

KAT-V1: Kwai-AutoThink Technical Report



<https://huggingface.co/Kwaipilot/KAT-V1-40B>

Zizheng Zhan^{*†}, Ken Deng*, Huaixi Tang*, Wen Xiang*, Kun Wu*, Weihao Li, Wenqiang Zhu, Jingxuan Xu, Lecheng Huang, Zongxian Feng, Shaojie Wang, Shangpeng Yan, Xuxing Chen, Jiaheng Liu, Zhongyuan Peng, Zuchen Gao, Haoyang Huang, Xiaojiang Zhang, Jinghui Wang, Zheng Lin, Mengtong Li, Huiming Wang, Ziqi Zhan, Yanan Wu, Yuanxing Zhang, Jian Yang, Guang Chen, Haotian Zhang, Bin Chen, Bing Yu

Kwaipilot Team

{zhanzizheng, dengken, zhanghaotian}@kuaishou.com

Abstract

We present Kwaipilot-AutoThink (KAT), an open-source 40B large language model developed to address the overthinking problem in reasoning-intensive tasks, where an automatic thinking training paradigm is proposed to dynamically switch between reasoning and non-reasoning modes based on task complexity. Specifically, first, we construct the dual-regime dataset based on a novel tagging pipeline and a multi-agent synthesis strategy, and then we apply Multi-Token Prediction (MTP)-enhanced knowledge distillation, enabling efficient and fine-grained reasoning transfer with minimal pretraining cost. Besides, we implement a cold-start initialization strategy that introduces mode-selection priors using majority-vote signals and intent-aware prompting. Finally, we propose Step-SRPO, a reinforcement learning algorithm that incorporates intermediate supervision into the GRPO framework, offering structured guidance over both reasoning-mode selection and response accuracy. Extensive experiments across multiple benchmarks demonstrate that KAT consistently matches or even outperforms current state-of-the-art models, including DeepSeek-R1-0528 and Qwen3-235B-A22B, across a wide range of reasoning-intensive tasks while reducing token usage. Notably, KAT outperforms all open-source models and even surpasses o3-mini on the leakage-controlled LiveCodeBench Pro. Beyond academic evaluation, KAT has been successfully deployed in Kwaipilot (i.e., Kuaishou's internal coding assistant), where it improves real-world development workflows with high accuracy, efficiency, and controllable reasoning behaviors. Moreover, we are actively training a 200B Mixture-of-Experts (MoE) model with 40B active parameters, and early results already show significant gains, further demonstrating the scalability of the AutoThink paradigm.

*Equal contribution. †Corresponding author.

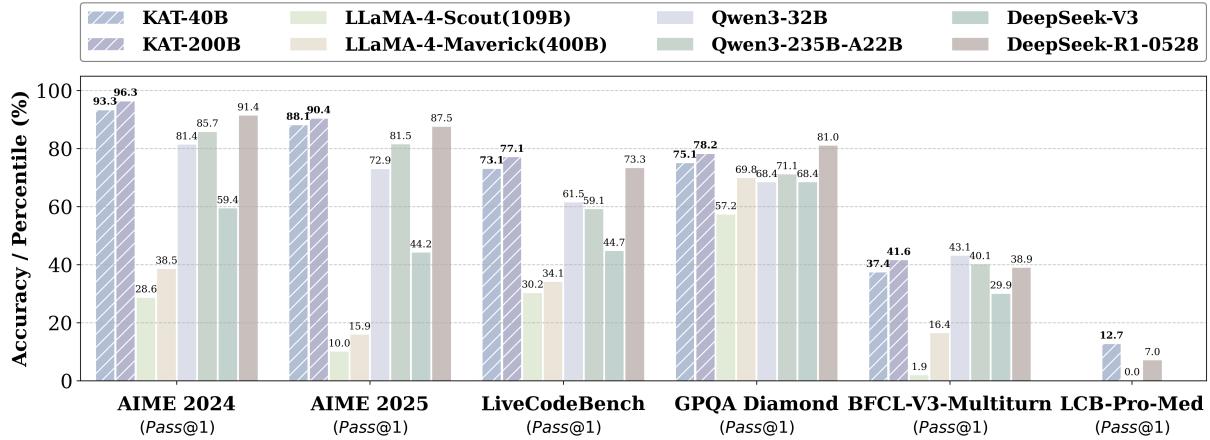


Figure 1 | Performance of Kwaipilot-AutoThink on various benchmarks.

1. Introduction

Large language models (LLMs) have recently achieved impressive, even human-level, performance across a broad spectrum of natural language understanding and generation tasks [1–15]. A key driver of this progress is the ability of LLMs to perform chain-of-thought (CoT) reasoning, which has substantially improved their problem-solving abilities on complex tasks.

However, recent studies have revealed that applying CoT reasoning indiscriminately to all inputs—regardless of their inherent complexity—often results in significant computational overhead, increased latency, and excessive token consumption, particularly for straightforward queries [16, 17]. This phenomenon, commonly referred to as overthinking, has emerged as a critical bottleneck for the real-world deployment of LLMs in interactive applications, where both efficiency and responsiveness are essential[18, 19].

To mitigate the overthinking problem, a growing body of research has focused on improving the efficiency by adaptively reasoning in large language models [19–23]. One of the representative approaches is to empower models with the ability to dynamically decide whether to engage in reasoning, based on the complexity of the input. Notable examples include AdaptThink [23] and AdaCoT [19], which employ reinforcement learning [24, 25] to train models that can selectively trigger CoT reasoning during inference. Despite their promising results, most of these efforts still face two fundamental limitations. First, they typically lack sufficient supervision to help models develop robust, query-specific preferences over reasoning strategies. Second, existing RL frameworks are often coarse-grained, providing limited intermediate feedback and failing to model the step-wise nature of reasoning and mode selection.

Moreover, research on knowledge distillation [26–28] has aimed to improve model efficiency by transferring reasoning capabilities from stronger teacher models to smaller or less expensive student models. However, conventional distillation approaches often require extensive pretraining or rely solely on final-answer alignment, which fails to capture the nuanced utility of intermediate reasoning steps. Although methods like Multi-Token Prediction (MTP) [29] have been proposed to model future utility, few have explored how MTP can be systematically integrated into the distillation pipeline for reasoning tasks.

Building upon these insights, as shown in Figure 2, we present **Kwaipilot-AutoThink (KAT)**, a 40B open-source LLM that addresses the overthinking challenge through a two-stage training framework (i.e., pre-training and post-training stages) and achieves adaptive reasoning, efficient

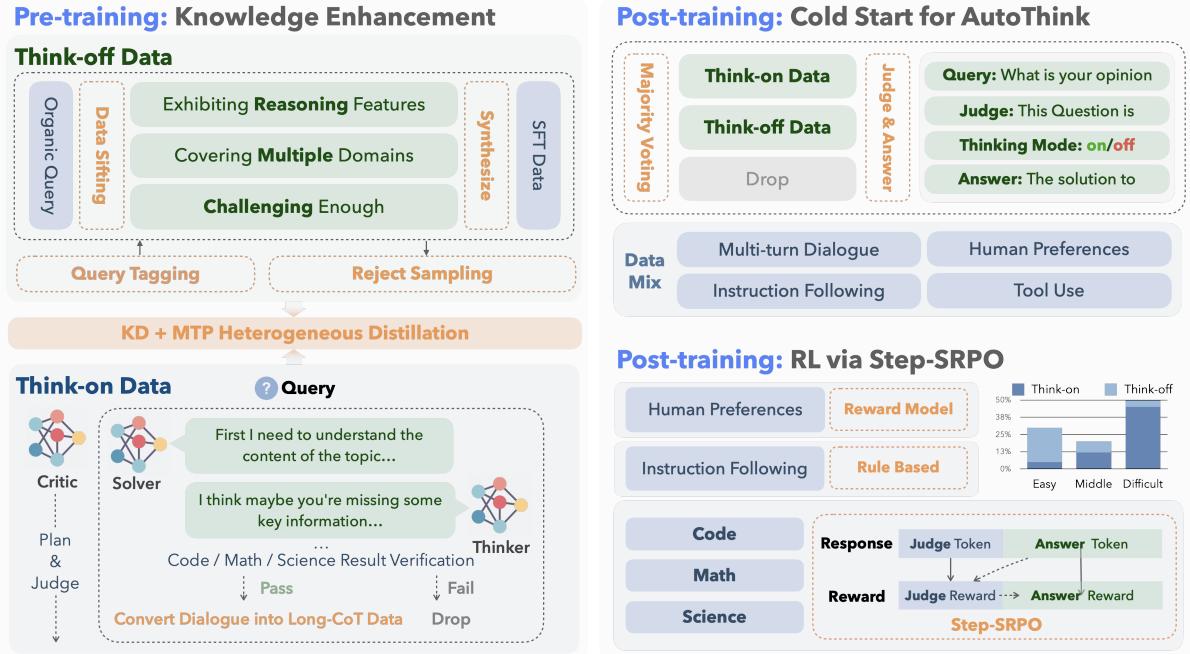


Figure 2 | The **two-stage** AutoThink training framework for controllable and efficient reasoning in large language models.

knowledge injection, and real-world applicability as follows:

In the pre-training stage¹, we introduce a knowledge enhancement strategy based on knowledge distillation and multi-token prediction. Specifically, we propose a novel data labeling system to curate a diverse and challenging set of training queries. Leveraging this system, we construct two distinct data regimes—Think-off data synthesized via our tagging system, and Think-on data generated through a multi-agent framework grounded in state-of-the-art LLMs. Unlike prior work, we combine knowledge distillation with Multi-Token Prediction (MTP) to train on this dual-regime data, enabling more granular utility modeling and efficient acquisition of high-quality reasoning capabilities while avoiding the cost of extensive pretraining.

