Extended Abstract

Motivation Large language models (LLMs) at test time typically generate a single response, but recent work has shown that investing extra compute by sampling multiple reasoning paths can substantially boost performance. Snell et al. (2024) In particular, the self-consistency strategy - sampling N reasoning paths and answers, then selecting the most common final answer - yields strong gains on many tasks. Wang et al. (2023) However, its underlying mechanisms remain unexplored; it's unknown whether self-consistency's success stems more from upweighting likely-to-appear reasoning traces or from leveraging diversity among reasoning traces, though the method's authors suspect the latter. Wang et al. (2023) Simultaneously, its efficacy has not been assessed in settings with high-dimensional outputs, like the *Countdown* task, where answers are arithmetic expressions whose equivalence is nontrivial to determine. We address three questions: **(1)** Does self-consistency improve performance in Countdown's high-dimensional answer space? **(2)** Which component - likelihood or diversity - drives its gains? **(3)** How does self-consistency compare to alternative test-time strategies, like a ground-truth verifier or stream-of-search? Gandhi et al. (2024)

Method We build on Qwen 2.5 0.5B Base, performing supervised finetuning (SFT) on the Warm-Start dataset, which imparts knowledge of the Countdown task, correct answer formatting, and a toolkit of reasoning strategies to think through the problem before generating a final answer.

From this SFT checkpoint, we run four test-time regimes. First, we characterize collisions, sampling N answers at temperature 0.6 and quantifying how often distinct answers "collide" (i.e., map to the same canonical form). To do this, we write a novel lightweight equivalentizer that normalizes expressions' parentheses, operator order, and whitespace. Second, we implement self-consistency, selecting the largest collision group as our final answer. We compare self-consistency to two other approaches: **(1)** a ground-truth verifier, which symbolically evaluates each sample and selects the first correct one, representing an upper bound on test-time performance, and **(2)** Stream of Search, for which we finetune SFT Qwen on solver trace sequences (state -> action -> next state) as in Gandhi et al. (2024). Being self-verifying but lacking access to ground truth, Stream of Search is a nice middle ground for comparison. Fourth, to test if self-consistency's strength comes from diversity in reasoning paths, we explicitly upweight answers stemming from more diverse reasoning paths. We resolve ties by using Bge-M3 embeddings to choose the most dispersed collision group of reasoning traces. We also implement diversity-incentivizing BEAM search to generate most diverse reasoning paths.

Finally, for the default project requirement, we implement RLOO on our SFT Qwen model.

Implementation & Results Evaluating on Countdown, we find performance improvements from test-time compute. With a single-sample baseline, our plaintext (SFT Qwen) model outputs the correct answer 34% of the time; by $N = 10$, a ground-truth verifier achieves 63% accuracy, while self-consistency reaches 46%, recovering almost half the possible test-time improvement (i.e., half the gap between single-sample and $N = 10$ ground-truth verifier performance). At one sample, Stream of Search underperforms the plaintext model, achieving 29% accuracy; but at $N = 2$ it surpasses self-consistency's performance and by $N = 10$ it achieves 52% accuracy. Diversity-incentivizing BEAM search degrades performance, but the Bge-M3 embeddings are very successful, correctly breaking ties 80% of the time as opposed to 50% with a random baseline. Surprisingly, improvement for all three test-time methods plateaus around $N = 7$. If the model cannot answer a problem correctly on the first try, it has a more than half chance of failing even given 9 further attempts.

