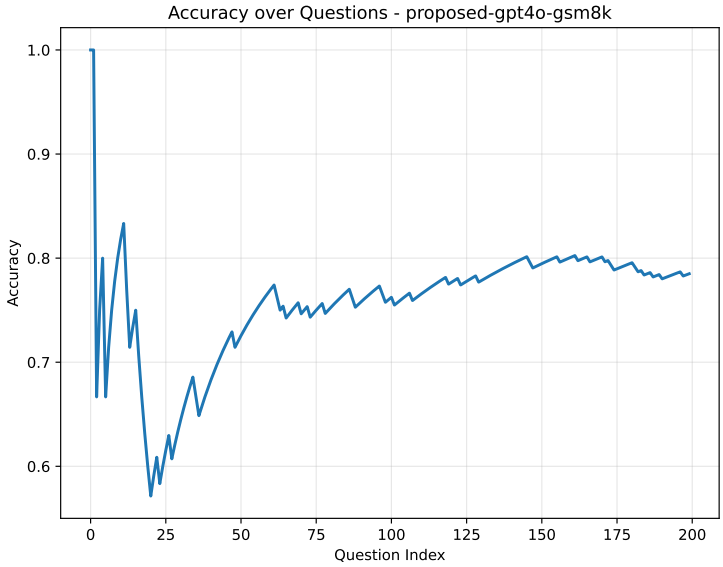

Figure 1. Cumulative accuracy over evaluation questions for an individual run; higher values indicate better performance.

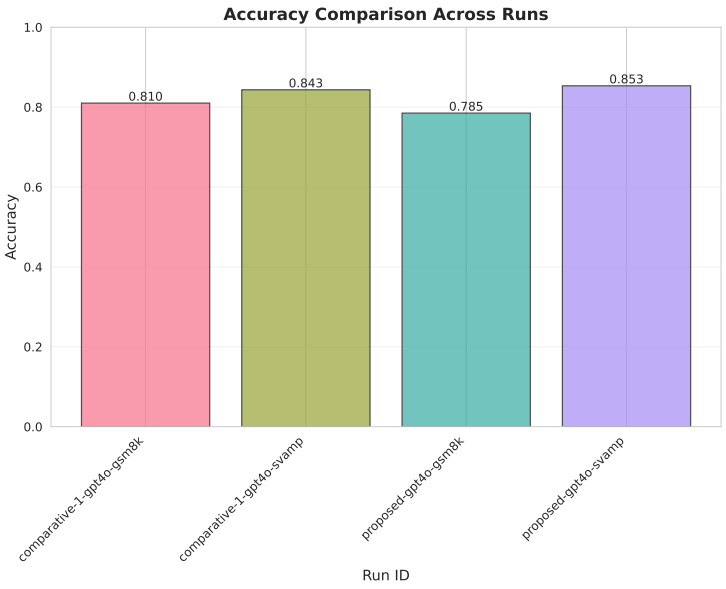

Figure 2. Accuracy comparison across runs; higher values indicate better performance.

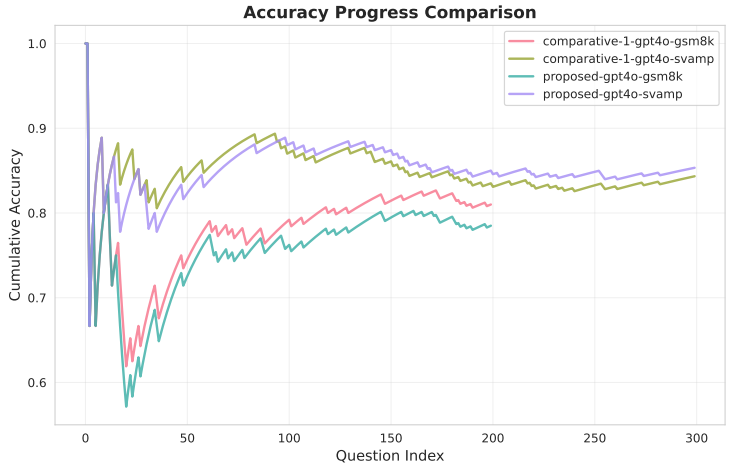

Figure 3. Overlay of cumulative accuracy progress across runs; higher values indicate better performance.