In the post-training stage, we introduce an auto-thinking method to achieve efficient token usage for reasoning LLMs. Specifically, we first establish a robust cold-start pipeline for AutoThink training. For each query, we employ majority voting to determine the appropriate thinking mode and condition generation on both query attributes and inferred user intent. Then, we propose Step-SRPO, an enhanced reinforcement learning algorithm that introduces intermediate supervision into the SRPO [30] framework, enabling structured guidance at two key levels: (1) accurate thinking-mode selection, and (2) final response correctness under the selected mode. This staged design better aligns the model’s internal decision process with the complexity of the input, stabilizing training and improving convergence.

Through this two-stage paradigm, Kwaipilot-AutoThink not only achieves state-of-the-art performance across multiple benchmarks—including math, code, commonsense, and scientific tasks—but also significantly reduces token consumption. For instance, our model achieves scores of 88.1 and 93.3 on AIME2025 and AIME2024, respectively, while using fewer tokens than the current state-of-the-art LLM. Notably, Kwaipilot-AutoThink ranks first among all open-source models on LiveCodeBench Pro[31], a challenging benchmark explicitly designed

¹In this paper, pre-training refers to continued pre-training based on the upscaled Qwen2.5-32B to a 40B scale.

to prevent data leakage, and even surpasses strong proprietary systems such as Seed[32] and o3-mini[33]. Furthermore, Kwaipilot-AutoThink demonstrates robust performance in real-world deployment: it has been integrated into Kwaipilot, Kuaishou’s internal code assistant, where it effectively handles a wide range of development tasks, underscoring the real-world practicality of our thinking control framework and cost-efficient training approach.

In summary, our contributions are fourfold:

- **Comprehensive Data Synthesis with MTP-Enhanced Knowledge Distillation:** We develop a versatile data synthesis framework that integrates both non-reasoning and long-chain reasoning regimes. This is combined with a Multi-Token Prediction (MTP)-enhanced knowledge distillation strategy, enabling fine-grained and efficient transfer of reasoning capabilities without requiring costly full-scale pretraining.
- **AutoThink with Intermediate Supervision:** We introduce AutoThink with Intermediate Supervision, a staged reinforcement learning framework (Step-SRPO) that imposes structured guidance on both reasoning mode selection and final response generation. This approach enhances convergence and strengthens the model’s ability to align its reasoning behavior with task complexity.
- **State-of-the-Art Performance with Efficiency Gains:** Kwaipilot-AutoThink achieves state-of-the-art results across multiple benchmarks. Remarkably, despite having only 40 billion parameters, our model matches or surpasses the performance of leading LLMs with several times more parameters. For example, Kwaipilot-AutoThink scores leads all open-source models and even outperforms proprietary competitors such as Seed and o3-mini on LiveCodeBench Pro. These results demonstrate that our model delivers both competitive performance and high efficiency, underscoring the effectiveness of our training paradigm, reasoning control framework, and the fine-grained data mixture strategy that balances domain diversity and data complexity.
- **Real-World Deployment and Practicality:** Kwaipilot-AutoThink has been successfully deployed in Kwaipilot, Kuaishou’s internal coding assistant. It shows robust performance across diverse software development scenarios, validating the practical value of our adaptive thinking mechanism and cost-effective training strategy in real-world environments.

Together, these contributions advance the frontier of controllable and efficient reasoning in large language models, laying a solid foundation for scalable, deployable, and high-performing AI systems in both academic and industrial settings.

2. Architecture: Upscaling from 32B to 40B

Building on recent studies that explore the relationship between model depth and performance, we scaled the Qwen2.5-32B model to a 40B-parameter variant via a targeted layer-wise expansion strategy. Specifically, we conducted a layer saturation analysis—measuring the cosine similarity between input and output token representations at each transformer layer, following methodologies introduced in [34]—to identify redundant or saturated layers (i.e., layers where representational change is minimal). This diagnostic revealed that approximately one-fourth of the model’s layers exhibited near-identity transformations, suggesting a limited contribution to feature abstraction.

To enhance model capacity while preserving training efficiency, we selectively upscaled these saturated layers by duplication and continued training with diverse, high-quality data spanning both reasoning-intensive and general tasks. In parallel, infrastructure optimizations

revealed that the 40B parameter scale achieves peak Model FLOPs Utilization (MFU) of up to 50%, providing an optimal balance between scale and computational efficiency. The resulting model, **KAT-V1-40B**, demonstrates state-of-the-art zero-shot and few-shot performance among models below the 40B parameter threshold. Detailed model configurations can be found in Table 1.

Table 1 | Model architecture details for Kwaipilot–AutoThink.

| Model | Layers | Tie Embedding | Heads (Q / KV) | Context Length |
|-------------------------|--------|---------------|----------------|----------------|
| Kwaipilot–AutoThink–40B | 80 | No | 40 / 8 | 32K |

3. Pre-training: Knowledge Enhancement

This stage marks the first step toward realizing the AutoThink paradigm, where the model is trained to follow externally provided think-mode instructions, enabling it to alternate between reasoning and non-reasoning behaviors based on input prompts. Specifically, we adopt trigger tokens such as `<think_on>` and `<think_off>` during training to indicate whether reasoning should be activated for each query. These tokens are not involved in loss computation, but serve as effective mode selectors during training.

This mechanism is aligned with prior works such as Llama-Nemotron [35] and Qwen3 [5], which also rely on external cues to control reasoning behavior. Although the model does not yet autonomously determine when to reason, this approach already addresses the overthinking problem observed in fully CoT-activated models by enabling passive switching between reasoning modes. It thus lays the foundation for more advanced capability development in subsequent stages, where dynamic mode selection becomes increasingly model-driven.

3.1. Knowledge Distillation with Multi-Token Prediction

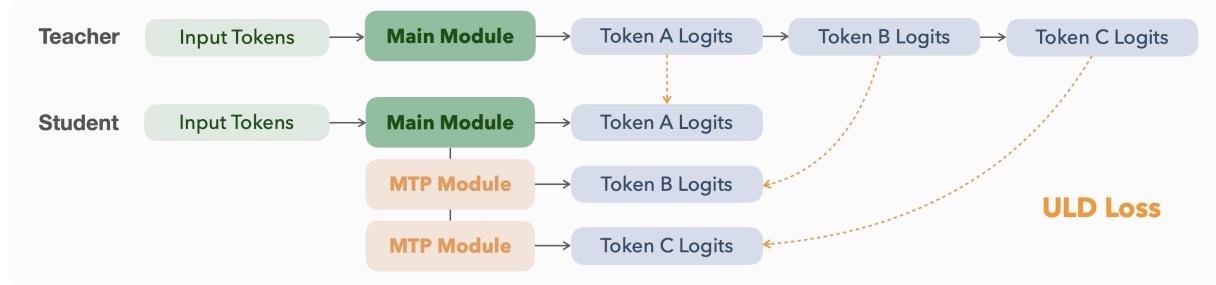


Figure 3 | Illustration of our heterogeneous knowledge distillation framework combining Multi-Token Prediction (MTP) and Universal Logit Distillation Loss (ULD Loss).

Technically, we proposed a heterogeneous distillation framework that combines Multi-Token Prediction (MTP) with a Universal Logit Distillation Loss (ULD Loss) to enhance the effectiveness of knowledge transfer from a larger teacher model to a smaller student model. As illustrated in Figure 3, both the teacher and student models process the same input tokens, while the student model incorporates additional MTP modules to predict future tokens (e.g., Token B, Token C) in parallel with the main decoding pathway. To accommodate the different reasoning requirements

of our dual-regime training data, we employed DeepSeek-V3 and DeepSeek-R1-0528 as teacher models for Think-off (non-reasoning) and Think-on (reasoning) data, respectively. These models were chosen due to their strong open-source performance and extensive community validation, making them well-suited for transferring high-quality knowledge in their respective domains.

