SPIRAL: LEARNING TO SEARCH AND AGGREGATE

Anonymous authors
Paper under double-blind review

ABSTRACT

Language model reasoning can be substantially improved at test time via scaffolds that scale inference compute across different primitives—sequential reasoning within a trace, independently sampled parallel traces, and aggregation of multiple reasoning traces into a final response. During post-training, however, language models are optimized only for sequential reasoning within a single trace. We introduce SEQUENTIAL-PARALLEL-AGGREGATIVE REINFORCEMENT LEARNING (SPIRAL), a framework in which a language model is trained to use all three primitives end-to-end, as part of a unified inference compute pipeline. Concretely, the language model first samples a set of independent traces in parallel, each produced through sequential chain-of-thought reasoning, and then generates a final aggregation trace conditioned on those traces; all components are optimized end-to-end against the reward of the final aggregated response. To train this system, SPIRAL uses set reinforcement learning to teach models to produce a set of traces that are collectively useful for an aggregator and standard reinforcement learning to teach models to aggregate the set into improved final responses. Our experiments on reasoning tasks show that SPIRAL effectively scales with inference compute, outperforming GRPO by up to 11$\times$ scaling efficiency and 15% higher performance when all three compute primitives are scaled.

1 INTRODUCTION

Scaling inference compute is a powerful way to extend the capabilities of language models. This scaling can take several complementary forms: a model can reason for longer, sample many candidate traces in parallel, and synthesize information across these candidates into a final response. Compositions of these primitives underpin frontier systems for difficult reasoning and open-ended discovery, enabling language models to solve problems that would be inaccessible to a single short generation (Alon et al., 2026; Novikov et al., 2025; Lopopolo, 2026; Feng et al., 2026).

Yet, there is a striking mismatch between how inference compute is used and how models are trained. Standard reasoning RL optimizes a model to produce a single sequential reasoning trace whose final answer receives reward (Shao et al., 2024; Guo et al., 2025). At test time, however, this same model is routinely embedded within a substantially richer inference pipeline: multiple traces are sampled, candidates may be compared or filtered, and information from different attempts is aggregated into a final solution. The policies governing this computation—how different attempts should explore, what information they should expose to one another, and how their discoveries should ultimately be synthesized—are typically specified through prompting and hand-designed scaffolds rather than learned end-to-end.

An approach that better embraces the "Bitter Lesson" (Sutton, 2019) would instead bring the inference procedure itself into the learning loop, allowing models to end-to-end learn how additional inference computation should be allocated, coordinated, and synthesized. In this work, we take a step toward this goal by exposing a language model during training to three fundamental primitives of inference compute: *sequential compute* within an individual trace, including reasoning tokens and tool calls; *parallel compute* across independently sampled reasoning traces; and *aggregative compute*, in which the model synthesizes a set of candidate traces into a final output (Figure 1). Crucially, we optimize these primitives jointly against the reward of the final aggregated response. The resulting objective does not prescribe what role each parallel trace should play, what ideas should be explored across the set, or how those ideas should be combined. Instead, these behaviors can emerge insofar as they serve the ultimate goal of improving task performance.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

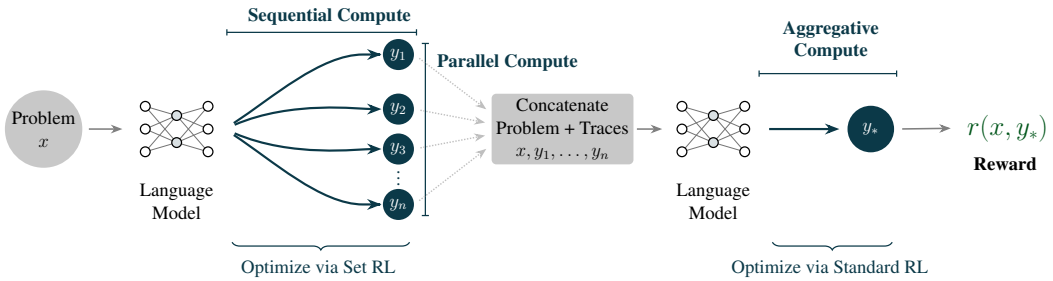


Figure 1: **Overview of SPIRAL.** SPIRAL trains a language model (LM) to use sequential, parallel, and aggregative inference compute end-to-end. Given an input x, we independently sample n parallel reasoning traces. The LM then synthesizes these into a final aggregation trace y_*. Using only the final reward $r(x, y_*)$, SPIRAL optimizes the parallel generations via set RL and the aggregation trace via standard RL. Consequently, the model jointly learns to generate traces that aggregate usefully and to synthesize them effectively.

We introduce SEQUENTIAL-PARALLEL-AGGREGATIVE REINFORCEMENT LEARNING (SPIRAL), a reinforcement learning framework for learning this sequential–parallel–aggregative inference pipeline end-to-end. To optimize the aggregation trace, SPIRAL applies standard reinforcement learning algorithms (Sutton et al., 1999), such as GRPO (Shao et al., 2024), to train the model to synthesize a set of candidate traces into an improved generation. Optimizing the parallel traces presents a different credit-assignment problem: the utility of an individual trace cannot be determined solely by whether it solves the task in isolation. A trace may instead contribute a useful intermediate idea, complementary approach, or piece of evidence that becomes valuable only when combined with the rest of the set. SPIRAL therefore leverages set reinforcement learning (Hamid et al., 2026; Orney et al., 2026), which assigns credit to parallel generations according to their marginal contribution to the downstream aggregation reward. Finally, we develop a scalable and practical training recipe whose sample complexity is comparable to standard RL for reasoning.

We empirically evaluate SPIRAL against standard reinforcement learning methods under an equal training-compute budget. Concretely, we fine-tune a `Qwen3-4B-Instruct-2507` using SPIRAL and compare against GRPO (Shao et al., 2024; Guo et al., 2025) on mathematical reasoning. At test time, we systematically scale different axes of inference compute and find that training on the full inference pipeline substantially improves the model's ability to benefit from additional compute. Under parallel compute scaling, SPIRAL achieves up to $11\times$ greater pass@k scaling efficiency on our test sets. When jointly scaling parallel and aggregative compute, SPIRAL achieves up to 15% higher performance. These results suggest that inference-compute scaling need not be treated solely as a test-time engineering problem: the strategies through which models search across and synthesize computation can themselves be learned.

2 PRELIMINARIES

Set Reinforcement Learning. Set reinforcement learning (set RL) (Hamid et al., 2026) is a framework that assigns a reward to a set of sampled actions, all of which are coupled under a shared learning signal. We refer the reader to (Orney et al., 2026) and (Hamid et al., 2026) for a discussion of the general formulation of set reinforcement learning; for language model training (and bandit settings), a practical formulation is obtained by applying a set-level objective over n i.i.d. generations from the same prompt. Let $y_{1:n} := (y_1, \ldots, y_n)$ be the multiset where each $y_i \overset{\text{i.i.d.}}{\sim} \pi_\theta(\cdot \mid x)$, and let $f(x, y_{1:n})$ be our set-level objective. The goal of set RL is to solve:

$$\max_\theta \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{y_{1:n}\sim\pi_\theta(\cdot|x)}[f(x, y_{1:n})]. \tag{1}$$

In this setting, a set-level reward is shared by all generations within the set. An example of such an objective is the polychromic objective defined as $f(x, y_{1:n}) = \frac{1}{n}\sum_{i=1}^{n} r(x, y_i) \cdot d(x, y_{1:n})$, where $d(x, y_{1:n})$ is a measure of the diversity of the generations (Hamid et al., 2026; Orney et al., 2026). This objective directly encourages the model to sample generations that balance exploration and

exploitation. In this paper, we will use a different objective under set reinforcement learning; in particular, our objective will not require access to any auxiliary objective or diversity function.

The objective in eq. (1) yields the following policy gradient:

$$\nabla_\theta \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{y_{1:n}\sim\pi_\theta(\cdot|x)}[f(x, y_{1:n})] = \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{y_{1:n}\sim\pi_\theta(\cdot|x)}[(f(x, y_{1:n}) - \hat{f}(x)) \sum_{i=1}^{n} \nabla_\theta \log \pi_\theta(y_i \mid x)] \tag{2}$$

where $\hat{f}(x)$ is a set-level baseline. The defining feature of eq. (2) is that all generations in the sampled set $y_{1:n}$ share the identical scalar learning signal, $f(x, y_{1:n}) - \hat{f}(x)$. Consequently, $\hat{f}(x)$ must be independent of any proper subset of $y_{1:n}$. This fundamentally differs from standard RL—where each generation receives its own advantage—because the objective f couples the samples, and the gradient assigns equal credit to all elements via the shared signal. Because this shared credit assignment must not be broken, techniques like leave-one-out estimators are strictly precluded.

General Recipe. Orney et al. (2026) propose a practical recipe for approximating the set RL gradient in eq. (2) by amortizing over a pool of samples. For a prompt x, first sample $N > n$ independent generations $y_{1:N} \sim \pi_\theta(\cdot \mid x)$. From this pool, construct subsets $G \subseteq \{y_1, \ldots, y_N\}$ of size n, either by enumerating all $\binom{N}{n}$ such subsets or by uniformly sampling K subsets without replacement. Each subset is scored using the set objective $f(x, G)$. Given the resulting collection $\mathcal{G}$ of evaluated subsets, define the set-level baseline $\hat{f}(x) = \frac{1}{|\mathcal{G}|}\sum_{G\in\mathcal{G}} f(x, G)$, and the corresponding set advantage of each set $A(x, G) = f(x, G) - \hat{f}(x)$. To obtain a learning signal for each sampled generation y_j, average the advantages of all evaluated subsets that contain it. Equivalently, letting $\mathcal{G}_j = \{G \in \mathcal{G} : y_j \in G\}$, define the marginal set advantage as $A_{\text{marg}}(x, y_j) = \frac{1}{|\mathcal{G}_j|}\sum_{G\in\mathcal{G}_j} A(x, G)$. This marginal set advantage assigns credit to an individual generation according to the average utility of the sets in which it participates, while preserving the set-level structure of the objective. In this work, we use this recipe to train parallel candidate traces according to their collective usefulness for downstream aggregation.

3 Learning Sequential, Parallel and Aggregative Inference

We aim to train a language model to optimize its usage of three fundamental primitives of inference compute. Specifically, during training, we expose the model to the following primitives:

Sequential compute. Sequential inference refers to computation allocated within an individual trace, including intermediate reasoning tokens, self-correction, verification, revision, and tool calls. The model must learn to optimally use this compute within a single chain of thought and acquire useful behaviors such as decomposing problems, setting intermediate subgoals, rigorously developing ideas, verifying partial solutions, and backtracking when necessary (Gandhi et al., 2025).

