CATS: CONFORMALIZED ADAPTIVE TEST-TIME SCALING

Sadegh Akhondzadeh [1] **Soroush H. Zargarbashi**[2] **Simone Antonelli**[2] **Aleksandar Bojchevski**[1]
[1] University of Cologne, [2] CISPA Helmholtz Center for Information Security
[akhondzadeh, zargarbashi, antonelli, bojchevski]@cs.uni-koeln.de

ABSTRACT

Reasoning language models (RLMs) have shown significant improvements through test-time scaling. However, these gains come at the cost of more resource-intensive and slower inference, leading to latencies that directly impact user experience. This cost can be reduced by avoiding overthinking and responding quickly to simple queries. Ideally, models should adapt their thinking strategy to the difficulty of the task. Such adaptivity is increasingly feasible because recent foundation models already provide discrete options for reasoning effort (e.g., low, high). We provide a framework for adaptive test-time scaling with a guarantee of controlled risk. Our method estimates task difficulty based on the probability of success at each reasoning level (while keeping additional cost minimal) and adjusts its output via conformal risk control. Specifically, our method guarantees that the probability of producing an incorrect answer remains below a user-specified tolerance level, while adaptively allocating less compute to easier queries and more to harder ones. Our results demonstrate that an appropriate reasoning level can be selected automatically while ensuring rigorous statistical guarantees.

1 INTRODUCTION

Large language models (LLMs) have demonstrated remarkable complex reasoning capabilities by generating extended chains-of-thought (Wei et al., 2022; Guo et al., 2025). In recent years, with the development of increasingly capable models, a new paradigm of scaling called *test-time scaling* has emerged. Test-time (inference) scaling shows that allocating additional computational resources at inference time, for example generating more tokens, or sampling multiple reasoning trajectories, can considerably improve the quality of the output in terms of accuracy, robustness or reasoning performance (Snell et al., 2024; Muennighoff et al., 2025; Wang et al., 2022).

These quality gains come at the cost of computational overhead and delayed responses, which directly impact resources and user experience. Thus, it is very important to properly decide when to use test-time scaling dynamically, based on downstream task difficulty. In other words, while empirically these techniques show a better performance, applying them indiscriminately across various hardness levels is both inefficient and undesirable. This is because the majority of simpler queries can be answered properly with minimal reasoning effort, meaning that this unnecessary *overthinking* only wastes energy and degrades the experience of the end user.

Fig. 1 [Right] verifies this claim by showing the performance gap. In this figure, the performance gap between thinking and non-thinking inference increases with the difficulty of the task. This highlights the importance of *adaptive* test-time scaling. Further, Fig. 1 demonstrates that, in the base case scenario, only a fraction (10%) of test-time scaling is in principle enough to gain almost the maximum performance if we can perfectly estimate difficulty with an oracle. Similarly, with our method, by scaling computation for fewer than half of the queries, we get close (less than 1% loss) to maximum performance. Recent models (He et al., 2025; Yang et al., 2025b; Agarwal et al., 2025) allow for various levels of chain of thought. For example, Qwen3 (Yang et al., 2025b) models allow thinking and non-thinking inference modes that are selectable at the beginning of prompting. Similarly GPT-OSS (Agarwal et al., 2025) allows for direct control over *reasoning effort*, which dictates the relative length of the thinking trajectory (low, medium, and high). Despite

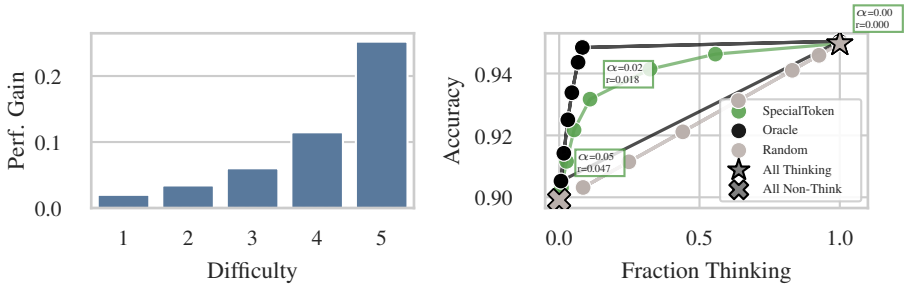

Figure 1: [Left] The performance gap between the thinking and non-thinking inference over the problem difficulty. The results are evaluated on Qwen-8B over the Math-500 dataset [Right] Calibration curves for thinking vs. non-thinking inference. Our SpecialToken approach provides the best trade-offs.

these advances, current systems often leave the decision to the end user, and in many evaluations they use a uniform budget for every query which leaves substantial efficiency on the table.

An alternative line of work uses additional inference-time compute to create several trajectories and returns the majority (Wang et al., 2022; Snell et al., 2024; Yao et al., 2023) as the answer. In this scenario, being able to run in batch and create trajectories in parallel becomes important. Therefore, the user should decide whether to use single inference or a majority-of-k approach with batch inference.

