
e1: Learning Adaptive Control of Reasoning Effort

Michael Kleinman*, Matthew Trager, Alessandro Achille, Wei Xia, Stefano Soatto
AWS Agentic AI

Abstract

Increasing the thinking budget of AI models can significantly improve accuracy, but not all questions warrant the same amount of reasoning. Users may prefer to allocate different amounts of reasoning effort depending on how they value output quality versus latency and cost. To leverage this tradeoff effectively, users need fine-grained control over the amount of thinking used for a particular query, but few approaches enable such control. Existing methods require users to specify the absolute number of desired tokens, but this requires knowing the difficulty of the problem beforehand to appropriately set the token budget for a query. To address these issues, we propose *Adaptive Effort Control*, a self-adaptive reinforcement learning method that trains models to use a user-specified fraction of tokens relative to the current average chain-of-thought length for each query. This approach eliminates dataset- and phase-specific tuning while producing better cost-accuracy tradeoff curves compared to standard methods. Users can dynamically adjust the cost-accuracy trade-off through a continuous effort parameter specified at inference time. We observe that the model automatically learns to allocate resources proportionally to the task difficulty and, across model scales ranging from 1.5B to 32B parameters, our approach enables approximately 3x reduction in chain-of-thought length while maintaining or improving performance relative to the base model used for RL training.

1 Introduction

Models can improve accuracy by increasing the amount of test-time compute, *i.e.*, the length of chain-of-thought (CoT). In fact, with unbounded test-time compute, many problems can be trivially solved through brute force search [Achille and Soatto, 2025]. However, this would result in untenable compute cost and latency. Instead, AI agents should learn to allocate their compute budget to achieve the best trade-off between cost/latency and accuracy, which depends on the task, the user, and the environment: When compute is cheap, producing more tokens may be desirable. When the user assigns low economic value to a task, compute usage should be proportionally lower, lest the user spend more than they gain.

Optimization of this tradeoff can be framed as optimizing the expected *net-reward*:

$$J = E[R - \lambda T]$$

where R is the task reward, T is the time used (number of tokens), and λ is the cost per token (which depends on the model and infrastructure). The user then selects λ based on the perceived value of the task, and the accuracy trade-off they want to achieve.

From an economic perspective, J is the right measure to evaluate model performance. However, using J as a training loss creates issues: when λ is fixed, complex tasks requiring many tokens cannot achieve positive net-reward, incentivizing the model to not attempt solutions. Recent work has addressed some of these training stability issues by using a modified penalty for

*Correspondence to Michael Kleinman <mjklein@amazon.com>

token usage [Arora and Zanette, 2025] or adaptive values of λ [Xiang et al., 2025]. However these approaches do not provide inference-time control over the compute-accuracy tradeoff.

To address these issues, we introduce **Adaptive Effort Control** (AEC), a simple auto-adaptive training method that teaches models to optimally allocate resources while maintaining stable learning dynamics. Rather than asking the model to achieve an absolute trade-off controlled by λ , we ask the model to produce an answer using less than r times the average amount of tokens used by correct solutions to the problem so far:

$$\begin{aligned} \mathcal{L} = \max R \\ \text{s.t. } \frac{T}{T_{\text{avg}}} < r. \end{aligned} \tag{1}$$

Choosing $r < 1$ forces the model to learn cheaper solutions than those it has already found, without requiring a λ tuned to the particular task. Since time is normalized by the average length of a correct solution, equal effort is devoted to optimizing both easy and difficult problems. This contrasts with the λ parametrization, which provides no learning signal for easy problems (where any reasoning chain trivially satisfies most λ values) or hard problems (which have no admissible reasoning chain for most λ values).

While this procedure does not directly optimize the net-reward J , we show that models trained this way achieve a better cost-versus-accuracy curve than models trained directly to optimize J . If desired, the resulting model can then be calibrated to take λ as input, or any other user-facing control-knob for effort allocation. Rather than fixing the hyperparameter r , we specify it in the prompt on a per-query basis during training, and use a reward function that depends on the value of r . Our resulting RL objective integrates directly with standard RL training procedures like GRPO [Shao et al., 2024].

After training with **AEC**, on our resulting model e1, increasing the effort parameter r leads a monotonic increase in both the number of generated tokens and task performance. We observe this across a range of tasks that require varying number of tokens.

Additionally, across model scales from 1.5B to 32B parameters, our approach enables significant efficiency gains. On math tasks, we find that our training approach results in an approximately 3x reduction in chain-of-thought length while maintaining or even improving performance relative to the base model used for our RL training. Furthermore, after training, it is possible to calibrate the effort parameter to provide linear control over either relative accuracy or relative token usage, enabling intuitive user control tailored to different applications.

2 Related Work

Test-Time Scaling for LLMs. Several works have shown that LLM performance can be improved by increasing computation at inference time in different ways. Methods can involve repeated sampling [Lightman et al., 2023, Brown et al., 2024], search-based techniques with a verifier [Wu et al., 2024, Snell et al., 2024, Manvi et al., 2024, Uscidda et al., 2025], or increasing the “reasoning,” or chain-of-thought computation, before producing an answer [Guo et al., 2025, Team et al., 2025]. However, increased accuracy does not necessarily lead to more “intelligent” behavior Achille and Soatto [2025], as the models can simply improve accuracy by using the additional resources for extended brute force search, without learning new strategies. On the other hand, learning to jointly minimizing both inference time and error rate leads to maximizing the *algorithmic information* in the trained model [Achille and Soatto, 2025, Theorem 1.1].

Efficient Reasoning Models. While accuracy typically improves with longer chain-of-thought reasoning, longer responses incur higher latency and computational costs. As a result, not only is more compute leading to diminishing return, but to decreasing net utility. For example, a model that improves performance in a math test but returns every answer after the time limit has passed is worth less than a model that guesses the answer, *i.e.*, performs at chance level. A recent class of methods aim for efficient reasoning [Arora and Zanette, 2025, Zhang and Zuo, 2025] using a penalty for the length of the response. [Xiang et al., 2025] use an adaptive length penalty depending on pass-rate of the model on the problem. These works typically use a fixed hyperparameter which leads to a single operating point along an accuracy-versus-tokens curve. In contrast, in our work we specify a control input along with the query, which allows us to trace the entire cost-control curve of the trained model and therefore operate at any point along the curve.

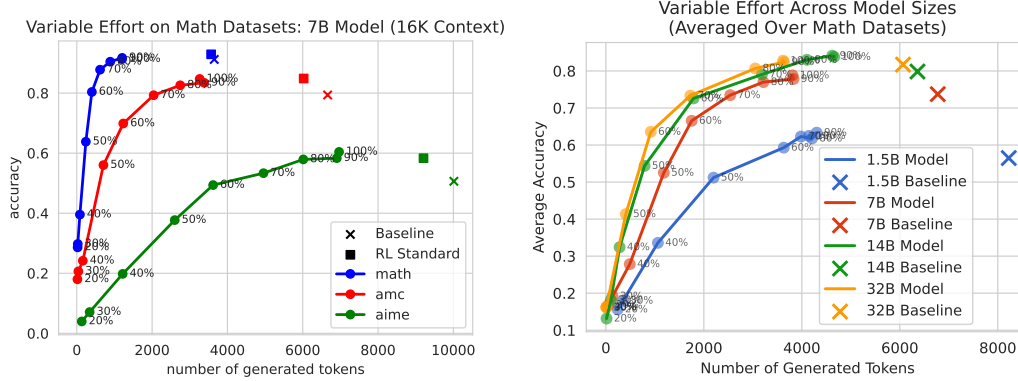


Figure 1: **Variable Effort Control and Improved Reasoning Efficiency.** (Left) After training, increasing r in the prompt leads to increasing accuracy and number of generated tokens across datasets of varying difficulty. Note that in addition to allowing control, this training approach also allows us to significantly reduce the number of generated tokens compared to the baseline (in crosses) and with regular RL training (squares). (Right) We apply our approach across model scales ranging from 1.5B to 32B and show results averaged across math datasets (see Appendix Fig. 13 for dataset-specific results), with our approach enabling more efficient reasoning and improved performance relative to the base models. We observe superior performance vs number of generated token curves with increasing model size.

