s1: Simple test-time scaling

Niklas Muennighoff [*1 3 4] **Zitong Yang** [*1] **Weijia Shi** [*2 3] **Xiang Lisa Li** [*1] **Li Fei-Fei** [1] **Hannaneh Hajishirzi** [2 3]
Luke Zettlemoyer [2] **Percy Liang** [1] **Emmanuel Candès** [1] **Tatsunori Hashimoto** [1]

Abstract

Test-time scaling is a promising new approach to language modeling that uses extra test-time compute to improve performance. Recently, OpenAI's o1 model showed this capability but did not publicly share its methodology, leading to many replication efforts. We seek the simplest approach to achieve test-time scaling and strong reasoning performance. First, we curate a small dataset **s1K** of 1,000 questions paired with reasoning traces relying on three criteria we validate through ablations: difficulty, diversity, and quality. Second, we develop budget forcing to control test-time compute by forcefully terminating the model's thinking process or lengthening it by appending "Wait" multiple times to the model's generation when it tries to end. This can lead the model to double-check its answer, often fixing incorrect reasoning steps. After supervised finetuning the Qwen2.5-32B-Instruct language model on **s1K** and equipping it with budget forcing, our model **s1-32B** exceeds o1-preview on competition math questions by up to 27% (MATH and AIME24). Further, scaling **s1-32B** with budget forcing allows extrapolating beyond its performance without test-time intervention: from 50% to 57% on AIME24. Our model, data, and code are open-source at https://github.com/simplescaling/s1.

1. Introduction

Performance improvements of language models (LMs) over the past years have largely relied on scaling up train-time compute using large-scale self-supervised pretraining (Kaplan et al., 2020; Hoffmann et al., 2022). The creation of these powerful models has set the stage for a new scaling paradigm built on top of them: *test-time scaling*. The aim of this approach is to increase the compute at test time to get better results. There has been much work exploring this idea (Snell et al., 2024; Welleck et al., 2024), and the viability of this paradigm was recently validated by OpenAI o1 (OpenAI, 2024). o1 has demonstrated strong reasoning performance with consistent gains from scaling test-time compute. OpenAI describes their approach as using large-scale reinforcement learning (RL) implying the use of sizable amounts of data (OpenAI, 2024). This has led to various attempts to replicate their models relying on techniques like Monte Carlo Tree Search (Gao et al., 2024b; Zhang et al., 2024a), multi-agent approaches (Qin et al., 2024), and others (Wang et al., 2024a; Huang et al., 2024b; 2025). Among these approaches, DeepSeek R1 (DeepSeek-AI et al., 2025) has successfully replicated o1-level performance, also employing reinforcement learning via millions of samples and multiple training stages. However, despite the large number of o1 replication attempts, none have openly replicated a clear test-time scaling behavior. Thus, we ask: what is the simplest approach to achieve both test-time scaling and strong reasoning performance?

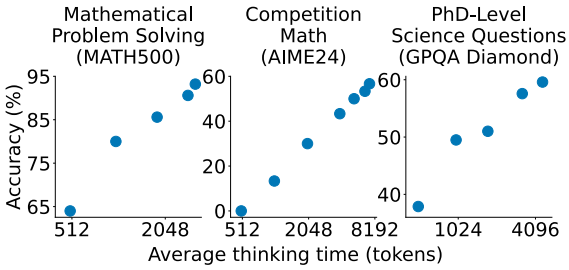


Figure 1. **Test-time scaling with s1-32B.** We benchmark **s1-32B** on reasoning-intensive tasks and vary test-time compute.

We show that training on only 1,000 samples with next-token prediction and controlling thinking duration via a simple test-time technique we refer to as *budget forcing* leads to a strong reasoning model that scales in performance with more test-time compute. Specifically, we construct **s1K**, which consists of 1,000 carefully curated questions paired with reasoning traces and answers distilled from Gemini Thinking Experimental (Google, 2024). We perform supervised fine-tuning (SFT) of an off-the-shelf pretrained model

[*]Equal contribution. ZY and NM started the project. WS, NM and ZY collected the prompts, XL, ZY and NM, built the data pipeline, LZ and WS proposed using a 1K subset and ZY and NM built budget forcing. [1] Stanford University. [2] University of Washington, Seattle. [3] Allen Institute for AI. [4] Contextual AI.

on our small dataset requiring just 26 minutes of training on 16 H100 GPUs. After training, we control the amount of test-time compute our model spends using *budget forcing*: **(I)** If the model generates more thinking tokens than a desired limit, we forcefully end the thinking process by appending an end-of-thinking token delimiter. Ending the thinking this way makes the model transition to generating its answer. **(II)** If we want the model to spend more test-time compute on a problem, we suppress the generation of the end-of-thinking token delimiter and instead append "Wait" to the model's current reasoning trace to encourage more exploration. Equipped with this simple recipe – SFT on 1,000 samples and test-time budget forcing – our model **s1-32B** exhibits test-time scaling (Figure 1). Further, **s1-32B** is the most sample-efficient reasoning model and outperforms closed-source models like OpenAI's o1-preview (Figure 2).

We conduct extensive ablation experiments targeting (a) our selection of 1,000 (1K) reasoning samples and (b) our test-time scaling. For **(a)**, we find that jointly incorporating difficulty, diversity, and quality measures into our selection algorithm is important. Random selection, selecting samples with the longest reasoning traces, or only selecting maximally diverse samples all lead to significantly worse performance (around −30% on AIME24 on average). Training on our full data pool of 59K examples, a superset of **s1K**, does not offer substantial gains over our 1K selection. This highlights the importance of careful data selection and echoes prior findings for instruction tuning (Zhou et al., 2023). For **(b)**, we define desiderata for test-time scaling methods to compare different approaches. Budget forcing leads to the best scaling as it has perfect controllability with a clear positive slope leading to strong performance.