Among various intermediate alignment strategies (e.g., embedding-level, MLP activations, LM head outputs), we empirically found that aligning token-level logits yielded the most effective transfer performance. The ULD Loss enables supervision to flow from the teacher’s sequential logits (Token A, B, C) to the student’s heterogeneous MTP modules, even when their prediction architectures differ. This allows for flexible and architecture-agnostic distillation. By predicting multiple future tokens, the MTP modules implicitly model “future rewards” and broaden the supervision space, while ULD Loss bridges the gap between sequential and parallel predictions. This design significantly improves the efficiency of knowledge injection, and enables rapid cold-start initialization with minimal computational overhead.

3.2. Data Construction and Distribution

To effectively enhance the capabilities of KAT-V1-40B, we curate a dual-regime dataset that is logically rich, cross-domain, and sufficiently challenging. This process begins with a novel data labeling system that leverages state-of-the-art LLMs to assess both the difficulty and the domain characteristics of each query. Queries are then classified into Think-off and Think-on categories, based on their intrinsic complexity and the availability of verifiable answers. For Think-off queries, we leverage DeepSeek-V3 to generate responses that do not require multi-step reasoning. Meanwhile, we employ multiple state-of-the-art LLMs to perform reject sampling, thereby ensuring high-quality outputs by selecting only the most accurate and relevant completions. In contrast, Think-on queries are processed through a multi-agent framework, composed of a solver, a thinker, and a critic. The solver provides initial answers, the thinker reflects on and iteratively improves the solution, and the critic supervises the full process to ensure logical consistency and output quality. Only verified outputs from this pipeline are retained and converted into long-CoT data.

Our Stage 1 training corpus comprises approximately 10 million examples, carefully curated to balance reasoning complexity and response efficiency. Among these, 34.8% of the data is annotated as Think-on, while 65.2% is labeled as Think-off, establishing a practical equilibrium between complex reasoning and lightweight generation. As illustrated in Figure 4, the dataset covers a diverse range of domains. This broad domain coverage enables the model to generalize its reasoning capabilities across a wide variety of real-world tasks and instruction types.

3.3. Analysis: Effectiveness of MTP-Enhanced Distillation

To evaluate the impact of our Multi-Token Prediction (MTP)-enhanced knowledge distillation, we compare it against standard knowledge distillation (KD) across several benchmarks. Our results show that integrating MTP with KD leads to approximately a 5 percent improvement in performance across these benchmarks.

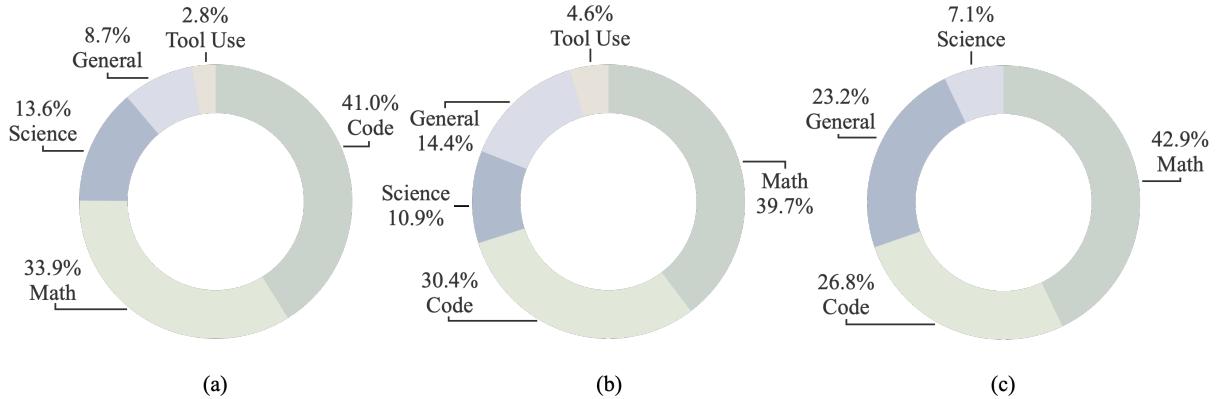


Figure 4 | Domain distributions of training data used in Stage 1 (a), Stage 2 (b), and Stage 3 (c) of the KAT training pipeline.

4. Post-training: Auto-Think

4.1. Cold Start

Building upon the large-scale knowledge enhancement achieved through the upscaled model and the efficient training strategy, we now take a critical step forward: empowering the model to autonomously determine the appropriate reasoning mode based on the input query. While the previous stage enabled passive mode switching via explicit tags, it still relied on externally provided instructions.

To overcome this limitation, we design a cold-start initialization mechanism that serves as the bridge from passive compliance to active decision-making. By introducing intent-aware analysis, majority-vote distillation, and carefully filtered supervision signals, we instill in the model an initial inductive bias to recognize whether a query warrants deep reasoning or a direct response. This transition marks a foundational advance in the AutoThink paradigm—shifting the model’s behavior from externally guided reasoning to internally inferred reasoning mode selection, setting the stage for reinforcement-based fine-tuning in the next phase.

4.1.1. Data Construction

As illustrated in Figure 2, we construct our training data through a structured pipeline designed to explore and supervise the model’s preference between Think-on and Think-off modes. Given a user query, we employ a majority voting mechanism across multiple model outputs to determine the preferred reasoning mode—either Think-on or Think-off. For the selected reasoning path, we sample corresponding responses using two distinct models: DeepSeek-R1 for the Think-On mode and DeepSeek-V3 for the Think-Off mode. This ensures a diverse and task-aligned generation process.

To enhance the model’s understanding of reasoning mode selection, we introduce an auxiliary evaluation step. Specifically, for each query-response pair, we use DeepSeek-V3 to generate a judgment rationale explaining the appropriateness of the selected mode. This step provides the training signal necessary for learning the alignment between query complexity and reasoning depth. Furthermore, to ensure coverage and avoid overfitting to biased query distributions, 1% of the queries are randomly assigned a mode (regardless of majority voting), enforcing the model’s exposure to diverse reasoning scenarios.

The final training samples are formatted in a unified structure that encapsulates both the judgment and the answer generation process. We define two templates as shown in Table 2 and Table 3.

This format enables the model to jointly learn when to engage in deeper reasoning and how to perform reasoning-aware answer generation. The explicit separation of reasoning analysis and final answer encourages structured alignment between mode selection, justification, and response quality.

Table 2 | Formatting templates used in the final training data. The Think-On and Think-Off modes follow a unified structure comprising judgment and answer segment.

| Think-On Mode | Think-Off Mode |
|---|--|
| <judge> {judge_analysis} </judge> | <judge> {judge_analysis} </judge> |
| <think_on> <think> {thinking_content} </think> | <think_off> <answer> {response} </answer> |
| <answer> {response} </answer> | |

Table 3 | Special tokens and their descriptions.

| Special Token | Description |
|----------------|--|
| <judge> | Analyzes input query to determine whether reasoning is required. |
| <think_on/off> | Specifies whether reasoning should be activated ("on") or skipped ("off"). |
| <think> | Marks the beginning of reasoning in "think-on" mode. |
| <answer> | Marks the beginning of the model's answer. |

4.1.2. Data Distribution

The composition of our dataset spans six distinct categories: **code, math, science, general, multi-turn, and tool use**. As illustrated in Figure 4, this distribution ensures broad coverage across both reasoning-intensive and general-purpose tasks, enabling the model to learn robust mode selection strategies under varying task complexities.

Code, Math, and Science We constructed a challenging question corpus from diverse public and proprietary sources, applying rigorous selection criteria to ensure broad coverage, disciplinary diversity, and balanced difficulty levels. Samples with verifiable results or test cases were prioritized. For mathematical and scientific questions meeting inclusion criteria but lacking definitive answers, we annotated soft labels using robust open-source models via majority voting. Difficulty levels were annotated granularly, and the dataset distribution was calibrated through stratified sampling. Excessively difficult questions were oversampled, with multiple responses collected per instance. Finally, response validity underwent multi-step verification, including rule-based validation, model-based response comparison, and sandboxed execution.

General and Multi-turn Dialogue We collected a diverse corpus of instructions spanning a broad spectrum of topics. Each instruction was annotated with multi-dimensional labels and difficulty ratings, followed by clustering and filtering. Particular attention was paid to balancing instances involving reasoning mode transitions within multi-turn dialogues. To preserve the model’s inherent preferences for reasoning patterns, we employed a prompt-based approach leveraging intermediate checkpoints to collect model responses via rejection sampling. For each query, responses underwent human preference evaluation using a reward model across multiple dimensions, retaining the optimal response.

