UNDERSTANDING THINKING-MODE SWITCH STRATEGIES IN HYBRID-REASONING LLMS

Anonymous authors
Paper under double-blind review

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 THINKSWITCH, a unified evaluation framework for studying thinking-mode switching in hybrid-reasoning LLMs. THINKSWITCH 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. THINKSWITCH provides reference implementations and a unified evaluation platform to support more controlled research on efficient reasoning in hybrid-reasoning LLMs. Our data, code and repository are available at https://anonymous.4open.science/r/ThinkSwitch.

1 INTRODUCTION

Recent reasoning-oriented LLMs, such as OpenAI o1 (Jaech et al., 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 (Yang et al., 2025a), gpt-oss Agarwal et al. (2025), and Seed-OSS, 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: **(i) 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. **(ii) 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. **(iii) 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.

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

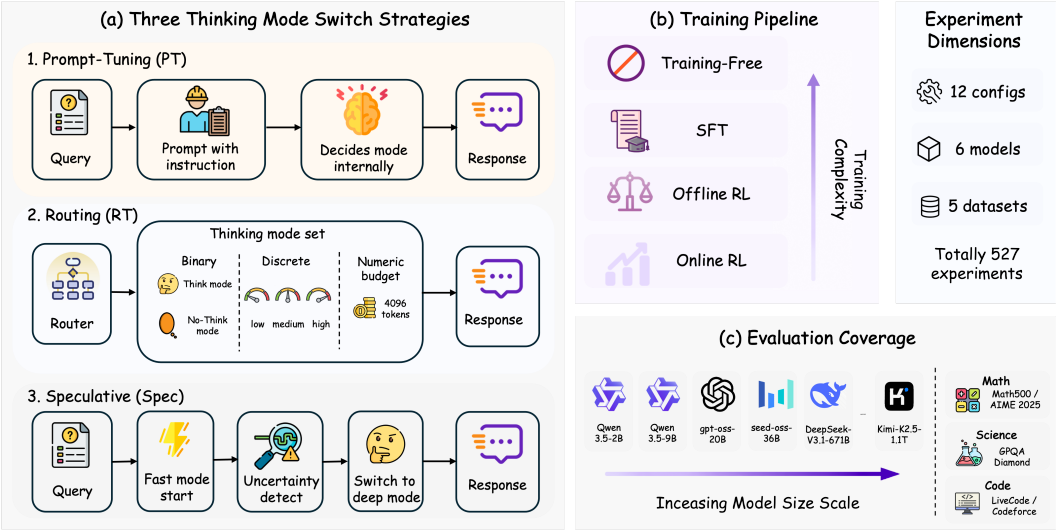


Figure 1: Overview of THINKSWITCH. **(a)** Three thinking-mode switch strategies: Prompt-Tuning, Routing, and Speculative. **(b)** Training pipeline spanning training-free to online RL. **(c)** Our evaluation coverage across 6 models spanning from 2B to 1.1T scale, 5 datasets, totaling 527 experiment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

Despite active progress, these methods are evaluated 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 present THINKSWITCH shown in Figure 1, a unified evaluation framework 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 study that 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 strategies 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 model 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: **i) 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 (Yang et al., 2025a),

DeepSeek-V3.1, Kimi-K2.5, and so on. **ii) 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. **iii) Numeric budget.** LLM family like Seed-OSS-36B 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 the Hybrid Fine-Tuning (HFT) cold start of (Jiang et al., 2026). RL-based methods include ACPO (Cheng et al., 2026), and the subsequent Hybrid Group Policy Optimization (HGPO) stage of (Jiang et al., 2026). DPO-based methods include AdaR1 (Luo et al., 2026) 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 ThinkSwitch 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: **i) A binary state space**: $\mathcal{M} = \{\text{think}, \text{no_think}\}$. **ii) A discrete effort space**: $\mathcal{M} = \{\text{low}, \text{mid}, \text{high}\}$. **iii) 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.

Table 1: The evaluation taxonomy of the THINKSWITCH. 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.

	Training-Free	SFT	Offline RL (DPO)	Online RL (GRPO)
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}

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:*

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

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. (2025c), and t^* is the token position at which $f(\tau_{1:t^*})$ is satisfied.