Parallel compute. Parallel inference refers to independently sampled traces conditioned on the same problem. This primitive provides the model an opportunity to explore a diverse range of plausible approaches, conjectures, intermediate hypotheses, and solution paths that can be combined or refined later (Madaan et al., 2023; Brown et al., 2024; Snell et al., 2024). The model must learn for itself to search broadly across traces while developing each attempt sufficiently for downstream aggregation to recover useful information.

Aggregative compute. Aggregative inference refers to computation that conditions on multiple candidate traces along with the original problem to produce a final output. This primitive allows the model to inspect, compare, verify, refine, and synthesize information from previous generations. Through training, the model must learn to effectively utilize information within and across generations to output an optimal final response (Li et al., 2025b; Venkatraman et al., 2025). Crucially, when no candidate trace is promising, the model should learn to recognize this failure mode and make a fresh attempt rather than merely recombining poor solutions.

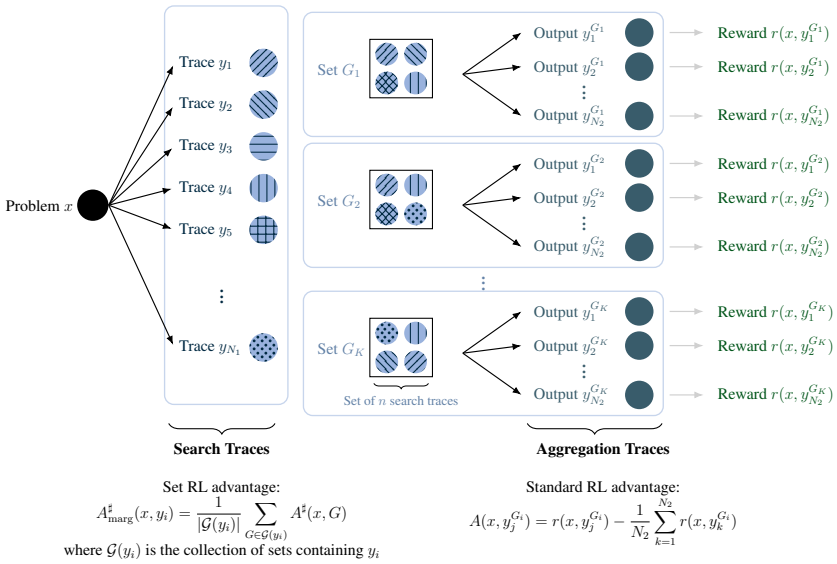


Figure 2: SPIRAL consists of two levels of generation. First, the model samples N_1 search traces, $y_1, \ldots, y_{N_1} \sim \pi_\theta(\cdot \mid x)$, from the problem x. It then uniformly samples K sets, each consisting of n search traces. For each set, it samples N_2 aggregation traces conditioned on both the problem and the set G_i, $y_1^{G_i}, \ldots, y_{N_2}^{G_i} \sim \pi_\theta(\cdot \mid x, G_i)$. Rewards are evaluated only at the end of this full inference compute pipeline compute pipeline, using the aggregation trace $r(x, y_j^{G_i})$.

3.1 PROBLEM FORMULATION

We formulate learning to effectively use inference compute as a reinforcement learning problem. For a given problem x, our objective is:

$$\max_{\theta,\phi} \mathbb{E}_{y_{1:n} \sim \pi_\theta(\cdot|x)} \left[\mathbb{E}_{y_* \sim \pi_\phi(\cdot|x, y_{1:n})} [r(x, y_*)] \right]. \tag{3}$$

Crucially, this objective only rewards the final output and our aim is to optimize the language model's performance across the entire inference compute pipeline (fig. 1). Optimizing this formulation requires the model to learn how to allocate sequential compute within a single chain of thought (i.e., each $y_i \sim \pi_\theta(\cdot \mid x)$), how to sample high-quality parallel traces that broadly explore the solution space (i.e., $y_{1:n} \sim \pi_\theta(\cdot \mid x)$), and how to aggregate these traces effectively to arrive at a correct answer (i.e., $y_* \sim \pi_\phi(\cdot \mid x)$). This formulation naturally permits using a different model for aggregation ($\phi \neq \theta$) or the same model for both stages ($\phi = \theta$).

Optimizing this objective via reinforcement learning allows the model to internalize general search strategies that subsume several rigid, hand-designed pipelines. Indeed, existing test-time scaling methods can be viewed as instantiations of the objective in eq. (3). For example, self-consistency (Wang et al., 2022) can be expressed as $\pi_\phi(y \mid x, y_{1:n}) = \mathbf{1}\{y = y_i \text{ for some } i \in \arg\max_j N(a(y_j))\}$, where $N(a) = \sum_{j=1}^{n} \mathbf{1}\{a(y_j) = a\}$ and $a(y_i)$ is the final answer extracted from y_i. Similarly, self-aggregation (Li et al., 2025b; Venkatraman et al., 2025) can be instantiated with $\pi_\phi = \pi_\theta$. However, optimizing this objective end-to-end teaches the model to proactively generate parallel traces that are highly useful for aggregation, while simultaneously mastering a more robust synthesis strategy.

Optimizing eq. (3) using standard reinforcement learning faces two immediate obstacles. First, the objective function is defined over a set of trajectories, whereas standard reinforcement learning utilizes an objective (i.e., the reward) defined over a single trajectory. Second, the objective function itself is possibly differentiable.

3.2 SPIRAL

In this section, we introduce our algorithm, SEQUENTIAL-PARALLEL-AGGREGATIVE REINFORCEMENT LEARNING (SPIRAL), for optimizing the following objective:

$$\mathbb{E}_{y_{1:n}\sim\pi_\theta(\cdot|x)}\left[\mathbb{E}_{y_*\sim\pi_\phi(\cdot|x,y_{1:n})}[r(x,y_*)]\right]. \tag{4}$$

Throughout the remainder of this paper, we will refer to the parallel generations $y_1,\ldots,y_n \sim \pi_\theta(\cdot \mid x)$ as the *search traces* and the final generation $y_* \sim \pi_\phi(\cdot \mid x, y_{1:n})$ as the *aggregation trace*.

We first introduce the recipe for when the same model is used to generate both the search traces and the aggregation trace (i.e., $\theta = \phi$). In §C, we discuss how our algorithm can also be extended to optimize two distinct models for the two levels of generation. The key insight we use is that the gradient of eq. (4) decomposes into a set RL gradient and a standard RL gradient. To see this, first, we define the following objective:

$$f_{\text{spiral}}(x, y_{1:n}) = \mathbb{E}_{y_*\sim\pi_\theta(\cdot|x,y_{1:n})}[r(x,y_*)]. \tag{5}$$

Now, we derive the policy gradient of eq. (4) as follows:

$$\begin{aligned}
&\nabla_\theta \mathbb{E}_{y_{1:n}\sim\pi_\theta(\cdot|x)}\left[\mathbb{E}_{y_*\sim\pi_\theta(\cdot|x,y_{1:n})}[r(x,y_*)]\right] \\
&= \underbrace{\mathbb{E}_{y_{1:n}\sim\pi_\theta(\cdot|x)}\left[f_{\text{spiral}}(x,y_{1:n})\sum_{i=1}^{n}\nabla_\theta \log \pi_\theta(y_i \mid x)\right]}_{\text{Set RL gradient}} \\
&\quad + \underbrace{\mathbb{E}_{y_{1:n}\sim\pi_\theta(\cdot|x)}\left[\mathbb{E}_{y_*\sim\pi_\theta(\cdot|x,y_{1:n})}\left[r(x,y_*)\nabla_\theta \log \pi_\theta(y_* \mid x, y_{1:n})\right]\right]}_{\text{Standard RL gradient}}.
\end{aligned} \tag{6}$$

These two gradient terms emerge naturally because we are jointly optimizing both the search traces within a set and the aggregation trace conditioned on that set. The set RL gradient updates the policy to generate an optimal set $y_{1:n}$ that the policy can effectively aggregate. Simultaneously, the standard RL gradient updates the policy to reliably synthesize a given set of generations into the optimal aggregation solution, y_*. As such, this framework captures a co-evolutionary procedure. Next, we discuss a scalable algorithm for optimizing eq. (6).

On-policy Data Collection. During each data collection step, for each prompt $x \sim \mathcal{D}$, we first sample N_1 search traces: $y_1,\ldots,y_{N_1} \sim \pi_\theta(\cdot \mid x)$. Next, we construct sets of size n, following the set RL recipe in (Orney et al., 2026). In particular, we uniformly sample K sets without replacement, $G_1,\ldots,G_K$, from the collection of all $\binom{N_1}{n}$ unordered sets of size n. In other words, each set $G_i = \{y_{i,1},\ldots,y_{i,n}\}$ contains n unique generations and no two sets in $\{G_1,\ldots,G_K\}$ are identical. Finally, we sample aggregation traces: conditioned on each set G_i, we sample $y_1^{G_i},\ldots,y_{N_2}^{G_i} \sim \pi_\theta(\cdot \mid x, y_{i,1},\ldots,y_{i,n})$. Each aggregation trace is evaluated using the reward function $r(x, y_j^{G_i})$ for trace index $j \in \{1,\ldots,N_2\}$ and set index $i \in \{1,\ldots,K\}$. This data collection pipeline is illustrated in fig. 2.

Policy Updates. The policy gradient of SPIRAL relies exclusively on the reward assigned to the aggregation traces. Each search trace is optimized via set reinforcement learning, and each aggregation trace is optimized via standard reinforcement learning. To optimize the search traces, we apply the general set RL recipe proposed by (Orney et al., 2026). First, we score each set G_i under the empirical objective function: $\hat{f}_{\text{spiral}}(x, G_i) = \frac{1}{N_2}\sum_{j=1}^{N_2} r(x, y_j^{G_i}), \quad \forall i \in \{1,\ldots,K\}$. We construct a Monte Carlo estimate of the baseline score across sets: $\hat{f}_{\text{spiral}}(x) = \frac{1}{K}\sum_{i=1}^{K}\hat{f}_{\text{spiral}}(x, G_i)$. This allows us to compute the set advantage of each set G_i as $A^\sharp(x, G_i; f_{\text{spiral}}) = \hat{f}_{\text{spiral}}(x, G_i) - \hat{f}_{\text{spiral}}(x)$. Finally, we compute the marginal set advantage for each individual search trace:

$$A^\sharp_{\text{marg}}(x, y; f_{\text{spiral}}) = \frac{1}{|\mathcal{G}(y)|}\sum_{G\in\mathcal{G}(y)} A^\sharp(x, G; f_{\text{spiral}}), \tag{7}$$

where $\mathcal{G}(y) = \{G \in \{G_1,\ldots,G_K\} \mid y \in G\}$ is the collection of sets containing y.