Length-Control for Reasoning Models. To enable control over the generation length, Muennighoff et al. [2025] proposed the S1 method which modifies inference by either forcing early termination at a specified number of tokens, or forcing a continuation if the model wanted to terminate early. Similarly, Aggarwal and Welleck [2025] train models to use either an exact or maximum number of tokens by modifying the reinforcement learning objective. However, as discussed in the introduction, the number of tokens depends on the difficulty of the task, and fixing a number can lead to unstable learning dynamics. Instead of fixing an hard thinking budget, concurrent works propose training model with soft prompts requesting “low”, “medium”, or “high” thinking effort [Agarwal et al., 2025, He et al., 2025]. In particular, He et al. [2025] presents a recipe to train models to reason at such effort levels though their approach involves multiple training stages, relies on custom system prompts per effort level, and doesn’t allow fine-grained control.

3 Method

Our objective is to train a model that can effectively balance solution quality and reasoning cost on a per-query basis according to user-specified preferences. Given a task instance x and a ground-truth answer y^* , a natural training objective to teach resource allocation is to optimize not just the expected reward obtained by the model, but rather the time-penalized net-reward:

$$J = R(y, y^*|x) - \lambda T_y \quad (2)$$

where T_y is the number of tokens used to answer. Alternatively, rather than using a soft-penalty, Aggarwal and Welleck [2025] introduce a time-constrained reward function:

$$\hat{R}(y, y^*|x, T_{\max}) = R(y, y^*|x) \cdot \mathbb{I}(T_y < T_{\max}), \quad (3)$$

which assigns zero reward to solutions that overrun the provided thinking budget $T_{\max}$. However, fixing λ or $T_{\max}$ during training results in suboptimal learning, since the objective makes it optimal to not even attempt a solution to complex tasks that would not fit the given budget. Making λ and $T_{\max}$ instance-specific parameters passed in the prompt partially addresses the issue. However, their optimal value depends both on the task difficulty and the current stage during training – e.g., using a small budget at the beginning of the training may prevent the model from finding any solution and prevent learning.

Adaptive Effort Control. To address these issues, we introduce *Adaptive Effort Control* (AEC), a modified training objective that automatically adapts the target effort level for the task to both the

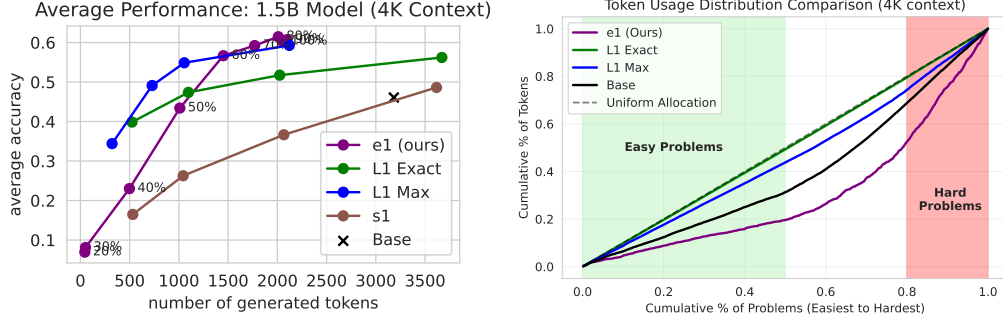


Figure 2: **(Left) Comparison against other length-control baselines.** Our controllable model (e1) obtains superior accuracy-token tradeoff compares to L1-Exact, and even slightly better than L1-Max (which does not allow precise control over the generation length) when the effort level is above 60%. See Appendix Fig. 12 for performance on individual datasets. **(Right) Adaptation to query difficulty.** In addition to providing control over the number of tokens, e1 is automatically able to adapt the length of the reasoning chain to the difficulty of the input task. We see this by ordering the problems from easiest to hardest (based on fraction of times the model solves a problem correctly) and plotting the cumulative percentage of total tokens used for solving those problems. A diagonal line indicates equal tokens allocation to each problem, and increasingly convex curves indicate fewer token allocation to easier problems. Our approach (shown for $r = 1$, purple) allocates a smaller percentage of tokens to easier problems than other length-control baselines and the base model used for RL training. In particular our model allocates only $\sim 20\%$ of tokens to the easiest 50% of problems.

complexity of the task and the current status of training. Given a training example x and a sampled target *effort ratio* $r > 0$, we proceed as follows. First, we construct an input x_r that encoded both the task x and the target effort-ratio r . In our experiments, we simply concatenate r to the prompt as follows:

$$x_r = \text{Concat}(x, \text{"Let's spend } \{r \cdot 100\}\% \text{ effort."})$$

Given x_r , we sample N chain-of-thought traces ($N = 16$ in our experiments) and denote with $S = \{h_1, \dots, h_{N_S(x_r)}\}$ the set of traces that terminate with a correct solution. We define the average time to find a solution to x_r as the average length of the successful traces:

$$T_{\text{avg}}(x_r) = \frac{\alpha}{N_S(x_r)} \sum_{i=1}^{N_S} \ell(h_i),$$

or $T_{\text{avg}}(x_r) = \infty$ if no correct solution was found. $\alpha = 2.5$ is a fixed scalar across all examples allowing $r \in [r_{\min}, 1]$ to vary in a more interpretable range when prompting the model. We then define the *AEC reward* for each trace h as:

$$R_{\text{AEC}}(y, y^* | x, r) = R(y, y^* | x) \cdot \mathbb{I} \left(\frac{\ell(h)}{T_{\text{avg}}(x_r)} < r \right). \quad (4)$$

That is, the objective gives no reward to any trace that uses more than a fraction r of the average time required to solve the problem. Crucially, since the constraint is relative to the *current* average time required to solve the *specific task* x , the AEC reward automatically adapts both to the complexity of the problem and the current stage during training (*i.e.*, it allows longer time to solve a task if the model hasn't yet learned any solution). If the model is not yet able to solve the task (and so $T_{\text{avg}}(x_r) = \infty$) the constraint is trivially satisfied, leaving the model free to explore.

Training Algorithm. We trained our model e1 using our AEC objective (Eq. 4). For optimization we used GRPO [Shao et al., 2024] without any modifications. We use binary rewards for answer correctness such that $R(y, y^* | x) = \mathbb{I}(y = y^*)$.