Tool Use To enhance the model’s capability in agent-based scenarios, we specifically collected data related to tool use, task planning, and reflective reasoning. We gathered diverse tool sets and multi-turn interaction trajectories from both public and internal sources, covering a wide range of scenarios. Targeting real-world application settings, we further collect interaction trajectories in software development contexts via multi-agent systems. Furthermore, we optimized the model’s tool calling scheme for end-to-end agent operation, which requires it to perform thorough reasoning and provide explanations before executing any calls.

In total, the Stage 2 training corpus comprises approximately 3.5 million examples, spanning the aforementioned categories, with the proportion of Think-on to Think-off examples approximately 2:1. As illustrated in Figure 4, the distribution maintains a balanced coverage across domains such as code, math, general reasoning, multi-turn dialogue, and tool-use trajectories, ensuring that the model is exposed to both structured reasoning and open-ended interaction patterns.

4.2. Step-SRPO

Typically, post-training of large language models (LLMs) involves separate reinforcement learning (RL) stages targeting different, often mutually exclusive, objectives. This sequential strategy may lead models to oscillate between conflicting objectives, causing instability similar to a “seesaw” effect. To address this challenge, we propose the **Step-SRPO** framework (the detailed AutoThink RL implementation will be released in a forthcoming companion paper), which unifies multiple reinforcement learning objectives into a single, coherent training session. Specifically, Step-SRPO aims to simultaneously achieve three interrelated goals.

Thinking Control. Although KAT acquires initial automatic reasoning abilities from the AutoThink cold-start training (Stage 2), these capabilities remain unstable, influenced by variables such as query domain and length. Therefore, subsequent reinforcement learning is necessary to stabilize and refine the model’s capability to autonomously activate explicit reasoning (“think-on”) for complex queries and deactivate it (“think-off”) for simpler tasks.

Reasoning RL. While we achieved strong initial performance during the cold-start phase, standalone reasoning-based RL training, particularly with challenging query-verifier pairs, effectively improves the model’s reasoning capability. However, this singular approach tends to bias the model towards producing answers predominantly in the think-on mode to maximize reward signals, thus negatively impacting its AutoThink gating ability. Incorporating reasoning-focused RL into the Step-SRPO structure mitigates this bias, ensuring balanced reasoning activation based on query complexity.

General RL. General RL aims to broadly enhance the model’s robustness and reliability across diverse scenarios through reward signals covering multiple tasks. Yet, applying general RL alone may degrade KAT’s AutoThink discriminative performance, necessitating integration within the Step-SRPO framework to maintain a balanced AutoThink capability.

4.2.1. Algorithm

Unlike conventional language models that apply uniform reasoning depth across all queries, Step SRPO introduces a "reasoning necessity assessment" phase. This allows models to determine whether deep reasoning is required before generating responses, achieving a dynamic balance between computational efficiency and output quality. The Step-SRPO reward structure comprises two interrelated components:

Judge Reward Calculated based on the overall correctness of the model's reasoning activation decisions, reinforcing accurate gate predictions across the entire query-response cycle.

Answer Reward Primarily determined by the correctness and quality of the final answer itself, further modulated by the Judge Reward. This ensures that the answer quality remains consistently aligned with the effectiveness of the reasoning decision.

This structured reward mechanism directs the model toward contextually appropriate reasoning utilization, balancing computational efficiency with response accuracy.

4.2.2. Reward

Math Responses are evaluated by symbolic equivalence checks (algebraic simplification and numeric approximation). Correct answers receive full reward; incorrect or absent solutions score zero.

Code Responses labeled as code are evaluated by running predefined unit tests. Passing all tests yields maximal reward; any test failure yields zero.

Science For multiple-choice questions, we extract the first letter of the submitted option and compare against the reference key. A match scores one; a mismatch scores zero.

General Use lightweight heuristics (keyword overlap, simple classifier) to assign a binary correct/incorrect signal.

4.2.3. Data Distribution

Maintaining an appropriate distribution of data difficulty is critical to the effectiveness of Step-SRPO in reinforcing the model's AutoThink capability. If the query-verifier pairs are too simple or too difficult, the model may fall into undesirable local optima—such as consistently avoiding reasoning or over-engaging in unnecessary reasoning. Conversely, a dataset consisting solely of moderately difficult instances may lead to unstable reasoning-mode preferences, preventing the model from learning robust decision boundaries.

To ensure robust learning dynamics, we constructed a dataset with broadly distributed difficulty levels, reflecting a spectrum from low to high complexity, while biasing toward harder examples that require nuanced reasoning. The difficulty distribution of this 45K-sample training corpus is illustrated in Figure 2, which shows a deliberate concentration in the high-difficulty region while retaining a non-negligible proportion of easier examples for stabilization. In addition to difficulty, domain coverage also plays a pivotal role in shaping AutoThink behavior. During reasoning-focused reinforcement learning, data is often skewed toward math and code, which risks overfitting the model to domain-specific reasoning shortcuts or "hacks." To mitigate this, we curated a dataset spanning math, code, and general-purpose reasoning queries. Each query-verifier pair was assigned a difficulty score based on the model's current capability, and the final dataset maintains a broad difficulty distribution with a bias toward higher-difficulty samples. A

small portion of lower-difficulty examples was also included to support general reinforcement learning objectives and stabilize training dynamics. As shown in Figure 4, this dataset maintains domain diversity, preventing over-specialization and encouraging generalizable AutoThink behavior.

4.3. Analysis

To rigorously evaluate the effectiveness of AutoThink reasoning control, we conduct a comprehensive analysis of both training and inference behaviors. Specifically, we focus on two key dimensions: reasoning mode activation (think-on vs. think-off) and token efficiency across training steps and diverse benchmark tasks.

During training, we examine how the model’s decision-making evolves under the Step-SRPO framework, tracking the frequency of reasoning-mode activations and the associated output length. These metrics provide insight into how the model learns to balance reasoning depth with efficiency. During inference, we further investigate how reasoning control generalizes across domains, highlighting how reasoning-intensive benchmarks sustain high think-on usage, while others benefit from adaptive reasoning suppression. Collectively, these trends reveal how KAT progressively develops fine-grained, context-sensitive reasoning behavior, delivering strong performance with reduced computational overhead.

4.3.1. Think-On vs. Think-Off and Token Count Dynamics During Training

We logged how often the model chose `<think_on>` versus `<think_off>` at each step. This shift—visualized in Figure 5—confirms that Step-SRPO not only improves final accuracy but also refines the model’s gating behavior, enabling it to skip heavy reasoning when unnecessary.

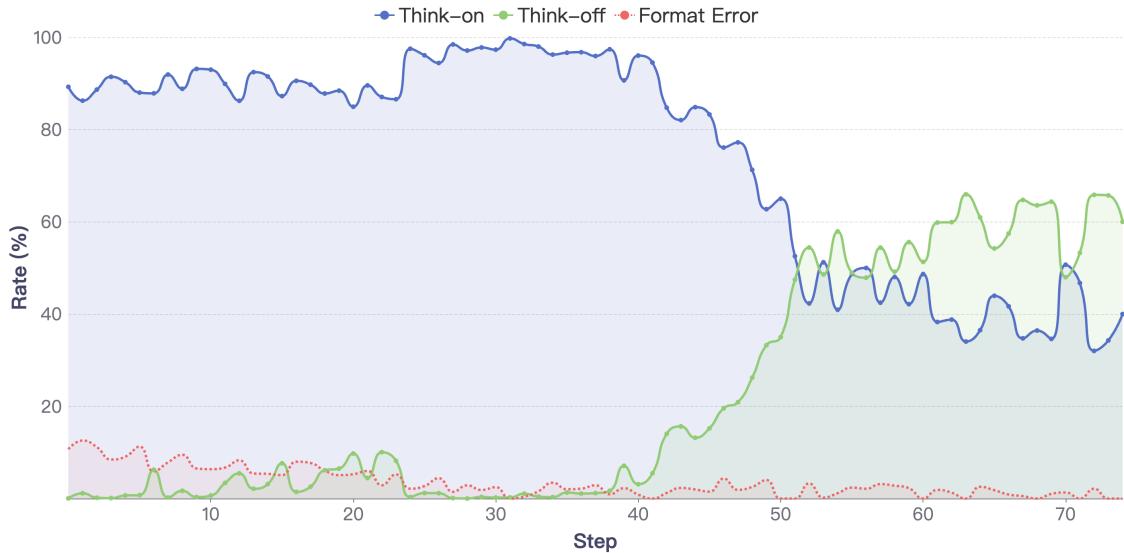


Figure 5 | **Training dynamics of reasoning-mode selection under Step-SRPO.** We track the model’s preference for `<think_on>` (blue) versus `<think_off>` (green) across training steps.

During AutoThink RL training, the average output token count per prediction exhibits a clear downward trend (see Figure 6). This reduction indicates that the model progressively learns to generate more concise responses, selectively invoking explicit reasoning only when necessary.