To optimize the aggregation traces, we compute the standard RL advantage. The advantage of each aggregation trace is computed within its respective set, rather than across all sets:

$$A(x, y_j^{G_i}) = r(x, y_j^{G_i}) - \frac{1}{N_2} \sum_{k=1}^{N_2} r(x, y_k^{G_i}). \tag{8}$$

We observe that using a per-set baseline leads to significantly lower variance, which helps the model learn how to reliably aggregate traces.

Having computed the advantages for each generation across both levels, we can now plug the advantages into standard reinforcement learning algorithms to optimize the model, following the recipe proposed in (Orney et al., 2026). In our implementations, we substitute the standard advantage function in REINFORCE with importance sampling using Tinker (Lab, 2026). The pseudocode and additional implementation details for our final algorithm can be found in §B.

We show that our set RL gradient estimator is unbiased if we sample K sets from all $\frac{N!}{(N-n)!}$ possible ordered tuples of size n. We show this via the following proposition (the proof, following a similar strategy to (Orney et al., 2026), is in §D):

Proposition 3.1. *Fix a prompt x, and let $y_1, \cdots, y_N \overset{\text{i.i.d.}}{\sim} \pi_\theta(\cdot \mid x)$ be our independently sampled N generations and let $f : \mathcal{X} \times \mathcal{Y}^{\oplus n} \to \mathbb{R}$ be our set objective. Then,*

$$\mathbb{E}[\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(y_i \mid x) \widehat{A^\sharp_{\text{marg}}}(x, y_i; f)] = M \nabla_\theta \mathbb{E}_{y_{1:n} \sim \pi_\theta(\cdot|x)} N\big[f(x, y_{1:n})\big],$$

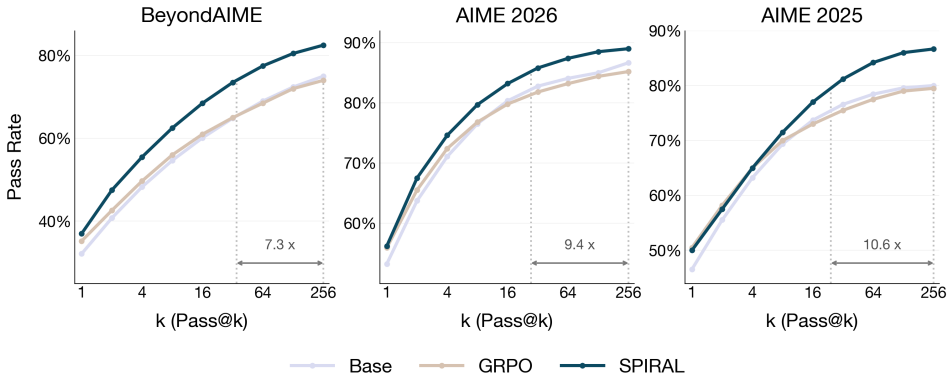
where $M \in \mathbb{R}_{>0}$ is a scaling factor. In particular, when we sample, uniformly, $K > 1$ sets without replacement from $K_{\text{all}} := \frac{N!}{(N-n)!}$ sets, we have that $M = \frac{N}{n} q_K - 1$, where $q_K := \Pr_{\mathcal{S}_K}(C_i(\mathcal{S}_K) > 0) = 1 - \frac{\binom{(N-1)_n}{K}}{\binom{(N)_n}{K}}$ and $(N)_n := \frac{N!}{(N-n)!}$. Consequently, after scaling the learning rate, the estimator is an unbiased estimator of the set RL gradient.

4 EXPERIMENTS

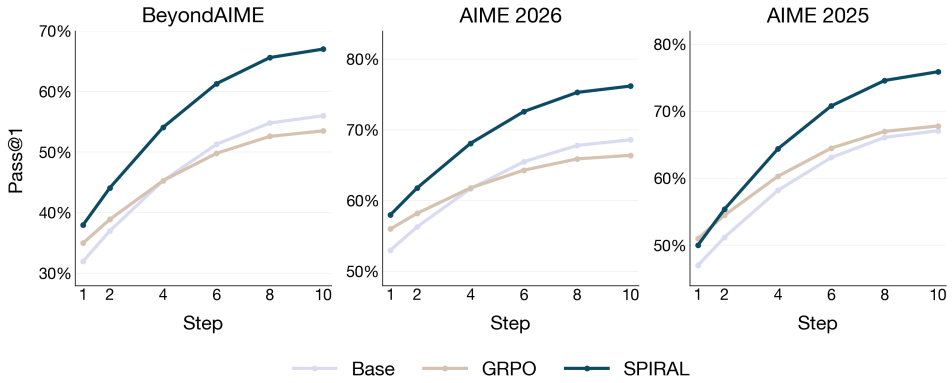
In this section, we empirically evaluate SPIRAL on reasoning tasks. Our goal is to study whether SPIRAL enables the model to use test-time compute more effectively. We use `Qwen3-4b-Instruct-2507` as our base model and train on a filtered subset of `POLARIS-53k` An et al. (2025), AceReason Chen et al. (2025b) and OmniMath Gao et al. (2024). We use 256 problems per batch, 24 rollouts per problem, 2 epochs of RL training, and dynamic sampling (Yu et al., 2025). We compare against GRPO (Shao et al., 2024; Guo et al., 2025; Liu et al., 2025). Both methods are compute matched, which, alongside other implementation details, we discuss in §B.

We first evaluate the methods' pass@k performance, which is summarized in 3a. In these evaluations, we independently sample k attempts per problem and evaluate the average number of problems the model can solve at least once. Note that in this setting, none of the models get the chance to aggregate their generations—we are only scaling independently sampled parallel inference compute. We observe that the pass@k performance of SPIRAL scales more strongly compared to GRPO, showing a gain of up to 11$\times$ higher efficiency.

The pass@k performance suggests that SPIRAL achieves higher effective diversity in its search traces. Recall that the search traces are optimized via set reinforcement learning, which rewards all generations in a set equally based on the model's ability to generate a correct aggregation trace when conditioning on that set. As such, the model is not encouraged to immediately collapse its entropy on a generation; search traces that are incorrect but still enable the model to craft high quality aggregation traces are explicitly encouraged by SPIRAL. In fact, as fig. 4 shows, the token-level entropy under SPIRAL does not collapse as readily as it does under GRPO. In fig. 5, we see that the average number of clusters of generations, where the clustering is done with respect to the similarity in strategy used as done in Orney et al. (2026), SPIRAL achieves higher notably higher diversity in model generations than GRPO. Such diversity in parallel samples is particularly helpful in cases where the model can use an oracle verifier (for example, LEAN) to verify each trace and



(a) **Pass@k evaluation on test sets.** The x-axis is the number of independent attempts, k, used in the evaluation and the y-axis is the coverage of the test set.



(b) **Pass@1 evaluation under recursive self-aggregation (Venkatraman et al., 2025).** The x-axis is the number of recursive self-aggregation steps used and the y-axis is the pass@1 rate.

return the optimal one. We now evaluate our model's ability to scale performance with aggregative test-time compute.

Recursive self-aggregation (RSA) (Venkatraman et al., 2025) is a test-time compute method that repeatedly converts parallel candidate traces into a new aggregated trace. The results are shown in fig. 3b. At the first level, we sample a population of independent candidate traces. At each subsequent level, we partition the traces from the previous level into groups, prompt the model with the original problem and the four candidate traces, and sample one aggregated solution for each group. We grade only the final aggregated traces at each level, yielding the pass@1 performance of the recursive aggregation pipeline as a function of the number of aggregation steps. In our experiments, we used a population size of 8 parallel traces at each step and set size 4. We find that SPIRAL benefits substantially more from recursive self-aggregation than both the base model and GRPO, achieving up to 13.5% higher performance. This suggests that SPIRAL learns search and refinement behaviors that are better suited to scaling parallel and aggregative inference compute at test time.

Finally, we make a comparison of the various inference compute methods — sequential compute, majority voting, and recursive self-aggregation — in terms of the efficiency in their usage of parallel threads and tokens. For sequential compute scaling, we simply increasing the maximum generation length up to the context length of the model. The majority@k performance is calculated by sampling k independent search traces and then grading the answer selected by a majority of the generations. The results are visualized in fig. 6a and fig. 6b. Sequential compute could be scaled only up to a certain limit due to the context length of the base model (i.e. `Qwen3-4b-Instruct-2507`). In our experiments on majority voting, the minimum number of traces we sampled is 4 and maximum number of tokens we allowed the traces to use is 16k, which is why majority voting requires at least a 64k token budget. Similarly, for recursive self-aggregation, we recursed over at least 1 step, leading to a minimum budget of 36k tokens. We observe that while sequential compute scaling can hit a wall,

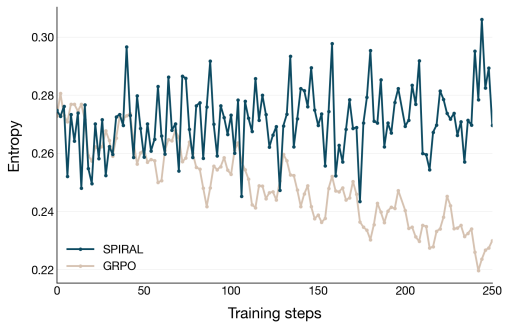


Figure 4: **Token-level entropy over training.** For SPIRAL, we plot the entropy over the search traces only to make the comparison to GRPO fair.

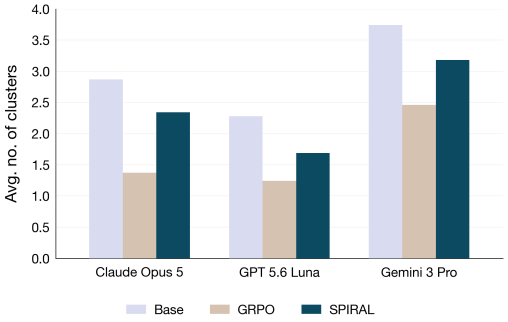


Figure 5: **Diversity in model generations.** The x-axis is the LLM-judge used while the y-axis is the average number of unique reasoning-strategy clusters across generations per prompt.

other methods that leverage parallel and aggregative compute like majority voting and recursive self-aggregation can be scaled for much longer with larger token budgets. Between these, recursive self-aggregation scales more favorably in terms of performance, showing that model-based aggregation can be more powerful than hand-designed rule based ones. Finally, SPIRAL scales significantly better under recursive self-aggregation than majority voting, suggesting that end-to-end learning of the primitives enable better performance.