4 THINKSWITCH 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.

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

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

Datasets. As shown in Table 2, we evaluate on five benchmarks spanning three task domains: **(i) Mathematics**: AIME 2025 (competition-level math problems) and MATH500 (high school math problems) Lightman et al. (2023). **(ii) Science**: GPQA-Diamond (graduate-level questions ranging from physics, chemistry, to biology) (Rein et al., 2023). **(iii) Code**: Live Code Bench (LCB) (live programming problems with execution-based evaluation) (Jain et al., 2025) and Codeforces (competition-level programming problems).

Models. In this paper, we evaluate 6 hybrid-reasoning LLMs spanning 2B to 1.1T parameters, covering three existing thinking modes: **(i) Qwen3.5-2B** and **Qwen3.5-9B** (Yang et al., 2025a): Binary switch (`think`/`no_think`). **(ii) gpt-oss-20B**: Discrete thinking mode switch (e.g., High/Medium/Low reasoning effort). **(iii) Seed-OSS-36B-Instruct**: Thinking mode switch via numeric token budget ($b \leq B_{\max}$). **(iv) DeepSeek-V3.1-671B** Liu et al. (2024): Binary switch (`think`/`no_think`). **(v)Kimi-K2.5-1.1T** Team et al. (2026): Binary switch (`think`/`no_think`).

4.2 EVALUATION METRICS

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

- **Acc**: Pass@1 accuracy (%). We report a single greedy sample, so the reported accuracy reflects the model's Pass@1 performance. Throughout the paper, **AVG** denotes *question-level pooled* accuracy: the total number of correctly solved problems divided by the total number of problems across all five benchmarks, avoiding over-weighting smaller benchmarks.
- **Tok**: Average end-to-end output token cost. Because the three strategies invoke different numbers of generation stages, we measure **Tok** identically for every strategy and always include the full cost

of any auxiliary stage. For RT, this means the router call is always added to the answering pass and is never omitted from the reported cost. For SPEC, when a trigger fires mid-generation, the discarded fast-mode pass is still charged in full alongside the re-generated deep-reasoning pass.

4.3 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 F. We categorize implementations into two parts:

- **Training-Free (TF) Implementation**: We compare the following three strategies:
  - **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., 2025b) 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).
- **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, enabling comparison rather than an absolute ranking of prior methods. Reproduction details and any deviations from original papers are documented in Appendix F.8.

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)?*

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.

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

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*:

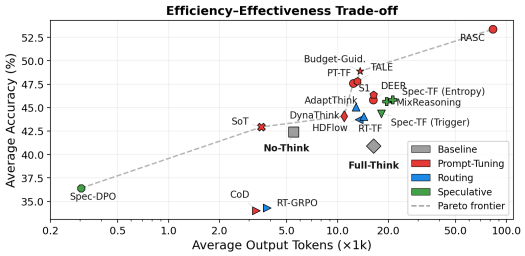


Figure 2: Overall efficiency–effectiveness trade-off. Each point represents a method averaged over 5 datasets; dashed line shows the Pareto frontier.

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 to difficulty, thereby avoiding unnecessary reasoning on simpler problems.

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

Finding 3: SPEC boosts accuracy at extra token cost. Unlike PT and RT, Spec-TF increases tokens relative to Full-Think, but in return yields notable accuracy increases, particularly on code tasks where the "try-then-verify" mechanism excels. Since a fired trigger discards the fast pass and re-generates in deep mode, SPEC works 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*, and 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

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

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

at 2B, all three strategies perform similarly (13.2–14.1%). RT generally ranks last but remains competitive at larger model scales (77.8% at 1.1T). This scale-dependent ranking also suggests that no single strategy can dominate in effectiveness.

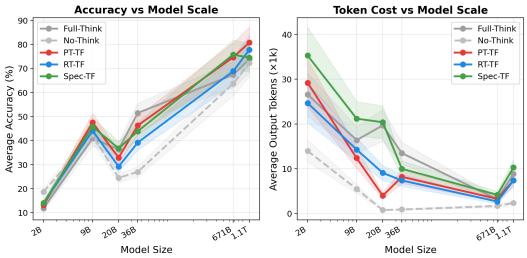


Figure 3: Strategy effectiveness (left) and efficiency (right) across model scales (2B–1.1T). All three strategies are evaluated across 6 models.