Such behavior demonstrates the effectiveness of the Step-SRPO reward design in promoting efficient use of chain-of-thought and controlling response length across diverse queries.

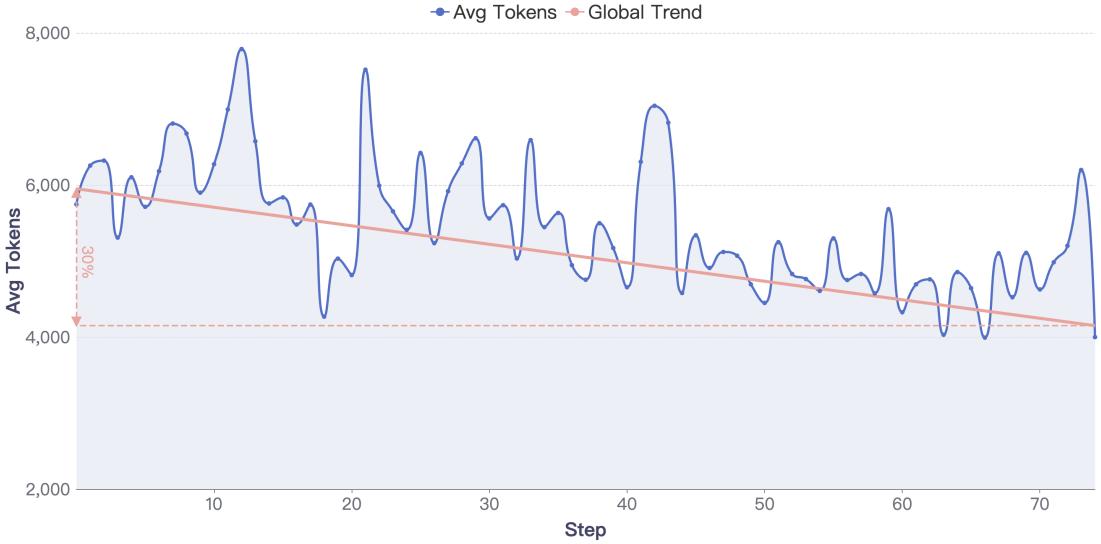


Figure 6 | **Token usage trend during reinforcement learning in Step-SRPO.** During training, the average output token count steadily declines.

4.3.2. Think-On vs. Think-Off and Token Count Dynamics During Inference

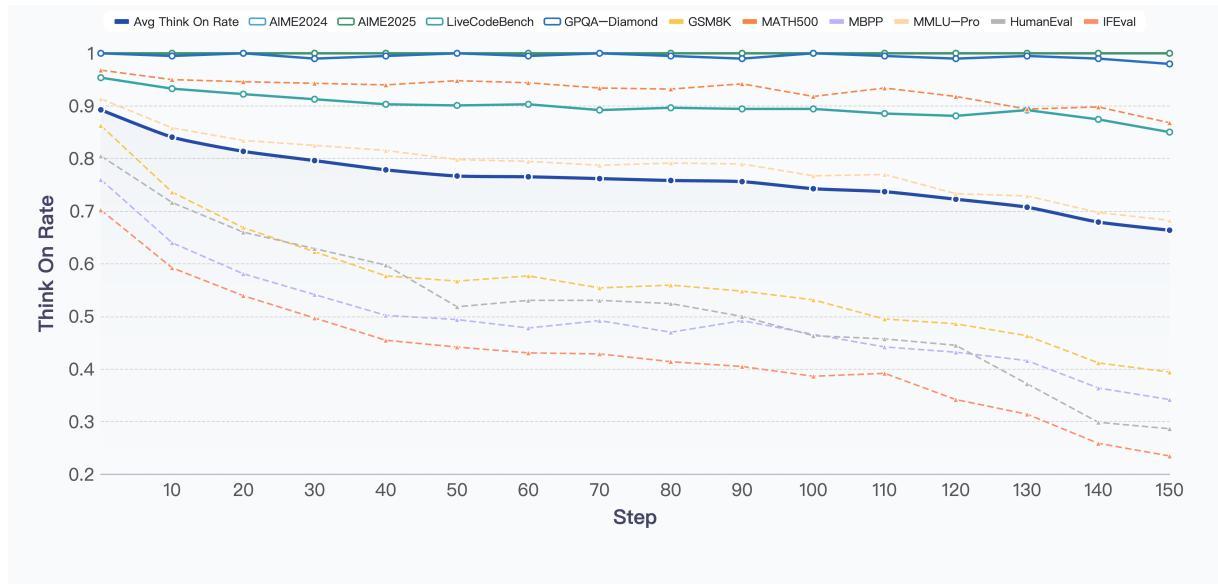


Figure 7 | Think-on rate dynamics during inference across 10 benchmark tasks.

We systematically evaluated the evolution of reasoning control (think-on versus think-off activation) and token efficiency across multiple benchmark tasks. Figure 7 shows the proportion of think-on activations across different evaluation datasets during inference. Reasoning-intensive tasks, including AIME2024, AIME2025, GPQA-Diamond, and LiveCodeBench, consistently demonstrate high think-on activation rates (>80%) throughout training. Conversely, tasks that

require less explicit reasoning—such as GSM8K, MBPP, HumanEval, CEval, and IFEval—exhibit a clear downward trend in think-on activation as training progresses. Across all datasets, the average think-on rate decreased significantly from approximately 90% to 66%, representing a relative reduction of about 24% and highlighting the model’s enhanced capability for selectively invoking reasoning.

Consistent with reasoning mode trends, Figure 8 illustrates the corresponding dynamics in average token counts per generated response during inference. Datasets with consistently high think-on activation maintain higher average token counts, indicative of sustained explicit reasoning. In contrast, datasets with reduced think-on activation show significant decreases in token usage over the course of training. Specifically, the average number of generated tokens across all datasets decreased from an initial 9,887 to 8,037 tokens , reflecting a relative reduction of approximately 19% and underscoring improvements in inference efficiency achieved through adaptive reasoning control.

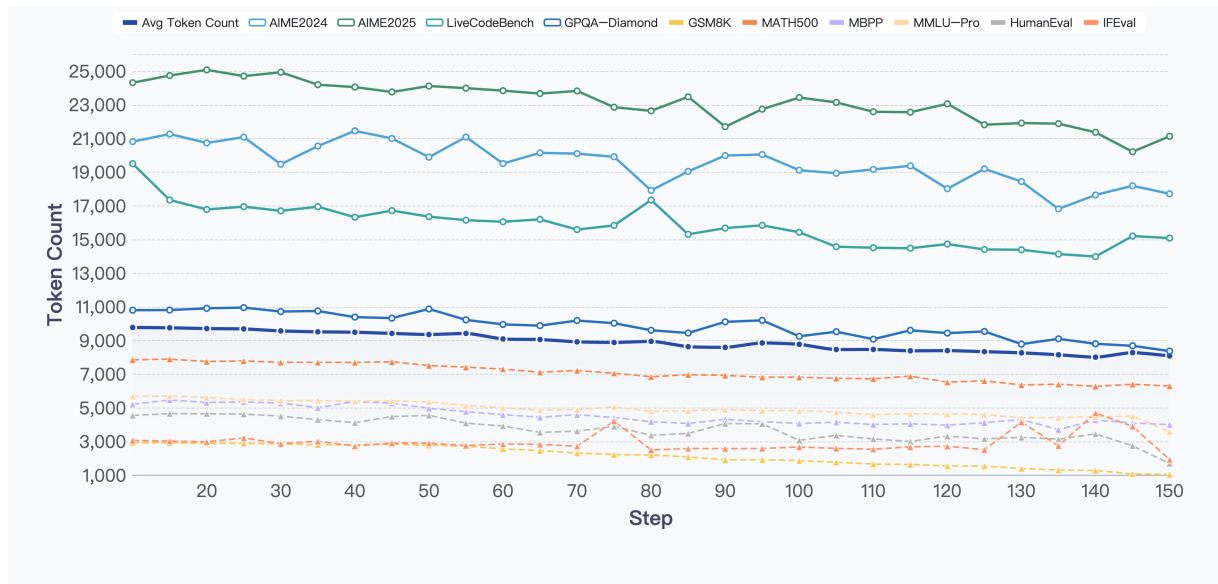


Figure 8 | Token Count Dynamics during inference across 10 benchmark tasks.

5. Results

To evaluate the effectiveness of KAT, we compared our model with several leading open-source state-of-the-art (SOTA) models, including LLaMA-4-Scout, LLaMA-4-Maverick, Qwen3-32B, Qwen3-235B-A22B, DeepSeek-V3, and DeepSeek-R1-0528 across a variety of benchmarks. As shown in Table 4, KAT consistently matches or outperforms these models in terms of accuracy across tasks in general reasoning, math and text reasoning, as well as agent and coding tasks.

