HRBench: Benchmarking and Understanding Thinking-Mode Switch Strategies in Hybrid-Reasoning LLMs

Anonymous ACL submission

Abstract

Hybrid-reasoning large language models (LLMs) expose explicit controls over reasoning effort, allowing users or systems to trade off answer quality against inference cost. However, existing methods for adaptive thinking-mode selection are typically evaluated under different models, datasets, and implementation assumptions, making it difficult to compare their practical behavior. We introduce HRBench, a unified evaluation framework for studying thinking-mode switching in hybrid-reasoning LLMs. HRBench organizes the design space along two axes: three switching strategy families, prompt-based selection, external routing, and speculative execution, and four training regimes, training-free, SFT, offline and online RL, yielding 12 controlled evaluation settings. We evaluate these settings across 6 LLMs, from Qwen3.5-2B to Kimi-K2.5-1.1T, and 5 reasoning benchmarks covering mathematics, science, and code, while reimplementing 12+ representative prior methods within the same pipeline. Our analysis characterizes how different switching strategies occupy distinct effectiveness-efficiency trade-off regions: prompt-based methods often provide favorable token-accuracy trade-offs, routing methods offer more stable cost reduction, and speculative methods tend to improve accuracy at higher token cost. We further find that training affects strategies differently, and that the preferred strategy varies with model scale and task domain. HRBench provides reference implementations and a unified evaluation platform to support more controlled research on efficient reasoning in hybrid-reasoning LLMs. [1]

1 Introduction

Recent reasoning-oriented LLMs, such as OpenAI o1 (OpenAI, 2024) and DeepSeek-R1 (Guo et al., 2025), are achieving remarkable success on complex tasks through extended chain-of-thought (CoT) reasoning (Wei et al., 2022), but at the cost of substantial token overhead. To address this, a new generation of *hybrid-reasoning LLMs* has emerged, including Qwen3.5 (Qwen Team, 2025), gpt-oss (Agarwal et al., 2025), and Seed-OSS (ByteDance, 2025), that expose explicit *thinking-mode switches*: users can select between deep reasoning (`think`) and direct answering (`no_think`), specify discrete reasoning effort levels (e.g., High/Medium/Low), or set numeric budgets (e.g., ≤ 4096 tokens). This raises a question: **when should the model think, and how much?**

A growing body of work tackles this efficiency–effectiveness trade-off by proposing *adaptive thinking-mode switch* methods. These can be categorized into three thinking-mode switch strategies:

- **Prompt-Tuning (PT)** guides the model to determine its thinking mode during a single inference pass through carefully designed prompts. For example, methods such as S1 (Muennighoff et al., 2025) and TALE (Han et al., 2025) inject token-budget or difficulty-aware instructions that let the model control reasoning length. Furthermore, RL-based approaches like ACPO (Cheng et al., 2026) directly optimize this internal decision.
- **RouTing (RT)** adopts a classify-then-generate strategy, where a router evaluates the query difficulty before dispatching it to the appropriate thinking mode. As representative examples, AdaptThink (Zhang et al., 2025a) trains such a router via GRPO, while HDFlow (Yao et al., 2024) uses rule-based difficulty classification.
- **Speculative (SPEC)** methods allow the model to begin in a fast mode and dynamically switch to deep reasoning upon detecting uncertainty signals. For instance, MixReasoning (Lu et al., 2025) uses entropy-based triggers for this escalation, while ADR (Zhang et al., 2025c) learns the switching policy through SFT and RL.

Despite active progress, these methods are evalu-

[1] Our data, code and repository are available at https://anonymous.4open.science/r/HRBench-1714.

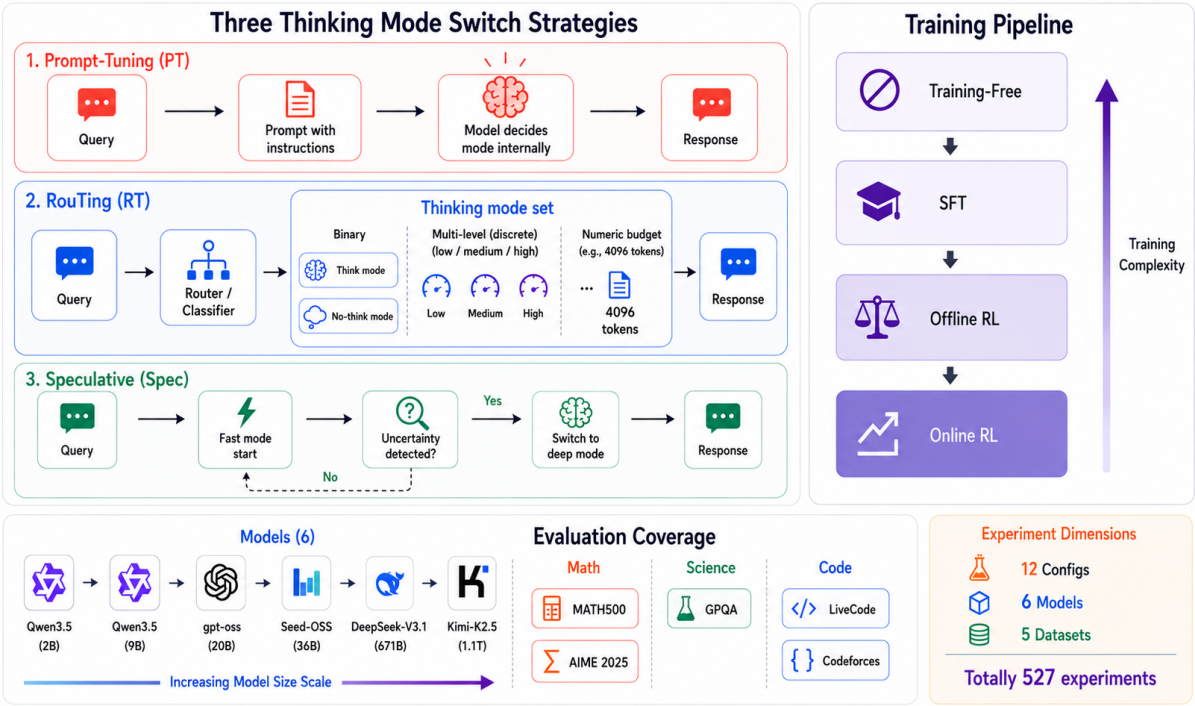

Figure 1: Overview of HRBENCH. **Left**: Three thinking-mode switch strategies—Prompt-Tuning, Routing, and Speculative. **Right**: Training pipeline spanning training-free to online RL. **Bottom**: Our evaluation coverage across 6 models spanning from 2B to 1.1T scale, 5 datasets, totaling 527 experiment runs.

ated under incomparable settings—different LLMs, datasets, metrics, and decoding configurations—making it impossible to answer: *which strategy truly works best?* and *how much does the training process help each strategy?*

To address this, we propose HRBENCH shown in Figure 1, a unified benchmark for understanding how different thinking-mode switch strategies behave across strategies, training regimes, model scales, and task domains. We orthogonally combine the three strategies with both training-free and training-based (SFT, offline RL, online RL) approaches, yielding 12 evaluation configurations that cover representative existing methods. Under a unified pipeline—6 LLMs spanning Qwen3.5-2B to Kimi-K2.5-1.1T, 5 benchmarks covering math, code, and science, and unified metrics—we systematically characterize how strategies navigate the efficiency–effectiveness trade-off, how training signals interact with strategy choice, and how the optimal strategy–training combination shifts with model scale and task domain. We further integrate 12+ existing methods into the pipeline, providing the unified comparison across the full taxonomy.

Our contributions are summarized as follows:

- **Unified evaluation framework.** We present the first benchmark that systematically covers 12 configurations for the thinking-mode switch, enabling controlled cross-strategy comparison under identical conditions.
- **Systematic empirical analysis of thinking switch mechanisms.** We reveal that:
  - The three strategy exhibit fundamentally different trade-off profiles: PT achieves a "win-win" (higher accuracy and fewer tokens), RT offers moderate token savings with preserved accuracy, while SPEC improves accuracy at additional token cost.
  - Training gains are strategy-dependent: GRPO achieves the highest token reduction for RT while all training methods maintain comparable accuracy across strategies.
  - Both effects shift with LLM size scale and task domain: SPEC surpasses PT at the 20B and 671B scales, while PT performs better on math and SPEC on code tasks.
- **Open-source baselines and platform.** Reference implementations for all 12 configurations and 12+ integrated prior methods, forming a plug-and-play platform for the community.