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 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 domain-dependent strategy preferences, demonstrating that the nature of the task influences strategy selection:

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

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

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.

6 EFFECT OF TRAINING PIPELINE ON SWITCHING STRATEGIES

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

To answer RQ2, we train each of the three strategies under three regimes (i.e., SFT, DPO, and GRPO) and compare against the Training-Free (TF) baselines from §5. We run this full grid on three models chosen to expose structurally different switching interfaces, yielding 27 trained configurations that we analyze jointly, with a model-scale synthesis in §6.3; the reason the trained grid stops at 36B is given in Appendix F.2. Figure 4 traces how accuracy and token reduction evolve as training strengthens for every strategy on all three models.

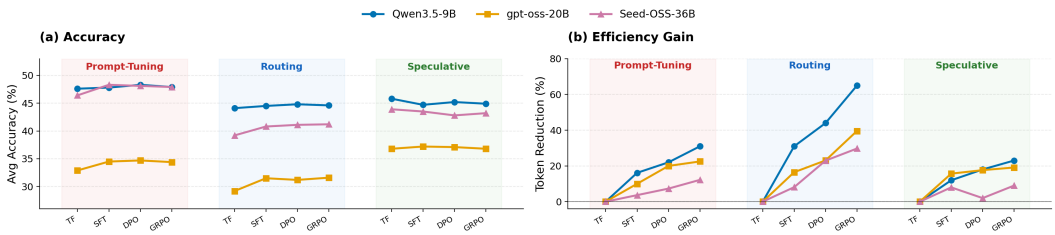


Figure 4: Training effect on switching capacity across three models. **(a)** Average accuracy over 5 benchmarks; **(b)** token reduction relative to each configuration's training-free baseline.

6.1 Overall Trade-off Pattern

Finding 7: Training universally improves switching capacity, with larger gains in efficiency than accuracy. On all three models, training (SFT, DPO, and GRPO) leads to marginal accuracy increase, while cutting token cost by as much as 65% (Figure 4b). This indicates that training primarily teaches the model *when* to skip unnecessary reasoning, rather than improving the reasoning itself. The pattern is remarkably consistent across the binary, discrete-effort, and budget interfaces, which suggests it is a property of the switching strategies rather than of any single model.

Finding 8: RT benefits the most from training in efficiency, consistently across interfaces. On every model the routing strategy yields the largest token reduction from training: -65% on Qwen3.5-9B (binary), -40% on gpt-oss-20B (discrete effort), and -30% on Seed-OSS-36B (numeric budget), each ahead of PT and SPEC on the same model. A plausible explanation is that RT's binary route-or-not decision is well-matched to a coarse correctness reward: the training signal directly reflects routing quality, letting the router learn a sharper difficulty boundary. The efficiency ranking is also monotone in training strength: within almost every configuration the reduction grows along SFT $<$ DPO $<$ GRPO. Because this finding holds across three distinct models rather than being tied to a single one, we can attribute it to the routing strategy itself rather than to any particular model.

6.2 Training Regimes Effect

Finding 9: The three training regimes offer distinct trade-off profiles. The regimes specialize along complementary axes rather than dominating one another. GRPO delivers the largest token savings and is the natural choice when efficiency is the priority; DPO leans toward accuracy, giving the best effectiveness on Qwen3.5-9B; and SFT sits between them as a low-cost, balanced default when the optimization target is unspecified. These are tendencies, not a fixed ranking: the accuracy-optimal regime is itself model-dependent (DPO on Qwen3.5-9B, whereas SFT is at least as strong on gpt-oss-20B and Seed-OSS-36B), so practitioners should pick the regime to match their operating priority and model rather than assume one regime dominates.

6.3 Model Scale Effect

Finding 10: The qualitative effect of training holds across model sizes, but its quantitative profile is model-specific. Across the three trained model sizes, one pattern consistently holds: training improves efficiency substantially more than accuracy. However, the magnitude and optimal training regime vary considerably across models. Under the same GRPO training, for example, RT reduces token usage by 65% on Qwen3.5-9B but by only 30% on Seed-OSS-36B. Moreover, as shown in Finding 9, the accuracy-optimal regime shifts with size, from DPO at 9B to SFT at 20B and 36B. Thus, training yields a consistent qualitative pattern across sizes, but the achievable gains and optimal operating points remain model-dependent.