In summary, our contributions are: We develop simple methods for creating a sample-efficient reasoning dataset (§2) and test-time scaling (§3); Based on these we build **s1-32B** which is competitive with o1-preview (§4); We ablate subtleties of data (§5.1) and test-time scaling (§5.2). We end with a discussion to motivate future work on simple reasoning (§6). Our code, model, and data are open-source at `https://github.com/simplescaling/s1`.

2. Reasoning data curation to create s1K

In this section, we describe our process for creating a large dataset first in §2.1 and then filtering it down to **s1K** in §2.2.

2.1. Initial collection of 59K samples

We collect an initial 59,029 questions from 16 sources following three guiding principles. **Quality**: Datasets should be high-quality; we always inspect samples and ignore datasets with, e.g., poor formatting; **Difficulty**: Datasets should be challenging and require significant reasoning effort; **Diversity**: Datasets should stem from various fields to cover different reasoning tasks. We collect datasets of two categories:

Curation of existing datasets Our largest source is NuminaMATH (LI et al., 2024) with 30,660 mathematical problems from online websites. We also include historical AIME problems (1983-2021). To enhance diversity, we add OlympicArena (Huang et al., 2024a) with 4,250 questions spanning Astronomy, Biology, Chemistry, Computer Science, Geography, Mathematics, and Physics from various Olympiads. OmniMath (Gao et al., 2024a) adds 4,238 competition-level mathematics problems. We also include 2,385 problems from AGIEval (Zhong et al., 2023), which features questions from standardized tests like SAT and LSAT, covering English, Law, and Logic. We refer to Table 7 in §C for our other sources.

New datasets in quantitative reasoning To complement these existing datasets, we create two original datasets. s1-prob consists of 182 questions from the probability section of Stanford University's Statistics Department's PhD Qualifying Exams (`https://statistics.stanford.edu`), accompanied by handwritten solutions that cover difficult proofs. The probability qualifying exam is held yearly and requires professional-level mathematical problem-solving. s1-teasers comprises 23 challenging brain-teasers commonly used in interview questions for quantitative trading positions. Each sample consists of a problem and solution taken from PuzzledQuant (`https://www.puzzledquant.com/`). We only take examples with the highest difficulty level ("Hard").

For each question, we generate a reasoning trace and solution using the Google Gemini Flash Thinking API (Google, 2024) extracting its reasoning trace and response. This yields 59K triplets of a question, generated reasoning trace, and generated solution. Examples from our dataset are in §D.2. We decontaminate all samples against our evaluation questions (MATH500, GPQA Diamond, AIME24; §C.5) using 8-grams and deduplicate the data.

2.2. Final selection of 1K samples

We could directly train on our pool of 59K questions, however, our goal is to find the *simplest* approach with minimal resources. Thus, we go through three stages of filtering to arrive at a minimal set of 1,000 samples relying on our three guiding data principles: Quality, Difficulty, and Diversity.

Quality We first remove any questions where we ran into any API errors reducing our dataset to 54,116 samples. Next, we filter out low-quality examples by checking if they contain any string patterns with formatting issues, such as ASCII art diagrams, non-existent image references, or inconsistent question numbering reducing our dataset to 51,581 examples.

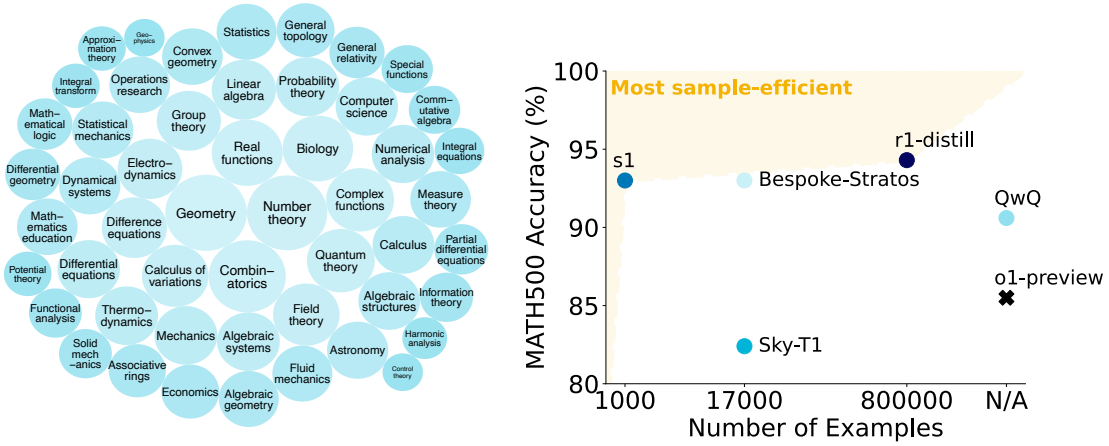


Figure 2. **s1K and s1-32B.** *(left)* **s1K** is a dataset of 1,000 high-quality, diverse, and difficult questions with reasoning traces. *(right)* **s1-32B**, a 32B parameter model finetuned on **s1K** is on the sample-efficiency frontier. See Table 1 for details on other models.

From this pool, we identify 384 samples for our final 1,000 samples from datasets that we perceive as high-quality and not in need of further filtering (see §C.4 for details).