2 Related Work

2.1 Hybrid-Reasoning LLMs

Recent work has introduced LLMs with user-controllable thinking-mode switches that enable

flexible allocation of inference compute (Wang et al., 2025). These *hybrid-reasoning LLMs* offer three forms of control over reasoning depth:

- **Binary switch.** The most common way provides a `think`/`no_think` switch, where the former activates extended chain-of-thought and the latter generates direct answers. Current LLMs adopting this design include Qwen3.5 (Qwen Team, 2025), DeepSeek-V3.1, Kimi-K2.5, and so on.
- **Discrete reasoning effort.** Certain LLMs expose tiers of reasoning effort, e.g, gpt-oss-20B (Agarwal et al., 2025) introduces High/Medium/Low settings. This approach affords coarse-grained control of test-time compute.
- **Numeric budget.** LLM family like Seed-OSS-36B (ByteDance, 2025) also accept explicit token budgets $b \leq B_{\max}$, enabling fine-grained, continuous control over the token of reasoning.

2.2 Adaptive Thinking-Mode Switch

Existing adaptive thinking-mode switch methods can be categorized into three categories based on when and how the mode decision is made.

Prompt-Tuning. PT-based methods guide mode selection through prompt engineering within a single inference pass—the model itself decides whether and how deeply to reason. Both training-free and training-based approaches have been explored. Training-free approaches include S1 budget forcing (Muennighoff et al., 2025) and TALE token-budget-aware reasoning (Han et al., 2025). SFT-based methods include OThink-R1 (Zhang et al., 2025b) and HGPO (Jiang et al., 2025). RL-based methods include ACPO (Cheng et al., 2026), and Think-Only (HGPO) (Jiang et al., 2025). DPO-based methods include AdaR1 (Luo et al., 2025) and Think-in-Blocks (Zhu et al., 2025).

Routing. RT-based methods employ an explicit two-stage process: a router first assesses query difficulty and selects the appropriate mode, then the model generates under that mode. Training-free routers include HDFlow (Yao et al., 2024) and CP-Router (Su et al., 2026). SFT-trained routers include Self-Route (He et al., 2025), and ThinkSwitcher (Liang et al., 2025). For example, AdaptThink (Zhang et al., 2025a) trains a routing policy via GRPO that decides `think`/`no_think` per query, while Self-Route uses a lightweight SFT-trained linear classifier on hidden-state features.

Speculative. Spec-based methods dynamically switch modes during inference. The model begins in a fast mode (typically `no_think`) and triggers a switch to deep reasoning upon detecting uncertainty signals mid-stream. For example, training-free approaches include MixReasoning (Lu et al., 2025), which uses entropy-based triggers to detect when the fast-mode output is unreliable. In addition, ADR (Zhang et al., 2025c) combines SFT and GRPO stages for learned switching policies.

However, these approaches are evaluated in isolation. No prior work provides a unified framework that enables systematic cross-strategy comparison under controlled conditions, which is precisely the gap HRBENCH addresses.

3 Preliminary

Definition 1 (Thinking Mode)**.** *In a hybrid-reasoning LLM π_θ, a thinking mode m is defined as a control parameter that dictates the token budget allocated for intermediate chain-of-thought τ before generating a final answer a.*

The set of all available thinking modes, denoted as $\mathcal{M}$, typically takes one of the following forms depending on the model architecture:

- A binary state space: $\mathcal{M} = \{\text{think}, \text{no_think}\}$.
- A discrete effort space: $\mathcal{M} = \{\text{low}, \text{mid}, \text{high}\}$.
- A continuous token budget space: $\mathcal{M} = \{b \mid b \in [0, B_{\max}]\}$, where b specifies the maximum number of intermediate reasoning tokens.

Problem 1 (Thinking Mode Switch)**.** *Given a query q and the thinking modes set $\mathcal{M}$, the model π_θ adaptively selects approximate thinking modes to generate a response $r = (\tau, a)$, where τ denotes the chain-of-thought and a is the final answer.*

In this paper, the above problem can be solved by the following three strategies:

Strategy 1 (Prompt-Tuning based Switch)**.** *The model π_θ implicitly selects a thinking mode m and generates the response:*

$$r \sim \pi_\theta(\cdot \mid q, T_{\text{PT}}, m) \tag{1}$$

where T_{PT} is a prompt template that encodes mode-selection instructions, and m is implicitly determined by the model π_θ during inference.

Strategy 2 (Routing based Switch)**.** *A router first explicitly selects a thinking mode, then the model π_θ generates based on the routed thinking mode:*

$$r \sim \pi_\theta(\cdot \mid q, \hat{m}), \hat{m} = \pi_\psi(q) \tag{2}$$

	Training-Free	SFT	Online RL	Offline RL
Prompt-Tuning	S1, TALE	OThink-R1, HFT	AdaR1	ACPO, HGPO, SABER
Routing	HDFlow, CP-Router	Self-Route, ThinkSwitcher	*Ours*	AdaptThink
Speculative	MixReasoning	ADR_{SFT}	*Ours*	ADR_{GRPO}

Table 1: The evaluation taxonomy of the HRBENCH. Each cell represents a unique (strategy, training regime) configuration. Representative external methods are listed for each cell. *Ours* marks 2 configurations without prior baselines, first explored in this work.

where π_ψ is the router policy, mapping the query to a specific thinking mode $\hat{m} \in \mathcal{M}$.

Strategy 3 (Speculative based Switch)**.** *The model π_θ initiates decoding under an initial thinking mode m_0 and monitors the partial output. Upon a trigger signal, it will switch to an alternative thinking mode m_t:*

$$r \sim \begin{cases} \pi_\theta(\cdot \mid q, m_0) & \textit{if } f(\tau_{1:t}) \textit{ not triggered} \\ \pi_\theta(\cdot \mid q, m_t, \tau_{1:t^*}) & \textit{if } f(\tau_{1:t^*}) \textit{ triggered} \end{cases} \tag{3}$$

where $m_0, m_t \in \mathcal{M}$ are distinct thinking modes, $\tau_{1:t} \sim \pi_\theta(\cdot \mid q, m_0)$ is the partial chain-of-thought under the initial thinking mode m_0, f is a trigger function (Yang et al., 2025b), and t^* is the token position at which $f(\tau_{1:t^*})$ is satisfied.

4 HRBench Construction

4.1 Evaluation Taxonomy

We organize the evaluation into a systematic taxonomy (Table 1), crossing three strategies with four training regimes to yield 12 configurations.

4.2 Datasets

As shown in Table 2, we evaluate on five benchmarks spanning three task domains:

- **Mathematics**: AIME 2025 (competition-level math problems) and MATH500 (high school math problems) (Lightman et al., 2023).
- **Science**: GPQA-Diamond (graduate-level questions ranging from physics, chemistry, to biology) (Rein et al., 2024).
- **Code**: Live Code Bench (LCB) (live programming problems with execution-based evaluation) (Jain et al., 2024) and Codeforces (competition-level programming problems).

4.3 Models

We evaluate 6 hybrid-reasoning LLMs spanning 2B to 1.1T parameters, covering three thinking modes:

- **Qwen3.5-2B** and **Qwen3.5-9B** (Qwen Team, 2025): Binary switch (`think`/`no_think`).

Dataset	Domain	#Prob.	Difficulty
MATH500	Math	500	High School
AIME 2025	Math	30	Competition
GPQA-Dia.	Science	198	Graduate
LCB	Code	167	Graduate
Codeforces	Code	366	Competition

Table 2: Overall dataset statistics for the five benchmarks used in HRBENCH.

- **gpt-oss-20B**: Discrete thinking mode switch (e.g., High/Medium/Low reasoning effort).
- **Seed-OSS-36B-Instruct**: Thinking mode switch via numeric token budget ($b \leq B_{\max}$).
- **DeepSeek-V3.1-671B** (DeepSeek-AI, 2025): Binary switch (`think`/`no_think`).
- **Kimi-K2.5-1.1T** (Team et al., 2026): Binary switch (`think`/`no_think`).

4.4 Metrics

In this paper, we use accuracy and token cost to investigate the effectiveness-efficiency tradeoff:

- **Acc**: Pass@1 accuracy (%).
- **Tok**: Average output token cost (including CoT).

4.5 Baselines and Implementations

Fixed baselines. **Full-Think** (always `think`), **No-Think** (always `no_think`), and **Budget-Aware** (High/Medium/Low reasoning effort tiers).

Our implementations. For each of the 12 taxonomy cells, we provide a reference implementation using verl (Sheng et al., 2024) for training and vLLM (Kwon et al., 2023) for inference. All methods are evaluated under identical decoding parameters. Details are provided in Appendix D. We categorize implementations into two parts:

- **Training-Free (TF) Implementation**:
  - **Prompt-Tuning (PT-TF)**: We craft model-specific prompts mapping to reasoning effort levels (e.g., `think`/`no_think`, token budgets), enabling the LLM to auto-select its mode.

- **Routing (RT-TF)**: We employ the LLM itself as a router to assess query difficulty before dispatching to the appropriate mode.
- **Speculative (Spec-TF)**: We operate via two mechanisms. **Spec-TF (Trigger)** constructs a model-specific uncertainty keyword library (e.g., *wait*, *hmm*) that varies across models, triggering a re-think during inference. **Spec-TF (Entropy)** monitors token-level output probabilities and triggers mode escalation when entropy exceeds a calibrated threshold.