7 Unified Comparison of Existing Methods

RQ3: *When existing methods are adapted onto a fixed backbone and unified pipeline (e.g., same model and data), are the strategy-level trade-offs they exhibit consistent with our own findings?*

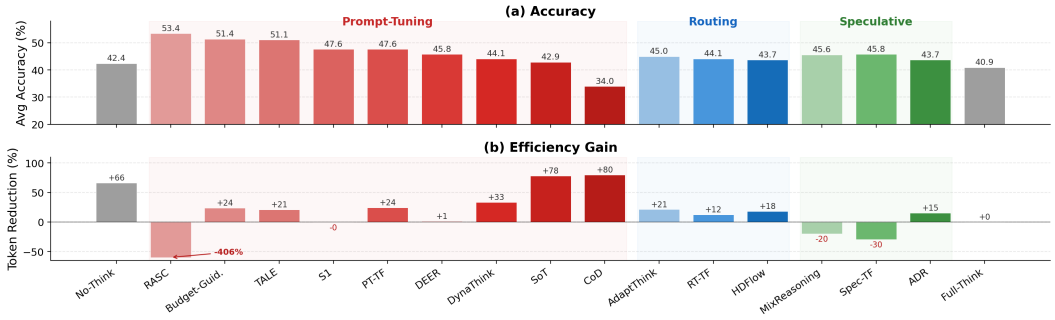


Figure 5: Unified comparison of 12 methods under matched conditions on Qwen3.5-9B. (a) Average accuracy across five benchmarks. (b) Token reduction relative to Full-Think (positive = saving). Methods grouped by strategy.

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. Since a single averaged operating point can conflate regimes, we also report an iso-budget view (Appendix C) with realized end-to-end consumption.

Finding 11: 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 12: No single external method dominates across all task domains. Existing method performance varies by domain: e.g., 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, Limitations, and Future Work

In this paper, we presented ThinkSwitch, a unified evaluation framework 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 find 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.

Our evaluation targets single-turn reasoning over five benchmarks in mathematics, science, and code, so trade-offs under multi-turn, tool-use, or other task domains may differ. Promising directions include extending training-based evaluation to large scales, studying thinking-mode switching under multi-turn and tool-use workloads where token cost is also shaped by inter-turn environment feedback, and broadening ThinkSwitch's task-domain coverage beyond mathematics, science, and code.

AI USE STATEMENT

In this work, we used generative AI tools to clean and reformat raw experimental logs into the analysis tables and figures reported in the paper. Additionally, we used generative AI tools to assist with drafting and polishing the paper text, creating and refining scientific figures (e.g., Figures 2–5). We have reviewed all AI-assisted text, figures, and code for correctness, and we take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

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.

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

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

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.

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

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

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Lingjie Jiang, Xun Wu, Shaohan Huang, Qingxiu Dong, Zewen Chi, Li Dong, Xingxing Zhang, Tengchao Lv, Lei Cui, and Furu Wei. Think only when you need with large hybrid-reasoning models. *Advances in Neural Information Processing Systems*, 38:159355–159382, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

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

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

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

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

Haotian Luo, Haiying He, Yibo Wang, Jinluan Yang, Rui Liu, Naiqiang Tan, Xiaochun Cao, Dacheng Tao, and Li Shen. Ada-r1: Hybrid-cot via bi-level adaptive reasoning optimization. *Advances in Neural Information Processing Systems*, 38:59353–59377, 2026.

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

Jiabao Pan, Yan Zhang, Chen Zhang, Zuozhu Liu, Hongwei Wang, and Haizhou Li. 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*, pp. 14686–14695, 2024.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

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

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

Guangya Wan, Yuqi Wu, Jie Chen, and Sheng Li. 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)*, pp. 3613–3635, 2025.

Shouren Wang, Wang Yang, Xianxuan Long, Qifan Wang, Vipin Chaudhary, and Xiaotian Han. Demystifying hybrid thinking: Can llms truly switch between think and no-think? *arXiv preprint arXiv:2510.12680*, 2025.

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

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

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.