Difficulty For difficulty, we use two indicators: model performance and reasoning trace length. We evaluate two models on each question: Qwen2.5-7B-Instruct and Qwen2.5-32B-Instruct (Qwen et al., 2024), with correctness assessed by Claude 3.5 Sonnet comparing each attempt against the reference solution (see §C.3 for the grading protocol). We measure the token length of each reasoning trace to indicate problem difficulty using the Qwen2.5 tokenizer. This relies on the assumption that more difficult problems require more thinking tokens. Based on the grading, we remove questions that either Qwen2.5-7B-Instruct or Qwen2.5-32B-Instruct can solve correctly and thus may be too easy. By using two models we reduce the likelihood of an easy sample slipping through our filtering due to a rare mistake on an easy question of one of the models. This brings our total samples down to 24,496, setting the stage for the next round of subsampling based on diversity. While filtering with these two models may be optimized for our setup as we will also use Qwen2.5-32B-Instruct as our model to finetune, the idea of model-based filtering generalizes to other setups.

Diversity To quantify diversity, we classify questions into domains using Claude 3.5 Sonnet based on the Mathematics Subject Classification (MSC) system (e.g., geometry, combinatorics, etc.) from the American Mathematical Society.[1] The taxonomy focuses on topics in mathematics but also includes other sciences such as biology, physics, and economics. To select our final examples from the pool of 24,496 questions, we first choose one domain uniformly at random. Then, we sample one problem from this domain according to a distribution that favors longer reasoning traces (see §C.4 for details) as motivated in *Difficulty*. We repeat this process until we have 1,000 total samples spanning 50 domains.

In §5.1, we will show that using our three criteria in combination is important, as only relying on quality, diversity, or difficulty in isolation leads to worse datasets. Some distilled generations are incorrect, which we allow in our data as we focus on capturing the reasoning process rather than entirely correct solutions. Our grader (§C.3) deems 53.6% correct in **s1K** and 63.0% in our follow-up **s1K-1.1** (see §A).

3. Test-time scaling

3.1. Method

We classify test-time scaling methods into **1) Sequential**, where later computations depend on earlier ones (e.g., a long reasoning trace), and **2) Parallel**, where computations run independently (e.g., majority voting) (Snell et al., 2024; Brown et al., 2024). We focus on sequential scaling as intuitively we believe it should scale better, since later computations can build on intermediate results, allowing for deeper reasoning and iterative refinement. We propose new sequential scaling methods and ways to benchmark them.

Budget forcing We propose a simple decoding-time intervention by forcing a maximum and/or minimum number of thinking tokens. Specifically, we enforce a maximum token count by simply appending the end-of-thinking token delimiter and optionally "`Final Answer:`" to early exit

[1] https://mathscinet.ams.org/mathscinet/msc/msc2020.html

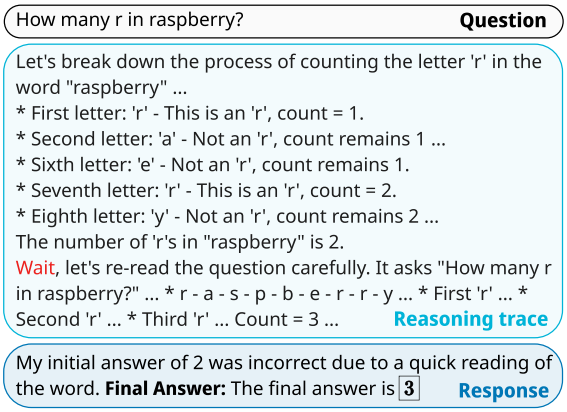


Figure 3. **Budget forcing with s1-32B.** The model tries to stop after "...is 2.", but we suppress the end-of-thinking token delimiter instead appending "Wait" leading **s1-32B** to self-correct its answer.

the thinking stage and make the model provide its current best answer. To enforce a minimum, we suppress the generation of the end-of-thinking token delimiter and optionally append the string "Wait" to the model's current reasoning trace to encourage the model to reflect on its current generation. Figure 3 contains an example of how this simple approach can lead the model to arrive at a better answer.

Baselines We benchmark budget forcing with: **(I) Conditional length-control methods**, which rely on telling the model in the prompt how long it should generate for. We group them by granularity into (a) Token-conditional control: We specify an upper bound of thinking tokens in the prompt; (b) Step-conditional control: We specify an upper bound of thinking steps, where each step is around 100 tokens; (c) Class-conditional control: We write two generic prompts that tell the model to either think for a short or long amount of time (see §E.1 for details). **(II) Rejection sampling**, which samples until a generation fits a predetermined compute budget. This oracle captures the posterior over responses conditioned on its length.

3.2. Metrics

We establish a set of desiderata as evaluation metrics to measure test-time scaling across methods. Importantly, we do not only care about the accuracy a method can achieve but also its controllability and test-time scaling slope. For each method we consider, we run a set of evaluations $a \in \mathcal{A}$ varying test-time compute on a fixed benchmark, e.g. AIME24. This produces a piece-wise linear function f with compute as the x-axis measured in thinking tokens and accuracy as the y-axis (see Figure 1, where the rightmost dot for AIME24 corresponds to $f(7320) = 57\%$). We measure three metrics:

$$\text{Control} = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \mathbb{I}(a_{\min} \leq a \leq a_{\max}) \tag{1}$$

where $a_{\min}, a_{\max}$ refer to a pre-specified minimum and maximum amount of test-time compute; in our case thinking tokens. We usually only constrain $a_{\max}$. As tokens generated correspond to the amount of test-time compute spent, this metric measures the extent to which a method allows controllability over the use of that test-time compute. We report it as a percentage with 100% being perfect control.

$$\text{Scaling} = \frac{1}{\binom{|\mathcal{A}|}{2}} \sum_{\substack{a,b \in \mathcal{A} \\ b>a}} \frac{f(b) - f(a)}{b - a} \tag{2}$$