Discussion & Conclusion Our results show that self-consistency remains useful even in high-dimensional output spaces, recovering half the possible test-time improvement with a suitable equivalentizer. Given that explicitly prioritizing diverse reasoning paths leads to better tie-breaking, our results suggest that diversity of reasoning traces plays an important role in the self-consistency effect, though further experiments are needed. Self-consistency on the plaintext model does underperform many-shot Stream of Search. Indeed, Stream of Search applied to our Qwen model essentially gains self-verification abilities, never once committing an arithmetic mistake, in stark contrast to the original paper on the smaller GPT-Neo model. Gandhi et al. (2024)

Refining and Expanding the Self-Consistency Strategy

Charlotte Nicks
Departments of Mathematics, Electrical Engineering
Stanford University
cnicks13@stanford.edu

Teddy Ganea
Departments of Mathematics, Classics
Stanford University
tganea@stanford.edu

Eli Myers
Department of Mathematics
Stanford University
epmeyers@stanford.edu

Abstract

We investigate the mechanisms and efficacy of the self-consistency approach - sampling N reasoning chains and voting on the most frequent answer - in the high-output-dimensional Countdown arithmetic task. Prior literature conjectures self-consistency's success stems from a **diversity bonus** (diverse, convergent reasoning paths signal correctness), but a **likelihood bonus** (more likely, common paths are simply higher quality) could also explain gains. To investigate whether self-consistency can handle high-dimensional output spaces, and to disentangle these effects, we finetune a Qwen-2.5-500M-Base model finetuned on the WarmStart dataset, then compare three test-time strategies: self-consistency; a ground-truth verifier that, picking the first symbolically correct sample, represents the upper bound of test-time compute capabilities; and a Stream-of-Search approach, where we finetune Qwen on solver traces in a structured, tree-based search approach to Countdown. Single-sample SFT achieves 34% accuracy; self-consistency rises to 46% at $N = 10$, recovering half the possible improvement from test-time compute; and Stream-of-Search peaks at 52%. We further introduce two diversity interventions - semantic tie-breaking for self-consistency via Bge-M3 embeddings and a diversity-incentivizing BEAM search - finding that the former boosts accuracy, correctly breaking ties in 80% of cases, suggesting the diversity bonus at play. All methods plateau by $N = 7$, highlighting diminishing returns on test-time compute. Our results confirm that self-consistency remains effective for high-dimensional output spaces, that diversity in reasoning paths aids performance, and that learned search strategies can outperform majority voting.

1 Introduction

Large language models (LLMs) at test time typically generate a single response, but recent work has shown that investing extra compute by sampling multiple reasoning paths can substantially boost performance. Snell et al. (2024) In particular, the self-consistency strategy - sampling N reasoning paths and answers, then selecting the most common final answer - yields strong gains on many tasks. Wang et al. (2023) However, its underlying mechanisms remain unexplored. The authors conjecture this is effective because when diverse sets of reasoning paths converge on the same answer, this indicates the answer is more likely to be correct, especially in quantitative domains. Wang et al. (2023) However, self-consistency could also be improving performance because it upweights likely-to-appear reasoning traces, which, being of higher probability in the model, might also be of higher quality. We term the former effect a **diversity bonus** and the latter a **likelihood bonus**.

Stanford CS224R 2025 Final Report

Simultaneously, its efficacy has not been assessed in settings with high-dimensional outputs, like the *Countdown* task, where answers are arithmetic expressions combining 4 numbers and 4 arithmetic operations, whose equivalences are nontrivial to determine, and where there is a combinatorial explosion in the number of unique answers possible. With so many answers possible, even with careful equivalentization, it is unclear *prima facie* that our model will produce sufficient "collisions" (i.e., repeat answers from reasoning traces) to yield any performance boost from the self-consistency strategy.

In this paper, we address three questions: **(1)** Does self-consistency improve performance in Countdown's high-dimensional answer space? **(2)** Which component - likelihood or diversity - drives its gains? **(3)** How does self-consistency compare to alternative test-time strategies, like a ground-truth verifier or stream-of-search?