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*, 2025b.

Wang Yang, Xiang Yue, Vipin Chaudhary, and Xiaotian Han. Speculative thinking: Enhancing small-model reasoning with large model guidance at inference time. *arXiv preprint arXiv:2504.12329*, 2025c.

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

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

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

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

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

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 THINKSWITCH 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: Answer Scoring Protocol

Every benchmark in ThinkSwitch ships with ground-truth answers, so correctness is determined by deterministic rule-based scoring and no model-based grader is involved at any point. Concretely, MATH500 and AIME 2025 responses are parsed for the final answer and compared against the reference by string match plus symbolic equivalence checking, so that mathematically equivalent forms (e.g., $1/2$ and 0.5) are both accepted; GPQA-Diamond is scored by extracting the selected option and matching it against the ground-truth option; LiveCodeBench and Codeforces solutions are

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	83.3	17.5k	63.4	10.1k	29.5	9.8k	30.5	9.5k	47.8	10.4k
PT	DPO	85.8	4.8k	83.3	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	76.7	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	76.7	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	66.7	28.1k	53.0	13.5k	36.5	15.3k	34.8	16.1k	44.7	16.0k
SPEC	DPO	86.4	6.2k	70.0	26.5k	53.5	12.8k	37.7	14.1k	35.2	14.8k	45.2	14.9k
SPEC	GRPO	86.0	5.8k	70.0	25.2k	52.5	12.2k	37.1	13.5k	35.5	14.0k	44.9	14.1k

Table 6: Full per-dataset training results (Qwen3.5-9B). Acc: Pass@1 (%); Tok: average output tokens.

scored by executing the generated program against the official test cases. This protocol is applied identically to every strategy, training regime, and external method, so reported accuracy differences are never confounded by grader variability.

B Full Per-Dataset Results

B.1 Unified Comparison: Additional Findings

Key findings from the unified comparison: Our unified evaluation reveals distinct performance characteristics across strategy families. Among PT methods, RASC achieves the highest accuracy (53.4% average) but at significantly higher token cost (-406%), while Budget-Guidance and TALE offer favorable token-accuracy trade-offs (51.4% and 51.1% accuracy, $+23.9\%$ and $+21.2\%$ token reduction). Among RT methods, AdaptThink and RT-TF deliver similar accuracy (45.0%, 44.1%) with competitive token reduction (+21.3%, +12.5%). Speculative methods generally occupy the middle ground, with Spec-DPO achieving 45.2% accuracy and +9.1% token reduction. Our PT-TF (47.6% accuracy, +24.4% reduction) and RT-TF baselines (44.1%, +12.5%) sit at competitive positions within their respective strategy families. When averaged across all methods, token reduction correlates positively with efficiency gains while accuracy varies by task domain.

B.2 Training Effect Per-Dataset Results

Table 6 reports the complete per-dataset training results for Qwen3.5-9B, whose 5-benchmark averages are the Qwen3.5-9B line in Figure 4.

C Iso-Budget Comparison

Averaging each method at its own natural operating point can conflate methods that make fundamentally different budget assumptions: self-consistency methods such as RASC deliberately spend multiple samples, whereas CoD and SoT are aggressive compression methods. To make these as-

sumptions explicit we constrain every method to matched token budgets on MATH500 (Qwen3.5-9B) and report accuracy at each budget alongside its unconstrained realized cost.

We enforce the budget per regime: sampling methods cap the number of samples so the *summed* token count over all samples meets the target; compression, prompt-tuning, and speculative methods cap `max_tokens` (for SPEC the cap covers both the discarded fast pass and the deep pass); routing methods cap the combined judge-plus-solve budget. The *natural cost* column reports each method's realized end-to-end consumption at its own unconstrained operating point—this is the quantity that includes extra samples, discarded speculative passes, and routing overhead, and it is what makes a face-value average misleading. RASC's natural cost (30k) is nearly $2\times$ the largest matched budget, so its apparent lead evaporates once budgets are equalized; conversely, CoD and SoT sit far below the smallest tier (0.7–0.9k), so their accuracy is flat across all three budgets—they cannot use the extra allowance, by design rather than by defect.