Training data. We create our training dataset as follows. We start with a training dataset $\mathcal{D} = \{(x^i, y^{i*})\}_{i=1}^N$ for reinforcement learning where x^i denotes the input question, and y^{i*} denotes the corresponding answer. For each question x^i we sample $r \in [r_{\min}, 1]$. This augmentation results in a

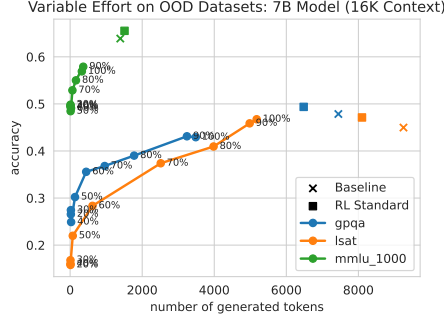


Figure 3: **Transferability of effort control skill across different domains:** Despite training on math questions, we find that after training with our objective that we can modulate the number of generated tokens and performance on different domains by varying the effort parameter r . Across all of the datasets we find that the number of generated tokens and accuracy increases with the effort parameter supplied in the prompt.

new training dataset of $\mathcal{D}^{new} = \{(x_r^i, y^{i*})\}_{i=1}^N$ that we use for our reinforcement learning training. At inference time, we augment the questions in a test dataset with a particular effort level r of interest and quantify the ability to control the number of generated tokens and performance.

4 Experimental Setup

Models and Datasets. We start with an R1-Distilled Qwen {1.5,7,14,32}B as the base model, unless otherwise stated. This is a distilled version of the DeepSeek R1 model [Guo et al., 2025] onto the Qwen 2.5 family of models [Qwen et al., 2025]. This model had strong mathematical capabilities but does not have the ability to modulate its thinking effort. We train using the DeepScaler dataset [Luo et al., 2025] which consists of $\sim 40,000$ math problems consisting of question and ground-truth output. We focus our evaluations on a variety of math datasets of varying difficulty (AIME 2024, AMC, and MATH500 [Hendrycks et al., 2021]) and also evaluate the ability to transfer to out-of-domain datasets. We also evaluate the model out-of-domain using GPQA [Rein et al., 2024], LSAT [Zhong et al., 2023], and MMLU [Hendrycks et al., 2020].

Hyperparameters and Training Details. We train and evaluate with the relative effort $r \in [0.2, 1]$. We set $\alpha = 2.5$. Unless explicitly stated we trained and evaluated with a context length of 16K, which was significantly larger than the initial average number of tokens outputted over our training set. For training, we use a batch size of 320, and a learning rate of $1e-6$. We train for 1000 steps unless otherwise stated. We trained with GRPO using 16 rollouts per question. We used the VERL framework for training [Sheng et al., 2024].

Evaluation. We evaluate the ability for our approach to modulate both the performance and number of generated tokens. We assess this by plotting the accuracy against number of generated tokens as we vary the effort level parameter r across the datasets described above. When doing evaluations, we sample 16 times per question and report the average accuracy (fraction of questions solved correctly). We evaluate by augmenting the evaluation dataset using with the effort $r \in [0.2, 0.3, \dots, 1]$.

Baselines. We compare our approach against length-control baselines L1 [Aggarwal and Welleck, 2025] and S1 Muennighoff et al. [2025]. S1 modifies inference to extend inference by appending "Wait" if the model wanted to conclude before the user’s targeted number of tokens, or by forcing termination at the user’s specified number of tokens. In our experiments with S1, we enforce strict following of the token specification. L1 trains a model using reinforcement learning to output an answer while following a desired number of tokens specified in the prompt. L1 consists of two variants: L1-exact aims to exactly follow of the number of tokens specified by a user, whereas L1-max aims to satisfy a maximum number of tokens constraint.

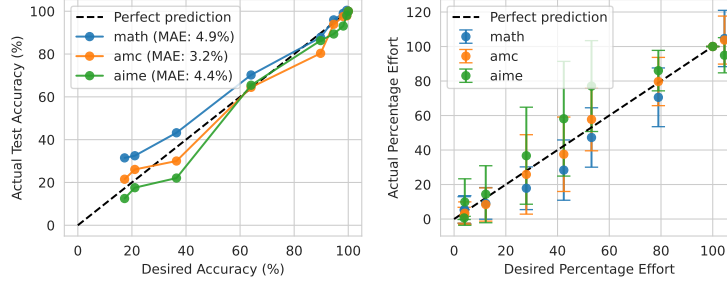


Figure 4: **Calibration for interpretable control over percentage effort.** After training, we can reparameterize the effort parameter r to linearly control either the **(left)** relative accuracy or **(right)** relative number of tokens and enable more intuitive user control.

5 Results

5.1 Variable Effort Control and Improved Reasoning Efficiency

We first assess whether our approach enables variable effort control on math evaluation datasets. In Fig. 1a we show that after training, increasing the effort r in the prompt leads to increasing accuracy and number of generated tokens across datasets of varying difficulty. We find that the model uses more tokens on more difficult problems, for example allocating more budget for problems in the AIME dataset which is generally considered more difficult than the Math500 dataset, where our model correctly allocates less budget. Note that in addition to allowing control, this training approach also allows us to significantly reduce the number of generated tokens while increasing accuracy (even allowing us to achieve over 60% on AIME 2024 using a 7B model). Our training approach also leads to significant compression ($\sim 3x$) compared to the base model used for RL training and models trained with standard reinforcement learning.

In Fig. 1b we show that our approach can be applied to models of various sizes (ranging from 1.5B to 32B), enabling more efficient reasoning and improved performance relative to the base model used for our RL training. We observe better performance vs number of generated token curves with increasing model size. We show aggregated results in Fig. 1b, and we show dataset-specific results in Appendix Fig. 13.

5.2 Adaptation to query difficulty and comparison to length-control baselines

In addition to providing a new mechanism for controlling the length of generation, we show that our approach has superior performance-cost tradeoffs to other token-based specifications methods (Fig. 2a). We do this ablation on 1.5B models trained on a smaller context length (4096) using the DeepScaler model [Luo et al., 2025] as the base model to align with previous work and allow us to leverage available online checkpoints for comparison [Aggarwal and Welleck, 2025]. Additionally, we show that training with our approach leads to smaller token allocation to easier problems, and larger token allocation to more difficult problems (Fig. 2b). Our approach (shown for maximum effort $r = 1$) allocates fewer tokens to easier problems than uniform allocation and the baseline model, allocating $\sim 20\%$ of tokens to 50% of problems. This differs from token-based specification which uses the specified number of tokens regardless of question difficulty.

5.3 Transferability of effort control skill across different domains

Despite training on math questions, we find that after training with our objective that we can modulate the number of generated tokens and performance on datasets in different domains by varying the effort parameter r (Fig. 3). We observe improved performance on LSAT with reduced number of generated tokens compared to the model before our RL training (on math questions) with our effort objective and compared a RL model trained without the effort objective. We observe reduced performance on GPQA and MMLU compared to the baseline and an RL trained model, though our approach uses significantly fewer tokens. Importantly, across all of the datasets we find that the number of generated tokens and accuracy increases with the effort parameter supplied in the prompt.