In addition to competitive accuracy, KAT demonstrates improvements in token efficiency. As shown in Figure 9, across various benchmarks, token savings are consistently achieved. These improvements are enabled by KAT’s dynamic reasoning control, which selectively activates reasoning only when necessary, significantly reducing token usage in simpler queries. The model adapts its reasoning depth based on task complexity, resulting in fewer tokens consumed without sacrificing response quality.

In summary, Kwaipilot-AutoThink offers both high performance and enhanced efficiency, driven by a strong reasoning control strategy and an adaptive approach to token usage. This

makes KAT a resource-efficient solution for real-world applications, as evidenced by its integration into Kwaipilot, Kuaishou’s internal coding assistant.

Table 4 | Comparison of KAT and leading SOTA models across diverse benchmark tasks. The numbers, enclosed in a **boxed**, **bolded**, and underlined, respectively indicate the top three performance rankings of the models on the current benchmark.

| | KAT -V1-40B | KAT -V1-200B | DeepSeek -R1-0528 | DeepSeek -V3 | Qwen3 -235B-A22B | Qwen3 -32B | LLaMA-4 -Maverick | LLaMA-4 -Scout |
|----------------------------------|----------------|-----------------|----------------------|-----------------|---------------------|---------------|----------------------|-------------------|
| Architecture | Dense | MoE | MoE | MoE | MoE | Dense | MoE | MoE |
| # Total Params | 40B | 200B | 671B | 671B | 235B | 32B | 400B | 109B |
| # Act Params | 40B | 40B | 37B | 37B | 22B | 32B | 17B | 17B |
| <i>General Tasks</i> | | | | | | | | |
| MMLU-Pro[36] | 77.8 | 82.3 | 81.9 | 81.2 | 80.8 | 76.2 | 80.5 | 74.3 |
| DROP[37] | 91.2 | 91.6 | 92.2 | 90.5 | 91.9 | 91.1 | 90.7 | 89.2 |
| WildBench[38] | 8.73 | 8.77 | 8.80 | 8.74 | 8.72 | 8.56 | 8.29 | 8.21 |
| GPQA-D[39] | <u>75.1</u> | <u>78.2</u> | <u>81.0</u> | 68.4 | 71.1 | 68.4 | 69.8 | 57.2 |
| <i>Math & Text Reasoning</i> | | | | | | | | |
| MATH500[40] | 97.4 | 97.6 | 97.3 | 94.0 | 98.0 | 97.2 | 90.6 | 82.6 |
| AIME2024[41] | 93.3 | 96.3 | 91.4 | 59.4 | 85.7 | 81.4 | 38.5 | 28.6 |
| AIME2025[42] | 88.1 | 90.4 | 87.5 | 44.2 | 81.5 | 72.9 | 15.9 | 10.0 |
| AutoLogi[43] | 82.1 | <u>84.9</u> | 84.3 | 76.1 | 89.0 | 87.3 | 75.2 | 56.8 |
| <i>Agent & Coding Tasks</i> | | | | | | | | |
| HumanEval[44] | 95.1 | 98.2 | 97.0 | 93.3 | 30.5 | 90.2 | 88.4 | 83.5 |
| MBPP[45] | 85.2 | 93.2 | 96.2 | 83.6 | 65.4 | 74.6 | 81.0 | 72.0 |
| LiveCodeBench[46] | 73.1 | 77.1 | 73.3 | 44.7 | 59.1 | 61.5 | 34.1 | 30.2 |
| LCB-Pro-Med[31] | 12.7 | N/A | 7.0 | 0.0 | N/A | N/A | 0.0 | N/A |
| BFCL-V3-Mul[47] | 37.4 | 41.6 | 38.9 | 29.9 | <u>40.1</u> | 43.1 | 16.4 | 1.9 |

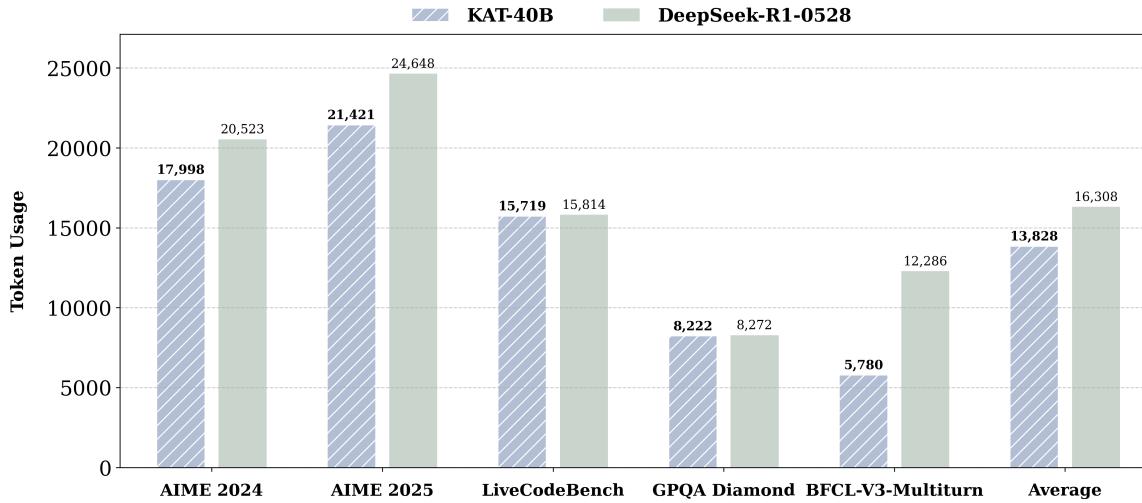


Figure 9 | Comparison of Token Usage between KAT-V1-40B and DeepSeek-R1-0528.

6. Work with KwaiPilot: Real-World Applications

Beyond strong benchmark performance, KAT-V1-40B has also demonstrated its practical value through deployment in Kwaipilot, Kuaishou’s internal coding assistant. To validate the

real-world applicability of our thinking control mechanism and training pipeline, we conducted a series of controlled product-level evaluations.

6.1. Avoiding Overthinking in Simple Queries

| User Query | KAT-V1 | | | | | | | | | | | | |
|---|--|-------------|---------------------------|--------------------------|-------------------------|------------------------|-----------------|---|-------------------------------|----------------------|----------------------------------|----------------------------|--|
| How to create a virtual env in conda? | KAT-V1 | | | | | | | | | | | | |
| Other SOTA Reasoning Model | Judge | | | | | | | | | | | | |
| Thoughts [379 Tokens in Extra 17 Seconds] | <ul style="list-style-type: none"> Request procedural knowledge, requiring only the listing of Conda environment creation commands. Answer without in-depth analysis, use THINK-OFF mode. | | | | | | | | | | | | |
| We need to use Conda to create a new virtual environment. Typically, ... [omitted for brevity...] Therefore, we provide the following code: | To create a virtual environment in Anaconda (Conda) , follow these steps: 1. Open Anaconda Prompt (Windows) or Terminal (macOS/Linux) Ensure Anaconda is installed and added to your system's PATH during installation. 2. Create a New Environment Use the <code>conda create</code> command with the desired environment name and Python version: Command <code>conda create -n myenv python=3.9</code> | | | | | | | | | | | | |
| Answer To create a new virtual environment using Conda, follow these steps: 1. Basic creation command Command <code>conda create --name your_env_name python=x.x</code> Replace <code>your_env_name</code> with your environment name (e.g., <code>myenv</code>). Replace <code>x.x</code> with the Python version (e.g., <code>3.9</code>). 2. Complete operation example Command <code># Create an environment named "ml_project" with Python 3.9 conda create --name ml_project python=3.9</code> <code># Activate the environment conda activate ml_project # Common for Windows/Linux/macOS</code> <code># Verify the environment (check Python path) which python # Linux/macOS where python # Windows</code> | 3. Confirm Installation Type <code>y</code> and press Enter to proceed. 4. Activate the Environment • Windows: Command <code>conda activate myenv</code> • macOS/Linux: Command <code>source activate myenv # For older Conda versions conda activate myenv # For newer Conda versions</code> | | | | | | | | | | | | |
| 3. Common options <table border="1"> <thead> <tr> <th>Option</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><code>--name or -n</code></td> <td>Specify environment name</td> </tr> <tr> <td><code>python=x.x</code></td> <td>Specify Python version</td> </tr> <tr> <td><code>-y</code></td> <td>Skip confirmation prompt (automatic installation)</td> </tr> <tr> <td><code>cudatoolkit=11.3</code></td> <td>Install CUDA toolkit</td> </tr> <tr> <td><code>pytorch torchvision</code></td> <td>Install specified packages</td> </tr> </tbody> </table> | Option | Description | <code>--name or -n</code> | Specify environment name | <code>python=x.x</code> | Specify Python version | <code>-y</code> | Skip confirmation prompt (automatic installation) | <code>cudatoolkit=11.3</code> | Install CUDA toolkit | <code>pytorch torchvision</code> | Install specified packages | 5. Verify Activation Your terminal prompt will show (<code>myenv</code>), indicating the environment is active. Additional Options <ul style="list-style-type: none"> Specify Packages: Install packages during environment creation: Command <code>conda create -n myenv python=3.9 numpy pandas</code> Use YAML File: Save environment configuration for reproducibility: Command <code>conda env export > environment.yml conda env create -f environment.yml</code> |
| Option | Description | | | | | | | | | | | | |
| <code>--name or -n</code> | Specify environment name | | | | | | | | | | | | |
| <code>python=x.x</code> | Specify Python version | | | | | | | | | | | | |
| <code>-y</code> | Skip confirmation prompt (automatic installation) | | | | | | | | | | | | |
| <code>cudatoolkit=11.3</code> | Install CUDA toolkit | | | | | | | | | | | | |
| <code>pytorch torchvision</code> | Install specified packages | | | | | | | | | | | | |
| 4. Create an environment with specific packages Command <code>conda create -n dl_env python=3.10 numpy pandas scikit-learn</code> | Deactivate the Environment Command <code>conda deactivate</code> | | | | | | | | | | | | |
| 5. Manage environments Command <code># List all environments conda env list</code> <code># Export environment configuration conda env export > environment.yml</code> <code># Delete environment conda env remove -n your_env_name</code> | Delete an Environment Command <code>conda env remove -n myenv</code> | | | | | | | | | | | | |
| Notes: <ul style="list-style-type: none"> Install Anaconda or Miniconda for the first use Use Anaconda Prompt on Windows to execute commands After creating the environment, it must be activated with <code>conda activate env_name</code> to use For example: Create an environment for PyTorch Command <code>conda create -n pytorch_env python=3.9 pytorch torchvision cudatoolkit=11.3 -c pytorch</code> | Troubleshooting <ul style="list-style-type: none"> If <code>conda</code> isn't recognized, reinstall Anaconda or run <code>conda init</code> to fix path issues. Use <code>conda update -n base -c defaults</code> to update Conda if commands fail. Let me know if you need further clarification! 😊 | | | | | | | | | | | | |