Scaling is the average slope of the piece-wise linear function. It must be positive for useful methods and larger is better.

$$\text{Performance} = \max_{a \in \mathcal{A}} f(a) \tag{3}$$

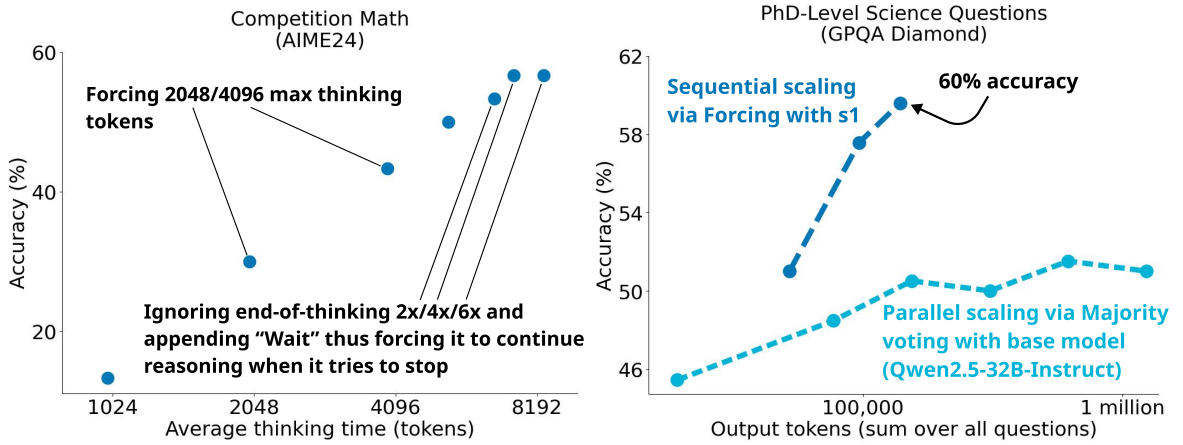
Performance is simply the maximum performance the method achieves on the benchmark. A method with monotonically increasing scaling achieves 100% performance on any benchmark in the limit. However, the methods we investigate eventually flatten out or further scaling fails due to control or context window limitations.

4. Results

4.1. Setup

Training We perform supervised finetuning on Qwen2.5-32B-Instruct using **s1K** to obtain our model **s1-32B** using basic hyperparameters outlined in §D. Finetuning took 26 minutes on 16 NVIDIA H100 GPUs with PyTorch FSDP.

Evaluation We select three representative reasoning benchmarks widely used in the field: **AIME24** (of America, 2024) has 30 problems that were used in the 2024 American Invitational Mathematics Examination (AIME) held from January 31 – February 1, 2024. AIME tests mathematical problem-solving with arithmetic, algebra, counting, geometry, number theory, probability, and other secondary school math topics. High-scoring high school students in the test are invited to participate in the United States of America Mathematics Olympiad (USAMO). All AIME answers are integers ranging from 000 to 999, inclusive. Some AIME problems rely on figures that we provide to our model using the vector graphics language Asymptote as it cannot take image inputs. **MATH500** (Hendrycks et al., 2021) is a benchmark of competition math problems of varying difficulty. We evaluate on the same 500 samples selected by OpenAI in prior work (Lightman et al., 2023). **GPQA Diamond** (Rein et al., 2023) consists of 198 PhD-level science questions



(a) Sequential scaling via budget forcing

(b) Parallel scaling via majority voting

Figure 4. **Sequential and parallel test-time scaling.** *(a):* Budget forcing shows clear scaling trends and extrapolates to some extent. For the three rightmost dots, we prevent the model from stopping its thinking 2/4/6 times, each time appending "Wait" to its current reasoning trace. *(b):* For Qwen2.5-32B-Instruct we perform 64 evaluations for each sample with a temperature of 1 and visualize the performance when majority voting across 2, 4, 8, 16, 32, and 64 of these.

from Biology, Chemistry and Physics. Experts with PhDs in the corresponding domains only achieved 69.7% on GPQA Diamond (OpenAI, 2024). When we write "GPQA" in the context of evaluation in this work, we always refer to the Diamond subset. We build on the "lm-evaluation-harness" framework (Gao et al., 2021; Biderman et al., 2024). Unless otherwise specified, we evaluate with a temperature of 0 (greedy) and measure accuracy (equivalent to pass@1).

Other models We benchmark **s1-32B** against: **OpenAI o1 series** (OpenAI, 2024), closed-source models that popularized test-time scaling; **DeepSeek r1 series** (DeepSeek-AI et al., 2025), open-weight reasoning models with up to o1-level performance; Qwen's **QwQ-32B-preview** (Team, 2024), a 32B open-weight reasoning model without disclosed methodology; **Sky-T1-32B-Preview** (Team, 2025) and **Bespoke-32B** (Labs, 2025), open models with open reasoning data distilled from QwQ-32B-preview and r1; **Google Gemini 2.0 Flash Thinking Experimental** (Google, 2024), the API that we distill from. As it has no official evaluation scores, we use the Gemini API to benchmark it ourselves. However, the "recitation error" of the Gemini API makes evaluation challenging.[2] We circumvent this, by manually inserting all 30 AIME24 questions in its web interface where the error does not appear. However, we leave out MATH500 (500 questions) and GPQA Diamond (198 questions), thus they are N.A. in Table 1. Our model, **s1-32B**, is fully open including weights, reasoning data, and code.