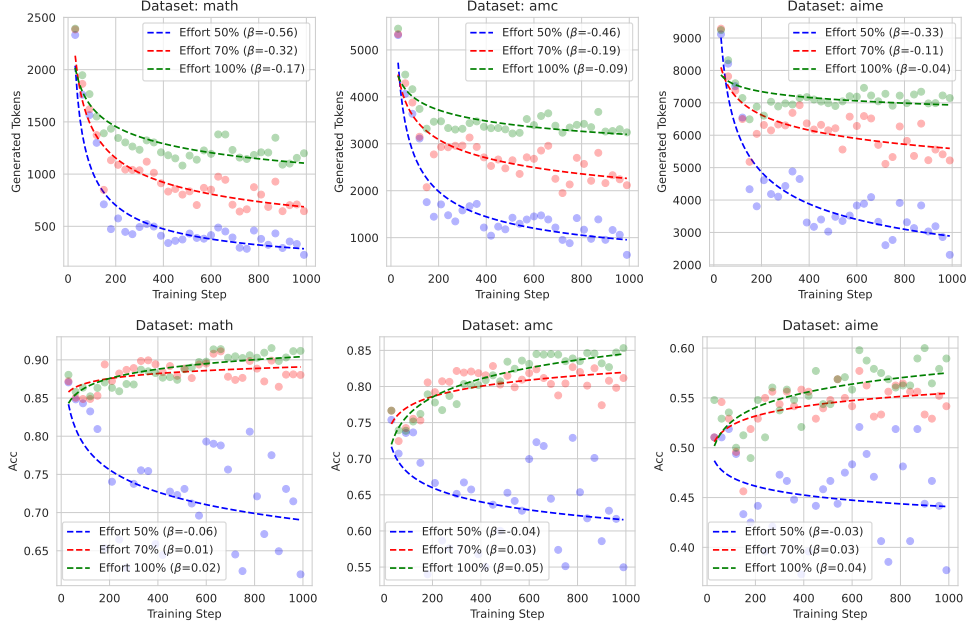


Figure 5: **Learning dynamics of AEC training objective.** We plot the accuracy and number of tokens as a function of training step for the three math evaluation datasets. Early in training, the model has similar accuracy and number of generated tokens regardless of the effort level, which means that the model needed to learn to become sensitive to the effort level. We find that the number generated tokens roughly decreases as a power law with exponent β across training steps. We find that shorter problems (Math 500) decrease faster (larger β) than longer problems (AIME). Despite token lengths decreasing through training, we find that accuracy increases (provided that the effort level was above 70%).

5.4 Calibration for interpretable control over percentage effort

We saw previously that our approach enables increasing tokens and performance with increasing r , but it was not precisely clear how changing r affects the number of generated tokens or accuracy. We show that a simple reparameterization of the effort parameter after training can enable linear control for either relative tokens or relative accuracy (defined with respect to the tokens or accuracy with $r = 1$).

When calibrating for the relative number of tokens, we define the maximum effort for a problem x to be the average number of tokens generated $T(x, r = 1)$ when using the largest value of r . On a small validation set (15 examples from each of Math500, AMC and AIME) we compute the desired percentage effort as $d(r) = \mathbb{E}_x \left[\frac{T(x, r)}{T(x, r=1)} \right]$, allowing us to map from user-specified desired percentage effort d to r (that is used to augment the prompt). On new examples, we similarly compute the outputted percentage effort as $o(x, r) = \frac{T(x, r)}{T(x, r=1)}$. We find that the outputted percentage effort is highly correlated with the desired percentage effort (Fig. 4b) even though the absolute number of tokens differs significantly across these datasets. We summarize these results by computing the average relative error $e = \mathbb{E}_{x \in D, r} [o(x, r) - d(r)]$ over the different datasets D , finding that the average relative error is $\sim 11\%$ for each of these datasets (Fig. 14).

We similarly calibrate based on accuracy by computing $a(r) = \frac{\text{Acc}(r)}{\text{Acc}(r=1)}$ using the same validation set. We compare the desired accuracy $a(r)$ that a user can specify against the accuracy on the test set in Fig. 4a, finding an average error of $\sim 4\%$.

5.5 Learning dynamics of our AEC objective

We have observed that after our reinforcement learning training that specifying the effort level in the prompt enables varying the accuracy and number of tokens on a per-question basis. But how does this ability emerge through the RL training?

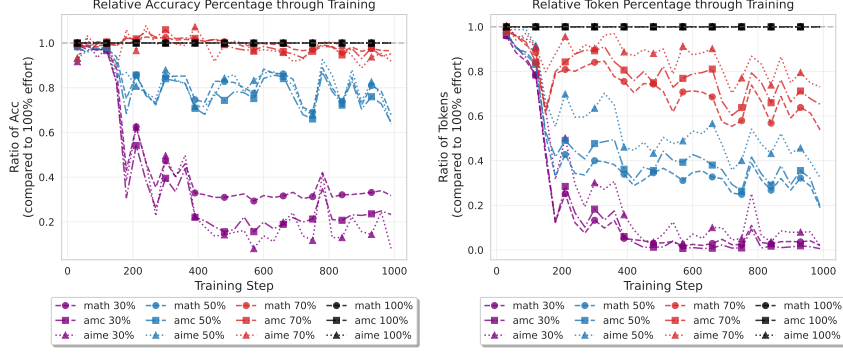


Figure 6: **Relative accuracy and relative token length through training.** We evaluate the relative accuracy and relative number of tokens through training by comparing relative to 100% effort. Despite problems being of varying difficulty and taking varying length, we find that the relative accuracy across datasets remains roughly constant over training for a fixed effort level, as the curves are overlapping. We observe a similar phenomenon when analyzing the ratio of number of tokens.

We analyze how the accuracy and number of generated tokens vary over learning (Fig 5). We plot the accuracy and number of tokens as a function of training step for the three math evaluation datasets. At the beginning of training, the number of tokens and accuracy was similar regardless of the effort level supplied in the prompt. We find that the number of tokens roughly decreases as a power law. We find that shorter problems (Math500) decrease faster than longer problems (AIME). Despite token lengths decreasing through training, we find that accuracy increases (provided that the effort level was above 70%). This highlights that in addition to enabling controllable inference-time performance, our training enables both more efficient responses and performance increase. We observe similar learning dynamics across models of various sizes (Appendix Fig. 16, Fig. 17, Fig. 18), with larger models decreasing the number of tokens faster during our training.

In Fig. 6, we observe that the relative number or tokens (or accuracy) compared to 100% effort is consistent across datasets throughout training for various effort level parameters r .

5.6 Effect of objective and robustness to prompt

Here, we aim to ablate what aspects of our reward function resulted in the ability for inference-time control using the effort parameter. We examine the effect of the prompt and the objective function, and show this ablation for a 1.5B model. For the following experiments we use a different prompt:

$$x_i^{new} = "(\{p\} \text{ pts}) \{x_i\} \text{ We'll get } \{p\} \text{ points for correctly answering the question.}"$$

We varied $p \in \{1, 2, 3, 4, 5\}$, which corresponded to linearly varying our effort parameter $r \in [0.48, 0.8]$ in Eq. 4. In Fig. 7 we show that training with this prompt variant also leads to increasing performance and number of generated tokens with increasing effort (now specified through “points”). This shows that our sensitivity to the effort parameter is not dependent on the exact prompt used.

To ablate whether the dependency on the effort parameter depended on the specifics of the reward objective used, we compared our reward objective with a modified objective that correspond to a length penalty denoted by $R_\lambda(y, y^*)$ where

$$R_\lambda(y, y^*) = \mathbb{I}(y = y^*) - \frac{\lambda}{p} T_y. \quad (5)$$

In Fig. 7, we used either a small length penalty $\lambda = 1e - 4$ or a large length penalty $\lambda = 1e - 3$ and similarly varied $p \in \{1, 2, 3, 4, 5\}$. When the length penalty is small with $\lambda = 1e - 4$ (shown in squares in the figure) while we observe a reduction in token lengths (and increase in accuracy compared to the baseline), we do not observe the ability to modulate the number of generated tokens though this parameter p . When we used a larger length penalty of $\lambda = 1e - 3$, we observed the ability to modulate the number of generated tokens and performance through the parameter p , though the number of tokens and performance was significantly reduced (triangles in plot). The training with a length penalty is therefore highly sensitive to the value of λ , whereas our approach is more robust as it depends on the statistics of the model during a rollout, without requiring this hyperparameter.