Figure 10 | Comparison between KAT-V1-40B and a state-of-the-art reasoning model on a simple procedural query.

As shown in Figure 10, when handling a query that does not require explicit reasoning, the state-of-the-art reasoning model still engages in unnecessary step-by-step analysis, generating

a verbose response of nearly 400 tokens and incurring an additional 17 seconds of latency. In contrast, KAT-V1-40B correctly recognizes the simplicity of the task, promptly activates the think-off mode, and produces a high-quality response.

This example underscores KAT-V1-40B's ability to avoid overthinking in low-complexity scenarios, significantly improving response efficiency while reducing token consumption. Moreover, thanks to the multi-agent generated reasoning data introduced during the knowledge enhancement stage, KAT-V1-40B exhibits strong interaction capabilities and a fine-grained understanding of when reasoning is contextually appropriate, further reinforcing its practical value in real-world deployments.

6.2. Reasoning Mode for Complex Query

User Query

How to optimize this slowly executing SQL query?

SQL Query

```
SELECT task_id FROM kvq_tool_task WHERE task_status=0 AND tenant='cape' LIMIT 3000;
```

KAT-V1

Judge

- Task:** Find bottleneck and fix slow SQL, Need **THINK-ON** mode.
- Steps:** Index/query/config (Idx/Qry/Cfg), monitoring, step-by-step.

Thoughts

- Idx:** Check if task_status, tenant are indexed. task_id for covering index.
- Plan:** Use EXPLAIN for plan.
- Dist:** Check filter cardinality.
- Other:** Consider ORDER BY task_id for LIMIT.
- IdxRes:** Composite index on (task_status, tenant, task_id) is best.

Answer

Composite Index

SQL Statements

```
CREATE INDEX idx_task_status_tenant_task_id ON kvq_tool_task (task_status, tenant, task_id);
EXPLAIN SELECT task_id FROM kvq_tool_task WHERE task_status=0 AND tenant='cape' ORDER BY task_id LIMIT 3000;
```

Check Existing Indexes

- Show indexes: SHOW INDEXES FROM kvq_tool_task;
- Use EXPLAIN to check index usage and bottlenecks.

Other Suggestions

- Dist: Check selectivity of task_status=0 and tenant='cape'.
- Rewrite: Avoid LIKE '%...' and ensure type consistency.
- Order: Composite index order matters.
- Validate: Use EXPLAIN and measure performance before/after.

Other Options

SQL Statements

```
CREATE INDEX idx_task_status ON kvq_tool_task (task_status);
CREATE INDEX idx_tenant ON kvq_tool_task (tenant);
```

- Pros:** Easy to manage, good for single filter.
- Cons:** May scan more rows, no index-only scan for task_id.

Data & Query

SQL Statements

```
ANALYZE TABLE kvq_tool_task;
CREATE MATERIALIZED VIEW mv_task_status_cape AS SELECT task_id FROM kvq_tool_task WHERE task_status=0 AND tenant='cape';
```

- Partition by tenant or task_status for huge tables.
- Tune DB config, use replicas, validate/rebuild indexes.

Figure 11 | Comparison between KAT-V1-40B and a state-of-the-art reasoning model on a complex SQL performance tuning task.

When faced with a more complex scenario involving SQL performance tuning (Figure 11), KAT-V1-40B automatically engages its reasoning mode (think-on). It delivers a structured, multi-step analysis within just 15 seconds of thinking, while another reasoning model we host took 53 seconds. The response is rated on par with the state-of-the-art reasoning model in correctness, but outperforms it in depth, architectural insight, and scalability suggestions. This

demonstrates KAT-V1-40B's capacity to adjust its reasoning depth dynamically and efficiently in real-world engineering tasks.

6.3. End-to-End Mode Switching with Multi-Turn Capability

Problem Statement

The `fowlkes_mallows_score` function produces a `RuntimeWarning` and returns `nan` when the product of `pk` and `qk` exceeds the `int32` limit (`2**32`):

```
sklearn\metrics\cluster\supervised.py:859: RuntimeWarning: overflow encountered in int_scalars
    return tk / np.sqrt(pk * qk) if tk != 0. else 0.
```

This occurs because the calculation `pk * qk` uses 32-bit integers, which can overflow when the values are sufficiently large. The overflow causes the result to be invalid, leading to `nan` being returned instead of the expected float value.

Steps to Reproduce:

1. Compute the Fowlkes-Mallows Index where `pk` and `qk` are large enough such that `pk * qk > 2**32`
2. Observe a `RuntimeWarning` and `nan` output

Expected Behavior:

The function should compute the Fowlkes-Mallows Index as:

$$FMI = \frac{tk}{\sqrt{(pk * qk)}}$$

and return a valid float value without warnings, even for large values of `pk` and `qk`.

Actual Behavior:

The function returns `nan` and raises a `RuntimeWarning` due to integer overflow in the calculation.

Agent Trajectory

Step 1: Problem Diagnosis Think: OFF

Judge The query involves a direct implementation change based on a bug report.

Answer Identified an **overflow issue** in `fowlkes_mallows_score()` **due to large inputs**. Proposed fix: replace `tk / np.sqrt(pk * qk)` with `np.sqrt(tk / pk) * np.sqrt(qk / pk)` to **avoid overflow**.

Action View file
`cat -n /testbed/sklearn/metrics/cluster/supervised.py`

Step 2: Fix Implementation Think: ON

Judge Requires think-on mode to analyze the codebase and implement the fix.

Answer Located the **problematic line**. Confirmed **mathematical equivalence** and suggested replacement:

```
return np.sqrt(tk / pk) * np.sqrt(qk / pk) if tk != 0. else 0.
```

Action Replace line
`str_replace_editor /testbed/sklearn/metrics/cluster/supervised.py`

Step 3: Change Verification Think: OFF

Judge Requires think-off mode to confirm the changes and verify the fix.

Answer Confirmed the **fix was correctly implemented**. Suggested running a **test case** for verification.

Action Submit
`diff --git a/supervised.py b/supervised.py`

```
- return tk / np.sqrt(pk * qk) if tk != 0. else 0.
+ return np.sqrt(tk / pk) * np.sqrt(qk / pk) if tk != 0. else 0.
```

Step 4: Final Submission Think: OFF

Judge Requires think-off mode to confirm the final version.

Answer Displayed the **final diff** of changes and confirmed **successful submission**.

Action Submit
`@@ def fowlkes_mallows_score(...):
- return tk / np.sqrt(pk * qk) if tk != 0. else 0.
+ return np.sqrt(tk / pk) * np.sqrt(qk / pk) if tk != 0. else 0.`

Figure 12 | Example of KAT-V1-40B executing a full multi-turn agent trajectory while dynamically switching between reasoning modes.