[2] https://github.com/google/generative-ai-docs/issues/257

Table 1. **s1-32B is a strong open reasoning model.** We evaluate **s1-32B**, Qwen, and Gemini (some entries are unknown (N.A.), see §4). Other results are from the respective reports (Qwen et al., 2024; Team, 2024; OpenAI, 2024; DeepSeek-AI et al., 2025; Labs, 2025; Team, 2025). # ex. = number examples used for reasoning finetuning; BF = budget forcing. See §A for our better **s1.1** model.

Model	# ex.	AIME 2024	MATH 500	GPQA Diamond
API only				
o1-preview	N.A.	44.6	85.5	73.3
o1-mini	N.A.	70.0	90.0	60.0
o1	N.A.	**74.4**	**94.8**	**77.3**
Gemini 2.0 Flash Think.	N.A.	60.0	N.A.	N.A.
Open Weights				
Qwen2.5-32B-Instruct	N.A.	26.7	84.0	49.0
QwQ-32B	N.A.	50.0	90.6	54.5
r1	≫800K	**79.8**	**97.3**	**71.5**
r1-distill	800K	72.6	94.3	62.1
Open Weights and Open Data				
Sky-T1	17K	43.3	82.4	56.8
Bespoke-32B	17K	**63.3**	93.0	58.1
s1 w/o BF	**1K**	50.0	92.6	56.6
s1-32B	**1K**	56.7	**93.0**	**59.6**

4.2. Performance

Test-time scaling Figure 1 shows the performance of **s1-32B** with budget forcing scales with more test-time compute. In Figure 4 (left), we expand the plot from Figure 1 (middle) showing that while we can improve AIME24 performance using our budget forcing technique (§3) and more test-time compute it does eventually flatten out at six times. Suppressing the end-of-thinking token delimiter too often can lead the model into repetitive loops instead of continued reasoning. In Figure 4 (right), we show that after training Qwen2.5-32B-Instruct on our 1,000 samples to produce **s1-32B** and equipping it with the simple budget forcing technique, it operates in a different scaling paradigm. Scaling test-time compute on the base model via majority voting cannot catch up with the performance of **s1-32B** which validates our intuition from §3 that sequential scaling is more effective than parallel. We provide example generations of **s1-32B** in Figure 5.

Sample-efficiency In Figure 2 (right) and Table 1 we compare **s1-32B** with other models. We find that **s1-32B** is the most sample-efficient open data reasoning model. It performs significantly better than our base model (Qwen2.5-32B-Instruct) despite just training it on an additional 1,000 samples. The concurrently released r1-32B shows stronger performance than **s1-32B** while also only using SFT (DeepSeek-AI et al., 2025). However, it is trained on 800 × more reasoning samples. It is an open question whether one can achieve their performance with just 1,000 samples. Finally, our model nearly matches Gemini 2.0 Thinking on AIME24. As the data for **s1-32B** is distilled from Gemini 2.0, this shows our distillation procedure was likely effective.

5. Ablations

5.1. Data Quantity, Diversity, and Difficulty

In §2 we outlined our three guiding principles in curating **s1K**: Quality, Difficulty, and Diversity. Here we test the importance of combining them and the overall efficacy of our selection. **Only Quality (1K-random)**: After obtaining our high-quality reasoning chains from Gemini, we select 1,000 samples at random; not relying on our difficulty and diversity filtering at all. Table 2 shows this approach performs much worse than **s1K** across all benchmarks. **Only Diversity (1K-diverse)**: For this dataset, we sample uniformly across domains to maximize diversity disregarding any notion of difficulty. This approach also leads to poor performance similar to 1K-random. **Only Difficulty (1K-longest)**: Here we rely on one of our difficulty indicators introduced in §2 by selecting the 1,000 samples with the longest reasoning traces. This approach significantly boosts GPQA performance but overall still falls short of using **s1K**. **Maximize Quantity**:

Table 2. **s1K data ablations.** We budget force (BF) a maximum of around 30,000 thinking tokens for all scores in this table. This performs slightly better than the scores without BF (Table 1) as it allows the model to finish with a best guess when stuck in an infinite loop. We report 95% paired bootstrap confidence intervals for differences relative to the **s1K** model using 10,000 bootstrap samples. E.g., the interval [-13%, 20%] means that, with 95% confidence, the true difference between 59K-full and **s1K** is between -13% and +20%. If the entire interval is negative, e.g. [-27%, -3%], we can confidently say that the performance is worse than **s1K**.

Model	AIME 2024	MATH 500	GPQA Diamond
1K-random	36.7 [-26.7%, -3.3%]	90.6 [-4.8%, 0.0%]	52.0 [-12.6%, 2.5%]
1K-diverse	26.7 [-40.0%, -10.0%]	91.2 [-4.0%, 0.2%]	54.6 [-10.1%, 5.1%]
1K-longest	33.3 [-36.7%, 0.0%]	90.4 [-5.0%, -0.2%]	59.6 [-5.1%, 10.1%]
59K-full	53.3 [-13.3%, 20.0%]	92.8 [-2.6%, 2.2%]	58.1 [-6.6%, 8.6%]
s1K	50.0	93.0	57.6

Finally, we compare with just training on all of our 59K samples, a superset of all the 1K-sample versions. This leads to a strong model but uses much more resources. To finetune on 59K samples, we use 394 H100 GPU hours while **s1-32B** only required 7 H100 GPU hours. Moreover, relying only on **s1K** is extremely competitive as shown in §2. Overall, combining all three criteria – *Quality*, *Difficulty*, *Diversity* – via our methodology in §2 is key for sample-efficient reasoning training.

5.2. Test-time scaling methods

Table 3. **Ablations on methods to scale test-time compute on AIME24.** $|\mathcal{A}|$ refers to the number of evaluation runs used to estimate the properties; thus a higher value indicates more robustness. **Bold** indicates our chosen method and the best values. BF = budget forcing, TCC/SCC/CCC = token/step/class-conditional control, RS = rejection sampling.