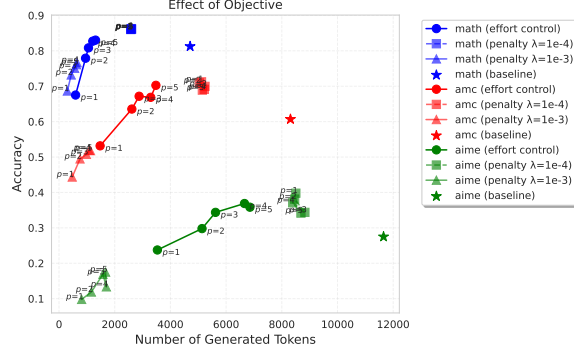


Figure 7: **Effect of RL training objective and robustness to prompt.** We compare our effort objective a modified objective (Eq. 5) that penalizes the length of the generation regardless of problem difficulty. For this experiment we use a modified prompt that specifies the effort through a “points” parameter p . After training with the effort objective (circles in plot), we observe the ability to increase accuracy and the number of generated tokens by increasing p , highlighting that the dependence on the effort parameter does not depend on the specific prompt used. Training with a small length penalty ($\lambda = 1e - 4$) does not allow modulating the number of generated tokens or accuracy by varying the effort parameter p (squares). Training with a small length penalty ($\lambda = 1e - 3$, triangles) leads to the ability to modulate the number of tokens and accuracy (within a narrow range), but results in significantly fewer generated tokens (less than 2000 for all datasets).

5.7 Selecting effort level for a task: conceptual depth of the trained model

As we have anticipated, once the model is trained, its operation on a particular environment should be *evaluated* with a criterion that combines space and time, with a unit conversion factor λ specific to that environment, task, user, and model. We call the compound criterion *conceptual depth*, a property of a trained model, in analogy with the notion of *logical depth* of the data used to train it Bennett [1988]. Logical depth refers for the time it takes a Turing Machine to solve a problem using a program not too much longer than the minimum (its Kolmogorov Complexity). Theorem 1.1 of Achille and Soatto [2025] implies that training with our AEC objective maximizes the *algorithmic information* in the trained model. The resulting model exhibits the ability to trade off time T and reward R or error E . We call their combination, modulated by the environment-specific $\lambda_{\mathcal{D}}$, the conceptual depth of the trained model, defined as

$$\kappa(\mathcal{D}, T, \lambda_{\mathcal{D}}) = E(\mathcal{D}) + \lambda_{\mathcal{D}}T, \quad (6)$$

where $\lambda_{\mathcal{D}}$ represents the cost of time as determined by the environment, $E(\mathcal{D})$ denotes the error rate on a dataset $\mathcal{D}$, and T denotes the average length of the chain-of-thought when solving queries from the task. A particular error rate carries different value depending on the time available or the cost of time, which is determined by the environment. For example, if an exam has a time limit, the value of any outcome rendered after the limit is zero. We should emphasize that the above Eq. 6 is for *evaluating* a trained model, not for training a model to be an optimal solver (for that, see [Achille and Soatto, 2025]).

To optimally specify the number of tokens/effort for a task given this objective, we should select the optimal chain-of-thought length T^* such that

$$T^*(\mathcal{D}, \lambda_{\mathcal{D}}) = \arg \min_T \kappa(\mathcal{D}, T, \lambda_{\mathcal{D}}).$$

If the error rate decreases with increasing T , the minimum occurs when:

$$\frac{dE_{\mathcal{D}}(T)}{dT} = -\lambda_{\mathcal{D}}.$$

In Fig. 8, we plot the conceptual depth curves for various values of $\lambda_{\mathcal{D}}$ and math tasks of varying difficulty. We set $\lambda_{\mathcal{D}}$ values inversely proportional to the time constraints in actual testing scenarios: $\lambda_{\text{MATH}} = \lambda$, $\lambda_{\text{AMC}} = \lambda/3$, $\lambda_{\text{AIME}} = \lambda/12$, reflecting that students receive 12 minutes per AIME problem and 3 minutes per AMC problem, while we approximate (as this is not a standardized exam) 1 minute per MATH500 problem. We vary the value of λ across the different panels while keeping the

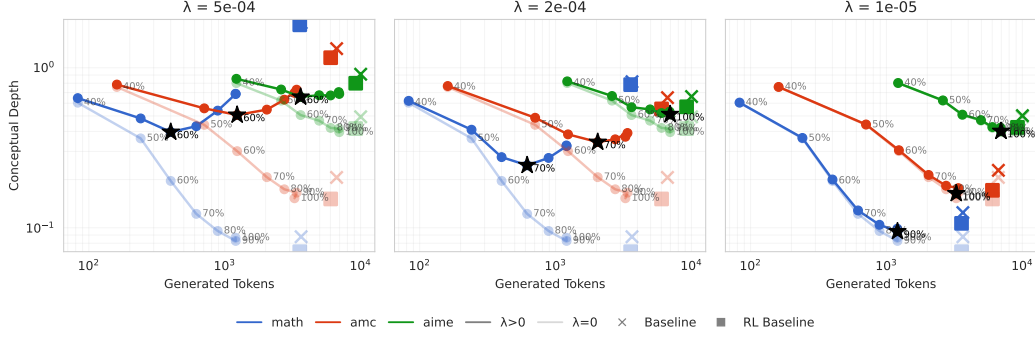


Figure 8: **Selection of effort level for a task.** A user or agent should select the number of chain of thought tokens for a task to balance the performance on the task and the cost incurred for achieving that performance. When these two criteria are combined into an objective (the conceptual depth), the minimum of this curve is the optimal operating point for the model on the task (shown in black stars). The error rate (where $\lambda = 0$) is shown in a lighter shade. While the optimal operating point that minimizes the conceptual depth can be specified by either specifying the number of tokens or the effort level, we find that the optimal operating point is approximately consistent across our effort levels, even though the absolute number of tokens differs significantly.

scaling across the datasets fixed. We show that even though the number of tokens needed to minimize $\kappa(\mathcal{D})$ varies significantly across the different math tasks, the effort level needed to minimize $\kappa(\mathcal{D})$ is similar across the different tasks for a given value of λ . We show this is the case for various values of λ (shown across the 3 panels). For larger values of λ the models use a smaller effort level, and for smaller values of λ , the models use a larger effort level.

The minimum of $\kappa(\mathcal{D})$ occurs at similar effort levels across the different math datasets as $\frac{dE_{\mathcal{D}}(T)}{dT}$ evaluated at the different effort levels is inversely proportional to the (real) time allocated per question, which scales $\lambda_{\mathcal{D}}$ (Fig. 19). Controlling the effort therefore provides a simple and practical knob to select an optimal operating point across different tasks with different cost-of-time.

6 Conclusion

We introduced AEC, a reinforcement learning method for enabling prompt-based control over the *relative* amount of chain-of-thought tokens for reasoning models, while being adaptive to the difficulty of a problem. In addition to enabling control over the generation length, our approach enables an approximately 3x reduction in chain of thought length while maintaining or improving performance relative to the base model.

Our approach requires just a simple modification to traditional RL training. In this work we specified a single query-dependent parameter in the input during RL training and using a corresponding query-dependent reward objective, which allowed inference-time control after training. Future work could extend our approach to specify multiple query-relevant parameters and using analogous example-dependent reward function to what we used. Additionally we applied our approach for a reasoning model – future work could extend our training approach to multi-agent systems.