We first finetune Qwen-2.5-500M-Base on the WarmStart dataset, which imparts knowledge of the Countdown task, correct answer formatting, and a toolkit of reasoning strategies to think through the problem before generating a final answer. Using this SFT Qwen model, trained to think in English plaintext before outputting its answer, we implement three test-time regimes: **(1)** self-consistency; **(2)** a ground-truth verifier approach, which symbolically evaluates each sample and selects the first correct one, and **(3)** Stream of Search, for which we finetune SFT Qwen on solver trace sequences (state -> action -> next state) as in Gandhi et al. (2024). We experiment with N ranging from 1 to 10. The ground-truth verifier represents an upper bound on test-time performance, since it only fails if none of the N model generations yield a correct answer. Meanwhile, Stream of Search is a nice middle ground for comparison to self-consistency, being self-verifying but lacking access to ground truth. (Indeed, our Stream-of-Search model never made an arithmetic mistake in all the generations we sampled, a difference from the 15% arithmetic mistakes of the original Stream-of-Search paper, which used a smaller model. Gandhi et al. (2024) This in effect made our Stream-of-Search self-verifying, because search paths either terminated successfully, or the model ran out of context length.)

Having tackled questions **(1)** and **(3)**, we address **(2)** by augmenting the role of diversity in reasoning paths, reasoning that if the **diversity bonus** plays a key role in self-consistency's performance, then strengthening it should further improve performance. We do so in two ways. We implement a diversity-incentivizing BEAM search to elicit more diverse reasoning paths from our SFT Qwen model. We also build upon the self-consistency algorithm to explicitly upweight diversity of reasoning paths in selecting answers; specifically, we break ties by embedding reasoning traces using the bge-M3 model and selecting the answer for which the reasoning traces are more dissimilar.

2 Related Work

Countdown and Stream-of-Search. Our setting is the Countdown task, a generalization of the 24 Game, where four numbers are combined using arithmetic operations to reach a target number. Gandhi et al. (2024) Countdown is challenging due to its combinatorial complexity, requiring careful planning, search, and backtracking. *Stream-of-search* (SoS) represents search processes as serialized strings and fine tunes models on search trajectories to autonomously navigate search trajectories. Using SoS, other authors have achieved 36% accuracy on the Countdown using GPT-Neo 250M. Gandhi et al. (2024)

Self-Consistency. The self-consistency strategy samples multiple chains of thought and selects the most frequent answer, improving accuracy by significant margins accross different tasks. The most extreme improvements are on arithmetic tasks, while multiple choice and yes-or-no tasks undergo smaller improvements. Wang et al. (2023)

Self-consistency was tested on tasks that have a much smaller solution space than the Countdown task (as the arithmetic tasks tend to have answers that are small, integer numbers). Additionally, self-consistency was initially designed for freeform reasoning, while SoS search trajectories are highly formal steps of tokenized reasoning. Gandhi et al. (2024) Wang et al. (2023) So, the investigation of self-consistency's performance in this new setting is a novel contribution.

Encouraging Diversity. Self-consistency seeks to obtain diverse reasoning pathways via high temperature and top-k sampling. We explore more sophisticated methods of encouraging diversity, since we want to elicit specifically diverse, good reasoning paths (changing temperature and top-k may compromise quality of reasoning paths).

3 Methods and Experiments

3.1 Supervised Finetuning

In supervised finetuning, three methodological approaches enabled us to achieve high performance, cumulatively taking our score from 0.06 to nearly 0.4. First was response parsing. We found that our model tended to fill all the available space in its context window with gibberish even after solving the problem, leading to its final answer being of lower quality. Taking the final answer after its first reasoning trace (demarcated by special <think> tags) improved score to 0.19. Second, we found that aggressive gradient accumulation greatly improved performance. Thirdly, after experimenting with many learning rate schedulers, we found that a consistently high $(3 * 10^{-5})$ and unchanging learning rate was key to maximizing model score. Note that the scores discussed in this section are not answer accuracy; following the scoring scheme in Gandhi et. al., we also assign a score of 0.1 to incorrect, but correctly-formatted, responses. Gandhi et al. (2025)

3.2 RLOO

For our RLOO implementation, we struggled with memory as we were only able to store one set of logprobs at a time (we were using a 40G A100). However, by performing a backpropagation step after each generation's logprob calculation, we were able to effectively mean over the generations without holding them all in memory at one time. We used 16 gradient accumulation steps and a $k = 8$.