While existing approaches to this trade-off either set a uniformly high inference cost (which leads to wasting resources for a subgroup of queries) or rely on user-adjusted assessments, which come without any statistical guarantee, we address this issue through a risk-controlled decision problem: how to minimize the cost while preserving the accuracy of the always-maximal-cost setup. We propose conformal adaptive test-time scaling (CATS), a method that decides the scaling level based on the difficulty of the prompt. Our method operates automatically while providing a guarantee that the mistake rate (or more formally risk) remains below a predefined tolerance level on the data from the same distribution. CATS supports two directions of test-time scaling: (i) extended reasoning: CATS decides the predefined reasoning length the model should use. (ii) CATS decides whether or not the given query needs majority voting over parallel inference. Our key idea is to estimate the prompt-conditional success probability under each reasoning budget level. We propose various classifiers to predict the required budget before generation. Then through conformal risk control, we suggest a Pareto-front in the trade-off between tolerated risk (bounded performance degradation), and the computational cost. Among our classification approaches, we propose "internal special token probing" where we add a trainable special token at the end of the prompt which is responsible for deciding the reasoning effort of the downstream task. Our approach keeps the LLM weights frozen, and only introduces a very small number of parameters ($\sim 0.002\%$ of the LLM parameters) to train a lightweight classifier over the hidden state of that token. Interestingly after predicting the difficulty from it, the key-value cache of the rest of the prompt remains usable for the generation. We further propose a dataset which is a mix of tasks with different difficulties, to simulate the performance of the model in real-life.

2 Formal Framework for Mode Selection

We study the following mode-selection problem. Consider a language model $\mathcal{M}$ that, for each prompt, must choose among t ordered inference options represented as a set of inference policies $\mathcal{Z} = \{z_1, \ldots, z_t\}$, where higher-indexed options entail greater computational cost (e.g., latency) but typically yield better performance. Given a prompt x, and conditional on any choice of inference z, the model returns a response $y \sim \mathcal{M}(x \mid z)$. Note that, here $\mathcal{M}$ is inherently a randomized model, meaning that y is a single sample from the space of the outputs. Notably, while the free-form space of outputs is complex, we consider y to be a projection into the space of answers; which could in practice be computed through another forward pass even on a much lighter language model. In the same space, there exists the ground truth y^*, and with that we define the success rate of the scale

z as $p(x \mid z) := \Pr_{y \sim \mathcal{M}(x|z)}(y = y^* \mid x)$. To start, we add an assumption that by increasing the scaling level z, the chance of returning the true label increases. While this is commonly correct, there are corner cases where a lower scale performs better. We discuss this later in § 3. Note that in this work, we address test-time scaling regimes where the scale z can only be selected prior to the LLM inference; e.g. sequential scaling where the model receives the thinking effort z alongside the prompt before starting the generation, or parallel scaling where the final answer is determined through majority vote. Even in the latter case we should decide the mode prior to the generation as majority voting can gain significant speed-ups by resampling via batched prediction.

Threshold Policy Function Let $s : \mathcal{X} \times \mathcal{Z} \to [0, 1]$ be a score function tending to align with the success probability $p(x \mid z)$ with an extra constraint that $s(x, \cdot)$ is increasing in z. We define the following threshold-based policy function:

$$\pi_\lambda(x) \;:=\; \inf\Big(\{z \in \mathcal{Z} : s(x, z) \geq \lambda\} \cup \{z_t\}\Big).$$

This policy clearly favors the smallest scale that passes a constant threshold λ. Clearly, by increasing λ, our policy favors higher scales, which makes the setup more conservative, leading to potentially higher accuracy. Assuming that the probability of success increases over z we want to ensure that our policy $\pi_\lambda(x)$ does not degrade the performance of the model more than an acceptable tolerance level w.r.t. full-scale inference. Consider the example of binary thinking/non-thinking model, we want to ensure that the probability of success at the selected policy does not decrease by more than α compared to the thinking mode (which is the maximum chain of thought). For a policy π_λ we define the following:

Definition 2.1 (Policy accuracy)**.** The expected accuracy of the system under the policy π_λ is:

$$\mathrm{Acc}(\lambda) = \mathbb{E}_{x \sim \mathcal{D}_\mathcal{X}}\Big[\sum_{z \in \mathcal{Z}} p(x \mid z) \cdot \mathbb{I}[\pi_\lambda(x) = z]\Big], \tag{1}$$

where $\mathcal{D}_\mathcal{X}$ denotes the distribution over input prompts.

For $\lambda_{\max}$ which sets the policy to always generate with highest scaling level, $\mathrm{Acc}_{\max} = \mathrm{Acc}(\lambda_{\max})$ is roughly equal to the model's accuracy (the only discrepancy is with the corner cases where lower scale performs better). For each x, we define the risk as the proportion of the success probability that we lose by the policy π_λ; formally $L(x, \lambda) = p(x \mid z_t) - \sum_{z \in \mathcal{Z}} p(x \mid z) \cdot \mathbb{I}[\pi_\lambda(x) = z]$. This risk will be 0 at $\lambda_{\max}$ since the policy always favors the highest scale, and a non-negative value for any other λ. For any λ, the expected future risk is defined as

$$R(\lambda) := \mathbb{E}_{x \sim \mathcal{D}_\mathcal{X}}\left[L(x, \lambda)\right] = \mathbb{E}_{x \sim \mathcal{D}_\mathcal{X}}\left[p(x \mid z) - \sum_{z \in \mathcal{Z}} p(x \mid z) \cdot \mathbb{I}[\pi_\lambda(x) = z]\right] = \mathrm{Acc}_{\max} - \mathrm{Acc}(\lambda), \tag{2}$$

Here we provide a procedure to tune the variable λ that comes with a guarantee of bounded risk; i.e. $\mathrm{Acc}(\lambda) \geq \mathrm{Acc}_{\max} - \alpha$ for any predefined tolerance α. Where $\mathrm{Acc}(\cdot)$ is evaluated over the inputs from $\mathcal{D}_\mathcal{X}$. We follow conformal risk control, originally proposed by Angelopoulos et al. (2022), that allows such adjustment and provides the test-time guarantee.