References

- Alessandro Achille and Stefano Soatto. AI agents as universal task solvers. *arXiv preprint arXiv:2510.12066*, 2025.
- Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.
- Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning. *arXiv preprint arXiv:2503.04697*, 2025.
- Daman Arora and Andrea Zanette. Training language models to reason efficiently. *arXiv preprint arXiv:2502.04463*, 2025.
- Charles H Bennett. *Logical depth and physical complexity*. Citeseer, 1988.
- 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.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- Qianyu He, Siyu Yuan, Xuefeng Li, Mingxuan Wang, and Jiangjie Chen. Thinkdial: An open recipe for controlling reasoning effort in large language models. *arXiv preprint arXiv:2508.18773*, 2025.
- Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.
- Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, Li Erran Li, et al. Deepscaler: Surpassing o1-preview with a 1.5 b model by scaling rl. *Notion Blog*, 2025.
- Rohin Manvi, Anikait Singh, and Stefano Ermon. Adaptive inference-time compute: Llms can predict if they can do better, even mid-generation. *arXiv preprint arXiv:2410.02725*, 2024.
- Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.
- Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. 2025. URL <https://arxiv.org/abs/2412.15115>.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
- Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.
- 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.
- Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.
- Theo Uscidda, Matthew Trager, Michael Kleinman, Aditya Chattopadhyay, Wei Xia, and Stefano Soatto. Latts: Locally adaptive test-time scaling. *arXiv preprint arXiv:2509.20368*, 2025.
- Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.
- Violet Xiang, Chase Blagden, Rafael Rafailov, Nathan Lile, Sang Truong, Chelsea Finn, and Nick Haber. Just enough thinking: Efficient reasoning with adaptive length penalties reinforcement learning. *arXiv preprint arXiv:2506.05256*, 2025.
- Jixiao Zhang and Chunsheng Zuo. Grpo-lead: A difficulty-aware reinforcement learning approach for concise mathematical reasoning in language models. *arXiv preprint arXiv:2504.09696*, 2025.
- WanJun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. Agieval: A human-centric benchmark for evaluating foundation models, 2023.

Appendix

7 Additional Details

7.1 Difficulty of a problem

In the paper we have occasionally referred to the “difficulty” of a problem. This is not some objective entity or a property of a problem: some problems are easy to some and difficult to others. So, “difficult” is model- (or individual-)dependent. Human polling allows us to measure empirically the average perceived difficulty of a problem. For example, AIME problems are considered on average more difficult than MATH500 problems, which is why the organizers of the competition allocate more time for the former. We define a (necessarily) model-dependent notion of difficulty for a trained model as follows.

We define the the difficulty of a problem with respect to a model through it’s difficulty profile

$$d(T|x) = 1/p_{correct}(T|x) \quad (7)$$

where

$$p_{correct}(T|x) = E_{h \sim p_{model}(h|x)} [\mathbb{I}(e(h) = y^*) \cdot \mathbb{I}(\text{len}(h) \leq T)], \quad (8)$$

where h denotes a model’s generated text, $\text{len}(h)$ denotes the length of the generated text, $e(h)$ denotes the extracted answer, and T denotes a token budget. This is an extension of the definition proposed by [Lightman et al., 2023, Snell et al., 2024] where we also factor in the tokens needed to produce the solution. In this way, the difficulty depends both on the probability of providing a correct solution and the length of the reasoning needed to produce the solution. We say a problem is more difficult than another if its difficulty profile dominates another, in that it is higher for all T . Note that $d(T|x)$ corresponds to the expected number of attempts needed to produce a correct answer to the question.

7.2 Qualitative Examples

We show qualitative examples of the responses to an easy problem at effort level 100% and 50% in Fig. 9 and Fig. 10 respectively². Note that for this question, while both solutions obtain the correct answer, the length of the solution is significantly shorter for at 50% effort level. We also show a generation for a more complicated AIME math question (Fig. 11).

7.3 Additional Figures

While we showed aggregate results across datasets in Fig. 1, in Fig. 13 we show results for on individual datasets for the different model sizes. Similarly we show results for different datasets for our comparison against token-based control approaches (Fig. 12). We also discuss additional experiments below.

²We leveraged the LaTeX formatting in Aggarwal and Welleck [2025] for presenting the generations of our effort control model.

Example

Question

You have seven bags of gold coins. Each bag has the same number of gold coins. One day, you find a bag of 53 coins. You decide to redistribute the number of coins you have so that all eight bags you hold have the same number of coins. You successfully manage to redistribute all the coins, and you also note that you have more than 200 coins. What is the smallest number of coins you could have had before finding the bag of 53 coins?

Let's think step by step and output the final answer within boxed{}. Let's spend 100% effort.

Effort Percentage Specified

100%

Model Response

Thinking Process

Let me try to figure out this problem step by step. So, the setup is that I have seven bags of gold coins, each containing the same number of coins. Then, I find an additional bag with 53 coins, making it eight bags in total. I redistribute all the coins so that each of the eight bags has the same number of coins. After redistribution, I note that I have more than 200 coins in total. I need to find the smallest number of coins I could have had before finding the 53-coin bag.

Okay, let me break this down. Let's denote the number of coins in each of the original seven bags as 'n'. So, the total number of coins I had initially is $7n$. Then, I found a bag with 53 coins, so the total number of coins after finding it becomes $7n + 53$.

Now, I redistribute all these coins equally among eight bags. Let's denote the number of coins in each of the eight bags after redistribution as 'm'. So, we have $8m = 7n + 53$. This equation tells me that $7n + 53$ must be divisible by 8 because m has to be an integer.

[Thinking process truncated for visualization ...]

Therefore, the smallest n is when $k=3$:

$$n = 8 \cdot 3 + 5 = 24 + 5 = 29.$$

So, $n=29$.

Let me verify this. If $n=29$, then the initial total is $7 \cdot 29 = 203$. Adding 53 gives $203 + 53 = 256$.

256 divided by 8 is 32, which is an integer. So, that works. Also, 256 is more than 200, as required.

Therefore, the smallest number of coins I could have had before finding the bag of 53 coins is 203.

The smallest number of coins you could have had before finding the bag of 53 coins is 203.

Correct Answer

203

Generated Tokens

1183

Figure 9: Example model response with effort level 100% for a problem from Math500. 1183 tokens were generated.

Example

Question

You have seven bags of gold coins. Each bag has the same number of gold coins. One day, you find a bag of 53 coins. You decide to redistribute the number of coins you have so that all eight bags you hold have the same number of coins. You successfully manage to redistribute all the coins, and you also note that you have more than 200 coins. What is the smallest number of coins you could have had before finding the bag of 53 coins?
Let's think step by step and output the final answer within boxed{ }. Let's spend 50% effort.

Effort Percentage Specified

50%

Model Response

Thinking Process

Let n be the original number per bag. Total before: $7n + 53$. After adding 53, total is $8n$. So $7n + 53 = 8n$. Also $8n > 200 \Rightarrow n > 25$. So $7n = 8n - 53$. $8n - 53$ must be divisible by 7. $8n \equiv 53 \pmod{7}$. $53 \pmod{7} = 4$, $8 \equiv 1 \pmod{7}$, so $n \equiv 4 \pmod{7}$. Smallest $n > 25$ with $n \equiv 4 \pmod{7}$: 26? $26 \pmod{7} = 5$. 27 $\pmod{7} = 6$. 28 $\pmod{7} = 0$. 29 $\pmod{7} = 1$. 30 $\pmod{7} = 2$. 31 $\pmod{7} = 3$. 32 $\pmod{7} = 4$. So $n = 32$. Total coins = $8 \cdot 32 = 256$. Before: $256 - 53 = 203$.