3.3 Best of N Suitability

When we began our extension, we wanted to understand whether or not we could actually implement best of N sampling in the Countdown task. This algorithm relies on selecting the most popular answer out of a set of model generations. However, in the Countdown task, many different textual outputs can be considered equivalent. For example, we would want $(6 + 3) + 9$ to be counted as the same as $(9 + 3) + 6$. To do this, we first wrote an equivalence checker, which determined if two equations belonged to the same equivalence class. We did this by checking whether, if the numbers were replaced with python variables, the expressions were the same. We grouped answers in this way and sampled with temperature at 0.6.

3.3.1 Experiment 1

In our first experiment, we examined the size of the collisions (equivalence classes) that resulted from a growing N. We also measured the average and maximum group sizes. We did so by computing N samples (we used N from 1 to 10) on 100 different prompts.

3.3.2 Experiment 2

In our second experiment, we investigated how frequently there were any collisions as well as how frequently there were tied collisions. We did so by computing N samples (we used N from 1 to 10) on 100 different prompts.

3.4 Comparing Test-Time Compute Strategies

3.4.1 Stream of Search

We fine-tuned the base QWEN model on a stream-of-search database, following the methodology presented in Gandhi et al. (2024). We used random breadth first and depth first search trajectories as the training data. The main difference between our approach and the one in the paper was the fact that the paper trained a model from scratch and we fine-tuned a model that already had quite a bit of pretraining, so our training took many fewer examples and much less time and compute to complete.

We wanted to try combining the best of N algorithm with stream of search. However, as it turned out, in the *vast* majority of cases, stream of search's failure mode was not identifying the answer before running out of time (as the max length was capped) rather than producing an incorrect answer. Thus, were were nearly no collisions.

As a side note, if it were possible to get stream of search to produce incorrect answers, possibly by increasing the difficulty of moving between nodes, the it would be much easier to evaluate how similar two reasoning paths were (since computing the alignment of states gives a clear metric for how similar two search paths were).

However, stream of search still presented an interesting test-time compute strategy. We try running the stream of search algorithm N times. We select the first generation that actually leads to an output, and we extract the equation from the search trajectory. While we considered also running a single stream of search with the entire test time compute budget, we found that it was very common for the model to get stuck in an incomprehensible search loop. Running separate generations seemed to provide necessary diversification.

Because we extract and format the solution, it does not make sense to give credit for formatting. Thus, in this paper, we report only the percentage of times where the correct solution is identified rather than the average reward.

3.4.2 Verifier Selection

In this section, we implemented a verifier to check using our ground truth understanding (the numbers and the target) whether a solution was valid. We sampled N solutions from the model, and we selected the first solution where the verifier was able to confirm the solution's correctness. If it finds none, it selects a properly-formatted solution to at least score formatting points (on the leaderboard – this does not matter in the results we report here since we report accuracy rather than reward).

3.4.3 Best of N Selection

In this section, we wanted to understand how we can select a good sample without the ground truth verifier, since there are many applications for which this is not available. We sample N solutions, group them into equivalency classes as described above (or discard them if they do not contain a vaild equation), and select a random solution from the largest equivalence class, breaking ties randomly.

3.4.4 Experiment 3

We compared the accuracy that can be obtained by using each of the above methods, scaling N from 1 to 10.

3.5 Diversity Boosting

3.5.1 Experiment 4

In order to better understand the underlying drivers of the best of N algorithm's effectiveness, we wanted to see whether boosting diversity in sample generations could help improve the algorithm. The logic behind this was that if more diverse reasoning paths agree on an answer, it is perhaps better evidence for that answer than two similar reasoning paths giving the same answer.

We tried using a beam search sampling strategy to get our samples. We used 10 beams with a diversity penalty of 1 using HuggingFace sampling. This allowed us to produce a more diverse set of outputs.

3.5.2 Experiment 5

After experiment 4, we determined that it would be better to leave model sampling alone to ensure that we were still getting the highest possible generation quality.