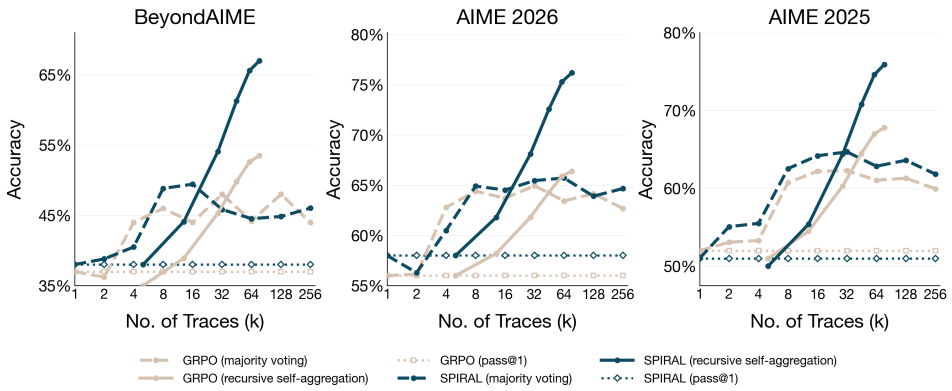
5 Related Work

Inference-time compute can be scaled either sequentially, by extending a model's chain of thought, or across multiple reasoning traces (OpenAI et al., 2026; Guo et al., 2025; Wang et al., 2022; Brown et al., 2024). Parallel approaches such as self-consistency and best-of-N sample multiple independent traces and reduce them to a final answer through voting or selection (Wang et al., 2022; Cobbe et al., 2021; Wang et al., 2024). More general aggregation methods instead condition on multiple traces and synthesize a new solution, allowing information from several candidate trajectories to be combined (Li et al., 2025b; Venkatraman et al., 2025; Teng et al., 2026; Schroeder et al., 2025). Other work introduces additional structure through tree search, pairwise comparison, evolutionary refinement, or task decomposition (Yao et al., 2023; Zhou et al., 2026; Zhang et al., 2025; Yang et al., 2025; Pan et al., 2025). Most of these methods, however, specify how parallel inference compute is used at test time rather than learning the entire inference procedure end-to-end.

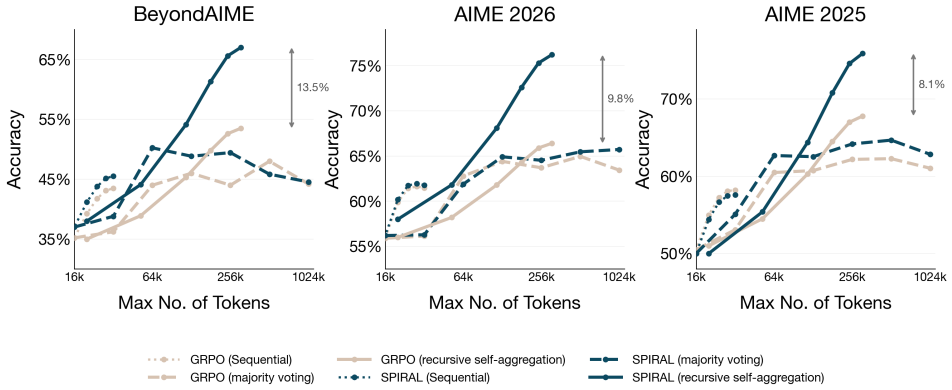
A growing line of work instead trains models to exploit parallel inference compute. ParaThinker trains a model to aggregate several independently generated traces through supervised fine-tuning (Wen et al., 2025), while PaCoRe and related approaches use reinforcement learning to improve the aggregation or synthesis stage while treating the input traces as fixed (Hu et al., 2026; Qi et al., 2025; Venkatraman et al., 2025). Related work also jointly trains generation and verification, but supplies separate learning signals to the different components (Singh et al., 2026). In contrast, SPIRAL trains the complete search-and-aggregation pipeline through the reward of the final synthesized solution: the quality of the generated set is therefore learned jointly with the model's ability to aggregate that set. This is enabled by set reinforcement learning, which provides learning signals to the parallel search traces through their collective contribution to the final output (Hamid et al., 2026; Orney et al., 2026). We provide a broader discussion of policy-gradient methods, inference-compute scaling, and related training approaches in §A.

6 Conclusion

We introduced SEQUENTIAL-PARALLEL-AGGREGATIVE REINFORCEMENT LEARNING (SPIRAL), a framework that combines set reinforcement learning with standard reinforcement learning to train a model to effectively use sequential, parallel, and aggregative inference compute. Our results suggest that jointly training a model to use these primitives enables it to better



(a) **Comparison of inference methods scaling parallel traces.** The x-axis is the number of parallel traces sampled. GRPO is not trained to aggregate, whereas SPIRAL trains a model to search and aggregate. Majority voting is a rule-based aggregation procedure whereas self-aggregation is model-based. We plot pass@1 value for reference.



(b) **Comparison of inference compute scaling and model pairs against token usage.** The x-axis is the maximum token budget provided to each model and inference compute method pair. Sequential compute scaling is up to 32k tokens due to the context length of our base model.

search, verify, and refine its generations, leading to improved scaling of performance at test time. Our work has several limitations. First, our experiments focus on domains with verifiable final rewards, and it remains unclear how effectively the framework extends to settings with sparse, noisy, or subjective feedback. Third, while SPIRAL learns to generate and aggregate sets of reasoning traces end-to-end, the overall inference structure—including the number of samples, aggregation stages, and connectivity between them—is specified in advance rather than learned adaptively. Developing methods that jointly learn both the reasoning policy and the inference-compute structure is an important direction for future work.

AI USE STATEMENT

In this work, we used generative AI tools to assist with implementing methods and creating or editing software code, as well as to edit the manuscript to improve readability and clarity. We also used generative AI tools to help identify potentially relevant literature.

We have not used generative AI tools to generate synthetic datasets; develop theoretical models or conceptual frameworks; formulate mathematical claims; provide critical ingredients for proving mathematical claims; assist in writing proofs; propose or refine hypotheses; design or provide feedback on research methodology or experiments; assist with translation; clean or reformat datasets; support qualitative or thematic data analysis; or interpret experimental results.

We have reviewed and tested all AI-assisted work. AI-assisted manuscript edits were reviewed by the authors, and suggestions were incorporated only at the authors' discretion. Literature identified

with the assistance of generative AI was independently checked against the original sources for relevance and accuracy. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Noga Alon, Thomas F. Bloom, W. T. Gowers, Daniel Litt, Will Sawin, Arul Shankar, Jacob Tsimerman, Victor Wang, and Melanie Matchett Wood. Remarks on the disproof of the unit distance conjecture, 2026. URL https://arxiv.org/abs/2605.20695.

Chenxin An, Zhihui Xie, Xiaonan Li, Lei Li, Jun Zhang, Shansan Gong, Ming Zhong, Jingjing Xu, Xipeng Qiu, Mingxuan Wang, and Lingpeng Kong. Polaris: A post-training recipe for scaling reinforcement learning on advanced reasoning models, 2025. URL https://hkunlp.github.io/blog/2025/Polaris.

Shalabh Bhatnagar, Richard S. Sutton, Mohammad Ghavamzadeh, and Mark Lee. Natural actor–critic algorithms. *Automatica*, 45(11):2471–2482, 2009. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2009.07.008. URL https://www.sciencedirect.com/science/article/pii/S0005109809003549.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456): 885–890, 2019. doi: 10.1126/science.aay2400. URL https://www.science.org/doi/abs/10.1126/science.aay2400.

Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025a.

Yang Chen, Zhuolin Yang, Zihan Liu, Chankyu Lee, Peng Xu, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Acereason-nemotron: Advancing math and code reasoning through reinforcement learning, 2025b. URL https://arxiv.org/abs/2505.16400.

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

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Thomas Degris, Martha White, and Richard S. Sutton. Off-policy actor-critic, 2013. URL https://arxiv.org/abs/1205.4839.

Tony Feng, Trieu H Trinh, Garrett Bingham, Dawsen Hwang, Yuri Chervonyi, Junehyuk Jung, Joonkyung Lee, Carlo Pagano, Sang-hyun Kim, Federico Pasqualotto, et al. Towards autonomous mathematics research. *arXiv preprint arXiv:2602.10177*, 2026.

Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D. Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars, 2025. URL https://arxiv.org/abs/2503.01307.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. Omni-math: A universal olympiad level mathematic benchmark for large language models, 2024. URL https://arxiv.org/abs/2410.07985.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Anthony GX-Chen, Ankit Anand, Gheorghe Comanici, Zaheer Abbas, Eser Aygün, David Smalling, Shibl Mourad, Doina Precup, André Barreto, and Mark Rowland. Using reward uncertainty to induce diverse behaviour in reinforcement learning, 2026. URL https://arxiv.org/abs/2606.03962.

Jubayer Ibn Hamid, Ifdita Hasan Orney, Ellen Xu, Chelsea Finn, and Dorsa Sadigh. Polychromic objectives for reinforcement learning, 2026. URL https://arxiv.org/abs/2509.25424.

Jingcheng Hu, Yinmin Zhang, Shijie Shang, Xiaobo Yang, Yue Peng, Zhewei Huang, Hebin Zhou, Xin Wu, Jie Cheng, Fanqi Wan, Xiangwen Kong, Chengyuan Yao, Kaiwen Yan, Ailin Huang, Hongyu Zhou, Qi Han, Zheng Ge, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Pacore: Learning to scale test-time compute with parallel coordinated reasoning, 2026. URL https://arxiv.org/abs/2601.05593.

Sham M Kakade. A natural policy gradient. In T. Dietterich, S. Becker, and Z. Ghahramani (eds.), *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001. URL https://proceedings.neurips.cc/paper_files/paper/2001/file/4b86abe48d358ecf194c56c69108433e-Paper.pdf.

Joongwon Kim, Wannan Yang, Kelvin Niu, Hongming Zhang, Yun Zhu, Eryk Helenowski, Ruan Silva, Zhengxing Chen, Srinivasan Iyer, Manzil Zaheer, Daniel Fried, Hannaneh Hajishirzi, Sanjeev Arora, Gabriel Synnaeve, Ruslan Salakhutdinov, and Anirudh Goyal. Scaling test-time compute for agentic coding, 2026. URL https://arxiv.org/abs/2604.16529.

Thinking Machines Lab. Tinker, 2026. URL https://thinkingmachines.ai/tinker/.

Yoonho Lee, Joseph Boen, and Chelsea Finn. Feedback descent: Open-ended text optimization via pairwise comparison, 2025. URL https://arxiv.org/abs/2511.07919.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses, 2026. URL https://arxiv.org/abs/2603.28052.