Method	Regime	Natural cost (realized tok)	Accuracy (%) at matched budget ∼4k	∼8k	∼16k
RASC	sampling	30.0k	87.2	89.6	89.8
CoD	compression	0.7k	74.2	73.6	73.2
SoT	compression	0.9k	82.6	82.2	82.2
AdaptThink	routing	5.8k	82.4	85.8	86.4
PT-TF (ours)	prompt-tuning	6.5k	84.4	86.2	85.8
Spec-TF (ours)	speculative	11.1k	82.2	83.4	85.4

Table 7: Iso-budget comparison on MATH500 (Qwen3.5-9B). *Natural cost*: realized end-to-end tokens at the method's own unconstrained operating point, including extra samples (RASC), discarded speculative passes (SPEC), and judge overhead (routing). Accuracy columns constrain every method to a matched token budget; compression methods (CoD, SoT) read flat because their natural cost already lies below the smallest tier.

D SENSITIVITY TO IMPLEMENTATION CHOICES

A strategy-level comparison is only meaningful if it reflects intrinsic strategy behaviour rather than one particular prompt or threshold. We therefore vary, for each strategy, its single most sensitive implementation component, holding the other two strategies at their original implementation as reference (Qwen3.5-9B).

Swept component	Variant	Acc (%)	Tok	Acc rank	Tok rank
PT system prompt	original	47.6	12.4k	1	1
	concise	46.2	11.2k	1	1
	few-shot	48.8	13.5k	1	1
RT judge prompt	original	44.1	14.3k	3	2
	concise	44.2	13.2k	3	2
	few-shot	44.5	15.6k	3	2
SPEC entropy τ	0.10	45.8	21.2k	2	3
	0.06	44.6	23.5k	2	3
	0.14	45.2	23.2k	2	3

Table 8: Sensitivity of each strategy to its most influential implementation component (Qwen3.5-9B). Ranks are the position of the swept strategy among the three strategies on accuracy and token cost.

Across all nine variants the absolute numbers move only modestly—accuracy within about 2.6 points and tokens within roughly 2.4k of the original setting—and, crucially, the *relative* ordering of the three strategies never changes: PT stays most accurate and most token-efficient (rank 1/1), RT stays least accurate but middling in cost (3/2), and SPEC stays in between on accuracy while most expensive in tokens (2/3). Reporting the absolute values, rather than ranks alone, confirms that our strategy-level conclusions are not an artifact of a particular prompt or threshold choice.

D.1 Sensitivity to Sampling Randomness (Multi-Seed Stability)

Our main results (Table 3) use a single greedy decode (temperature=0) per configuration. To check that this is not a source of instability, we additionally sample the five core Training-Free configurations (Full-Think, No-Think, PT, RT, SPEC (Entropy)) on Qwen3.5-9B under non-greedy decoding (temperature=0.6) across 3 random seeds, on the two smallest benchmarks where sampling noise is most likely to matter: AIME 2025 (30 problems) and GPQA-Diamond (198 problems).

Method	Dataset	Seed 1		Seed 2		Seed 3		Acc Mean±Std
		Acc	Tok	Acc	Tok	Acc	Tok	
Full-Think	AIME 2025	70.0	23.1k	73.3	22.6k	70.0	20.2k	71.1 ± 1.9
No-Think	AIME 2025	53.3	11.8k	53.3	11.6k	56.7	13.3k	54.4 ± 2.0
PT-TF	AIME 2025	80.0	20.2k	76.7	21.1k	80.0	19.5k	78.9 ± 1.9
RT-TF	AIME 2025	73.3	22.4k	73.3	23.1k	73.3	22.8k	73.3 ± 0.0
SPEC-TF	AIME 2025	63.3	32.4k	60.0	32.7k	63.3	33.1k	62.2 ± 1.9
Full-Think	GPQA-Dia.	53.0	15.9k	54.5	16.2k	54.5	15.6k	54.0 ± 0.9
No-Think	GPQA-Dia.	53.0	3.2k	53.0	3.3k	54.5	3.1k	53.5 ± 0.9
PT-TF	GPQA-Dia.	63.1	12.1k	63.1	12.4k	63.1	12.3k	63.1 ± 0.0
RT-TF	GPQA-Dia.	52.0	11.2k	52.5	11.3k	52.0	11.0k	52.2 ± 0.3
SPEC-TF	GPQA-Dia.	55.6	18.4k	55.6	17.8k	54.5	18.2k	55.2 ± 0.6

Table 9: Stability of the five core Training-Free configurations across 3 random seeds (Qwen3.5-9B, temperature=0.6) on the two smallest benchmarks. These non-greedy runs are a separate batch from the greedy main table and are not meant to replace it; they are reported solely to bound sampling variance.