However, it is still possible to give a bonus to diversity by selecting equivalence classes that have the greatest amount of diversity within their samples (rather than just the largest number of samples). This required that we come up with a way to measure how similar samples were, which we did by using a `bgem3` embedding to get a vector representing the model's output. Then, we took the sine dissimilarity between pairs of these embeddings to quantify how different the embeddings were from one another (this was because in the embedding space, direction matters more than magnitude). We summed the pairwise difference measurements in each group, and we selected the group that had the most diversity within it.

4 Results and Discussion

4.1 Best of N Suitability

4.1.1 Experiment 1

We found that with $N = 10$, we were able to get large groups of collisions (Figure 1). Note that by collision, we mean two generations that give equivalent answers.

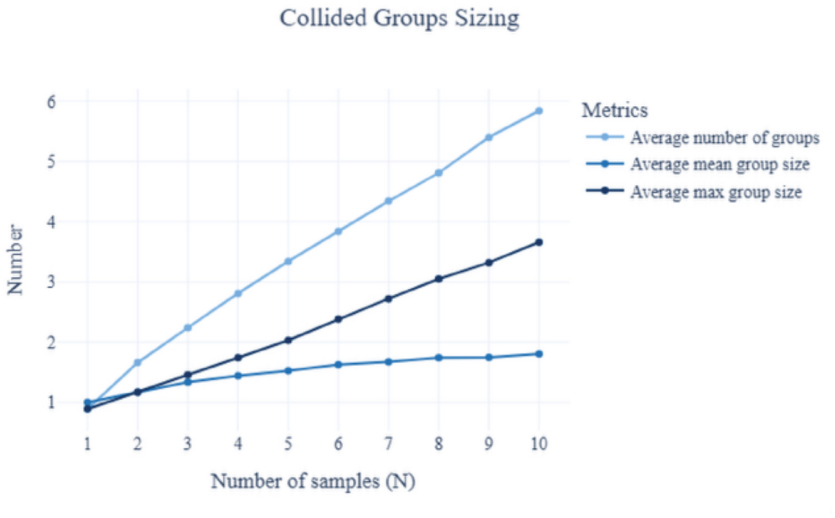

Figure 1: Group count and size growth as the number of generations N increases.

4.1.2 Experiment 2

Additionally, we found that at $N = 10$ there was a nontrivial number of ties, indicating that there were collisions between wrong answers as well (Figure 2). This means that selecting the largest group won't be infallible, making for an interesting environment to explore methods of selecting the correct group.

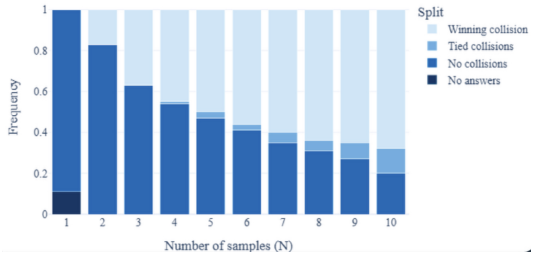

Figure 2: Nontrivial collisions are the norm after $N = 4$. Ties between collision groups become common around $N = 9$.

4.2 Comparing Test-Time Compute Strategies

4.2.1 Experiment 3

A comparison of the three test-time compute methods can be found in Figure 4. As one might expect, the verifier-based strategy performed best. The number of samples needed on average before stream of search actually found a solution (whether or not it was correct – though in the huge majority of cases, found solutions were correct, and we were only able to find one or two counterexamples across all experimentation) can be found in Figure 3.