Dacheng Li, Shiyi Cao, Chengkun Cao, Xiuyu Li, Shangyin Tan, Kurt Keutzer, Jiarong Xing, Joseph E. Gonzalez, and Ion Stoica. S*: Test time scaling for code generation, 2025a. URL https://arxiv.org/abs/2502.14382.

Michael Y. Li, Jubayer Ibn Hamid, Emily B. Fox, and Noah D. Goodman. Neural garbage collection: Learning to forget while learning to reason, 2026. URL https://arxiv.org/abs/2604.18002.

Zichong Li, Xinyu Feng, Yuheng Cai, Zixuan Zhang, Tianyi Liu, Chen Liang, Weizhu Chen, Haoyu Wang, and Tuo Zhao. Llms can generate a better answer by aggregating their own responses, 2025b. URL https://arxiv.org/abs/2503.04104.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1509.02971.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective, 2025. URL https://arxiv.org/abs/2503.20783.

Ryan Lopopolo. Harness engineering: Leveraging codex in an agent-first world. https://openai.com/index/harness-engineering/, February 2026. OpenAI.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

Yuzhen Mao, Michael Y. Li, and Emily B. Fox. Forget, then recall: Learnable compression and selective unfolding via gist sparse attention, 2026. URL https://arxiv.org/abs/2604.20920.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. Skeleton-of-thought: Prompting llms for efficient parallel generation. In *International Conference on Learning Representations*, volume 2024, pp. 917–967, 2024.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich, Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Bohan Zhang, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian Osband, Ignasi Clavera Gilaberte, Ilge Akkaya, Ilya Kostrikov, Ilya Sutskever, Irina Kofman, Jakub Pachocki, James Lennon, Jason Wei, Jean Harb, Jerry Twore, Jiacheng Feng, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joaquin Quiñonero Candela, Joe Palermo, Joel Parish, Johannes Heidecke, John Hallman, John Rizzo, Jonathan Gordon, Jonathan Uesato, Jonathan Ward, Joost Huizinga, Julie Wang, Kai Chen, Kai Xiao, Karan Singhal, Karina Nguyen, Karl Cobbe, Katy Shi, Kayla Wood, Kendra Rimbach, Keren Gu-Lemberg, Kevin Liu, Kevin Lu, Kevin Stone, Kevin Yu, Lama Ahmad, Lauren Yang, Leo Liu, Leon Maksin, Leyton Ho, Liam Fedus, Lilian Weng, Linden Li, Lindsay McCallum, Lindsey Held, Lorenz Kuhn, Lukas Kondraciuk, Lukasz Kaiser, Luke Metz, Madelaine Boyd, Maja Trebacz, Manas Joglekar, Mark Chen, Marko Tintor, Mason Meyer, Matt Jones, Matt Kaufer, Max Schwarzer, Meghan Shah, Mehmet Yatbaz, Melody Y. Guan, Mengyuan Xu, Mengyuan Yan, Mia Glaese, Mianna Chen, Michael Lampe, Michael Malek, Michele Wang, Michelle Fradin, Mike McClay, Mikhail Pavlov, Miles Wang, Mingxuan Wang, Mira Murati, Mo Bavarian, Mostafa Rohaninejad, Nat McAleese, Neil Chowdhury, Neil Chowdhury, Nick Ryder, Nikolas Tezak, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Patrick Chao, Paul Ashbourne, Pavel Izmailov, Peter Zhokhov, Rachel Dias, Rahul Arora, Randall Lin, Rapha Gontijo Lopes, Raz Gaon, Reah Miyara, Reimar Leike, Renny Hwang, Rhythm Garg, Robin Brown, Roshan James, Rui Shu, Ryan Cheu, Ryan Greene, Saachi Jain, Sam Altman, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Santiago Hernandez, Sasha Baker, Scott McKinney, Scottie Yan, Shengjia Zhao, Shengli Hu, Shibani Santurkar, Shraman Ray Chaudhuri, Shuyuan Zhang, Siyuan Fu, Spencer Papay, Steph Lin, Suchir Balaji, Suvansh Sanjeev, Szymon Sidor, Tal Broda, Aidan Clark, Tao Wang, Taylor Gordon, Ted Sanders, Tejal Patwardhan, Thibault Sottiaux, Thomas Degry, Thomas Dimson, Tianhao Zheng, Timur Garipov, Tom Stasi, Trapit Bansal, Trevor Creech, Troy Peterson, Tyna Eloundou, Valerie Qi, Vineet Kosaraju, Vinnie Monaco, Vitchyr Pong, Vlad Fomenko, Weiyi Zheng, Wenda Zhou, Wenting Zhan, Wes McCabe,

Wojciech Zaremba, Yann Dubois, Yinghai Lu, Yining Chen, Young Cha, Yu Bai, Yuchen He, Yuchen Zhang, Yunyun Wang, Zheng Shao, and Zhuohan Li. Openai o1 system card, 2026. URL https://arxiv.org/abs/2412.16720.

Ifdita Hasan Orney, Jubayer Ibn Hamid, Shreya S Ramanujam, Shirley Wu, Hengyuan Hu, Noah Goodman, Dorsa Sadigh, and Chelsea Finn. Poly-epo: Training exploratory reasoning models, 2026. URL https://arxiv.org/abs/2604.17654.

Jiayi Pan, Xiuyu Li, Long Lian, Charlie Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer, and Alane Suhr. Learning adaptive parallel reasoning with language models, 2025. URL https://arxiv.org/abs/2504.15466.

Pouya Pezeshkpour and Estevam Hruschka. Large language models sensitivity to the order of options in multiple-choice questions. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 2006–2017, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-naacl.130. URL https://aclanthology.org/2024.findings-naacl.130/.

Jianing Qi, Xi Ye, Hao Tang, Zhigang Zhu, and Eunsol Choi. Learning to reason across parallel samples for llm reasoning, 2025. URL https://arxiv.org/abs/2506.09014.

Philip Schroeder, Nathaniel Morgan, Hongyin Luo, and James Glass. Thread: Thinking deeper with recursive spawning, 2025. URL https://arxiv.org/abs/2405.17402.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017a. URL https://arxiv.org/abs/1502.05477.

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

Omar Shaikh, Shardul Sapkota, Shan Rizvi, Eric Horvitz, Joon Sung Park, Diyi Yang, and Michael S. Bernstein. Creating general user models from computer use, 2025. URL https://arxiv.org/abs/2505.10831.

Omar Shaikh, Valentin Teutschbein, Kanishk Gandhi, Yikun Chi, Nick Haber, Thomas Robinson, Nilam Ram, Byron Reeves, Sherry Yang, Michael S. Bernstein, and Diyi Yang. Learning next action predictors from human-computer interaction, 2026. URL https://arxiv.org/abs/2603.05923.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Harman Singh, Xiuyu Li, Kusha Sareen, Monishwaran Maheswaran, Sijun Tan, Xiaoxia Wu, Junxiong Wang, Alpay Ariyak, Qingyang Wu, Samir Khaki, Rishabh Tiwari, Long Lian, Yucheng Lu, Boyi Li, Alane Suhr, Ben Athiwaratkun, and Kurt Keutzer. v_1: Unifying generation and self-verification for parallel reasoners, 2026. URL https://arxiv.org/abs/2603.04304.

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

Richard S. Sutton. The bitter lesson, March 2019. URL http://www.incompleteideas.net/IncIdeas/BitterLesson.html. Accessed: 2026-09-24.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'99, pp. 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

Fahim Tajwar, Guanning Zeng, Yueer Zhou, Yuda Song, Daman Arora, Yiding Jiang, Jeff Schneider, Ruslan Salakhutdinov, Haiwen Feng, and Andrea Zanette. Maximum likelihood reinforcement learning. In *Proceedings of the 43rd International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2026.

Yunhao Tang, Kunhao Zheng, Gabriel Synnaeve, and Rémi Munos. Optimizing language models for inference time objectives using reinforcement learning. *arXiv preprint arXiv:2503.19595*, 2025.

Fengwei Teng, Quan Shi, Zhaoyang Yu, Jiayi Zhang, Yuyu Luo, Chenglin Wu, and Zhijiang Guo. Atom of thoughts for markov llm test-time scaling. *Advances in Neural Information Processing Systems*, 38:74010–74040, 2026.

Siddarth Venkatraman, Vineet Jain, Sarthak Mittal, Vedant Shah, Johan Obando-Ceron, Yoshua Bengio, Brian R Bartoldson, Bhavya Kailkhura, Guillaume Lajoie, Glen Berseth, et al. Recursive self-aggregation unlocks deep thinking in large language models. *arXiv preprint arXiv:2509.26626*, 2025.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, 2024.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay, 2017. URL https://arxiv.org/abs/1611.01224.

Hao Wen, Yifan Su, Feifei Zhang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. Parathinker: Native parallel thinking as a new paradigm to scale llm test-time compute, 2025. URL https://arxiv.org/abs/2509.04475.

Xinyu Yang, Yuwei An, Hongyi Liu, Tianqi Chen, and Beidi Chen. Multiverse: Your language models secretly decide how to parallelize and merge generation, 2025. URL https://arxiv.org/abs/2506.09991.

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

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 15476–15488. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/639a9a172c044fbb64175b5fad42e9a5-Paper-Conference.pdf.

Yanzhi Zhang, Yitong Duan, Zhaoxi Zhang, Jiyan He, and Shuxin Zheng. Population-evolve: a parallel sampling and evolutionary method for llm math reasoning, 2025. URL https://arxiv.org/abs/2512.19081.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization, 2025. URL https://arxiv.org/abs/2507.18071.

Shang Zhou, Wenhao Chai, Kaiyuan Liu, Huanzhi Mao, Qiuyang Mang, and Jingbo Shang. Opendeepthink: Parallel reasoning via bradley–terry aggregation, 2026. URL https://arxiv.org/abs/2605.15177.

A Extended Related Works

Policy Gradient Methods. Policy gradient methods Sutton et al. (1999); Sutton & Barto (2018); Kakade (2001) are a foundational class of reinforcement learning algorithms that directly optimize a policy to maximize expected reward. Advances in variance reduction and sample efficiency Bhatnagar et al. (2009); Degris et al. (2013); Lillicrap et al. (2016); Wang et al. (2017); Schulman et al. (2017a;b) have made these methods widely adopted for language model (LM) fine-tuning. In the LM setting, policy-gradient methods often omit a learned critic and instead rely on empirical advantage estimates computed from sampled generations (Guo et al., 2025; Yu et al., 2025; Zheng et al., 2025; Liu et al., 2025; Chen et al., 2025a; Tajwar et al., 2026).