Accuracy varies by at most ± 2.0 points on AIME 2025 (30 problems, so this is within the noise of a single problem flipping across seeds) and by at most ± 0.9 points on GPQA-Diamond (198 problems). More importantly, the *relative ranking* of the five configurations under seed-averaging is identical to the greedy ranking reported in Table 3 on both benchmarks: PT > RT > Full-Think > SPEC > No-Think on AIME 2025, and PT > SPEC > Full-Think > No-Think > RT on GPQA-Diamond. This indicates that the accuracy comparisons underlying our Findings are not an artifact of a single deterministic decode.

E 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 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.

F IMPLEMENTATION DETAILS

This appendix provides comprehensive implementation details for all methods in THINKSWITCH, complementing the overview in §4.3.

F.1 TRAINING HYPERPARAMETERS

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 10: Training hyperparameters.

F.2 SCOPE OF THE TRAINED GRID

The training study in §6 covers three models (Qwen3.5-9B, gpt-oss-20B, and Seed-OSS-36B), selected so that each of the three structurally different switching interfaces, i.e., binary `think`/`no_think`, discrete reasoning effort, and numeric token budget, is represented by exactly one trained model. Running the full SFT/DPO/GRPO grid at the 671B and 1.1T scales was beyond our academic compute budget, so those two models are evaluated in the training-free setting only (§5), and all training-related findings (Findings 7–10) should be read as statements about models up to 36B parameters rather than about the entire 2B–1.1T range.

F.3 MODEL-SPECIFIC PARAMETER MAPPING

A key design principle of THINKSWITCH 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.

F.4 UNIFIED TRAINING DATA CONSTRUCTION

All training-based methods (SFT, DPO, GRPO) across all three strategies follow a unified rejection fine-tuning (RFT) pipeline. Given a training set from MathLightEval Hendrycks et al. (2021), our data construction proceeds as follows. MathLightEval repackages the same Hendrycks MATH corpus into a 7,500-problem train split and a 5,000-problem test split; all training data in this paper is drawn exclusively from the train split, which is disjoint from the test split underlying MATH500, so there is no overlap between our training pool and the MATH500 evaluation set.

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. We follow the reward configuration of DeepSeek-R1 (Guo et al., 2025), which likewise assigns the dominant weight to a rule-based correctness term and a smaller weight to the auxiliary shaping term, and we do not tune α or β per strategy or per model, so that any difference we observe between strategies is not an artifact of strategy-specific reward tuning. 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.

F.5 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.3) 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.

F.6 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.4). Here "judge" denotes the *difficulty classifier* that produces the routing decision inside the RT strategy; it is part of the method under evaluation and is never used to score correctness (scoring is rule-based, see Appendix A.7). 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.

F.7 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 Appendix §A.5).
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:

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

   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 $\rightarrow$ 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.

F.8 External Method Reproduction

Table 11 summarizes, for each of the 12 external methods integrated into ThinkSwitch, the mechanism used in the original paper, how we reproduce it within our unified pipeline, and the main point of deviation. We adapt each method's core mechanism to our unified environment as a fixed backbone, so as to isolate the effect of agent scaffolding from that of our analysis. Three properties are enforced by the pipeline itself and therefore hold for every method without exception: (i) inference runs on the same unforked vLLM stack with the decoding configuration of Appendix F.3; (ii) token cost is measured with the single end-to-end accounting rule of §4.2, so auxiliary router calls and discarded speculative passes are always charged in full; and (iii) correctness is scored by the same deterministic ground-truth protocol of Appendix A.7. Method-internal hyperparameters (thresholds, budgets, sampling counts) are kept at the values reported in each original paper unless a deviation is listed below, and the deviations that remain are confined to the method-specific mechanism rather than to the shared evaluation conditions.