203

Correct Answer

203

Generated Tokens

224

Figure 10: Example model response with effort level 50% for a problem from Math500. 224 tokens were generated.

Example

Question

Each vertex of a regular octagon is independently colored either red or blue with equal probability. The probability that the octagon can then be rotated so that all of the blue vertices end up at positions where there were originally red vertices is $\frac{m}{n}$, where m and n are relatively prime positive integers. What is $m + n$?
Let's think step by step and output the final answer within boxed{ }. Let's spend 50% effort.

Effort Percentage Specified

50%

Model Response

Thinking Process

The probability is the number of colorings fixed by some rotation divided by the total number of colorings. But maybe it's easier to use Burnside's lemma. However, the problem is about rotations where blue vertices end up on originally red positions. So we need colorings where the blue vertices are a subset of the red vertices under some rotation. This is equivalent to the set of blue vertices being a union of some orbits under rotation. Since the octagon is regular, the group is cyclic of order 8. The possible rotations that could fix the condition are those by multiples of 45 degrees. For each divisor d of 8, we look at rotations by $360/d$ degrees. The number of colorings fixed by rotation by $360/d$ degrees is $2^{8/d}$. But I might be mixing things up. Alternatively, [Thinking process truncated for visualization ...]

But wait, no, this counts the number of colorings fixed by each rotation, but overlapping colorings are counted multiple times.

So to get the actual number, inclusion-exclusion is needed, but it's complicated.

Alternatively, since the problem is about probability, and the total colorings is 256, and the average number of fixed colorings is $290/8=36.25$, which is not an integer. So perhaps this approach is wrong.

Alternatively, maybe the number is 36, as computed before, but $36/256=9/64$, which is $m/n=9/64$, $m+n=73$.

But I'm not sure.

The probability is $\frac{9}{64}$, so $m + n = 9 + 64 = 73$.

73

Correct Answer

371

Generated Tokens

3503

Figure 11: Example model response with effort level 50% for a problem from AIME. 3503 tokens were generated.

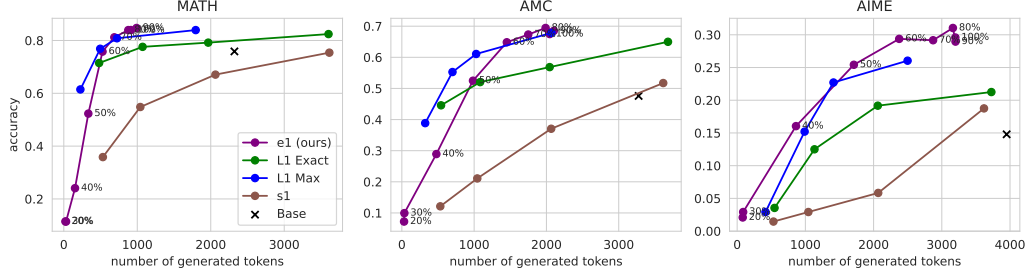


Figure 12: Comparison against other token-based control approaches separated per dataset. Same setup as Fig. 2a.

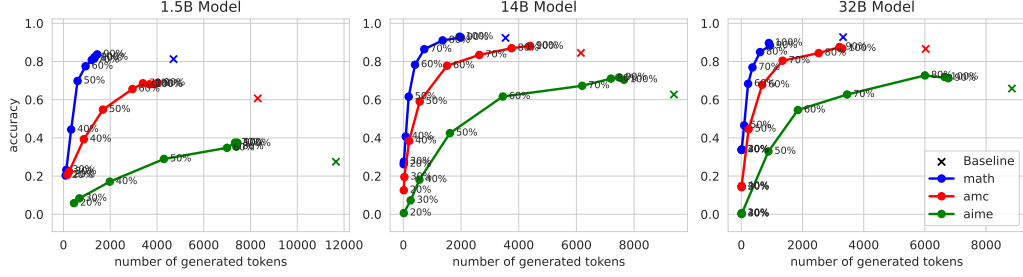


Figure 13: Same as Fig. 1a for various model sizes. These were aggregated to form Fig. 1c.

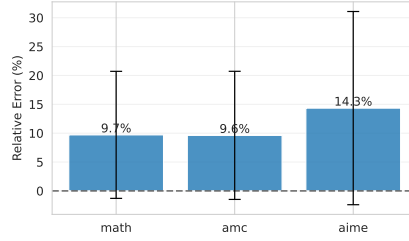


Figure 14: Calibration of percentage effort across datasets. We plot the average error (and standard deviation) across effort values from Fig. 4d for each dataset.

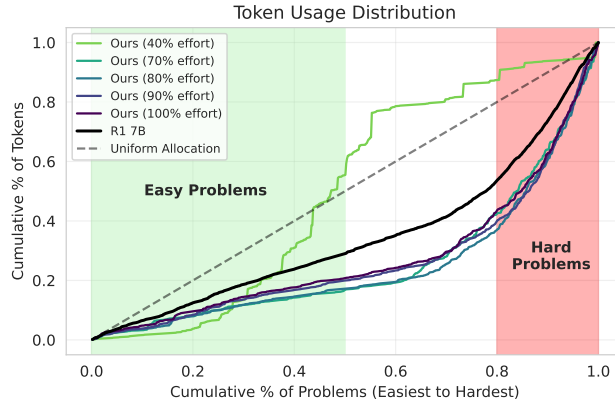


Figure 15: **Adaptation to query difficulty for 7B model, using R1 Distilled Qwen as Base Model.** In addition to providing control over the number of tokens, we show that training with our approach leads to smaller token allocation to easier problems, and larger token allocation to more difficult problems. We show results for various values of r . For values of $r \geq 0.7$, we observe fewer token allocation to easier problems, but not for low percentage effort when the total number of generated tokens and accuracies were significantly lower.

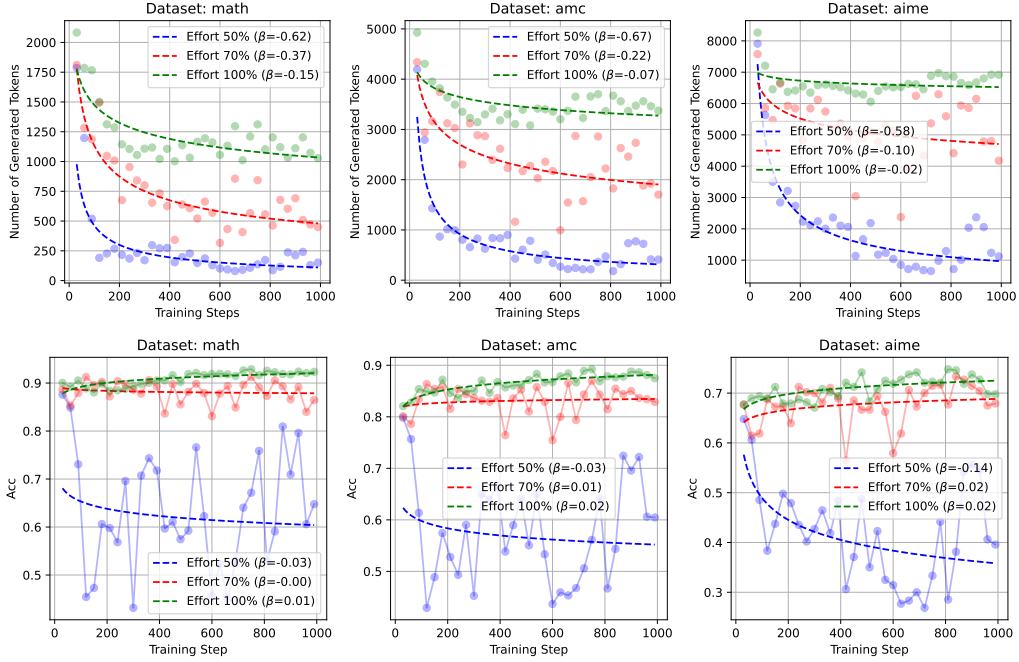


Figure 16: **Learning dynamics of AEC training objective for 32B model.** We plot the accuracy and number of tokens as a function of training step for the three math evaluation datasets. We find that the number of tokens roughly decreases as a power law with exponent β . Despite token lengths decreasing through training, we find that accuracy increases (provided that the effort level was 100%).