Inference Compute. Inference compute has a rich history in artificial intelligence. In game-playing systems such as AlphaGo and AlphaZero, inference compute is scaled through simulation-based search for Go, chess, and shogi Silver et al. (2017). Similarly, Brown & Sandholm (2019) scale search algorithms for poker. More recently, reasoning models Zelikman et al. (2022); Shao et al. (2024); OpenAI et al. (2026); Comanici et al. (2025) scale inference compute through thinking tokens, allowing models to deliberate before producing a final answer OpenAI et al. (2026); Guo et al. (2025), as well as to perform other internal actions Li et al. (2026); Mao et al. (2026); Shaikh et al. (2025; 2026). In this setting, inference compute is typically allocated sequentially within a single chain of thought.

A complementary approach scales inference compute across parallel traces Ning et al. (2024); Yao et al. (2023); Brown et al. (2024); Madaan et al. (2023); Kim et al. (2026); Li et al. (2025a). Self-consistency samples multiple reasoning traces in parallel and aggregates them by selecting the answer that appears most frequently among the samples (Wang et al., 2022). Best-of-N methods similarly sample several traces independently, but use a verifier or reward model to select the final output (Cobbe et al., 2021; Wang et al., 2024; Brown et al., 2024). Both self-consistency and best-of-N are filtering-based methods: they reduce a set of parallel traces to a single sampled output. Self-Refine and Reflexion instead scale inference compute sequentially, where a model first generates an attempt, critiques or reflects on that attempt, and then conditions on the resulting feedback to generate a revised solution (Madaan et al., 2023; Shinn et al., 2023). In contrast to filtering-based approaches, self-aggregation allows the model to condition on a set of parallel traces and synthesize a new generation Li et al. (2025b); Venkatraman et al. (2025); Teng et al. (2026); Schroeder et al. (2025); Lee et al. (2025). When a value function is available, tree-search methods can further allocate inference compute by filtering or expanding partial traces (Yao et al., 2023). OpenDeepThink introduces additional structure into parallel reasoning by aggregating traces through pairwise Bradley–Terry comparisons (Zhou et al., 2026). Several recent works also study parallel sampling followed by evolutionary refinement. For example, PopulationEvolve samples a population of parallel traces, evolves them over several steps using an evolution prompt, and then performs self-consistency-based answer extraction (Zhang et al., 2025). Other methods use feedback from the environment to support aggregation and refinement Lee et al. (2025; 2026).

These methods primarily operate at test time. During training, the model is typically optimized only for sequential chain-of-thought reasoning, leaving the use of parallel and aggregative inference compute to hand-designed test-time procedures.

Another line of work scales parallel thinking by decomposing a task into subtasks rather than by sampling independent solution traces. In these methods, the model decomposes an overall objective

into subproblems that can be solved in parallel. Yang et al. (2025) and Pan et al. (2025) train language models to decompose tasks into parallelizable subtasks, typically using supervised fine-tuning rather than reinforcement learning. Our work is complementary to this direction: rather than learning to decompose a problem into distinct subtasks, we train a model to generate and use sets of independently sampled reasoning traces.

Training for Inference Compute. The closest line of work studies how to train models to use parallel inference compute. Wen et al. (2025) consider a setup closely related to ours: a model first generates several independent reasoning traces in parallel, and then conditions on all of them to synthesize a final answer. However, they train this behavior through supervised fine-tuning rather than reinforcement learning. In our setting, set reinforcement learning is the key ingredient that allows the model to learn how to generate effective sets of parallel traces, rather than merely learning how to aggregate a fixed set of traces (Hamid et al., 2026; Orney et al., 2026). PaCoRe also trains models with reinforcement learning for inference-compute scaling, but constrains learning to the synthesis step and does not train the model to generate optimal sets of parallel traces (Hu et al., 2026). Similarly, Qi et al. (2025) and Venkatraman et al. (2025) train models to aggregate sets of traces effectively while treating the generation of the input set as fixed. Singh et al. (2026) train a single model to both generate and verify. However, their framework provides an independent learning signal to each component, whereas our framework trains a unified inference pipeline whose search and aggregation components are coupled through the reward of the final synthesized solution (see fig. 1).

Optimizing over Sets. Orthogonally, Tang et al. (2025) introduce the problem of optimizing over n samples and propose algorithms for improving pass@k and self-consistency, or majority@k, performance. Since both pass@k and self-consistency are filtering-based objectives, the final selected output is one of the independently sampled generations. This structure enables leave-one-out estimators for reinforcement learning, in which the output selected by pass@k or self-consistency receives higher credit assignment than the other samples in the set. However, this approach does not directly apply to settings such as self-aggregation, where the final output is a new generation synthesized from the set and there is no canonical way to assign individualized credit to each input trace.

In contrast, Hamid et al. (2026) introduce set reinforcement learning, a framework for training models with set-level rewards where the objective does not necessarily provide individualized credit assignment; more specifically, given a set of n actions sampled, one cannot always leave $k < n$ out to understand which specific generation provided credit, because the credit could emerge from more than one action appearing together in the set, as opposed to the existence of individual elements. In this setting, all actions or generations in a set are coupled through a shared learning signal: credit is assigned to the set as a whole, and the goal is to train a policy to sample sets of actions that are collectively useful. We use this framework to train our model to independently sample sets of parallel traces that can then be interleaved and aggregated into a final answer. Orney et al. (2026) propose a general set-RL recipe for training language models, which we adopt in our work. Set-level optimization has also been studied by GX-Chen et al. (2026), who use reward uncertainty to induce exploration.

B Implementation Details

In this section, we discuss various implementation details of our proposed algorithm. We present the pseudocode for SPIRAL in algorithm 1. In our experiments with the `Qwen-3-4b` model, we found that the model's response length was quite large. To make training feasible on the compute available to us, we modified the advantage to discourage response length from growing very rapidly. Concretely, suppose the max response length is $L_{\max}$ and suppose our target response length is L_{target} such that $L_{\text{target}} < L_{\max}$. Then, for any generation y whose length is L, we set

$$A_{\text{modified}}(x, y) = \begin{cases} A(x, y) & \text{if } A(x, y) < 0, \text{ or } A(x, y) > 0 \text{ and } L < L_{\text{target}} \\ 0 & \text{if } A(x, y) > 0, L > L_{\text{target}} \end{cases} \tag{9}$$

where $A(x, y)$ is the advantage of the generation as computed in algorithm 1. Note that search traces are optimized with set reinforcement learning while aggregation traces are optimized with standard reinforcement learning.

Hyperparameters. For SPIRAL, we sample $N_1 = 8$ search traces per problem, construct $K = 4$ sets of size $n = 4$, and sample $N_2 = 4$ aggregation traces per problem. Across both levels, the maximum number of tokens we allow the model to sample is $L_{\max} = 4096$. Altogether, per problem, SPIRAL samples:

$$\Big(\underbrace{4096}_{\text{Max length}} \times \underbrace{8}_{\text{\# Search Traces}}\Big) + \Big(\underbrace{4096}_{\text{Max length}} \times \underbrace{16}_{\text{\# Aggregation Traces}}\Big) = 98304 \text{ tokens.}$$

On the other hand, for GRPO, we sample 12 generations per problem where we allow the model to sample at most $L_{\max} = 8192$ tokens per problem. For GRPO, since there is no aggregation stage, we allow the model to sample all 8192 tokens at once. Per problem, GRPO samples:

$$\Big(\underbrace{8192}_{\text{Max length}} \times \underbrace{12}_{\text{\# Generations}}\Big) = 98304 \text{ tokens.}$$

The full set of hyperparameters are are provided in table 1.

Parameter	**Value**
Base model	`Qwen3-4b-Instruct-2507`
Search Traces per prompt	8
Aggregation Traces per set	4
Set size (for set RL)	4
Number of sets (for set RL)	4
Max prompt length	1024
Learning rate	2×10^{-5}
KL coefficient	0.0
Entropy coefficient	0.0
Rollout temperature	1.0
Prompts per batch	256
Max response length	4096
LoRA Rank	32
Device	Tinker Lab (2026)

Table 1: Training hyperparameters.

Prompts. The prompt for the search trace generation is shown in B.1 and the prompt for the aggregation trace is shown in B.2. For the aggregation trace, we explicitly ask the model to audit the search traces and synthesize the ideas into a correct output, in order to force the model to not generate a completely new independent attempt that does not pay attention to the search traces.

Prompt B.1: Search Trace Prompt

```
{problem}

Please reason step by step, and put your final answer within \boxed
    {}.
```

Prompt B.2: Aggregation Trace Prompt

```
You are given a problem and several candidate solution traces. Some
     candidate solutions may be incorrect, incomplete, or truncated.

Problem:

{problem}
```

```
Solution 1:
{candidate_solution 1}

Solution 2:
{candidate_solution 2}

Solution 3:
{candidate_solution 3}

Solution 4:
{candidate_solution 4}

Your task is to audit the candidate traces and then synthesize a
    correct answer to the original problem.

Follow this process:

1. Briefly audit each candidate trace. Identify promising
    strategies, equations, intermediate results, or final answers
    that are explicitly supported by the visible text.
2. Note any apparent errors or unsupported leaps. If a candidate
    trace is truncated, use only the claims that are explicitly
    supported by the visible text.
3. Check the useful ideas for correctness against the original
    problem. Do not assume a candidate is correct just because it is
     confident or because multiple candidates agree.
4. After the audit, write a line exactly as follows: ### Final
    Solution:
5. Under that label, synthesize one coherent, self-contained final
    solution that can be judged without referring back to the
    candidates. Combine only the valid parts. If the candidates
    disagree, resolve the disagreement by reasoning from the
    original problem.

Put the final answer within \boxed{}.
```

Algorithm 1 SEQUENTIAL-PARALLEL-AGGREGATIVE REINFORCEMENT LEARNING (SPIRAL)