- **Training-Based Implementation**: Built on MathLightEval (Hendrycks et al., 2021), all training variants utilize a unified data construction pipeline based on Rejection Fine-Tuning (RFT) with multiple rollouts per problem:
  - **SFT**: We train on the sample that are both *correct* and *token-minimal* in multiple rollout results. For Prompt-Tuning (PT-SFT) and Speculative (Spec-SFT), the model is directly fine-tuned on these samples to autonomously select modes or trigger escalation. We choose the Spec-TF (Entropy) for Spec-SFT because it achieves a better performance. For Routing, the optimal mode serves as the ground-truth label to train either the LLM itself (RT-SFT).
  - **DPO**: The RFT process naturally yields preference pairs. The chosen sample is the correct, token-minimal response. Rejected samples are longer correct answers, incorrect answers, or sub-optimal routing modes. This optimizes both prompt-tuning (PT-DPO), router (RT-DPO), and speculative (Spec-DPO).
  - **GRPO**: In on-policy RL training, a unified reward structure is applied during rollouts to optimize autonomous mode selection (PT-GRPO), router policies (RT-GRPO), and speculative decoding triggers (Spec-GRPO).

External methods. We integrate 12 representative methods from the community into our unified pipeline, covering all three strategies:

- **Prompt-Tuning**: S1 (Muennighoff et al., 2025), TALE (Han et al., 2025), Budget-Guidance (Li et al., 2025), Sketch-of-Thought (SoT) (Aytes et al., 2025), Chain-of-Draft (CoD) (Xu et al., 2025), DynaThink (Pan et al., 2024), DEER (Yang et al., 2025a) and RASC (Wan et al., 2025).
- **Routing**: AdaptThink (Zhang et al., 2025a) (GRPO-trained router) and HDFlow (Yao et al., 2024) (rule-based difficulty routing).

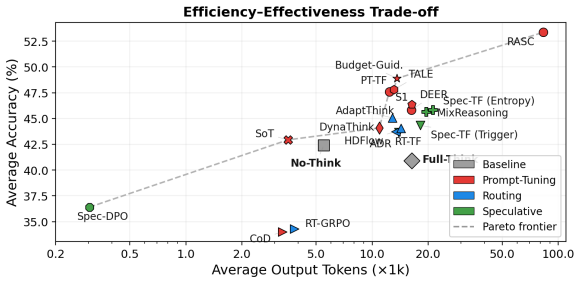

Figure 2: Efficiency–effectiveness trade-off on Qwen3.5-9B. Each point represents a method averaged over 5 datasets; dashed line shows the Pareto frontier.

- **Speculative**: MixReasoning (Lu et al., 2025) (entropy-based) and ADR (Zhang et al., 2025c) (SFT+GRPO trained switching policy).

All external methods are re-implemented within our unified pipeline and evaluated under identical conditions for fair comparison. Reproduction details and any deviations from original papers are documented in Appendix D.

5 Effectiveness–Efficiency Trade-off of Switching Strategies

RQ1: *How do different thinking-mode switch strategies (*PT/RT/SPEC*) trade off between effectiveness (accuracy) and efficiency (token cost)?*

To answer RQ1, we evaluate all three strategy implementations across all five benchmarks and six LLMs, examining how each strategy balances accuracy against token cost.

5.1 Overall Trade-off Patterns

Table 3 and Figure 2 reveal that the *three strategies exhibit fundamentally different trade-off patterns between effectiveness and efficiency*:

Finding 1: PT consistently achieves Pareto-optimal trade-offs. As shown in Figure 2, PT-TF simultaneously improves accuracy over Full-Think while substantially reducing token cost. This "win-win" pattern is unique to Prompt-Tuning: the prompt guides the model to allocate reasoning effort proportionally to difficulty, thereby avoiding unnecessary reasoning on simpler problems. Across all PT implementations in our benchmark, this Pareto-dominant behavior holds robustly.

Finding 2: RT preserves effectiveness with moderate efficiency gains. RT-TF maintains accuracy comparable to Full-Think while achieving moderate token savings through selective routing. The router correctly identifies easier problems (e.g.,

Method	MATH500		AIME 2025		GPQA		LiveCode		Codeforces		AVG	
	Acc	Tok	Acc	Tok	Acc	Tok	Acc	Tok	Acc	Tok	Acc	Tok
Baselines (Qwen3.5-9B)												
Full-Think	85.0	9.1k	70.0	23.1k	54.5	15.9k	21.6	16.1k	23.5	17.6k	40.9	16.4k
No-Think	**86.8**	1.7k	53.3	12.8k	53.0	3.1k	35.9	5.0k	32.8	4.8k	42.4	5.5k
Our Strategy Implementations (Training-Free)												
PT-TF	85.4	6.5k	**80.0**	20.2k	**63.1**	12.2k	29.9	11.7k	29.5	11.2k	**47.6**	12.4k
RT-TF	85.8	6.2k	73.3	22.4k	52.0	11.1k	29.9	15.1k	29.2	16.8k	44.1	14.3k
Spec-TF (Trigger)	86.2	7.8k	60.0	32.2k	52.5	15.0k	37.1	17.2k	35.8	18.6k	44.3	18.2k
Spec-TF (Entropy)	84.8	11.1k	63.3	32.7k	55.6	18.4k	43.1	20.8k	32.2	23.0k	45.8	21.2k

Table 3: Strategy-level trade-off on Qwen3.5-9B. PT achieves the best accuracy with 24% token reduction; RT preserves accuracy with 13% saving; SPEC boosts accuracy at extra token cost.

Model	Size	Full-Think		No-Think		PT-TF		RT-TF		Spec-TF	
		Acc	Tok	Acc	Tok	Acc	Tok	Acc	Tok	Acc	Tok
Qwen3.5-2B	2B	11.8	26.6k	18.7	14.0k	13.2	29.2k	14.0	24.7k	**14.1**	35.3k
Qwen3.5-9B	9B	40.9	16.4k	42.4	5.5k	**47.6**	12.4k	44.1	14.3k	45.8	21.2k
gpt-oss-20B	20B	36.4	19.7k	24.5	0.8k	32.9	4.0k	29.2	9.1k	**36.8**	20.4k
Seed-OSS-36B	36B	51.5	13.5k	27.0	915	**46.3**	8.2k	39.2	7.4k	43.9	10.0k
DeepSeek-V3.1	671B	67.4	3.2k	63.7	1.7k	74.7	3.3k	68.9	2.7k	**75.8**	4.2k
Kimi-K2.5	1.1T	73.9	8.9k	72.3	2.4k	**80.8**	7.4k	77.8	7.4k	74.5	10.3k

Table 4: Model scale modulation: effectiveness and efficiency of fixed baselines and three Training-Free strategies across 6 models (2B–1.1T), averaged over five benchmarks. Spec-TF reports the Entropy variant.

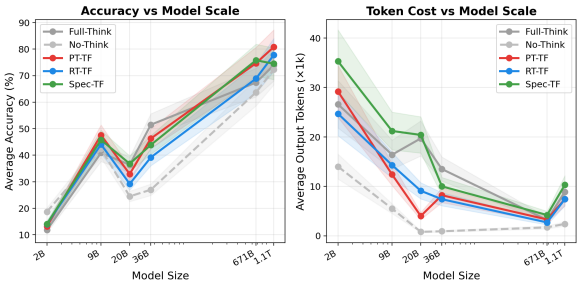

Figure 3: Strategy effectiveness (left) and efficiency (right) across model scales (2B–1.1T). All three strategies are evaluated across 6 models.

∼60% of MATH500) and routes them to `no_think` mode, while conservatively keeping harder benchmarks in full reasoning mode. This conservative strategy yields steady but limited improvements.

Finding 3: SPEC boosts accuracy at extra token cost. Unlike PT and RT, Spec-TF increases token usage relative to Full-Think, but in return yields notable accuracy improvements, particularly on code tasks where the "try-then-verify" mechanism excels. The no-think initial pass catches easy problems efficiently, but re-triggering deep reasoning when uncertainty is detected adds overhead. SPEC thus functions as an effectiveness-enhancing rather than efficiency strategy.

5.2 Model Scale Effect

To validate that trade-off patterns shift with model scale, we evaluate all six models (2B–1.1T) under Training-Free configurations. Table 4 reports averaged results, and Figure 3 visualizes the strategy ranking evolution across scales.

We observe that the *effectiveness–efficiency trade-off of each strategy shifts substantially with model scale*—neither strategy ranking nor efficiency advantage is consistent across scales:

Finding 4: Strategy effectiveness ranking varies across model scales. The best strategy choice differs depending on the size (Table 4): at 9B and 1.1T, PT leads (47.6% and 80.8% respectively); at 20B and 671B, SPEC overtakes (36.8% vs. 32.9% at 20B; 75.8% vs. 74.7% at 671B); while at 2B, all three strategies perform similarly (13.2–14.1%). RT generally ranks last but remains competitive at larger scales (77.8% at 1.1T). This scale-dependent ranking suggests that no single strategy universally dominates in effectiveness.