6.1. GSM8K pilot

Under SC majority vote, accuracy is 0.810, corresponding to 162 correct out of 200. Under FAB-SC, accuracy is 0.785, corresponding to 157 correct out of 200. The absolute difference (FAB-SC minus SC) is -0.025, which equals 5 fewer correct predictions in this slice. The evaluation pipeline logs cumulative accuracy across the question index; an example of this trace is provided in Figure 1, and the overlay view used for comparison is shown in Figure 3.

6.2. SVAMP pilot

Under SC, accuracy is 0.8433, corresponding to 253 correct out of 300. Under FAB-SC, accuracy is 0.8533, corresponding to 256 correct out of 300. The absolute difference is $+0.0100$, which equals 3 additional correct predictions in this slice. The across-run bar summary in Figure 2 reflects this small improvement.

6.3. Runtime overhead

FAB-SC substantially increases end-to-end evaluation runtime relative to SC. On GSM8K, the SC run reports a runtime of 1,164 seconds, while FAB-SC reports 20,587 seconds, approximately 17.7$\times$ slower. On SVAMP, SC reports 1,548 seconds, while FAB-SC reports 27,764 seconds, approximately 17.9$\times$ slower. This overhead is consistent with FAB-SC's call structure: for each of the m forward samples, FAB-SC adds one backsolve verification call and k falsification calls, so the number of model invocations per question increases from m (SC) to $(1+1+k)m$ (FAB-SC); with $k=3$, this equals $5m$ invocations.

6.4. Interpretation and limitations supported by the logged metrics

The pilots demonstrate that adding prompt-time verification to SC is not uniformly beneficial under the fixed prompts and hyperparameters used here. FAB-SC slightly improves SVAMP accuracy but degrades GSM8K accuracy, while introducing a large runtime cost. The current logged summaries focus on aggregate accuracy and runtime; they do not yet include mechanism-level diagnostics such as the rate of correcting wrong-majority SC outcomes, the distribution of per-sample weights w_i for correct versus incorrect candidates, or compute-normalized accuracy measured in generated tokens. Consequently, within this pilot evidence, the main supported conclusions are (i) FAB-SC can change accuracy in either direction depending on dataset, and (ii) the compute cost increase is large and must be justified by sufficiently consistent accuracy gains or by future compute-aware variants.

7. Discussion

The pilot results highlight both the potential and the challenges of prompt-only verification for self-consistency. While FAB-SC introduces theoretically motivated verification signals—backsolve constraint checking and adversarial falsification—its empirical outcomes depend critically on the interaction between the base model's self-verification capability, the specific problem distribution, and the hyperparameters governing aggregation.

On SVAMP, the small accuracy gain suggests that the verification signals can sometimes provide useful ranking information that corrects errors in the raw majority vote. However, on GSM8K, the accuracy decline indicates that the same signals can mis-rank candidates, potentially because the model's self-verification inherits the same biases that produced the original errors. This underscores a fundamental tension in prompt-only approaches: asking a model to critique its own reasoning may not break correlation structures in the error distribution if the critique itself relies on the same parametric knowledge and heuristics.

The substantial runtime overhead—approximately $18\times$ in these runs—raises important questions about the practical deployment of verification-augmented aggregation. Even if verification were to provide consistent accuracy gains, the compute cost would need to be justified against alternative uses of the same budget, such as sampling more forward solutions or using a stronger base model. This motivates the development of adaptive compute allocation strategies that can decide dynamically whether to invest in additional forward samples or in verification of existing candidates.