Require: Policy π_θ, batch of inputs B, number of search traces N_1, set size n, number of sampled sets K, number of aggregation traces per set N_2, reward function r

for each input $x \in B$ **do**
  Sample search traces $y_1, \ldots, y_{N_1} \sim \pi_\theta(\cdot \mid x)$
  // Consider all $\binom{N_1}{n}$ sets of size n we can construct from these N_1 generations.
  // Then, randomly sample K sets from them without replacement
  Sample sets $G_1, \cdots, G_K$ of n unique generations from $\{y_{1:N_1}\}$ without replacement
  for $l = 1, \cdots, K$ **do**
    Sample aggregation traces from each set $y_1^{G_l}, \cdots, y_{N_2}^{G_l} \sim \pi_\theta(\cdot \mid x, G_l)$
    Score set G_l using the reward: $\widehat{f}_{\text{spiral}}(x, G_l) \leftarrow \frac{1}{N_2} \sum_{i=1}^{N_2} r(x, y_i^{G_l})$
  end for
  Compute set baseline $b = \frac{1}{K} \sum_{l=1}^{K} \widehat{f}_{\text{spiral}}(x, G_l)$
  Compute set advantage $\widehat{A^\sharp}(x, G_l; \widehat{f}_{\text{spiral}}) = \widehat{f}_{\text{spiral}}(x, G_l) - b$ for $l = 1, \cdots, K$
  // Let $\mathcal{G}(y_j)$ be all sampled sets in $G_1, \cdots, G_K$ that contain y_j
  Compute marginal set advantage for each search trace y_j where $j \in \{1, \cdots, N\}$

$$\widehat{A^\sharp_{\text{marg}}}(x, y_j; \widehat{f}_{\text{spiral}}) := \frac{1}{|\mathcal{G}(y_j)|} \sum_{G \in \mathcal{G}(y_j)} \widehat{A^\sharp}(x, G; \widehat{f}_{\text{spiral}})$$

  Compute set-RL gradient for search traces:

$$\hat{g}_{\text{set}}(x) \leftarrow \sum_{j=1}^{N} \nabla_\theta \log \pi_\theta(y_j \mid x) \cdot \widehat{A^\sharp_{\text{marg}}}(x, y_j; \widehat{f}_{\text{spiral}})$$

  Compute standard-RL gradient for aggregation traces:

$$\hat{g}_{\text{std}}(x) \leftarrow \sum_{l=1}^{K} \sum_{i=1}^{N_2} \nabla_\theta \log \pi_\theta(y_i^{G_l} \mid x, G_l) \left(r(x, y_i^{G_l}) - \frac{1}{N_2} \sum_{j=1}^{N_2} r(x, y_j^{G_l}) \right)$$

  $\hat{g}(x) \leftarrow \hat{g}_{\text{set}}(x) + \hat{g}_{\text{std}}(x)$
end for
$\hat{g} \leftarrow \frac{1}{|B|} \sum_{x \in B} \hat{g}(x)$
return $\hat{g}$

C SPIRAL WITH DIFFERENT MODELS

In this section, we discuss how one can optimize two different models, π_θ to sample search traces and π_ϕ to sample aggregation traces, using a minor variant of SPIRAL. The key observation is that, in this case, the gradient of eq. (3) is computed as follows:

$$\begin{aligned} J(\theta, \phi) &= \mathbb{E}_{y_{1:n} \sim \pi_\theta(\cdot|x)} \left[\mathbb{E}_{y_* \sim \pi_\phi(\cdot|x, y_{1:n})} \left[r(x, y_*) \right] \right] \\ &= \mathbb{E}_{y_{1:n} \sim \pi_\theta(\cdot|x)} \left[f^\phi_{\text{spiral}}(x, y_{1:n}) \right], \end{aligned}$$

where

$$f^\phi_{\text{spiral}}(x, y_{1:n}) = \mathbb{E}_{y_* \sim \pi_\phi(\cdot|x, y_{1:n})} \left[r(x, y_*) \right]. \tag{10}$$

Taking gradients with respect to the search parameters θ and the aggregation parameters ϕ gives

$$\nabla_{\theta,\phi} J(\theta, \phi) = \left(\nabla_\theta J(\theta, \phi), \nabla_\phi J(\theta, \phi) \right).$$

The θ-component is

$$\begin{aligned}\nabla_\theta J(\theta,\phi) &= \nabla_\theta \mathbb{E}_{y_{1:n}\sim\pi_\theta(\cdot|x)}\left[f^\phi_{\text{spiral}}(x,y_{1:n})\right] \\ &= \underbrace{\mathbb{E}_{y_{1:n}\sim\pi_\theta(\cdot|x)}\left[f^\phi_{\text{spiral}}(x,y_{1:n})\nabla_\theta \log \pi_\theta(y_{1:n}\mid x)\right]}_{\text{Term 1: set RL gradient}}.\end{aligned}$$

The ϕ-component is

$$\begin{aligned}\nabla_\phi J(\theta,\phi) &= \mathbb{E}_{y_{1:n}\sim\pi_\theta(\cdot|x)}\left[\nabla_\phi \mathbb{E}_{y_*\sim\pi_\phi(\cdot|x,y_{1:n})}\left[r(x,y_*)\right]\right] \\ &= \underbrace{\mathbb{E}_{y_{1:n}\sim\pi_\theta(\cdot|x)}\left[\mathbb{E}_{y_*\sim\pi_\phi(\cdot|x,y_{1:n})}\left[r(x,y_*)\nabla_\phi \log \pi_\phi(y_*\mid x,y_{1:n})\right]\right]}_{\text{Term 2: standard RL gradient}}.\end{aligned}$$

Therefore,

$$\nabla_{\theta,\phi} J(\theta,\phi) = (\underbrace{\nabla_\theta \mathbb{E}_{y_{1:n}\sim\pi_\theta(\cdot|x)}\left[f^\phi_{\text{spiral}}(x,y_{1:n})\right]}_{\text{Term 1: set RL gradient}}, \underbrace{\mathbb{E}_{y_{1:n}\sim\pi_\theta(\cdot|x)}\left[\nabla_\phi \mathbb{E}_{y_*\sim\pi_\phi(\cdot|x,y_{1:n})}\left[r(x,y_*)\right]\right]}_{\text{Term 2: standard RL gradient}}).$$

D PROOFS

In SPIRAL, we optimize the search traces using the set RL recipe proposed in Orney et al. (2026). However, Orney et al. (2026) prove that their gradient estimator is unbiased under the assumption that the set objective is symmetric in its arguments. In our case, this is not necessarily true; $f_{\text{spiral}}(x,y_{1:n}) = \mathbb{E}_{y_*\sim\pi_\theta(\cdot|x,y_{1:n})}[r(x,y_*)]$ need not be equal to $f_{\text{spiral}}(x,y_{\sigma(1),\cdots,\sigma(n)})$ where σ is some permutation of $\{1,\cdots,n\}$, since the language model's distribution can depend on the ordering of the candidate search traces in its context. In fact, prior works have observed that language models are notorious for ordering biases Pezeshkpour & Hruschka (2024). In our implementation of SPIRAL, we uniformly sample K sets from the collection of *unordered* tuples, which implicitly assumes that f_{spiral} is symmetric. This is to ensure that the model is forced to aggregate sets of higher variety in terms of their constituents. Nevertheless, in this section, we extend the analysis in Orney et al. (2026) to show that our set RL gradient estimator is still unbiased if we sample K sets from all $\frac{N!}{(N-n)!}$ possible ordered tuples of size n.

Proposition 3.1 (Restated) Fix a prompt x, and let $y_1,\cdots,y_N \overset{\text{i.i.d.}}{\sim} \pi_\theta(\cdot\mid x)$ be our independently sampled N generations and let $f:\mathcal{X}\times\mathcal{Y}^{\oplus n}\to\mathbb{R}$ be our set objective. Then,

$$\mathbb{E}[\sum_{i=1}^{N}\nabla_\theta \log \pi_\theta(y_i\mid x)\,\widehat{A^\sharp_{\text{marg}}}(x,y_i;f)] = M\nabla_\theta \mathbb{E}_{y_{1:n}\sim\pi_\theta(\cdot|x)} N\big[f(x,y_{1:n})\big],$$

where $M\in\mathbb{R}_{>0}$ is a constant scaling factor that depends on the number of sets we construct. In particular, when we sample, uniformly, K sets without replacement from $K_{\text{all}} := \binom{N}{n}$ sets, we have that

$$M = \frac{N}{n}q_K - 1,$$

where

$$q_K := \Pr_{\mathcal{S}_K}(C_i(\mathcal{S}_K)>0) = 1 - \frac{\binom{(N-1)_n}{K}}{\binom{(N)_n}{K}}, \quad (N)_n := \frac{N!}{(N-n)!}.$$

Consequently, after also taking expectation over $x\sim\mathcal{D}$ and scaling the learning rate, the estimator is an unbiased estimator of the set RL gradient.

Proof. The proof strategy is largely similar to that in Orney et al. (2026). Let

$$\mathcal{S} := \{(s_1,\ldots,s_n)\in\{1,\ldots,N\}^n \mid s_a\neq s_b \text{ if } a\neq b\}$$

be the collection of all ordered tuples of n distinct indices. Define $K_{\text{all}} := |\mathcal{S}| = (N)_n$. For an ordered tuple $S=(s_1,\ldots,s_n)\in\mathcal{S}$, we will use the shorthand $f_S := f(x,y_{s_1},\ldots,y_{s_n})$. Now, let

$\mathcal{T} \subseteq \mathcal{S}$ be a uniformly sampled subset of size K, sampled without replacement and independently of $y_{1:N}$. For each $i \in \{1, \ldots, N\}$, define

$$C_i := C_i(\mathcal{T}) := \sum_{S \in \mathcal{T}} \mathbf{1}\{i \in S\}.$$

Then, our set-level baseline can be written as

$$\widehat{f}_{\mathcal{T}}(x) := \frac{1}{K} \sum_{S \in \mathcal{T}} f_S,$$

and the corresponding normalized marginal set advantage is

$$\widehat{A^{\sharp}_{\text{marg}}}(x, y_i; f \mid \mathcal{T}) := \mathbf{1}\{C_i > 0\} \frac{1}{C_i} \sum_{\substack{S \in \mathcal{T} \\ i \in S}} \left(f_S - \widehat{f}_{\mathcal{T}}(x) \right).$$

We will also use the following shorthand for convenience:

$$\nabla_i := \nabla_\theta \log \pi_\theta(y_i \mid x).$$

We first decompose the left-hand side as

$$\begin{aligned}
&\mathbb{E}_{y_{1:N}, \mathcal{T}} \left[\sum_{i=1}^{N} \nabla_i \widehat{A^{\sharp}_{\text{marg}}}(x, y_i; f \mid \mathcal{T}) \right] \\
&= \underbrace{\mathbb{E}_{y_{1:N}, \mathcal{T}} \left[\sum_{i=1}^{N} \nabla_i \mathbf{1}\{C_i > 0\} \frac{1}{C_i} \sum_{\substack{S \in \mathcal{T} \\ i \in S}} f_S \right]}_{\text{Term 1}} - \underbrace{\mathbb{E}_{y_{1:N}, \mathcal{T}} \left[\sum_{i=1}^{N} \nabla_i \mathbf{1}\{C_i > 0\} \widehat{f}_{\mathcal{T}}(x) \right]}_{\text{Term 2}}.
\end{aligned}$$