Finding 5: Strategy efficiency ranking is also scale-dependent. Token efficiency does not uniformly favor one strategy across scales. Notably, PT increases token usage at 2B (29.2k vs. 26.6k for Full-Think), while achieving strong savings at 36B (−39%) and 1.1T (−17%). In contrast, RT is the

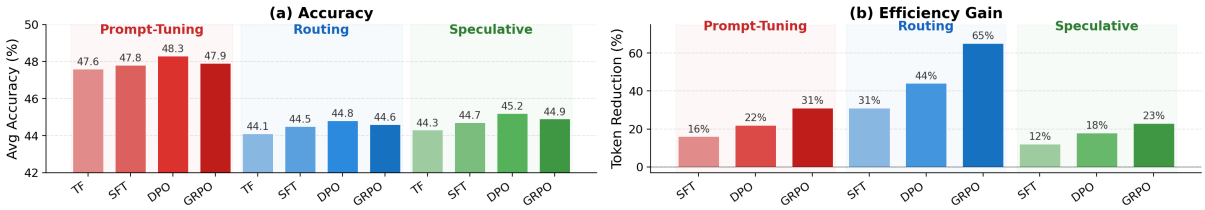

Figure 4: Training effect on switching capacity (Qwen3.5-9B). (a) Accuracy across training regimes, averaged over five benchmarks. (b) Token reduction relative to TF baseline.

Method	Math		Science		Code	
	ΔAcc	Red%	ΔAcc	Red%	ΔAcc	Red%
No-Think	−7.4	+62.8	−1.5	+80.6	**+11.8**	+70.6
PT-TF	+5.2	+20.5	**+8.6**	+23.2	+7.2	+31.8
RT-TF	+2.1	+17.4	−2.5	+30.4	+7.1	+5.4
Spec-TF	−4.4	−12.4	−2.0	+5.3	**+13.9**	−6.4

Table 5: Domain modulation on Qwen3.5-9B. ΔAcc: accuracy change (percentage points) vs. Full-Think; Red%: token reduction vs. Full-Think.

most consistent in reducing token cost: it achieves savings at every scale from 9B onward (e.g., −13% at 9B, −45% at 36B, −17% at 1.1T). SPEC consistently incurs extra tokens across all scales due to its re-think mechanism. These patterns indicate that efficiency-oriented deployment must carefully match the strategy to the target model scale.

5.3 Task Domain Effect

We further analyze how trade-off patterns vary across three task domains: math, science, and coding tasks. Table 5 reveals striking domain-dependent strategy preferences, demonstrating that the underlying nature of the task influences strategy selection:

Finding 6: The optimal strategy differs across task domains. No single strategy universally dominates: in Math and Science, PT is the clear winner, improving both accuracy and token efficiency; in Code, however, SPEC achieves the largest accuracy boost via its try-then-verify" mechanism, though PT and RT also yield efficient gains. This domain-dependent variation provides motivation for adaptive mode switching.

5.4 Summary

Overall, these domain-dependent patterns (§5.3), combined with model scale modulation (§5.2), confirm that no single strategy dominates universally. Consequently, an appropriate thinking mode switching strategy should carefully account for both the model scale and the expected task domain.

6 Effect of Training Pipeline on Switching Strategies

RQ2: *How do different training regimes (e.g., SFT/DPO/GRPO) affect the three thinking mode switch strategies?*

To answer RQ2, we train Qwen3.5-9B under three regimes (i.e., SFT, DPO, and GRPO) applied to each of the three strategies, and compare against the Training-Free (TF) baselines from §5. All training experiments use MathLightEval as the training data source. Figure 4 summarizes the accuracy and efficiency results across all 5 benchmarks.

Finding 7: Training universally improves switching capacity, with larger gains in efficiency than accuracy. Across all three strategies, training (SFT, DPO, and GRPO) maintains or slightly improves accuracy compared to TF, while achieving substantially larger gains in token reduction (Figure 4). This indicates that training primarily teaches the model when to skip unnecessary reasoning, rather than improving the reasoning itself. The accuracy improvements are modest (within 1-2 percentage points of TF), whereas efficiency gains range from 12% to 65% depending on the strategy and training method.

Finding 8: RT benefits the most from training in efficiency. The efficiency gains from training are strongly strategy-dependent: GRPO achieves 65% token reduction for RT, compared to 31% for PT and 23% for SPEC. This disparity arises because RT's binary routing decision is well-matched to the training signal—correctness feedback directly reflects routing quality, enabling the router to learn a sharper difficulty boundary. In contrast, PT and SPEC involve more fine-grained decisions (reasoning depth and trigger timing, respectively) that are harder to optimize with coarse reward signals.

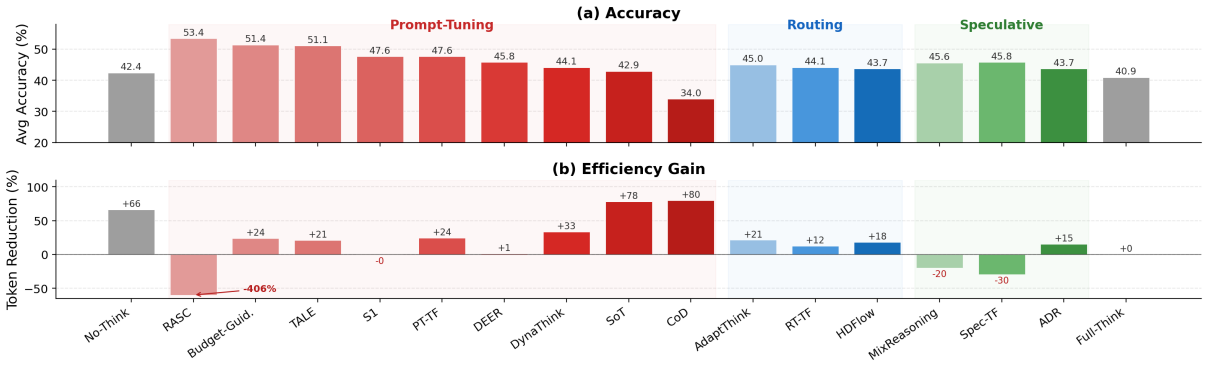

Figure 5: Fair comparison of 12 methods under unified evaluation on Qwen3.5-9B. (a) Average accuracy across five benchmarks. (b) Token reduction relative to Full-Think (positive = saving). Methods grouped by strategy.

Finding 9: Different training methods offer distinct trade-off profiles. The three training regimes serve a complementary role. DPO yields the largest accuracy improvements and is optimal for improving effectiveness. Conversely, GRPO achieves the greatest efficiency gains and token savings, making it ideal when efficiency is the priority. Meanwhile, SFT provides a balanced middle ground in both dimensions, serving as a safe default when specific optimization targets are unclear.

7 Fair Comparison of Existing Methods

RQ3: *Under a unified pipeline (e.g., same model and data), do the advantages claimed by existing methods hold, and are the conclusions consistent with our strategy-level findings?*

The RQ3 also aligns with the core motivation of HRBENCH: existing methods are developed and evaluated in isolation, making cross-method comparison unreliable. We re-implement 12 representative external methods within our unified pipeline and compare them against our TF baselines (PT-TF, RT-TF, Spec-TF) under identical conditions. Figure 5 visualizes the results, and full per-dataset breakdowns are in Table 6 (Appendix).

Finding 10: External methods exhibit similar strategy-level trade-off patterns. The unified evaluation validates our findings from §5:

- **Prompt-Tuning** methods span the widest Pareto frontier, ranging from RASC (53.4% accuracy, 5× more tokens) to CoD (34.0%, +80% token reduction), confirming PT's unique ability to deliver diverse operating points along the efficiency–effectiveness spectrum. Our PT-TF baseline (47.6%, +24% token reduction) sits at a competitive position within this range.
- **Routing** based methods consistently show moderate token savings (18–21%) with preserved accuracy (43–45%), consistent with RT's pattern of maintaining effectiveness while gaining modest efficiency. Our RT-TF (44.1%, +12.5%) also aligns with this trend.
- **Speculative** based methods mostly show negative token reduction (−11% to −30%), confirming their role as accuracy-boosting rather than efficiency-oriented approaches, which aligns with our analysis in §5.1. ADR is the only external Spec method achieving token reduction.

Finding 11: No single external method dominates across all task domains. As shown in Table 6, top methods vary by domain: RASC excels on AIME (83.3%), Budget-Guidance leads on MATH500 (87.8%), and SoT achieves the best GPQA accuracy. Overall, this also reinforces that practitioners should select suitable methods based on their target task domain rather than relying on the reported aggregate score.