Method	Original mechanism	Our reproduction & main deviation
Prompt-Tuning		
S1 (Muennighoff et al., 2025)	Suppresses the end-of-thinking stop token and injects "Wait" to force continued thinking, up to a token budget.	We control the same budget levels (1024/4096/16384 tokens) via the model's native `thinking_budget`/`reasoning_effort` parameter instead of literal stop-token suppression and text injection, since not all three model families expose a decoding-loop hook.
TALE-EP (Han et al., 2025)	Model self-estimates a token budget, then reasons within it.	Same mechanism (prompted self-estimation); we additionally report three fixed-budget variants (100/500/1500 tokens) not present in the original paper, for comparison with S1.
Chain-of-Draft (Xu et al., 2025)	Minimal draft-style steps ($\leq$5 words/step) via prompting only.	Identical prompt-only mechanism; paired with no-think/low-effort decoding since conciseness is prompt-driven rather than decoding-driven.
Budget-Guidance (Li et al., 2025)	Enforces a token budget inside the decoding loop via a forked `transformers` library.	Prompt-only approximation (explicit budget stated in the prompt; levels 128/512/2048 tokens), since we standardize all methods on a single unforked decoding stack (vLLM). This is the same simplification the original authors report as a reasonable approximation.
DynaThink (Pan et al., 2024)	Fast pass, then self-rated confidence; escalate to a slow pass if confidence is low.	Same two-step mechanism with threshold 0.7; confidence is elicited via a separate LLM call (black-box), matching the original paper's setting.
Sketch-of-Thought (Aytes et al., 2025)	Classifies queries into 3 cognitive paradigms via a trained DistilBERT classifier, then applies paradigm-specific compressed prompts.	We default to a lightweight domain$\rightarrow$paradigm rule (math/science/code) rather than the DistilBERT classifier, to avoid an extra model dependency; the classifier-based mode is implemented but not used for the reported numbers.
RASC (Wan et al., 2025)	Quality-aware self-consistency sampling with early stopping.	Same mechanism (max 8 samples, min 3, early-stop consistency 0.6); uses temperature 0.7 rather than greedy, since diversity sampling is intrinsic to the method (noted in §4.3).
DEER (Yang et al., 2025b)	True early exit during decoding at high-confidence transition points, saving real compute.	*Post-hoc approximation*: we run the full generation with logprobs and retrospectively identify the point at which an early exit would have fired, rather than interrupting decoding, because our unified stack standardizes on an unforked vLLM inference path. The token cost we report is therefore the full (non-exited) generation, which should be read as an *upper bound* on DEER's cost and correspondingly a *lower bound* on its efficiency; DEER's accuracy is unaffected by this approximation, and its realized savings under a true streaming early-exit implementation would be larger than our numbers indicate. We therefore do not interpret DEER's position in Figure 5b as evidence about its attainable efficiency.
Routing		
AdaptThink (Zhang et al., 2025a)	RL(PPO)-trained think/nothink policy.	We load the externally-trained checkpoint released by the authors and let the fine-tuned policy decide natively; inference, decoding configuration, and token accounting run inside our unified pipeline, and the reported think/nothink mode is inferred post-hoc from thinking-token count for logging only.
HDFlow (Yao et al., 2024)	Rule-based difficulty classification for routing.	Same rule-based routing mechanism as described in §2, with the method's internal thresholds kept at the original paper's default values; inference is executed through our unified vLLM stack under the shared decoding protocol and end-to-end token accounting.
Speculative		
MixReasoning (Lu et al., 2025)	Entropy-based trigger for escalation to deep reasoning.	Same entropy-triggered escalation mechanism as described in §2, with the entropy threshold kept at the original paper's default value; inference is executed through our unified vLLM stack under the shared decoding protocol and end-to-end token accounting.
ADR (Zhang et al., 2025c)	Two-stage SFT + GRPO trained switching policy.	Same two-stage switching policy as described in §2, trained with the original paper's reported recipe and hyperparameters; the resulting policy is then evaluated inside our unified pipeline under the shared decoding protocol and end-to-end token accounting.

Table 11: Reproduction fidelity of the 12 external methods integrated into THINKSWITCH. All methods share the same decoding protocol and token-accounting scheme; deviations are limited to the method-specific mechanism.