Several methodological improvements could address the limitations observed in these pilots. First, verification prompts could be calibrated on a held-out validation set to reduce systematic biases, for example by tuning the framing of the backsolve check or by learning prompt templates that better distinguish correct from incorrect solutions. Second, the aggregation formula could be extended to incorporate uncertainty estimates over the verification scores themselves, rather than treating them as deterministic signals. Third, hybrid methods that combine prompt-only verification with lightweight external tools (such as symbolic equation solvers for arithmetic problems) could provide more reliable constraint checks without requiring end-to-end training of a separate verifier.

Finally, the current evaluation focuses on aggregate accuracy and runtime, but does not yet quantify the hypothesized mechanism: the rate at which FAB-SC successfully corrects cases where SC produces a wrong majority. Computing this statistic from the saved per-question artifacts would provide direct evidence of whether the verification signals are functioning as intended, and would help diagnose whether accuracy changes are driven by fixing wrong majorities or by introducing new errors in cases where SC was already correct.

8. Conclusions

We examined a key weakness of self-consistency: treating agreement among sampled chain-of-thought solutions as evidence of correctness can fail when samples share correlated reasoning errors, producing a confident wrong majority. To address this failure mode without training an external verifier, we proposed FAB-SC, a prompt-only augmentation to SC that attaches two structured verification signals to each sampled solution. A backsolve plug-back check estimates whether the proposed answer satisfies explicit problem constraints, while an adversarial falsification ensemble estimates "survival under refutation" by attempting to find concrete flaws. FAB-SC combines these signals with an external length penalty into per-sample reliability weights and aggregates answers through importance-weighted voting.

Inference-only pilot experiments with GPT-4o under a matched forward-sampling budget ($m = 8$) show mixed outcomes. On SVAMP (first 300 test items), FAB-SC slightly improves accuracy from 0.8433 to 0.8533. On GSM8K (first 200 test items), FAB-SC reduces accuracy from 0.810 to 0.785. The method also increases end-to-end runtime by roughly $18\times$ in these runs, consistent with the additional verification calls per forward sample.

Taken together, the results suggest that prompt-only verification using the same base model is a delicate intervention: it can provide useful signals on some distributions but can also mis-rank candidates and negate the benefits of self-ensembling, all while imposing substantial compute costs. Future work should prioritize (i) computing mechanism-level diagnostics from saved artifacts (for example, how often FAB-SC overturns an incorrect SC majority), (ii) calibrating and stress-testing verification prompts to reduce systematic self-verification bias, and (iii) integrating compute-aware policies, including early-stopping ideas developed for SC [**?**], to make verification-augmented aggregation practically viable.

Author Contributions: For research articles with several authors, a short paragraph specifying their individual contributions must be provided. The following statements should be used: "Conceptualization, X.X. and Y.Y.; methodology, X.X.; software, X.X.; validation, X.X., Y.Y. and Z.Z.; formal analysis, X.X.; investigation, X.X.; resources, X.X.; data curation, X.X.; writing—original draft preparation, X.X.; writing—review and editing, X.X.; visualization, X.X.; supervision, X.X.; project administration, X.X.; funding acquisition, Y.Y. All authors have read and agreed to the published version of the manuscript." Please turn to the CRediT taxonomy for the term explanation. Authorship must be limited to those who have contributed substantially to the work reported.

Funding: This research received no external funding.

Data Availability Statement: All resources used in this study are openly available at

Acknowledgments: In this study, we automatically carried out a series of research processes—from hypothesis formulation to paper writing—using generative AI.

Conflicts of Interest: The authors declare no conflicts of interest.

. Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q.; Zhou, D. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models **2022**.
. Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.; Chi, E.; Narang, S.; Chowdhery, A.; Zhou, D. Self-Consistency Improves Chain of Thought Reasoning in Language Models **2022**.
. Ma, M.; Gong, C.; Zeng, L.; Yang, Y.; Wu, L. Escape Sky-high Cost: Early-stopping Self-Consistency for Multi-step Reasoning **2024**.