8 Conclusion

In this paper, we proposed HRBENCH, a unified benchmark for systematically understanding thinking-mode switch strategies in hybrid-reasoning LLMs. By evaluating 12 configurations (3 strategies × 4 training regimes) across 6 models and 5 datasets, and integrating 12+ external methods into the same pipeline, we reveal that: (1) the three strategies exhibit fundamentally different trade-off profiles; (2) training gains are strongly strategy-dependent; and (3) the optimal strategy–training combination shifts with model scale and task domain. We release all implementations and the unified evaluation framework to facilitate future research on efficient hybrid reasoning.

Limitations

Our work has several limitations. First, training-based evaluations are limited to Qwen3.5-9B due to computational constraints, scaling to 20B+ models would strengthen conclusions about training-scale interactions. Second, our evaluation focuses on single-turn reasoning; multi-turn and agentic scenarios where mode-switching decisions compound across steps remain unexplored. Third, while we cover mathematics, science, and code, other domains (e.g., creative writing, multilingual tasks) may exhibit different trade-off patterns.

Ethical Statement

We use LLMs for paper polishing and figure plotting, and have carefully verified all outputs for correctness. This work evaluates existing LLMs on publicly available benchmarks and does not involve human subjects, private data, or the generation of harmful content. All datasets used (MATH500, AIME 2025, GPQA-Diamond, LiveCodeBench, Codeforces) are publicly released for research purposes. Our benchmark focuses on improving inference efficiency of reasoning models, which may reduce computational costs and associated carbon emissions. We do not foresee direct negative societal impacts from this work. All model outputs are used solely for automated evaluation and are not deployed in user-facing applications.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Simon A Aytes, Jinheon Baek, and Sung Ju Hwang. 2025. Sketch-of-thought: Efficient llm reasoning with adaptive cognitive-inspired sketching. *arXiv preprint arXiv:2503.05179*.

ByteDance. 2025. Seed-oss open-source models. https://github.com/ByteDance-Seed/seed-oss.

Xiaoxue Cheng, Junyi Li, Zhenduo Zhang, Xinyu Tang, Xin Zhao, Xinyu Kong, and Zhiqiang Zhang. 2026. Incentivizing dual process thinking for efficient large language model reasoning. *Advances in Neural Information Processing Systems*, 38:152743–152764.

DeepSeek-AI. 2025. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2025. Token-budget-aware llm reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24842–24855.

Yang He, Xiao Ding, Bibo Cai, Yufei Zhang, Kai Xiong, Zhouhao Sun, Bing Qin, and Ting Liu. 2025. Self-route: Automatic mode switching via capability estimation for efficient reasoning. *arXiv preprint arXiv:2505.20664*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, and 1 others. 2024. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Zhiyu Jiang and 1 others. 2025. Think only when you need with large hybrid-reasoning models. *arXiv preprint arXiv:2505.14631*. NeurIPS 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with PagedAttention. In *SOSP*.

Junyan Li, Wenshuo Zhao, Yang Zhang, and Chuang Gan. 2025. Steering llm thinking with budget guidance. *arXiv preprint arXiv:2506.13752*.

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

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Haiquan Lu, Gongfan Fang, Xinyin Ma, Qi Li, and Xinchao Wang. 2025. Mixreasoning: Switching modes to think. *arXiv preprint arXiv:2510.06052*.

Haotian Luo, Haiying He, Yibo Wang, Jinluan Yang, Rui Liu, Naiqiang Tan, Xiaochun Cao, Dacheng Tao, and Li Shen. 2025. Adar1: From long-cot to hybrid-cot via bi-level adaptive reasoning optimization. *arXiv preprint arXiv:2504.21659*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. 2025. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20286–20332.

OpenAI. 2024. Learning to reason with LLMs.

Jiabao Pan, Yan Zhang, Chen Zhang, Zuozhu Liu, Hongwei Wang, and Haizhou Li. 2024. Dynathink: Fast or slow? a dynamic decision-making framework for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14686–14695.

Qwen Team. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

David Rein, Betty Li Hou, Asa Cooper Stickland, and 1 others. 2024. GPQA: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*.

Guangming Sheng and 1 others. 2024. verl: An open-source unified framework for post-training of large language models. *arXiv preprint*.

Jiayuan Su, Fulin Lin, Zhaopeng Feng, Han Zheng, Teng Wang, Zhenyu Xiao, Xinlong Zhao, Zuozhu Liu, Lu Cheng, and Hongwei Wang. 2026. Cp-router: An uncertainty-aware router between llm and lrm. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 33065–33073.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, SH Cai, Yuan Cao, Y Charles, HS Che, Cheng Chen, Guanduo Chen, and 1 others. 2026. Kimi k2. 5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*.

Guangya Wan, Yuqi Wu, Jie Chen, and Sheng Li. 2025. Reasoning aware self-consistency: Leveraging reasoning paths for efficient llm sampling. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3613–3635.

Yifan Wang and 1 others. 2025. Demystifying hybrid thinking in large language models. *arXiv preprint arXiv:2510.12680*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS*.

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. 2025. Chain of draft: Thinking faster by writing less. *arXiv preprint arXiv:2502.18600*.

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

Ziyi Yang and 1 others. 2025b. Speculative thinking: Enhancing small-model reasoning with large model guidance at inference time. *arXiv preprint arXiv:2504.12329*.

Wenlin Yao, Haitao Mi, and Dong Yu. 2024. Hdflow: Enhancing llm complex problem-solving with hybrid thinking and dynamic workflows. *arXiv preprint arXiv:2409.17433*.

Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. 2025a. Adaptthink: Reasoning models can learn when to think. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 3716–3730.

Shengjia Zhang, Junjie Wu, Jiawei Chen, Changwang Zhang, Zhe Li, Xingyu Lou, Wangchunshu Zhou, Sheng Zhou, Can Wang, and Jun Wang. 2025b. Othink-r1: Intrinsic fast/slow thinking mode switching for over-reasoning mitigation. *arXiv preprint arXiv:2506.02397*.

Yujian Zhang, Keyu Chen, Zhifeng Shen, Ruizhi Qiao, and Xing Sun. 2025c. Adaptive dual reasoner: Large reasoning models can think efficiently by hybrid reasoning. *arXiv preprint arXiv:2510.10207*.

Yekun Zhu, Guang Chen, and Chengjun Mao. 2025. Think in blocks: Adaptive reasoning from direct response to deep reasoning. *arXiv preprint arXiv:2508.15507*.

A Prompt Templates

This appendix provides the system prompts and user templates used by each strategy and model family. All prompts are centralized to ensure consistency between training data construction and inference. We reproduce them in full below.

A.1 Answer Format Instructions

Every user message in our benchmark is appended with a domain-specific answer format instruction:

Mathematics (MATH500, AIME).

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

Code (LiveCodeBench, Codeforces).

```
Write a Python solution. Read input from
stdin and print output to stdout. Do not
include any test code or examples. Only
provide the solution code.
```

Science (GPQA multiple-choice).

```
Provide your answer. If the problem
is multiple-choice, state the correct
option letter. Otherwise, provide a
clear, concise answer.
```

A.2 Problem Templates

The user-facing message is constructed by formatting the problem text into domain-specific templates:

Math user message.

```
{problem}
Put your final answer within \boxed{}.
```

Code user message.

```
{problem}
Write a Python solution. Read input from
stdin and print output to stdout. Do not
include any test code or examples. Only
provide the solution code.
```

Science user message.

```
{problem}
Provide your answer. If the problem
is multiple-choice, state the correct
option letter. Otherwise, provide a
clear, concise answer.
```

These templates are shared by all baselines (Full-Think, No-Think) and strategies. Strategies add their own system prompts (below) while keeping the user message format unchanged.

A.3 Prompt-Tuning System Prompts

PT-TF uses model-specific system prompts that teach the model to self-select reasoning depth using its native thinking-mode interface. The complete prompts are reproduced verbatim below.

Qwen3.5 series (`PROMPT_TUNING_SYSTEM_QWEN`).

```
You are an expert problem solver with
adaptive reasoning.

Before solving, assess the problem's
difficulty and choose your reasoning
depth:
- For simple problems: Keep your <think>
block empty or very brief. Answer
directly.
- For complex problems: Use your
<think> block for thorough step-by-step
reasoning.
- For medium problems: Use your <think>
block briefly for key observations only.

You decide the appropriate depth based
on the problem.
```

gpt-oss (`PROMPT_TUNING_SYSTEM_GPT_OSS`).

```
You are an expert problem solver. You
may adjust your reasoning level between
high, medium, and low based on problem
complexity.

- For simple problems: Use low reasoning
effort. Minimal analysis, direct
answer.
- For complex problems: Use high
reasoning effort. Thorough step-by-step
analysis.
- For medium problems: Use medium
reasoning effort. Brief analysis on key
steps.

Assess each problem and choose the
appropriate reasoning level yourself.
```

Seed-OSS (`PROMPT_TUNING_SYSTEM_SEED_OSS`).

```
You are an intelligent assistant with
reflective reasoning ability. You may
adjust your thinking budget based on
problem complexity.

- For simple problems: Set your thinking
budget to 0 or minimal. Skip the
thinking process and answer directly.
- For complex problems: Allow a
generous thinking budget (4096+
tokens). Think thoroughly and use
<seed:cot_budget_reflect> to track your
token usage.
- For medium problems: Set a moderate
thinking budget (512-1024 tokens).
Think on key steps, reflect on progress,
then answer.

Example reflection during thinking:
<seed:cot_budget_reflect>I have
used 200 tokens, and there
are 800 tokens remaining for
use.</seed:cot_budget_reflect>

Assess each problem and manage your
thinking budget accordingly.
```

PT-TF user template (PROMPT_TUNING_USER_TEMPLATE). This wraps the problem before appending the answer format instruction:

```
Solve the following problem. Decide whether it requires deep thinking or a direct answer.
Problem: {problem}
Provide your final answer.
```

The complete user message sent to the model is the combination of user messages and tailored system prompt.