Theorem 2.2 (Conformal Risk Control - rephrased)**.** *Let λ be a parameter (larger λ yields more conservative output), and $L_i : \Lambda \to (-\infty, B]$ for $i = 1, \ldots, n+1$ be exchangeable random functions. If (i) L_is are non-increasing right-continuous w.r.t. λ, (ii) for $\lambda_{\max} = \sup \Lambda$ we have $L_i(\lambda_{\max}) \leq \alpha$, and (iii) $\sup_\lambda L_i \leq B < \infty$, then we have:*

$$\mathbb{E}_\mathcal{D}[L_{n+1}(\hat{\lambda})] \leq \alpha \quad \textit{for} \quad \hat{\lambda} = \inf\{\lambda : \frac{\sum_{i=1}^n L_i(\lambda)}{n+1} + \frac{B}{n+1} \leq \alpha\}$$

The primary step towards our goal is to learn an estimator that can predict the success rate for each x, and each inference type z. Later we train a model to approximate s with $\hat{s} : \mathcal{X} \to [0, 1]^{|\mathcal{Z}|}$. Through this estimator we define a threshold-based inference policy. As mentioned, by increasing the scale level from z_k to z_{k+1} the computation effort, and (in almost all cases) the performance increase. Therefore for a threshold λ if λ increases we want either to remain in the same scale level, or jump to a higher level.

Our next step is to calibrate the policy over a representative set of prompts - prompts that are exchangeably sampled from the distribution of future inputs - and compute the conformal value for λ. Through Theorem 2.2 we compute a Pareto-front over the trade-off between relative accuracy, and efficiency as defined below.

Definition 2.3 (Efficiency)**.** Given a relative cost assignment $c_1, \ldots, c_t$ for each scale level, the *efficiency/cost* $C(\lambda)$ is the expected cost of the policy evaluated at parameter λ; i.e. a weighted average over $c_1, \ldots, c_t$ with weights equal to proportion of the times each scale is called. Formally

$$C(\lambda) = \mathbb{E}_{x \sim \mathcal{D}_\mathcal{X}}[\sum_{z \in \mathcal{Z}} c_z \cdot \mathbb{I}[\pi_\lambda(x) = z]] \tag{3}$$

Choice of score function Our approach works for any score function that is increasing over z. One simple example is to train a model to imitate $p(x \mid z)$ and take the probability estimations as $s(x, z) = \hat{p}(x \mid z)$. Notably this function will introduce a bias against harder examples. For example consider two input prompts x_1, and x_2 where the estimated success probability for them are $\boldsymbol{s}_1 = [0.3, 0.4, 0.5]$ and $\boldsymbol{s}_2 = [0.7, 0.8, 0.9]$. For $\lambda = 0.6$ x_1 always assigns full scale to x_1 and the minimum scale to x_2 while both methods benefit from highest scale (e.g. thinking) with the same probability. Therefore we define uplifted score function based on our probability estimations as $\hat{s}(x, z) = 1 - (s(x, z_t) - s(x, z))$ which assigns 1 to the highest scale and the relative difference from the full-scale to other values of z. Notably, we still rely on an assumption that $p(x \mid z)$ is increasing over z, which fails in a few corner cases. For this, we monotonize our risk function to be able to apply conformal risk control for it.

3 Risk-Controlled Mode Selection

The risk captures the expected accuracy loss incurred by selectively routing queries to the lower scales mode based on the threshold λ. One specific yet common example of our setup is the binary decision between thinking (T) and non-thinking (NT) mode in generation. That directly applies to many models like Qwen3. Without loss of generality we phrase the arguments for this mode. With the uplifted score function the score of thinking mode is always 1 and the other score will be $\hat{s}(x) := \hat{s}(x, z = \mathrm{NT}) = 1 - (s(x, z = \mathrm{T}) - s(x, z = \mathrm{NT}))$ where s is the estimated success probability score. For a fixed λ the policy is

$$\pi_\lambda(x_i) = \begin{cases} T & \text{if } s(x_i) \leq \lambda \\ NT & \text{o.w.} \end{cases} \tag{4}$$

Intuitively $s(x_i)$ estimates the gain in performance when using the Thinking mode over the Non-Thinking mode.

Discussion Our threshold-based policy traces a Pareto front in the efficiency-accuracy trade-off. Each value of λ corresponds to a point on this frontier, and via conformal risk control we estimate the resulting accuracy associated with each λ. While any score function can be used in our framework – meaning that the guarantee remains valid (and we can potentially target any accuracy between the model's accuracy at the lowest and highest scales) – the choice of score function influences other features about the policy.