Method	Control	Scaling	Performance	$\lvert\mathcal{A}\rvert$
BF	**100%**	15	**56.7**	5
TCC	40%	-24	40.0	5
TCC + BF	**100%**	13	40.0	5
SCC	60%	3	36.7	5
SCC + BF	**100%**	6	36.7	5
CCC	50%	**25**	36.7	2
RS	**100%**	-35	40.0	5

[AIME24]

Alice and Bob play the following game. A stack of n tokens lies before them. The players take turns with Alice going first. On each turn, the player removes either 1 token or 4 tokens from the stack. Whoever removes the last token wins. Find the number of positive integers n less than or equal to 2024 for which there exists a strategy for Bob that guarantees that Bob will win the game regardless of Alice's play.

The problem asks for the number of positive integers $n \leq 2024$ such that Bob has a winning strategy. This is a typical impartial game that can be analyzed using the Sprague-Grundy theorem or by directly finding the winning and losing positions. Let W be the set of winning positions (number of tokens n such that the player whose turn it is can force a win) and L be the set of losing positions. [...]

Let $P(n)$ be the statement "the player whose turn it is when there are n tokens has a winning strategy. $P(0)$ is false (the player whose turn it is and faces 0 tokens has already lost). Consider the small values of n:
$n = 1$: The player whose turn it is can take 1 token and win. So $P(1)$ is true. $1 \in W$.
$n = 2$: The player whose turn it is can take 1 token, leaving 1 token for the opponent. Since $1 \in W$, the opponent wins. So $P(2)$ is false. $2 \in L$.
$n = 3$: The player whose turn it is can take 1 token, leaving 2 tokens for the opponent. Since $2 \in L$, the player whose turn it is wins. So $P(3)$ is true. $3 \in W$.[...]

We are looking for the number of positive integers $n \leq 2024$ for which Bob has a winning strategy. This means that if the game starts with n tokens, and Alice goes first, Bob can win. This happens if and only if n is a losing position (because if $n \in L$, then Alice's first move must lead to a winning position for Bob). [...]
Let's try to find a pattern in the losing positions $L = \{0, 2, 5, 7, 10, 12, 15, 17, 20, 22, ...\}$. Differences between consecutive terms: $2, 3, 2, 3, 2, 3, 2, 3, 2, ...$ The pattern of differences is alternating 2, 3. [...]
The losing positions are of the form $5m$ or $5m + 2$, where $m \geq 0$. We need to find the number of positive integers $n \leq 2024$ that are in the set of losing positions L. [...]
Case 1: $n = 5m$, where $m \geq 1$. $1 \leq 5m \leq 2024$ $1/5 \leq m \leq 2024/5 = 404.8$ Since m is an integer, $1 \leq m \leq 404$. The number of values of m is 404.
Case 2: $n = 5m + 2$, where $m \geq 0$. $5m + 2 \leq 2024$ $5m \leq 2022$ $m \leq 2022/5 = 404.4$ Since m is an integer, $0 \leq m \leq 404$. The number of values of m is $404 - 0 + 1 = 405$.

The total number of positive integers $n \leq 2024$ that are losing positions is $404 + 405 = \boxed{809}$.

[MATH500]

Find the projection of a onto $b = \begin{pmatrix} 2 \\ 6 \\ 3 \end{pmatrix}$ if $a \cdot b = 8$.

The projection of vector $\mathbf{a}$ onto vector $\mathbf{b}$ is given by the formula:

$$\text{proj}_{\mathbf{b}}\mathbf{a} = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{b}\|^2}\mathbf{b}$$

We are given $\mathbf{b} = \begin{pmatrix} 2 \\ 6 \\ 3 \end{pmatrix}$ and $\mathbf{a} \cdot \mathbf{b} = 8$.

First, we need to find the magnitude squared of $\mathbf{b}$, which is $\|\mathbf{b}\|^2$.

$\|\mathbf{b}\|^2 = \mathbf{b} \cdot \mathbf{b} = 2^2 + 6^2 + 3^2 = 4 + 36 + 9 = 49$.

Now we can plug the values into the projection formula:

$$\text{proj}_{\mathbf{b}}\mathbf{a} = \frac{8}{49}\mathbf{b}$$

Substitute the vector $\mathbf{b}$:

$$\text{proj}_{\mathbf{b}}\mathbf{a} = \frac{8}{49}\begin{pmatrix} 2 \\ 6 \\ 3 \end{pmatrix}$$

Multiply the scalar by each component of the vector:

$$\text{proj}_{\mathbf{b}}\mathbf{a} = \begin{pmatrix} \frac{8}{49} \times 2 \\ \frac{8}{49} \times 6 \\ \frac{8}{49} \times 3 \end{pmatrix} = \begin{pmatrix} \frac{16}{49} \\ \frac{48}{49} \\ \frac{24}{49} \end{pmatrix}$$

Final Answer: The final answer is

$$\boxed{\begin{pmatrix} \frac{16}{49} \\ \frac{48}{49} \\ \frac{24}{49} \end{pmatrix}}$$

[GPQA]

A quantum mechanical particle of mass m moves in two dimensions in the following potential, as a function of the polar coordinates (r, θ):

$$V(r, \theta) = \frac{1}{2}kr^2 + \frac{3}{2}kr^2\cos^2(\theta)$$

Find the energy spectrum. Hint: Write the potential in Cartesian coordinates.