A.4 Routing Prompts

RT-TF uses a two-stage process. Each stage uses different prompts.

A.4.1 Stage 1: Judge Prompts

The judge stage uses model-specific prompts to classify problem difficulty. It runs in no-think/low-effort mode with `max_tokens=256`.

Qwen3.5 judge (ROUTING_JUDGE_SYSTEM_QWEN + ROUTING_JUDGE_USER_QWEN).

```
[System] You are a problem difficulty classifier.

[User] Assess the difficulty of the following problem and choose a reasoning mode.
Problem: {problem}
Available modes:
1 - Think: Complex problem, enable full thinking with <think> block
2 - NoThink: Simple problem, skip thinking, answer directly
3 - Budget Think: Medium problem, think within a limited token budget
Respond with ONLY a JSON object:
{"mode": "1" or "2" or "3", "budget": null or 1024 or 2048 or 4096}
Rules:
- Mode 1: budget = null (unlimited thinking)
- Mode 2: budget = null (no thinking)
- Mode 3: budget = 1024 / 2048 / 4096
```

gpt-oss judge (ROUTING_JUDGE_SYSTEM_GPT_OSS + ROUTING_JUDGE_USER_GPT_OSS).

```
[System] You are a problem difficulty classifier.

[User] Assess the difficulty of the following problem and choose a reasoning effort level.
Problem: {problem}
Available reasoning levels:
1 - High: Complex problem, needs thorough step-by-step analysis
2 - Low: Simple problem, minimal analysis, direct answer
3 - Medium: Moderate problem, brief analysis on key steps
Respond with ONLY a JSON object:
{"level": "high" or "medium" or "low"}
```

Seed-OSS judge (ROUTING_JUDGE_SYSTEM_SEED_OSS + ROUTING_JUDGE_USER_SEED_OSS).

```
[System] You are a problem difficulty classifier.

[User] Assess the difficulty of the following problem and choose a thinking strategy.
Problem: {problem}
Available thinking modes:
1 - Full Think: Complex problem, unlimited thinking budget
2 - No Think: Simple problem, skip thinking (budget = 0)
3 - Budget Think: Medium problem, think within a fixed token budget
Respond with ONLY a JSON object:
{"mode": "1" or "2" or "3", "budget": null or 512 or 1024 or 2048 or 4096}
Rules:
- Mode 1: budget = null (unlimited)
- Mode 2: budget = null (no thinking)
- Mode 3: budget = 512 / 1024 / 2048 / 4096
```

A.4.2 Stage 2: Solve Prompt

After routing, the problem is solved using a shared system prompt (`ROUTING_SOLVE_SYSTEM`):

```
You are a helpful assistant. Solve the given problem carefully.
```

The user message uses the standard problem template (§A.2).

A.5 Speculative Thinking Configuration

Speculative strategies do not use custom system prompts; they reuse the baseline message format (§A.2). Mode switching is controlled at the *token level* via two-pass generation.

Spec-Trigger: Complete keyword library (SPECULATIVE_TRIGGER_WORDS). The full 55-keyword library:

- **Hesitation/Uncertainty (9):** *wait, hmm, hm,, hold on, i'm not sure, i am not sure, not certain, unclear, confusing*
- **Self-correction/Backtracking (14):** *actually, on second thought, let me reconsider, i made a mistake, i made an error, that's wrong, that's incorrect, that doesn't seem right, this is wrong, correction:, i need to correct, scratch that, let me redo, start over, going back*

- **Re-examination/Verification (11):** *let me verify, let me check, let me re-examine, let me recalculate, double-check, double check, verify this, verify that, reconsider, re-examine, revisit*
- **Alternative approach (10):** *alternatively, another approach, another way, different approach, different method, try a different, let me try, instead,, perhaps, maybe i should*
- **Deeper reasoning (11):** *think again, think more carefully, think step by step, let me think, need to think, this requires, this is tricky, this is complex, this is harder, more carefully, closer look*
- **Contradiction/Confusion (9):** *but that contradicts, that contradicts, this contradicts, doesn't make sense, does not make sense, something is off, something is wrong, paradox, inconsistent*
- **Explicit re-reasoning (7):** *recap, summarize what we know, let me summarize, to be more precise, more precisely, to clarify, in other words*

All matching is case-insensitive. If *any* keyword appears in the no-think output, the response is discarded and regenerated in full think mode.

Spec-Entropy: Threshold configuration. Model-specific entropy thresholds (calibrated on held-out validation set): Qwen3.5 τ=0.10, gpt-oss τ=0.08, Seed-OSS τ=0.06. All use top-k=20 logprobs. Escalation is triggered when ≥ 3 tokens or $> 5\%$ of output tokens exceed the threshold.

Entropy is computed as normalized Shannon entropy from the top-k logprobs:

$$H_t = \frac{-\sum_{v \in \text{top-}k} \hat{p}_t(v) \log \hat{p}_t(v)}{\log k} \tag{4}$$

where $\hat{p}_t(v) = \exp(\ell_v) / \sum_{v'} \exp(\ell_{v'})$ is the renormalized probability over the top-k returned logprobs. Lower thresholds for gpt-oss/Seed-OSS reflect their generally tighter output distributions in no-think mode.

A.6 Training Prompts

Baseline SFT mode-selection (`SFT_MODE_SELECTION_SYSTEM`).

```
You are an adaptive reasoning assistant.
For each problem, you must first decide
your reasoning strategy, then solve the
problem accordingly.

Output format:
[MODE: think] or [MODE: nothink]
Then solve the problem.
```

Strategy-specific training prompts. The function `get_strategy_system_prompt(strategy, model_family)` returns the *same* prompt used at inference time:

- `strategy="pt"`: Returns the PT system prompt for the given model family (§A.3)
- `strategy="rt"`: Returns `ROUTING_SOLVE_SYSTEM` = "You are a helpful assistant. Solve the given problem carefully."
- `strategy="baseline"`: Returns "You are a helpful math assistant. Solve the problem step by step and provide your final answer."

This design guarantees prompt consistency between training data construction and evaluation.

A.7 Evaluation: LLM-as-Judge Prompt

For problems where rule-based answer extraction is insufficient, we use an LLM-as-judge for correctness evaluation:

```
[System] You are an expert evaluator.
Given a question, a reference answer,
and a student's answer, determine if the
student's answer is correct.

[User] Question:
{problem}

Reference Answer:
{reference}

Student's Answer:
{response}

Is the student's answer correct?
Consider mathematical equivalence (e.g.,
1/2 and 0.5 are equivalent, different
forms of the same expression are
equivalent).

Respond with ONLY a JSON object:
{"correct": true} or {"correct": false}
```

B Full Per-Dataset Results

B.1 Fair Comparison Per-Dataset Results

Table 6 provides the complete per-dataset breakdown of Figure 5.

B.2 Training Effect Per-Dataset Results

Table 7 reports the complete per-dataset training results visualized in Figure 4.