This score function can be analyzed along two directions: (i) the design of the score given a difficulty estimator, where in § 2 we discussed the effect of using an uplifted score $\hat{s}$ instead of the raw estimated success probability s, and (ii) the accuracy of the difficulty prediction itself, namely how well the predicted probability $s(x, z)$ aligns with the true $p(x \mid z)$. For the binary decision case, let $\mathrm{Acc}_{\min}$ denote the accuracy of the model at the lowest scale. At its worst, any accuracy in the range $[\mathrm{Acc}_{\min}, \mathrm{Acc}_{\max}]$ can be achieved by randomly routing prompts to the highest scale with probability λ, yielding an expected accuracy of $\lambda \cdot \mathrm{Acc}_{\max} + (1 - \lambda) \cdot \mathrm{Acc}_{\min}$. While both this naive approach — which corresponds to a random uniform score function — and training an estimator with the uplifted score function achieve the same accuracy guarantees and define a trade-off frontier, Fig. 1 shows that the latter yields a strictly dominant frontier. In other words, although both approaches attain the same accuracy targets, a better score function enables a more efficient path toward the same performance. Interestingly Fig. 1 shows that by increasing the performance of the estimator we can get very close to the accuracy at highest scale, while using the thinking mode less than 20% of the time.

Our goal is to find the *optimal threshold* λ^* that guarantees a target accuracy level while minimizing computational cost. In order to be able to use Theorem 2.2 we need the following assumption:

Assumption 3.1 (Loss Monotonicity)**.** The loss function $L(x, \lambda)$ is monotonically decreasing with respect to the threshold parameter λ. We assume that for any x if $\lambda_1 \leq \lambda_2$, then $L(\lambda_1) \geq L(\lambda_2)$. This implies that as the threshold increases (e.g. the policy tends to use Thinking mode more often), the loss does not increase. This assumption is satisfied when the score function $\hat{s}(x)$ is positively correlated with the true uplift $\Delta(x) = p(x \mid z = \mathrm{T}) - p(x \mid z = \mathrm{NT})$.

Monotonization of the Empirical Loss Theorem 3.1 implies that as threshold increases, thereby assigning more inputs to the *thinking* mode, the expected loss should be non-increasing. The reason that monotonicity property is central to conformal risk control, is that it ensures the set of thresholds satisfying a given risk constraint forms an interval, enabling efficient threshold selection without additional multiple-testing corrections. In practice, however, this assumption may be violated at the empirical level. In particular, there are corner cases where the model makes incorrect predictions while operating in the thinking mode, such that increasing reliance on this mode decreases the probability of returning the correct output. As a consequence, the empirically estimated loss $L(x, \lambda)$ may exhibit non-monotonic behavior due to finite-sample effects, stochastic prediction errors, or model mis-specification. To address this issue, we apply a *monotonization* procedure to the empirical loss curve. Let $\widehat{L}(\lambda)$ denote the empirical loss associated with threshold λ. We define the monotonized loss as

$$\widehat{L}_{\mathrm{mono}}(x, \lambda) \;=\; \sup_{\lambda' \geq \lambda} \widehat{L}(x, \lambda').$$

This operation replaces each value of the loss with the maximum loss observed at any smaller threshold, thereby enforcing monotonicity. The resulting loss function $\widehat{L}_{\mathrm{mono}}$ forms a conservative upper envelope of the original empirical loss curve.

Monotonization serves two purposes. First, it restores the monotonic structure required for valid threshold selection in Theorem 2.2, ensuring that threshold feasibility can be determined without introducing additional correction terms for searching over multiple candidate thresholds. Second, by upper-bounding the empirical loss, it preserves safety: any threshold that satisfies the risk constraint under the monotonized loss $\widehat{L}_{\mathrm{mono}}(x, \lambda)$ also satisfies it under the original empirical loss $L(x, \lambda)$. Importantly, this procedure does not alter the underlying decision rules or per-sample loss values. Rather, it modifies only the aggregation of empirical risks across thresholds, yielding a monotone surrogate that aligns empirical behavior with the theoretical assumptions underlying conformal risk guarantees.

In practice, the ground-truth success probabilities $p(x \mid z) = \mathbb{P}(\mathcal{M}(x; z) = y^\star \mid x)$ are not directly observable. Even when the correct answer $y^\star$ is known (which can be done in pre-processing for limited number of datapoints), exact evaluation of these probabilities would require integrating over the model's full stochastic generation process, which is computationally intractable for modern large language models. Instead, we approximate these quantities via Monte Carlo sampling. Concretely, for each prompt x_i and inference mode $z \in \{\mathrm{T}, \mathrm{NT}\}$, we query the model independently K times and record the binary outcomes $Y_{i,1}^{(z)}, \ldots, Y_{i,K}^{(z)} \in \{0, 1\}$, where $Y_{i,k}^{(z)} = 1$ indicates a correct prediction on the k-th run. We then define the empirical estimator $\tilde{p}(x_i, z) \;=\; \frac{1}{K} \sum_{k=1}^{K} Y_{i,k}^{(m)}$. By construction, $\tilde{p}(x_i, z)$ is an unbiased estimator of $p(x_i, z)$, i.e. $\mathbb{E}[\tilde{p}(x_i, z) \mid x_i] = p(x_i, z)$, and converges to the true success probability as $K \to \infty$. While $\tilde{p}(x_i, z)$ does not equal the exact success probability for finite K, this estimation noise is symmetric around the true value and does not introduce systematic bias. Crucially, our conformal calibration procedure operates on a collection of calibration samples whose success probabilities are all estimated using the same Monte Carlo protocol. As a result, the induced stochasticities are homogeneous across calibration points and are absorbed into the empirical risk estimation step. This ensures that the validity guarantees of conformal risk control remain applicable, provided that the calibration data and test data are generated under the same sampling procedure and inference protocol.