The problem asks for the energy spectrum of a particle in a given 2D potential. The potential is given in polar coordinates (r, θ):

$$V(r, \theta) = \frac{1}{2}kr^2 + \frac{3}{2}kr^2\cos^2(\theta).$$

We can rewrite the potential as:

$$V(r, \theta) = \frac{1}{2}kr^2(1 + 3\cos^2(\theta)).$$

The Hamiltonian for a particle in 2D is given by:

$$H = -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + V(x, y).$$

In polar coordinates (r, θ), the Laplacian operator is:

$$\nabla^2 = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2}{\partial \theta^2}.$$

So the Hamiltonian in polar coordinates is:

$$H = -\frac{\hbar^2}{2m}\left[\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2}{\partial \theta^2}\right] + V(r, \theta).$$

The time-independent Schrödinger equation is $H\psi = E\psi$. We can try to separate variables by assuming a solution of the form $\psi(r, \theta) = R(r)\Theta(\theta)$. [...]

This confirms the energy spectrum is:

$$E = \hbar\sqrt{\frac{k}{m}}\left(2n_x + n_y + \frac{3}{2}\right).$$

Final Answer: $E = \hbar\sqrt{\frac{k}{m}}\left(2n_x + n_y + \frac{3}{2}\right)$.

Figure 5. **Example model outputs.** We pick a question from AIME24 (*left*), MATH500 (*middle*), and GPQA (*right*), where our model generates the correct answer. The black text is the prompt, the light blue text is the reasoning trace, and the blue text is the answer of **s1-32B**. The gray ellipsis [...] indicates that the text was trimmed to fit this page, but the generated text is actually longer.

Budget forcing In Table 3 we compare the test-time scaling methods we have introduced in §3. Overall, we find that *budget forcing* provides perfect control, good scaling, and leads to our best AIME24 score. Thus, this is the method we use for **s1-32B** in Figure 1 and in §4. In Table 4, we compare different strings for extrapolating performance. We find that "Wait" generally gives the best performance.

Class-conditional control We provide benchmark scores for this method in §E.1 and summarize three findings here: **(1)** Token-conditional control fails without budget forcing, as our model cannot reliably count tokens - even when trained to do so. **(2)** Under step-conditional control, the model generates a similar total number of tokens when given different step targets, as the model goes from few steps with many tokens per step, to many steps with few tokens in each step. Thus, the model learns to hack its way around the compute constraint making the controllability of this method mediocre. **(3)** Class-conditional control can work - telling a model to simply think longer can increase its test-time compute and performance, which leads good scaling in Table 3.

Table 4. **Budget forcing extrapolation ablations.** We compare ignoring the end-of-thinking delimiter twice and appending none or various strings.

Model	AIME 2024	MATH 500	GPQA Diamond
No extrapolation	50.0	**93.0**	57.6
2x without string	50.0	90.2	55.1
2x "Alternatively"	50.0	92.2	**59.6**
2x "Hmm"	50.0	**93.0**	**59.6**
2x "Wait"	**53.3**	**93.0**	**59.6**

Rejection sampling Surprisingly, we find that simply sampling until the generation fits a specific length leads to an inverse scaling trend as depicted in Figure 6. In §E.2 we inspect a question, which was answered correctly by the model when rejection sampling for ≤ 4000, but not for the ≤ 8000 token setting. In the ≤ 4000 setting the model directly jumps to the correct approach, while for the ≤ 8000 setting it backtracks a lot. We hypothesize that there is a correlation such that shorter generations tend to be the ones where the model was on the right track from the start, whereas longer ones tend to be ones where the model made mistakes and thus backtracks or questions itself. This leads to longer samples often being wrong when rejection sampling and thus the inverse scaling trend.

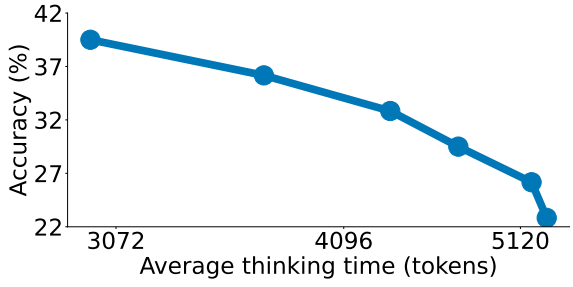


Figure 6. **Rejection sampling on AIME24 with s1-32B.** We sample with a temperature of 1 until all generations have less than (from left to right) 3500, 4000, 5000, 8000, and 16000 thinking tokens requiring an average of 655, 97, 8, 3, 2, and 1 tries per sample.

6. Discussion and related work

6.1. Sample-efficient reasoning

Models There are a number of concurrent efforts to build models that replicate the performance of o1 (OpenAI, 2024). For example, DeepSeek-r1 and k1.5 (DeepSeek-AI et al., 2025; Team et al., 2025) are built with reinforcement learning methods, while others rely on SFT using tens of thousands of distilled examples (Team, 2025; Xu et al., 2025; Labs, 2025). We show that SFT on only 1,000 examples suffices to build a competitive reasoning model matching o1-preview and produces a model that lies on the pareto frontier (Figure 2). Further, we introduce budget forcing which combined with our reasoning model leads to the first reproduction of OpenAI's test-time scaling curves (OpenAI, 2024). Why does supervised finetuning on just 1,000 samples lead to such performance gains? We hypothesize that the model is already exposed to large amounts of reasoning data during pretraining which spans trillions of tokens. Thus, the ability to perform reasoning is already present in our model. Our sample-efficient finetuning stage just activates it and we scale it further at test time with budget forcing. This is similar to the "Superficial Alignment Hypothesis" presented in LIMA (Zhou et al., 2023), where the authors find that 1,000 examples can be sufficient to align a model to adhere to user preferences.