C Failure Case Analysis

We identify representative failure modes for each strategy:

PT failure: Over-compression on hard problems. On AIME problems, PT-GRPO occasionally produces overly abbreviated reasoning (avg 4.1k tokens on MATH500 vs. 6.5k for PT-TF) that skips critical intermediate steps. While accuracy is maintained on average, per-problem inspection

Strategy	Method	MATH500 Acc	MATH500 Tok	AIME 2025 Acc	AIME 2025 Tok	GPQA Acc	GPQA Tok	LiveCode Acc	LiveCode Tok	Codeforces Acc	Codeforces Tok	AVG Acc	AVG Red%
Fixed Baselines													
	Full-Think	85.0	9.1k	70.0	23.1k	54.5	15.9k	21.6	16.1k	23.5	17.6k	40.9	—
	No-Think	86.8	1.7k	53.3	12.8k	53.0	3.1k	35.9	5.0k	32.8	4.8k	42.4	+66.5
Prompt-Tuning													
PT	RASC	**89.2**	42.5k	**83.3**	131.7k	56.6	89.1k	**49.7**	79.0k	**38.0**	72.5k	**53.4**	−406
PT	Budget-Guidance (M)	87.8	6.5k	73.3	19.3k	65.2	10.0k	45.5	13.5k	35.0	12.9k	51.4	+23.9
PT	TALE (EP)	87.2	6.8k	76.7	21.6k	67.2	10.3k	43.7	14.3k	30.6	12.0k	51.1	+21.2
PT	S1 (High)	84.4	16.3k	70.0	26.2k	54.5	16.0k	43.7	17.0k	35.2	18.0k	47.6	−0.4
PT	PT-TF	85.4	6.5k	80.0	20.2k	63.1	12.2k	29.9	11.7k	29.5	11.2k	47.6	+24.4
PT	DEER	85.6	8.3k	66.7	26.2k	52.5	16.8k	37.1	15.9k	37.2	13.9k	45.8	+1.1
PT	DynaThink	86.8	2.4k	56.7	15.5k	50.5	7.6k	40.1	14.6k	36.3	14.5k	44.1	+33.4
PT	Sketch-of-Thought	82.0	895	43.3	3.8k	**69.7**	3.2k	37.1	5.4k	32.5	4.4k	42.9	+78.3
PT	Chain-of-Draft	73.8	854	26.7	4.5k	65.7	3.6k	29.9	2.9k	24.0	3.7k	34.0	+79.9
Routing													
RT	AdaptThink	86.4	5.8k	73.3	20.8k	54.0	10.5k	31.1	13.2k	30.3	14.0k	45.0	+21.3
RT	RT-TF	85.8	6.2k	73.3	22.4k	52.0	11.1k	29.9	15.1k	29.2	16.8k	44.1	+12.5
RT	HDFlow	85.6	5.9k	70.0	21.5k	53.0	11.5k	30.5	13.8k	29.5	14.2k	43.7	+18.2
RT	RT-GRPO	85.8	2.1k	73.3	7.1k	54.0	3.9k	30.5	5.8k	29.4	6.2k	44.6	+69.5
Speculative													
Spec	Spec-TF (Entropy)	84.8	11.1k	63.3	32.7k	55.6	18.4k	43.1	20.8k	32.2	23.0k	45.8	−29.6
Spec	MixReasoning	85.4	9.8k	63.3	30.5k	54.5	17.2k	41.3	19.5k	33.6	20.8k	45.6	−19.6
Spec	Spec-TF (Trigger)	86.2	7.8k	60.0	32.2k	52.5	15.0k	37.1	17.2k	35.8	18.6k	44.3	−11.1
Spec	ADR	84.8	7.2k	63.3	22.5k	54.0	12.2k	34.7	13.5k	31.4	14.0k	43.7	+15.2
Spec	Spec-DPO	86.4	6.2k	63.3	26.5k	53.5	12.8k	37.7	14.1k	35.2	14.8k	45.2	+9.1

Table 6: Fair comparison of methods under unified evaluation on Qwen3.5-9B. All methods use an identical model, datasets, and evaluation pipeline. Red%: token reduction relative to Full-Think (positive = saving).

reveals that the model sometimes "summarizes" rather than fully reasons through multi-step proofs, occasionally leading to errors on problems requiring 4+ reasoning steps.

RT failure: Difficulty miscalibration on GPQA. Our router achieves only 52.0% accuracy on GPQA-Diamond, compared to 54.5% for Full-Think. Inspection reveals that the router systematically *overestimates* difficulty of science questions (routing 93% to think-mode), yet the think-mode response still fails. The issue is not routing quality but the model's intrinsic inability on this task—routing cannot improve performance when neither mode produces correct answers.

Spec failure: Confident wrong answers. On AIME, the speculative trigger's entropy-based mechanism fails when the model produces a *confident but incorrect* initial answer. With Spec-Entropy, all 30 problems enter "mixed" mode (attempting retrigger), but the retrigger adds tokens without correcting the fundamental reasoning error. The trigger fires too late—after the model has already committed to a wrong approach.

D Implementation Details

This appendix provides comprehensive implementation details for all methods in HRBench, complementing the overview in §4.5.

D.1 Training Hyperparameters

D.2 Model-Specific Parameter Mapping

A key design principle of HRBench is that each strategy maps to model-native parameters rather than using a one-size-fits-all approach.

Inference configuration. All experiments use greedy decoding (temperature=0) except RASC (temperature=0.7 for diversity sampling). Maximum output tokens are set to 32,768 for all strategies. The judge stage in RT-TF uses max_tokens=256 with no-think/low-effort mode to minimize overhead. The mode-decision heuristic post-generation classifies outputs as "think" if thinking_tokens > 100, "nothink" if thinking_tokens < 10, and "brief_think" otherwise.

D.3 Unified Training Data Construction

All training-based methods (SFT, DPO, GRPO) across all three strategies follow a unified rejection

Para.	Train	MATH500 Acc	MATH500 Tok	AIME 2025 Acc	AIME 2025 Tok	GPQA Acc	GPQA Tok	LiveCode Acc	LiveCode Tok	Codeforces Acc	Codeforces Tok	AVG Acc	AVG Tok
Baselines													
–	Full-Think	85.0	9.1k	70.0	23.1k	54.5	15.9k	21.6	16.1k	23.5	17.6k	40.9	16.4k
–	No-Think	86.8	1.7k	53.3	12.8k	53.0	3.1k	35.9	5.0k	32.8	4.8k	42.4	5.5k
Prompt-Tuning													
PT	TF	85.4	6.5k	80.0	20.2k	63.1	12.2k	29.9	11.7k	29.5	11.2k	47.6	12.4k
PT	SFT	85.6	5.2k	80.0	17.5k	63.4	10.1k	29.5	9.8k	30.5	9.5k	47.8	10.4k
PT	DPO	85.8	4.8k	80.0	16.3k	64.1	9.4k	31.1	9.2k	30.5	8.7k	48.3	9.7k
PT	GRPO	86.2	4.1k	80.0	14.8k	63.0	8.2k	30.1	7.8k	30.2	7.5k	47.9	8.5k
Routing													
RT	TF	85.8	6.2k	73.3	22.4k	52.0	11.1k	29.9	15.1k	29.2	16.8k	44.1	14.3k
RT	SFT	85.4	4.5k	73.3	15.8k	53.5	7.6k	29.9	10.2k	30.4	11.3k	44.5	9.9k
RT	DPO	85.6	3.8k	73.3	12.4k	54.0	6.2k	30.5	8.5k	30.6	9.1k	44.8	8.0k
RT	GRPO	85.8	2.1k	73.3	7.1k	54.0	3.9k	30.5	5.8k	29.4	6.2k	44.6	5.0k
Speculative													
SPEC	TF	84.8	11.1k	63.3	32.7k	55.6	18.4k	43.1	20.8k	32.2	23.0k	45.8	21.2k
SPEC	SFT	85.8	6.9k	63.3	28.1k	53.0	13.5k	36.5	15.3k	34.8	16.1k	44.7	16.0k
SPEC	DPO	86.4	6.2k	63.3	26.5k	53.5	12.8k	37.7	14.1k	35.2	14.8k	45.2	14.9k
SPEC	GRPO	86.0	5.8k	63.3	25.2k	52.5	12.2k	37.1	13.5k	35.5	14.0k	44.9	14.1k

Table 7: Full per-dataset training results (Qwen3.5-9B). Acc: Pass@1 (%); Tok: average output tokens.

Hyperparameter	SFT	DPO	GRPO
Learning rate	2e-5	5e-7	5e-7
Batch size	4	4	4
Epochs	2	1	1
Max length	122880	122880	122880
KL coefficient	–	0.1	0.01
GRPO n	–	–	8
Temperature	–	–	1.0
Framework	verl	verl	verl

Table 8: Training hyperparameters.

fine-tuning (RFT) pipeline. Given a training set from MathLightEval (Hendrycks et al., 2021), our data construction proceeds as follows:

Step 1: Multi-mode rollout. For each problem q, we generate K responses under each available thinking mode $m \in \mathcal{M}$ (e.g., `think`, `no_think`, or intermediate effort levels). Each rollout produces a response $r_{k,m}$ with associated token count $t_{k,m}$ and correctness label $c_{k,m} \in \{0, 1\}$.

Step 2: SFT sample selection. We select the response that is both correct and token-minimal:

$$r^* = \arg\min_{r_{k,m}} t_{k,m} \quad \text{s.t.} \quad c_{k,m} = 1 \tag{5}$$

This yields training pairs (q, r^*) that teach the model to produce efficient yet accurate responses.

Step 3: DPO pair construction. The RFT process naturally produces preference pairs. The chosen response r^+ is the correct response with minimum token cost; rejected responses r^- are drawn from (a) correct but longer responses, or (b) incorrect responses. Formally:

$$r^+ = r^*, \quad r^- \in \{r_{k,m} \mid t_{k,m} > t^* \text{ or } c_{k,m} = 0\} \tag{6}$$

Step 4: GRPO reward design. For GRPO, we perform $n = 8$ rollouts per problem and compute a composite reward:

$$R(r) = \alpha \cdot \mathbb{1}[\text{correct}(r)] + \beta \cdot \mathbb{1}[\text{correct}(r)] \cdot \max\left(0, 1 - \frac{t_r}{t_{\text{ref}}}\right) \tag{7}$$

where t_{ref} is the Full-Think reference token count, $\alpha = 1.0$ weights accuracy, and $\beta = 0.5$ weights efficiency. Critically, the efficiency term is *gated by correctness*: only correct responses receive efficiency bonuses, preventing degenerate compression. Per-group advantages are computed relative to the group mean reward across $n = 8$ rollouts.