4 Methods for Predicting Question Difficulty

As Fig. 1 demonstrates, an effective mode selection requires accurately predicting whether a given question would benefit from extended reasoning. In this section, we explore three different ap-

proaches for estimating question difficulty. The output of each method is used as the score for our risk control framework to select when to use thinking mode. Each method offers different trade-offs between computational cost and accuracy.

LLM-as-Judge (Self-Assessment) Our first approach directly queries the model itself to assess question difficulty. We further provide additional detail on judge prompting in § B.

External Encoder Model Our second approach leverages a pre-trained encoder model. Specifically, we use ModernBERT-base (Warner et al., 2025) fine-tuned on our labeled dataset to treat difficulty prediction as a multi-label regression task. BERT-style encoder models prepend a special `[CLS]` token to the input sequence, whose final hidden state serves as an aggregate representation of the entire input which is designed specifically for sequence-level classification tasks. Given a question q, we encode it using ModernBERT and extract the `[CLS]` token's hidden state $\boldsymbol{h}_{\text{CLS}} \in \mathbb{R}^d$ from the final layer. A classification head then maps this representation to two outputs: $\hat{p}_{\text{NT}}$ and $\hat{p}_{\text{T}}$, representing the predicted probability of success under non-thinking and thinking modes, respectively. Unlike approaches that freeze the encoder and only train the classification head, we fine-tune the entire network end-to-end, allowing the encoder to adapt its representations specifically for difficulty prediction. We minimize a combined loss:

$$\mathcal{L} = \mathcal{L}_{\text{BCE}}(\hat{p}, p) + \lambda \cdot \mathcal{L}_{\text{MSE}}(\hat{p}_{\text{T}} - \hat{p}_{\text{NT}}, p_{\text{T}} - p_{\text{NT}}), \tag{5}$$

where $\mathcal{L}_{\text{BCE}}$ is the binary cross-entropy loss for multi-label classification, and the second term encourages the model to correctly predict the *gap* between thinking and non-thinking performance. We set $\lambda = 0.1$ in our experiments. This approach is computationally more efficient at inference compared to the LLM-as-Judge, since encoder models are significantly smaller than the judge model.

Internal Probing with Learnable Token Inspired by prefix-tuning (Li & Liang, 2021), our third approach introduces a learnable special token into the model's vocabulary while keeping the base model parameters frozen. As demonstrated in Fig. 2, this token is appended as a suffix to the question and attends to all previous tokens and produces a representation that can be used for difficulty prediction based on the model's internal states. Furthermore, since this token is added as a suffix, the internal hidden representation of all other tokens remains unchanged. For training, we extend the tokenizer to include a special token `<MODE_PREDICTOR>`. Initially, the embedding of `<MODE_PREDICTOR>` is initialized randomly ($\boldsymbol{e}_{\texttt{<MODE\_PREDICTOR>}} \in \mathbb{R}^d$). For a question q, we construct the input as $[q; \texttt{<MODE\_PREDICTOR>}]$ and extract the hidden state ($\boldsymbol{h}_{\texttt{<MODE\_PREDICTOR>}}$) at the special token position from the final layer:

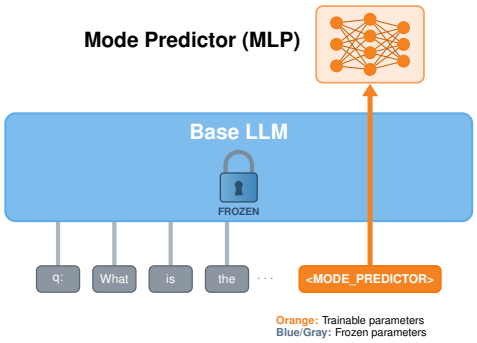

Figure 2: Architecture of the special token: a learnable token is appended to the prompt and its hidden state is used for classification.

$$\boldsymbol{h}_{\texttt{<MODE\_PREDICTOR>}} = \text{LLM}([q; \texttt{<MODE\_PREDICTOR>}])[-1]. \tag{6}$$

A small MLP f_θ maps this representation to predictions: $[\hat{p}_{\text{NT}}, \hat{p}_{\text{T}}] = \sigma(f_\theta(\boldsymbol{h}_{\texttt{<MODE\_PREDICTOR>}}))$, where σ denotes the sigmoid function. The classification head consists of two hidden layers with GELU activations and dropout regularization. Critically, all parameters of the base LLM remain frozen. Only the special token embedding $\boldsymbol{e}_{\texttt{<MODE\_PREDICTOR>}}$ and the classification head f_θ are optimized. This results in an extremely parameter-efficient approach; for example, in Qwen3-8B, only approximately 200K parameters (0.002% of total) are trainable. For finetuning, we use the same combined BCE and gap-prediction loss as explained in the external encoder approach.

Zero-Overhead Inference A major advantage of this approach is that it needs **negligible additional inference cost**. The special token can be appended to the prompt during the prefill phase that is already required for generation. After obtaining the hidden state and making the routing decision, the token is simply droped before proceeding with generation. This means the routing decision is made "for free" as part of the existing forward pass, with only the lightweight MLP classification head adding minimal overhead (∼0.1ms). This method probes the model's internal representations, potentially capturing task-specific features that an external model might miss.