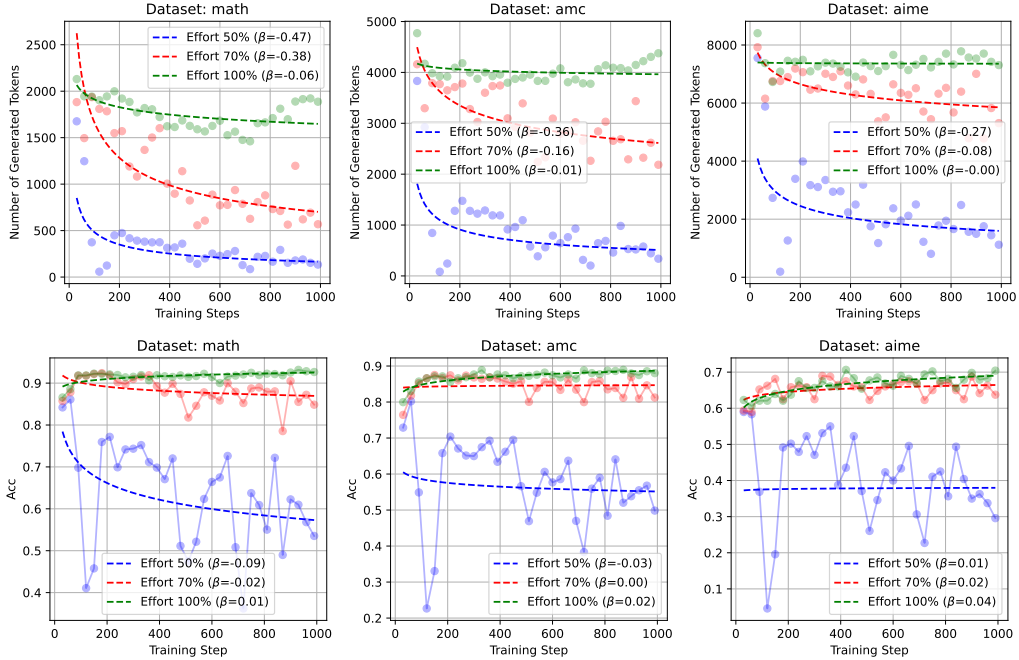


Figure 17: **Learning dynamics of AEC training objective for 14B model.** We plot the accuracy and number of tokens as a function of training step for the three math evaluation datasets. We find that the number of tokens roughly decreases as a power law with exponent β . Despite token lengths decreasing through training, we find that accuracy increases (when the effort level 100%).

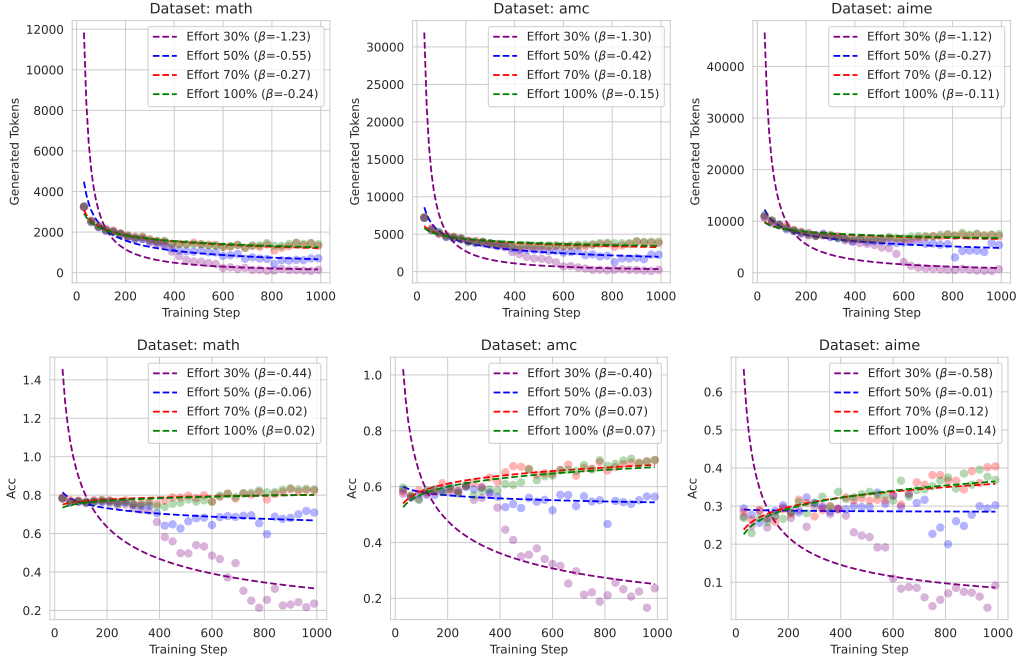


Figure 18: **Learning dynamics of AEC training objective for 1.5B model.** We first plot the accuracy and number of tokens as a function of training step for the three math evaluation datasets. We find that the number of tokens roughly decreases as a power law with exponent β (for effort levels greater than 50%). Despite token lengths decreasing through training, we find that accuracy increases (provided that the effort level was above 70%).

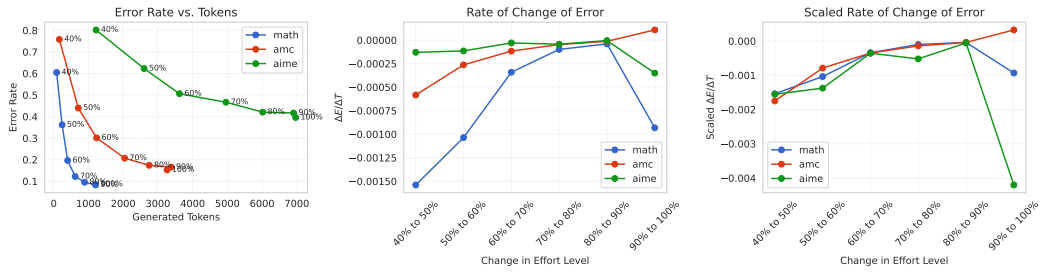


Figure 19: **(Left)** We plot the error rate across generated tokens for effort levels 40% to 100%. **(Center)** We compute the change in error over the change in tokens across the different effort levels (dashed lines). **(Right)** When rescaling these curves by the time allocated to problems from these datasets (12 minutes for AIME, 3 minutes for AMC, and we approximate 1 minute for Math500), we observe that the rescaled curves are approximately overlapping (solid lines). This means that the change in error rate $\frac{dE_D(T)}{dT}$ for different datasets $\mathcal{D}$ is inversely proportional to the real time allocated per question at different effort levels.