Prompt consistency. A key design choice: training-based strategies (PT-SFT, PT-DPO, PT-GRPO, RT-SFT, etc.) use the *identical* model-specific system prompt during both training data construction and inference.

D.4 Prompt-Tuning Implementations

PT-TF: Model-specific prompt design. Each hybrid-reasoning model exposes a different set of thinking mode controls. We design model-specific

prompts (full text in Appendix A §B.1) that leverage the native mode interface:

- **Qwen3.5 series**: The prompt guides the model to modulate its `<think>` block depth—empty/brief for simple problems, thorough for complex ones.
- **gpt-oss**: The prompt teaches mapping from difficulty assessment to three reasoning effort levels.
- **Seed-OSS**: The prompt teaches the model to self-allocate thinking budget and use the native `<seed:cot_budget_reflect>` tag for progress monitoring.

In all cases, the model autonomously decides the reasoning depth—the prompt provides the decision framework but does not hard-code the choice. After generation, we classify output mode based on thinking token count: `think` (> 100 tokens), `nothink` (< 10 tokens), or `brief_think` (intermediate).

PT-SFT. Training data is constructed per Step 1–2 above using the model-specific PT system prompt: for each problem, we sample under both think and no-think modes, then select the correct response with minimum tokens. The model learns to internalize the prompt-guided mode selection.

PT-DPO. Preference pairs are constructed per Step 3: the efficient correct response is chosen over verbose correct or incorrect alternatives. This teaches the model to prefer concise reasoning when prompt-guided.

PT-GRPO. The model generates multiple responses under the PT prompt, receiving rewards per Step 4 that encourage both correctness and brevity.

D.5 Routing Implementations

RT-TF: LLM-as-router (two-stage). The reasoning LLM itself serves as a router:

1. **Stage 1 (Judge):** A lightweight call with model-specific judge prompt (Appendix A §B.2). The judge runs in no-think/low-effort mode (`max_tokens=256`) to minimize overhead. It outputs a JSON object specifying the routing decision:
   - Qwen3.5: `{"mode": "1"|"2"|"3", "budget": N}` (think / nothink / budget-think)
   - gpt-oss: `{"level": "high"|"medium"|"low"}`
   - Seed-OSS: `{"mode": "1"|"2"|"3", "budget": N}` (same format as Qwen)
2. **Stage 2 (Solve):** The problem is dispatched according to the judge's decision.

JSON parsing includes fallback: if the response is malformed, the conservative default (full think) is used. No additional parameters are trained—the LLM's existing capabilities drive the routing decision.

RT-SFT & RT-DPO. For both RT-SFT and RT-DPO, we collect routing labels as follows:

1. For each problem q, run RFT under all available modes $m \in \mathcal{M}$.
2. Identify the mode m^* that produces correct answers with minimum average token cost.
3. The routing label for q is m^*.

This produces router training samples (q, m^*) for SFT, and preference pairs (q, m^+, m^-) for DPO where $m^+ = m^*$ and m^- is any alternative mode.

RT-GRPO. During GRPO training, *only the router is updated*—the backbone LLM is frozen. The router makes mode decisions, the LLM generates responses under the routed mode, and rewards are computed based on the final answer's correctness and token efficiency. This allows the router to learn optimal dispatching without modifying the LLM's reasoning capabilities.

D.6 Speculative Implementations

Both speculative variants follow a **two-pass architecture**:

Spec-Trigger: Keyword-based mode escalation.

1. **Pass 1:** Generate complete response in no-think mode.
2. **Decision:** Scan the response text for any match in the uncertainty keyword library (55 keywords across 6 categories; full list in §B.3).
3. **Pass 2 (if triggered):** Discard Pass 1 output; re-generate with full think mode. Total token count = Pass 1 tokens + Pass 2 tokens.

The keyword library is model-specific to account for different hedging patterns:

- **Qwen3.5**: 55 keywords including *wait*, *actually*, *let me reconsider*, *I'm not sure*, *hmm*, *alternatively*, *let me verify*, etc.
- **gpt-oss**: Same core library; models tend to use *hold on*, *let me think again*.
- **Seed-OSS**: Same core library; models tend to use *let me re-examine*, *actually no*.

Spec-Entropy: Token-level uncertainty trigger.

1. **Pass 1:** Generate complete response in no-think mode **with logprobs** (top-20 logprobs).

2. **Decision:** Compute normalized Shannon entropy for each output token:

$$H_t = \frac{-\sum_{v \in \text{top-}k} \hat{p}_t(v) \log \hat{p}_t(v)}{\log k} \quad (8)$$

where $\hat{p}_t$ is the renormalized distribution over the top-$k = 20$ tokens. Escalation fires if ≥ 3 tokens or $> 5\%$ of total output tokens exceed the model-specific threshold τ.
3. **Pass 2 (if triggered):** Re-generate with full think mode. Total token count includes both passes.

Spec-SFT/DPO. Training data follows the same RFT pipeline: we generate responses under both the initial no-think pass and the full speculative (trigger/entropy → re-think) pipeline, then select correct responses with minimum total tokens as SFT targets. For DPO, the efficient correct response is chosen over alternatives that either triggered unnecessarily (wasting tokens) or failed to trigger when needed (producing wrong answers).

Spec-GRPO. During training, the speculative mechanism runs end-to-end: the model begins in no-think mode, may trigger re-thinking, and produces a final answer. Multiple rollouts per problem receive rewards based on both answer correctness and total token cost (including any re-think overhead). The model learns when triggering is beneficial versus costly.

D.7 External Method Reproduction

Prompt-Tuning methods.

- **S1** (Muennighoff et al., 2025): Budget forcing via `thinking_budget` API parameter. Budget levels: Low=1024, Medium=4096, High=16384 tokens. We report the High variant. For Qwen3.5, this maps directly to `thinking_budget=16384`; for Seed-OSS, the same parameter is used.
- **TALE** (Han et al., 2025): Token-budget-aware explicit planning. The model first estimates a token budget (e.g., simple=100, medium=500, hard=1500), then reasons within that budget. We use the EP (self-Estimation Planning) variant with thinking enabled.
- **Budget-Guidance** (Li et al., 2025): Explicit token budget specified in system prompt. Budget levels: Low=128, Medium=512, High=2048. We evaluate the Medium variant. For Qwen3.5, we additionally set `thinking_budget` as a soft constraint matching the prompt budget.
- **SoT** (Aytes et al., 2025): Sketch-of-Thought with 3 cognitive paradigms (Chunked Symbolism / Conceptual Chaining / Expert Lexicons). We use domain-based paradigm selection (math→Chunked Symbolism, science→Conceptual Chaining, code→Expert Lexicons). Runs in no-think mode since conciseness is prompt-driven.
- **CoD** (Xu et al., 2025): Chain-of-Draft with domain-specific compressed prompts instructing ≤5 words per reasoning step. Runs in no-think mode.
- **DynaThink** (Pan et al., 2024): Three-stage (fast generation → confidence probe → optional regeneration). Confidence threshold=0.7.
- **DEER** (Yang et al., 2025a): Dynamic early exit monitoring 9 transition patterns (*Wait*, *Alternatively*, *Actually*, *Let me reconsider*, *On second thought*, *Hmm*, *No,*, *But wait*, paragraph breaks). Confidence threshold=0.85, minimum thinking tokens=50. Uses logprobs (top-10) to compute per-token confidence at transition points.
- **RASC** (Wan et al., 2025): Reasoning-aware self-consistency. Configuration: max_samples=8, min_samples=3 (early stopping), consistency_threshold=0.6, temperature=0.7. Scoring: 0.7×consistency + 0.3×brevity. Generates in think mode for quality.

Routing methods.

- **AdaptThink** (Zhang et al., 2025a): GRPO-trained model that internally decides think/nothink. At inference, we simply enable thinking and let the trained model choose. Mode is inferred from output: > 50 thinking tokens → think; < 5 → nothink. The model's RL training (with δ parameter controlling thinking ratio) internalizes the routing decision.
- **HDFlow** (Yao et al., 2024): Rule-based difficulty routing using query complexity heuristics. We faithfully implement their classification rules and route to think/nothink accordingly.

Speculative methods.

- **MixReasoning** (Lu et al., 2025): Entropy-based mode escalation during generation. We use their recommended entropy threshold and implement the same two-pass architecture as our Spec-Entropy.
- **ADR** (Zhang et al., 2025c): SFT+GRPO-trained adaptive switching policy. We retrain following their pipeline on our training set for fair comparison.