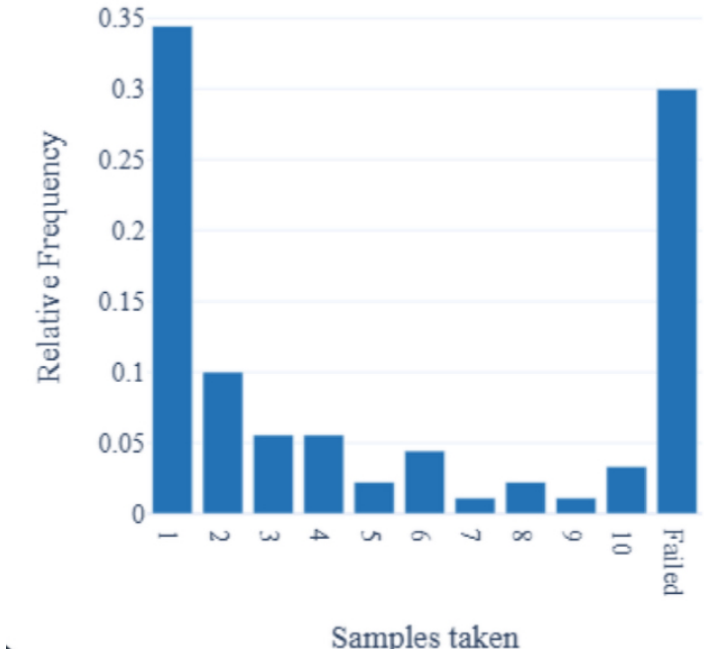

Figure 3: Distribution of the number of samples with max length of 1024 required before stream of search converges on an answer.

Stream of search performed better than best of N, however, this was also expected given that stream of search has undergone specialized fine tuning for a specific, well-tested method of solving this task. Best of N was actually able to recover about half of the verifier-based improvement (which represents the upper bound of performance for an algorithm selecting one of N samples from the base fine-tuned model). Overall, we were impressed by the strength of the best of N algorithm given its lack of ground truth knowledge or specialized fine tuning.

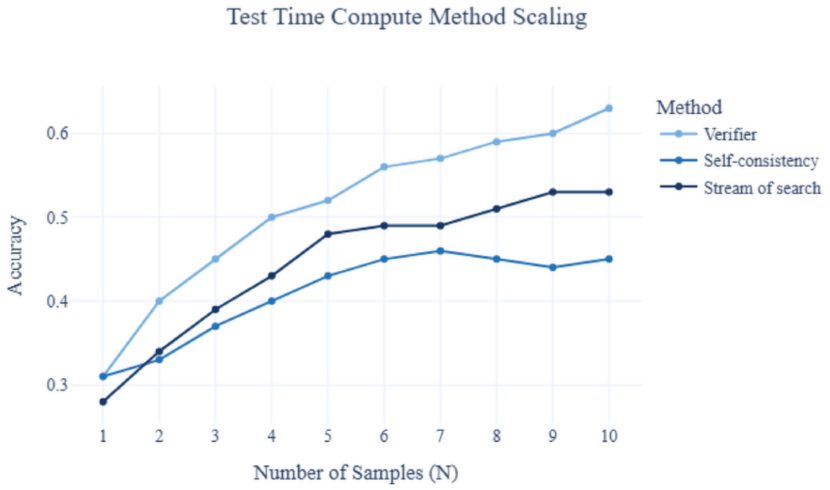

Figure 4: Accuracies of the three test-time compute methods over time.

4.3 Diversity Boosting

4.3.1 Experiment 4

When performing BEAM sampling, we found that overall, the model's generation quality was very poor due to the greedy algorithm. So, despite the increased diversity, the fact that the outputs were actually worse seemed to be too big of an issue to overcome. Ultimately, we decided not to proceed down this path and instead focused on the more promising embedding-based approach. However, we would be interested in trying to figure out how to sample stochastically while incorporating a diversity incentive in sampling to see if this would help.

4.3.2 Experiment 5

Diversity boosting in this fashion worked quite well, especially for tie-breaking. Indeed, while random tie-breaking selected the correct answer 50% of the time, tie-breaking by preferring the contending cluster with most sine-dissimilar reasoning traces achieved a 80% performance. Given how few ties there were, however, it is unclear to us whether this result is significant. Notably, this particular formulation did not change results much beyond tie-breaking; in our sampling, answers with less, but more diverse on average, reasoning traces never surpassed answers with more reasoning traces on this metric, probably because we were taking the sum over sine dissimilarities, the number of which increases quadratically with collision cluster size. More sophisticated mathematical formulae for upweighting diversity (perhaps the sum divided by the number of reasoning traces in a collision cluster) would yield even further performance improvements.