Benchmarks and methods To evaluate and push the limits of these models, increasingly challenging benchmarks have been introduced, such as Olympiad-level science competitions (He et al., 2024; Jain et al., 2024; Zhong et al., 2023) and others (Srivastava et al., 2023; Glazer et al., 2024; Su et al., 2024; Kim et al., 2024; Phan et al., 2025). To enhance models' performance on reasoning-related tasks, researchers have pursued several strategies: Prior works have explored continuing training language models on specialized corpora related to mathematics and science (Azerbayev et al., 2023; Yang et al., 2024), sometimes even synthetically generated data (Yu et al., 2024). Others have developed training methodologies specifically aimed at reasoning performance (Zelikman et al., 2022; 2024; Luo et al., 2025;

Yuan et al., 2025; Wu et al., 2024a). Another significant line of work focuses on prompting-based methods to elicit and improve reasoning abilities, including methods like Chain-of-Thought prompting (Wei et al., 2023; Yao et al., 2023a;b; Bi et al., 2023; Fu et al., 2023; Zhang et al., 2024b; Xiang et al., 2025; Hu et al., 2024; Diao et al., 2024). These combined efforts aim to advance the reasoning ability of language models, enabling them to handle more complex and abstract tasks effectively.

6.2. Test-time scaling

Methods As we introduce in §3, we differentiate two methods to scale test-time compute: **parallel** and **sequential**. The former relies on multiple solution attempts generated in parallel and selecting the best outcome via specific criteria. These criteria include choosing the most frequent response for majority voting or the best response based on an external reward for Best-of-N (Brown et al., 2024; Irvine et al., 2023; Levi, 2024). Unlike repeated sampling, previous sequential scaling methods let the model generate solution attempts sequentially based on previous attempts, allowing it to refine each attempt based on previous outcomes (Snell et al., 2024; Hou et al., 2025; Lee et al., 2025). Tree-based search methods (Gandhi et al., 2024; Wu et al., 2024b) offer a hybrid approach between sequential and parallel scaling, such as Monte-Carlo Tree Search (MCTS) (Liu et al., 2024; Zhang et al., 2023; Zhou et al., 2024; Choi et al., 2023) and guided beam search (Xie et al., 2023). REBASE (Wu et al., 2024b) employs a process reward model to balance exploitation and pruning during tree search. Empirically, REBASE has been shown to outperform sampling-based methods and MCTS (Wu et al., 2024b). Reward models (Lightman et al., 2023; Wang et al., 2024b;c) play a key role in these methods. They come in two variants: outcome reward models and process reward models. Outcome reward models (Xin et al., 2024; Ankner et al., 2024) assign a score to complete solutions and are particularly useful in Best-of-N selection, while process reward models (Lightman et al., 2023; Wang et al., 2024b; Wu et al., 2024b) assess individual reasoning steps and are effective in guiding tree-based search methods.

Limits to further test-time scaling We have shown that budget forcing allows extrapolating test-time compute in §4, e.g., improving AIME24 performance from 50% to 57%. However, it has two key limitations when scaling further: it eventually **flattens out** (Figure 4), and the **context window** of the underlying language model constrains it. Despite these constraints, our work shows test-time scaling across a wide range of accuracies (Figure 1), partly because scaling down test-time compute behaves predictably and does not suffer from these constraints.

Continuing test-time scaling will require approaches that can further extrapolate test-time compute. How can we get such extrapolation? There may be improvements to budget forcing such as rotating through different strings, not only "Wait", or combining it with frequency penalties or higher temperature to avoid repetitive loops. An exciting direction for future work is also researching whether applying budget forcing to a reasoning model trained with reinforcement learning yields better extrapolation; or if RL allows for new ways of test-time scaling beyond budget forcing. Our work defines the right metrics (§3.2) – Control, Scaling, and Performance – to enable future research and progress on extrapolating test-time compute.

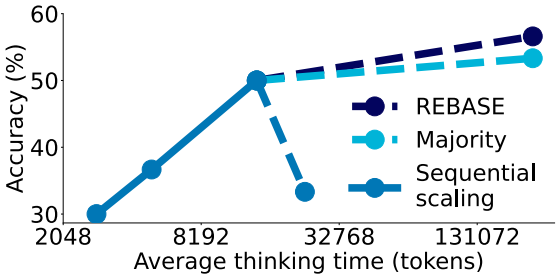


Figure 7. **Scaling further with parallel scaling methods.** All metrics averaged over the 30 questions in AIME24. Average thinking tokens for REBASE do not account for the additional compute from the reward model. For sequential scaling, we prompt the model to use up to (from left to right) 32, 64, 256, and 512 steps. For REBASE and majority voting we generate 16 parallel trajectories to aggregate across.

Parallel scaling as a solution Parallel scaling offers one solution to the limits of sequential scaling, thus we augment our sequentially scaled model with two methods: **(I) Majority voting:** After generating k solutions, the final solution is the most frequent one across generations; **(II) Tree search via REBASE:** We use the REBASE process reward model, which is initialized from LLaMA-34B and further finetuned on a synthetic process reward modeling dataset (Wu et al., 2024b). We then aggregate the solutions generated by REBASE via majority voting. As shown in Figure 7, augmenting our model with REBASE scales better than majority voting, and even sequential scaling in this scenario. However, REBASE requires an additional forward pass at each step for the reward model adding some computation overhead. For sequential scaling, when prompted to use up to 512 steps, for 12 out of the 30 evaluation questions the model generates a response that exceeds the context window leading to a large performance drop. Overall, we find that these parallel scaling methods complement sequential scaling thus they offer an avenue for scaling test-time compute even further; beyond fixed context windows.