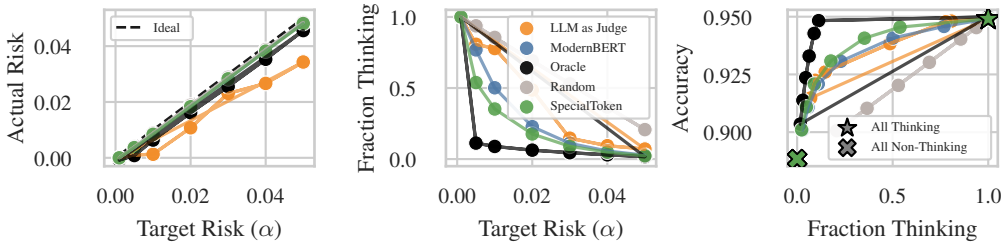

Figure 3: Risk-controlled thinking allocation (Sequential) for Qwen3-8B. [left] Calibration of CRC: Points on the diagonal indicate perfect calibration. [middle] Fraction of samples routed to thinking mode as target risk increases. [right] Accuracy vs. fraction of samples using thinking

5 Problem Setting

Datasets. We evaluate our framework on mathematical reasoning benchmarks, which provide objective correctness criteria essential for measuring success probabilities. Our evaluation suite includes: **GSM8K** (Cobbe et al., 2021), a dataset of grade-school math word problems; **MATH** (Hendrycks et al., 2021), a challenging benchmark of competition mathematics problems spanning algebra, geometry, number theory, and calculus; and **AIME** (Mathematical Association of America, 2024), problems from the American Invitational Mathematics Examination. To prevent data contamination between training and evaluation, we carefully partition each dataset: for MATH, we train on the original training set and evaluate on MATH500 (Lightman et al., 2023); for AIME, we use historical problems from 1983–2023 for training and reserve AIME 2024–2025 for testing; for GSM8K, we use the standard train/test split.

Data Generation for Score Predictors. Training our score predictors requires empirical estimates of success probability under each inference mode. For *sequential scaling* experiments with Qwen3-4B, Qwen3-8B and Qwen-14B (Yang et al., 2025a), we generate 10 independent samples per query with temperature $T = 1$ for both thinking and non-thinking modes. The success probability for each mode is estimated as the fraction of correct answers across these 10 trials, providing a smooth target for regression-based score prediction. For *parallel scaling* experiments, we generate 100 independent samples per query using Qwen3-8B in non-thinking mode with temperature $T = 1$. We then estimate the Pass@1 success probability as the fraction of correct single samples, and the Pass@Majority success probability by simulating majority voting over random subsets of 10 samples and computing the empirical success.

6 Experimental Results

We tested our approach in two different test-time scaling setups: sequential scaling (extended chain-of-thought reasoning), and parallel scaling, which relies on majority voting. Our method consistently delivered reliable accuracy guarantees while also cutting down computational costs.

Sequential Scaling: Extended Chain-of-Thought We applied the framework to Qwen3-8B, a model that can run in either standard (non-thinking) or extended chain-of-thought (thinking) mode. To estimate how likely each mode is to succeed on a given query, we trained two predictors: (1) SpecialToken, (2) ModernBERT encoder. Both were trained on a reserved training set, following the method described earlier. Using conformal risk control, we then set a threshold $\hat{\lambda}$ to keep the risk under a target level α.

We compared our method to several baselines: (1) Oracle, which has access to the true probability of success for each mode, representing the ideal scenario; (2) Random, which assigns modes randomly, based on probabilities that aim to get the model into thinking mode; and (3) LLM-as-Judge, which uses a language model to guess how hard a query is and routes it accordingly.

Fig. 3 shows our key results across three panels. In the left panel, we look at how well our conformal risk control method stays calibrated. The x-axis is the target risk level α, and the y-axis shows

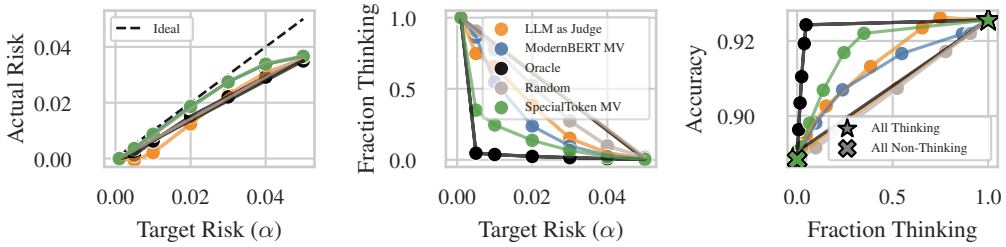

Figure 4: Parallel risk-controlled allocation. [Left] Calibration of CRC [Middle] Compute Efficiency: The fraction of samples routed to the thinking mode as the target risk level increases. [Right] The trade-off between accuracy and the fraction of samples requiring thinking.

the actual risk measured on test data. Ideally, points should stay on or below the diagonal (the black dashed line), meaning the method meets the risk target. Our methods (SpecialToken and ModernBERT) consistently stay close to the diagonal, demonstrating near-perfect calibration. While the framework guarantees risk stays below α, the monotonization step means we can't formally claim it's tight. However, empirically the *left panel* shows that we get very close. On the other hand, the LLM-as-Judge baseline strays further from the diagonal. That's likely because it makes discrete routing decisions, which break the continuity assumption of the score needed for tighter calibration.