We first simplify Term 1. While C_i is the number of sets in $\mathcal{T}$ that contain the specific generation y_{s_i}, define $\mathcal{G}_i := \{S \in \mathcal{S} : i \in S\}$ to be the collection of ordered tuples containing i. There are n possible positions for i, after which the remaining $n-1$ coordinates can be filled by an ordered selection from the remaining $N-1$ indices. Therefore, $d := |\mathcal{G}_i| = n(N-1)_{n-1}$. Then, we can rewrite Term 1 as follows:

$$\begin{aligned}
\text{Term 1} &= \mathbb{E}_{y_{1:N}, \mathcal{T}} \left[\sum_{i=1}^{N} \nabla_i \mathbf{1}\{C_i > 0\} \frac{1}{C_i} \sum_{\substack{S \in \mathcal{S} \\ i \in S}} \mathbf{1}\{S \in \mathcal{T}\} f_S \right] \\
&= \mathbb{E}_{y_{1:N}} \left[\sum_{i=1}^{N} \nabla_i \sum_{\substack{S \in \mathcal{S} \\ i \in S}} f_S \, \mathbb{E}_{\mathcal{T}} \left[\mathbf{1}\{C_i > 0\} \frac{\mathbf{1}\{S \in \mathcal{T}\}}{C_i} \right] \right] \\
&= \mathbb{E}_{y_{1:N}} \left[\sum_{i=1}^{N} \nabla_i \sum_{\substack{S \in \mathcal{S} \\ i \in S}} f_S \, \mathbb{E}_{\mathcal{T}} \left[\frac{\mathbf{1}\{S \in \mathcal{T}\}}{C_i} \right] \right],
\end{aligned}$$

Now, note that by uniform sampling of $\mathcal{T}$, the quantity $\mathbb{E}_{\mathcal{T}} \left[\frac{\mathbf{1}\{S \in \mathcal{T}\}}{C_i} \right]$ is the same for every $S \in \mathcal{G}_i$. Moreover,

$$\begin{aligned}
\sum_{S \in \mathcal{G}_i} \mathbb{E}_{\mathcal{T}} \left[\frac{\mathbf{1}\{S \in \mathcal{T}\}}{C_i} \right] &= \mathbb{E}_{\mathcal{T}} \left[\sum_{S \in \mathcal{G}_i} \frac{\mathbf{1}\{S \in \mathcal{T}\}}{C_i} \right] \\
&= \mathbb{E}_{\mathcal{T}} \left[\mathbf{1}\{C_i > 0\} \right] \\
&= q_K.
\end{aligned}$$

In the second line, we simply used the fact that $\sum_{S\in\mathcal{G}_i} \frac{1\{S\in\mathcal{T}\}}{C_i} = \frac{C_i}{C_i}$ if $C_i > 0$ (we are suppressing the fact that, by definition, we set this term to be 0 if $C_i = 0$). As such, for every $S \in \mathcal{G}_i$, recalling that $d = |\mathcal{G}_i|$, we can write:

$$\mathbb{E}_{\mathcal{T}}\left[\frac{\mathbf{1}\{S\in\mathcal{T}\}}{C_i}\right] = \frac{q_K}{d}.$$

As such, Term 1 can be simplified as follows:

$$\begin{aligned}
\text{Term 1} &= \frac{q_K}{d}\mathbb{E}_{y_{1:N}}\left[\sum_{i=1}^{N}\nabla_i \sum_{\substack{S\in\mathcal{S}\\ i\in S}} f_S\right] \\
&= \frac{q_K}{d}\mathbb{E}_{y_{1:N}}\left[\sum_{S\in\mathcal{S}} f_S \sum_{i\in S}\nabla_\theta \log \pi_\theta(y_i \mid x)\right] \\
&= \frac{q_K}{d}\sum_{S\in\mathcal{S}}\mathbb{E}_{y_{1:N}}\left[f_S \sum_{i\in S}\nabla_\theta \log \pi_\theta(y_i \mid x)\right].
\end{aligned}$$

For any fixed ordered tuple $S = (s_1, \ldots, s_n)$, the random variables $y_{s_1}, \ldots, y_{s_n}$ are i.i.d. samples from $\pi_\theta(\cdot \mid x)$. Because the ordering of the coordinates in S is retained, the score-function identity gives

$$\begin{aligned}
\mathbb{E}_{y_{1:N}}\left[f_S \sum_{i\in S}\nabla_i\right] &= \mathbb{E}_{y_{s_1},\ldots,y_{s_n}}\left[f(x, y_{s_1}, \ldots, y_{s_n})\sum_{r=1}^{n}\nabla_\theta \log \pi_\theta(y_{s_r} \mid x)\right] \\
&= \nabla_\theta \mathbb{E}_{y_{1:n}\sim\pi_\theta(\cdot|x)}\left[f(x, y_{1:n})\right].
\end{aligned}$$

Therefore,

$$\begin{aligned}
\text{Term 1} &= \frac{q_K}{d}K_{\text{all}}\nabla_\theta \mathbb{E}_{y_{1:n}\sim\pi_\theta(\cdot|x)}\left[f(x, y_{1:n})\right] \\
&= \frac{N}{n}q_K\nabla_\theta \mathbb{E}_{y_{1:n}\sim\pi_\theta(\cdot|x)}\left[f(x, y_{1:n})\right],
\end{aligned}$$

where we used

$$\frac{K_{\text{all}}}{d} = \frac{(N)_n}{n(N-1)_{n-1}} = \frac{N}{n}.$$

Next, we simplify Term 2. Recall that we can write the baseline as:

$$\widehat{f}_{\mathcal{T}}(x) = \frac{1}{K}\sum_{U\in\mathcal{T}} f_U,$$

we have

$$\text{Term 2} = \mathbb{E}_{y_{1:N}}\left[\sum_{i=1}^{N}\nabla_i \sum_{U\in\mathcal{S}} f_U\, \mathbb{E}_{\mathcal{T}}\left[\mathbf{1}\{C_i > 0\}\frac{\mathbf{1}\{U\in\mathcal{T}\}}{K}\right]\right].$$

For fixed $U \in \mathcal{S}$ and fixed i, there are two cases. First, suppose that $i \in U$. Now, if $U \in \mathcal{T}$, we necessarily have $C_i > 0$, and therefore

$$\begin{aligned}
\mathbb{E}_{\mathcal{T}}\left[\mathbf{1}\{C_i > 0\}\frac{\mathbf{1}\{U\in\mathcal{T}\}}{K}\right] &= \frac{1}{K}\Pr_{\mathcal{T}}(U\in\mathcal{T}) \\
&= \frac{1}{K}\frac{K}{K_{\text{all}}} \\
&= \frac{1}{K_{\text{all}}}.
\end{aligned}$$

Second, suppose that $i \notin U$. In this case, f_U is independent of y_i, and hence

$$\mathbb{E}_{y_{1:N}}[f_U\nabla_i] = \mathbb{E}_{y_{1:N}}[f_U]\,\mathbb{E}_{y_i}[\nabla_i] = 0.$$

It follows that

$$\begin{aligned}\text{Term 2} &= \frac{1}{K_{\text{all}}} \sum_{U \in \mathcal{S}} \mathbb{E}_{y_{1:N}} \left[f_U \sum_{i \in U} \nabla_i \right] \\ &= \frac{1}{K_{\text{all}}} \sum_{U \in \mathcal{S}} \nabla_\theta \mathbb{E}_{y_{1:n} \sim \pi_\theta(\cdot|x)} \left[f(x, y_{1:n}) \right] \\ &= \nabla_\theta \mathbb{E}_{y_{1:n} \sim \pi_\theta(\cdot|x)} \left[f(x, y_{1:n}) \right].\end{aligned}$$

Combining Term 1 and Term 2, we obtain

$$\begin{aligned}&\mathbb{E}_{y_{1:N}, \mathcal{T}} \left[\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(y_i \mid x) \, \widehat{A^{\sharp}_{\text{marg}}}(x, y_i; f \mid \mathcal{T}) \right] \\ &= \left(\frac{N}{n} q_K - 1 \right) \nabla_\theta \mathbb{E}_{y_{1:n} \sim \pi_\theta(\cdot|x)} \left[f(x, y_{1:n}) \right].\end{aligned}$$

Thus,

$$M_K = \frac{N}{n} q_K - 1.$$

It remains to verify the expression for q_K. For a fixed index i, the number of ordered tuples in $\mathcal{S}$ that do not contain i is

$$(N-1)_n.$$

Equivalently,

$$K_{\text{all}} - d = (N)_n - n(N-1)_{n-1} = (N-1)_n.$$

The event $C_i = 0$ occurs precisely when all K sampled ordered tuples are selected from these $(N-1)_n$ tuples. Therefore,

$$\Pr_{\mathcal{T}}(C_i = 0) = \frac{\binom{(N-1)_n}{K}}{\binom{(N)_n}{K}},$$

where $\binom{a}{K} = 0$ when $K > a$. Therefore, we can write

$$q_K = \Pr_{\mathcal{T}}(C_i > 0) = 1 - \frac{\binom{(N-1)_n}{K}}{\binom{(N)_n}{K}}.$$

Finally, when $K = 1$,

$$q_1 = \frac{d}{K_{\text{all}}} = \frac{n}{N},$$

and consequently $M_1 = 0$, as expected because a single sampled tuple has zero advantage relative to its own baseline. When $N > n$ and $K > 1$, we have that:

$$q_K > q_1 = \frac{n}{N}.$$

Therefore,

$$M_K = \frac{N}{n} q_K - 1 > 0.$$

□

Additionally, observe that we fixed the size of the sets of search traces to be n even though, at test-time, we can scale the number of parallel generations that we put into the context of the model when sampling its aggregate generation. This is done so that our training is stable and not sample expensive. However, in principle, one could define $f^{(n)}_{\text{spiral}}(x, y_1, \cdots, y_n) = \mathbb{E}_{y_* \sim \pi_\theta(\cdot|x, y_{1:n})}[r(x, y_*)]$ and optimize $f(x, y_{1:N_1}) = \sum_{n=1}^{N} f^{(n)}_{\text{spiral}}(x, y_{1:n})$ with N not exceeding the total number of search traces sampled; this objective trains the model to generate and aggregate sets of varying size (from $n = 1$ to $n = N$). The recipe for optimizing this objective would be the same as SPIRAL, although one would require constructing a large number of sets of varying sizes and sampling aggregation

traces from each which would be expensive. On the other hand, one can also scale the pipeline illustrated in fig. 1 by allowing the model to recursively sample and aggregate for more than just two steps. In our empirical evaluations, we will show that SPIRAL, despite training on two steps, enables significantly better scaling under recursive self-aggregation Venkatraman et al. (2025), by training the model to sample useful search traces and effective aggregation traces at any step.