5 Conclusion

In summary, our experiments demonstrate that majority-vote self-consistency, when equipped with a good equivalentizer, remains a powerful, verifier-free strategy even in the high-dimensional output space of the Countdown task, recovering nearly half of the available test-time performance gain. Crucially, our semantic tie-breaking intervention - selecting more diverse reasoning clusters via Bge-M3 embeddings - suggests that a true **diversity bonus** underlies much of self-consistency's benefit. These results show that the self-consistency strategy generalizes unexpectedly well, and that modified versions of the self-consistency strategy could continue to yield performance improvements even in more complex, difficult-to-verify, reasoning domains. Future work on explicitly incorporating information about the diversity of reasoning traces into a modified self-consistency approach, as well as applying the self-consistency strategy to more complex reasoning domains, both quantitative and not (e.g., legal work or strategy games), could yield results of note.

A key limitation of this paper is that, though the results provide the first suggestion that the diversity bonus indeed plays a major role in the self-consistency effect as conjectured, we do not conclusively show this. Pitting the diversity and likelihood effects more directly against each other is another avenue of future work. Furthermore, characterization remains to be done, e.g., scrutinizing when the largest collision clusters contain very similar reasoning traces, or very diverse traces.

Lastly, this work contains two cautionary tales about the importance of pretraining. First is that the performance of all test-time methods plateau around $N = 7$ generations. For nearly two-fifths of Countdown question, it seems that our model, even given near-unlimited sample generation opportunities, might never land on the right answer; these problems seem to be fundamentally unsolvable by SFT Qwen. In particular, SFT Qwen consistently struggles with multiplication of large numbers; here, without better arithmetic pretrained, there might be a cap on test-time improvements, even given unlimited compute. Second is our experience with Stream of Search. We'd initially intended to apply self-consistency onto Stream of Search; but, probably due to our larger Qwen model, Stream of Search made no arithmetic mistakes in our observations, a qualitative change from results reported in the original paper. Gandhi et al. (2024) In other words, results from test-time compute might no longer generalize with better, or different, pretraining regimes, an important consideration.

6 Team Contributions

All work for this project up until week 9 was done in joint sessions where the three of us worked together. This includes all programming for the milestone along with evaluators, SFT model, dataloaders, and the first few experiments for the extension. At the beginning of week 9, Teddy sustained a serious injury and was temporarily unable to work at a computer. However, he was present and active during group coding sessions, workshopping ideas, debugging, and coming up with experiments. The writing for the proposal, milestone, and final report was also completed synchronously.

If we had to divide the work, we would say that the following individuals led the following things in terms of the default project. However, each person participated in all activities listed below.

- **Charlotte.** All dataloaders. SFT model code. Proposal and milestone (along with Teddy).
- **Eli.** SFT model hyperparameter tuning. RLOO model code.
- **Teddy.** Evaluation and sampling. Writing reports.

We don't think it is possible to break down who led what on the extension since everybody was very involved in everything.

Changes from Proposal In the proposal, we divided each task separately, which just was not how we ended up completing the project. Our schedules worked out such that meeting to complete all work was feasible, and we found this style of work to be most productive and enjoyable.

References

Kanishk Gandhi, Denise Lee, 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 . (2025). https://arxiv.org/pdf/2503.01307

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winston Cheng, Archit Sharma, and Noah D. Goodman. 2024. Stream of Search (SoS): Learning to Search in Language. *arXiv preprint arXiv:2404.03683v1* (2024). https://arxiv.org/pdf/2404.03683

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:2408.03314v1* (2024). https://arxiv.org/pdf/2408.03314

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *arXiv preprint arXiv:2203.11171v4* (2023). https://arxiv.org/pdf/2203.11171