The *middle panel* shows how often queries are sent to the thinking mode depending on the target risk level α. As expected, aiming for lower risk (i.e., higher accuracy) means the system has to rely more on the slower, more compute-heavy thinking mode. But our predictors are much more efficient. They send fewer queries into thinking mode compared to both Random and LLM-as-Judge, while still meeting the same accuracy goals. For example, at a target risk of $\alpha = 0.02$, our methods use thinking mode in 25% of cases, whereas Random and LLM-as-Judge use 75% and 50% respectively. The *right panel* gives a direct look at the trade-off between accuracy and cost. It plots how accurate each method is based on how often it uses the thinking mode. The Oracle curve shows the best possible trade-off you could get if you had perfect knowledge of which queries would succeed. Our models come close to that curve, hitting near-optimal accuracy with less computation.

Parallel Scaling: Majority Voting In this parallel test-time scaling, we define two inference modes: (1) Pass@1, which returns the output of a single model sample, and (2) Pass@Majority, which selects the majority vote from multiple samples. Figure Fig. 4 presents analogous results for the parallel scaling scenario. The figure contains three-panel: calibration (left), compute allocation (middle), and the accuracy-efficiency trade-off (right). Our CRC framework extends naturally to this setting, offering the same theoretical risk guarantees while enabling more efficient use of parallel compute.

7 Conclusion

We proposed CATS, a principled framework for trading off efficiency and performance under statistical risk guarantees. By formulating mode selection within the conformal risk control framework, we derived statistically valid policies that bound the accuracy degradation incurred by reduced compute in selecting the mode. We evaluated CATS in two complementary scaling regimes: sequential scaling via extended chain-of-thought reasoning and parallel scaling via majority voting. Across both settings, our approach achieves substantial compute savings while maintaining accuracy above the specified nominal level. To enable efficient routing, we introduced lightweight difficulty predictors: an external encoder-based model (ModernBERT) and an internal special-token probing mechanism. The latter provides near-zero overhead inference-time decisions and yields the strongest empirical performance. Combined with conformal calibration, these predictors consistently draw a Pareto-frontier that is far from the random routing and LLM-as-Judge baselines, and approaching the oracle.

REFERENCES

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.

Anastasios N Angelopoulos, Stephen Bates, Adam Fisch, Lihua Lei, and Tal Schuster. Conformal risk control. *arXiv preprint arXiv:2208.02814*, 2022.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Gongfan Fang, Xinyin Ma, and Xinchao Wang. Thinkless: Llm learns when to think. *arXiv preprint arXiv:2505.13379*, 2025.

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, 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.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Xinglin Wang, Bin Sun, Heda Wang, and Kan Li. Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning. *arXiv preprint arXiv:2401.10480*, 2024.

Guosheng Liang, Longguang Zhong, Ziyi Yang, and Xiaojun Quan. Thinkswitcher: When to think hard, when to think fast. *arXiv preprint arXiv:2505.14183*, 2025.

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.

Wei Liu, Ruochen Zhou, Yiyun Deng, Yuzhen Huang, Junteng Liu, Yuntian Deng, Yizhe Zhang, and Junxian He. Learn to reason efficiently with adaptive length-based reward shaping, 2025. URL https://arxiv.org/abs/2505.15612.

Mathematical Association of America. American invitational mathematics examination (aime). https://www.maa.org/math-competitions/aime, 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.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms with preference data, 2024. *URL https://arxiv. org/abs/2406.18665*, 4, 2025.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. *URL https://arxiv. org/abs/2408.03314*, 20, 2024.

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.

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

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, et al. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2526–2547, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Canhui Wu, Qiong Cao, Chang Li, Zhenfang Wang, Chao Xue, Yuwei Fan, Wei Xi, and Xiaodong He. Beyond token length: Step pruner for efficient and accurate reasoning in large language models. *arXiv preprint arXiv:2510.03805*, 2025a.

Menghua Wu, Cai Zhou, Stephen Bates, and Tommi Jaakkola. Thought calibration: Efficient and confident test-time scaling. *arXiv preprint arXiv:2505.18404*, 2025b.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025b.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weiping Wang. Dynamic early exit in reasoning models. *arXiv preprint arXiv:2504.15895*, 2025c.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. *URL https://arxiv. org/abs/2305.10601*, 3:1, 2023.

Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. Adaptthink: Reasoning models can learn when to think, 2025. *URL https://arxiv. org/abs/2505.13417*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

A Related Work

Efficient Reasoning for Large Reasoning Models. Large Reasoning Models (LRMs) have shifted the scaling paradigm from only scaling parameters to optimizing test-time compute (Snell et al., 2024). Current approaches to improving reasoning efficiency generally fall into three categories: training-time optimization, dynamic inference-time monitoring, and pre-generation routing.

Training for Concise Reasoning. Several approaches attempt to optimize the chain-of-thought generation process during training. They usually incentivize the model to generate more concise reasoning paths. For example, LASER (Liu et al., 2025) and Step Pruner (Wu et al., 2025a) utilize reward shaping to penalize excessive token usage.

Dynamic Inference and Early Exiting. These approaches focus on post-training methods that reduce compute during the generation process. These methods monitor the model's internal state to trigger early exits when the answer is sufficiently confident. DEER (Yang et al., 2025c) and Thought Calibration (Wu et al., 2025b) employ confidence or entropy-based metrics to halt overthinking on simpler queries. Similarly, LATTS (Uscidda et al., 2025) and ESC (Li et al., 2024) dynamically adjust the number of sampling steps or reasoning depth based on real-time feedback. While orthogonal to our work and applicable on top of our method, these approaches also require continuous monitoring of the generation stream.

Reasoning Strategy Selection and Routing. The closest line of research to our method is selecting a model prior to generation. General routing frameworks like RouteLLM (Ong et al., 2025) and FrugalGPT (Chen et al., 2023) switch between different models to balance cost and quality. More recently, specific methods for reasoning models have emerged. AdaptThink (Zhang et al.) and Thinkless (Fang et al., 2025) utilize reinforcement learning or control tokens to predict whether a thinking phase is necessary for a given input. ThinkSwitcher (Liang et al., 2025) uses hidden representations of the model to select an appropriate reasoning mode. However, a critical limitation shared by all existing routing methods is their reliance on standard classifiers or policies that optimize for average-case performance without providing any formal guarantees on the resulting performance.

B Additional Details on Mode Selection

LLM-as-Judge LLM-as-Judge is simple to use; it doesn't need extra data or training and it taps into the model's general sense of what makes a math problem complex. It provides interpretable difficulty scores that align with human intuition about problem categories. However, it requires a separate forward pass with a different prompt structure. Moreover, this approach inherits the known biases and limitations of LLM-based evaluation (Zheng et al., 2023). Finally, since the models are trained on human-generated data, the resulting difficulty estimates primarily capture human-perceived complexity, which may differ from each model's own internal assessment of problem difficulty.

System Prompt for Difficulty Rating

```
You are an expert judge that rates the difficulty of mathematical questions.
Your task is to evaluate the difficulty of the given question on a scale from 1 to 5.
Difficulty Scale:

1: Very easy (basic arithmetic, simple facts, single-step problems)
2: Easy (standard textbook problems, straightforward applications)
3: Medium (requires multiple steps, moderate reasoning, or algebra)
4: Hard (competition-level problems, requires insight or creative thinking)
5: Very hard (olympiad-level, requires deep mathematical reasoning or multiple advanced
   concepts)

IMPORTANT: Your response should start with "The difficulty level is: Number". You
must respond with ONLY a single integer from 1 to 5. No explanations, no other text.
```

Prompting Strategy We design a structured prompt that instructs the model to rate question difficulty on an ordinal scale from 1 to 5. The model generates a single-token response representing

the difficulty rating $d \in \{1, 2, 3, 4, 5\}$. We provide the prompt in Box B. The prompt provides clear definitions for each difficulty level to guide the model's assessment.

C Additional Experiments

Sequential Scaling: Extended Chain-of-Thought We additionally report results for Qwen-4B, Qwen-14B. The evaluation is conducted in a sequential setting, comparing thinking and non-thinking modes. All other parameters are kept identical to those described in the main paper.

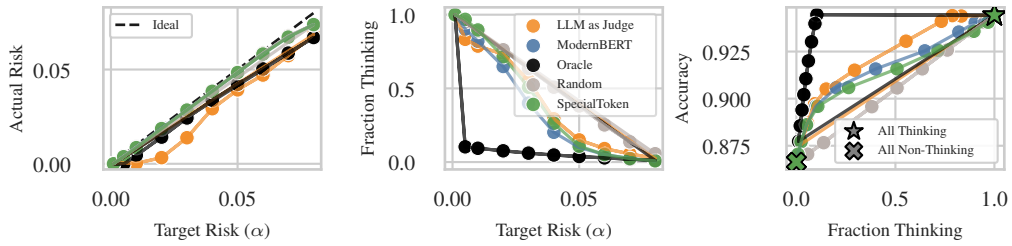

Figure 5: Risk-controlled thinking allocation (Sequential) for Qwen3-4B. [left] Calibration of CRC: target risk level (α) vs. actual risk. Points on the diagonal indicate perfect calibration. [middle] Fraction of samples routed to thinking mode as target risk increases. Lower values indicate more efficient compute. [right] Accuracy vs. fraction of samples using thinking, showing the accuracy-efficiency trade-off.

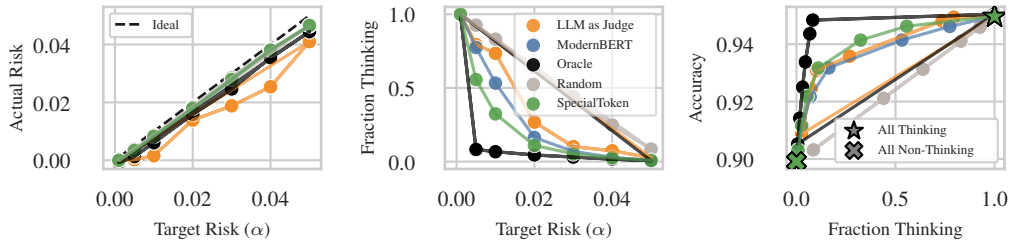

Figure 6: Risk-controlled thinking allocation (Sequential) for Qwen3-14B. [left] Calibration of CRC: target risk level (α) vs. actual risk. Points on the diagonal indicate perfect calibration. [middle] Fraction of samples routed to thinking mode as target risk increases. Lower values indicate more efficient compute. [right] Accuracy vs. fraction of samples using thinking, showing the accuracy-efficiency trade-off.