As shown in Figure 12, KAT-V1-40B demonstrates its end-to-end controllability over reasoning mode, accurately alternating between think-off and think-on across different steps in a multi-turn agent trajectory. It passively disables reasoning during file inspection (think-off) and actively enables deep reasoning and tool-based exploration (think-on) when diagnosis or code generation is required. The agent exhibits tool-using proficiency, context retention, and accurate task decomposition across multiple interaction rounds, validating KAT-V1-40B's ability to autonomously manage complex task flows in production settings.

6.4. User-Guided Mode Switching

In addition to autonomous reasoning control, KAT-V1-40B also supports user-guided control over reasoning mode, allowing users to flexibly override the model's default behavior. By embedding simple intent directives, such as explicit reasoning instructions or non-reasoning preferences, into the input query, users can directly influence whether the model engages in think-on or think-off mode.

As illustrated in Figure 13, on the left, the user enforces the non-reasoning mode with the

| | |
|--|---|
| Query | Query |
| <p>If 3 men can dig a 3 meters ditch in 3 days, then how many meters can 9 men dig in 9 days? DO NOT THINK!!!</p> | <p>Think deeper, how many 'i' in the word 'Kwaipilot'?</p> |
| User-enforced Think-off | User-enforced Think-on |
| <p> Judge</p> <p>Straightforwardly, the query asked for a direct calculation without reasoning. The core task is to compute the result based on the given ratio. Simply multiply the initial rate by the new variables to find the answer.</p> | <p> Judge</p> <p>It asked for deep thinking, indicating a reasoning mode. The core task is to count occurrences of 'i' in the word 'Kwaipilot', requiring logical analysis to identify and tally the letters.</p> |
| <p> Answer</p> <p>To solve this problem, we need to determine the rate at which men dig the ditch and then apply this rate to the new scenario.</p> <p>First, let's determine the rate at which one man digs the ditch. Given that 3 men can dig a 3-meter ditch in 3 days, we can calculate the digging rate per man per day:</p> <ul style="list-style-type: none"> • 3 men dig 3 meters in 3 days. • Therefore, 1 man digs $\frac{3 \text{ meters}}{3 \text{ men} \times 3 \text{ days}} = \frac{3}{9} = \frac{1}{3}$ meters per day. <p>Now, we want to find out how many meters 9 men can dig in 9 days. Using the rate of one man per day:</p> <ul style="list-style-type: none"> • 1 man digs $\frac{1}{3}$ meters per day. • Therefore, 9 men will dig $9 \times \frac{1}{3} = 3$ meters per day. • Over 9 days, 9 men will dig 3 meters/day \times 9 days = 27 meters. <p>Thus, 9 men can dig 27 meters in 9 days.</p> | <p> Thought [Key content retained for brevity.]</p> <p>We are counting the letter 'i' in the word 'Kwaipilot'. The word is: Kwaipilot. Let's break it down: <code>K w a i p i l o t</code></p> <p>We can see two 'i's: one at the 4th position and one at the 6th position. Steps: 1. Convert the word to lowercase. 2. Count the occurrences of 'i'. Original: "Kwaipilot" Convert to lowercase: "kwaipilot" Count the occurrences: <code>k, w, a, i, p, i, l, o, t -> two 'i's</code></p> <p>Answer: 2</p> <p> Answer</p> <p>The word "Kwaipilot" contains the letter 'i' at two positions:</p> <ol style="list-style-type: none"> 1. Position 4: After "Kwa", the fourth character is 'i' (forming "Kwai"). 2. Position 6: After "Kwaip", the sixth character is another 'i' (forming "Kwaipi"). <p>Thus, there are two 'i's in the word.</p> <p>Verification:</p> <ul style="list-style-type: none"> • Breaking the word: K-w-a-i-p-i-l-o-t • Both occurrences are lowercase 'i', and there are no uppercase 'I's in the given word. <p>Conclusion: The count is 2.</p> |

Figure 13 | Examples of user-guided mode switching in KAT-V1-40B.

directive "DO NOT THINK," leading KAT-V1-40B to skip reasoning steps and directly compute the answer. On the right, the instruction "Think deeper" prompts the model to engage in step-by-step reasoning. These examples demonstrate KAT-V1-40B's ability to interpret and follow user-specified intent for reasoning depth, enabling flexible and controllable interaction.

This user-in-the-loop controllability further enhances KAT-V1-40B's practicality in real-world systems, enabling flexible trade-offs between efficiency and interpretability based on task requirements or user preferences. It also provides a natural interface for integrating KAT-V1-40B into interactive workflows, where explicit control over reasoning depth is desirable.

6.5. Insights from Real-World Use Cases

These case studies collectively demonstrate that Kwai-AutoThink is not only competitive with state-of-the-art models on standard benchmarks but also excels in real-world production environments. Through comprehensive evaluations across diverse coding tasks—including but not limited to SQL optimization and agent-style multi-turn interactions—KAT-V1-40B has shown the ability to dynamically select appropriate reasoning modes based on task complexity, achieving strong performance in both response quality and efficiency.

In addition to its autonomous reasoning control, KAT-V1-40B supports user-guided thinking mode switching via prompt-level intent specification, giving users explicit control over reasoning depth when needed. This dual-mode flexibility, adaptive when desired and controllable when required, further enhances its utility in practical deployments. In general, these findings highlight the deployability, efficiency, and controllability of our thinking control framework, validating KAT-V1-40B not just as a research prototype, but as a solid foundation for building scalable, high-performance, and production-ready reasoning systems.

7. Conclusion

In this work, we present Kwaipilot-AutoThink, a 40B open-source large language model designed to address the overthinking problem in reasoning-intensive tasks. We propose a novel automatic thinking training framework that enables dynamic switching between reasoning and non-reasoning modes, guided by task complexity. Our approach combines a diverse dual-regime data synthesis pipeline, MTP-enhanced knowledge distillation, and Step-SRPO, a staged reinforcement learning algorithm with intermediate supervision. Together, these components allow the model to learn fine-grained reasoning behavior with minimal pretraining cost.

Through extensive experiments across multiple benchmarks, KAT achieves state-of-the-art performance while significantly reducing token usage—matching or exceeding models with several times more parameters. Furthermore, the model has been successfully deployed in real-world production, demonstrating strong practical value in Kuaishou's internal coding assistant, Kwaipilot.

Our findings highlight the importance of adaptive reasoning control, efficient knowledge transfer, and structured supervision for building scalable, high-performance, and deployable LLMs. We believe KAT offers a promising direction toward closing the gap between academic progress and real-world usability.

8. Future Works

Looking ahead, we will release a companion paper detailing our full AutoThink training framework, including cold-start initialization and reinforcement learning strategies. We will also open-source the associated training data, RL codebase, and a suite of models ranging from 1.5B, 7B to 13B parameters. Additionally, we plan to open-source our 200B-parameter Mixture-of-Experts (MoE) model upon completion of training, offering the community a high-capacity, sparsely activated LLM built on the AutoThink foundation.

Beyond the current scope, we aim to extend AutoThink to multi-modal and interactive agent settings, further enhancing the controllability, generality, and applicability of reasoning-capable language models in complex real-world environments.

References

- [1] 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.
- [2] Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. [arXiv preprint arXiv:2405.04434](#), 2024.
- [3] Xiao Bi, Deli Chen, Guanting Chen, Shanhua Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. Deepseek llm: Scaling open-source language models with longtermism. [arXiv preprint arXiv:2401.02954](#), 2024.
- [4] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming—the rise of code intelligence. [arXiv preprint arXiv:2401.14196](#), 2024.
- [5] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengan Huang, Chenxu Lv, et al. Qwen3 technical report. [arXiv preprint arXiv:2505.09388](#), 2025.
- [6] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025.
- [7] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report, 2024.
- [8] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. [arXiv preprint arXiv:2309.16609](#), 2023.
- [9] Meta-AI. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation. <https://ai.meta.com/blog/llama-4-multimodal-intelligence/>, 2025. Accessed: 2025-07-04.
- [10] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. [arXiv preprint arXiv:2407.21783](#), 2024.

- [11] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. [arXiv preprint arXiv:2307.09288](#), 2023.
- [12] Pei Wang, Yanan Wu, Zekun Wang, Jiaheng Liu, Xiaoshuai Song, Zhongyuan Peng, Ken Deng, Chenchen Zhang, Jiakai Wang, Junran Peng, et al. Mtu-bench: A multi-granularity tool-use benchmark for large language models. [arXiv preprint arXiv:2410.11710](#), 2024.
- [13] Jiaheng Liu, Dawei Zhu, Zhiqi Bai, Yancheng He, Huanxuan Liao, Haoran Que, Zekun Wang, Chenchen Zhang, Ge Zhang, Jiebin Zhang, et al. A comprehensive survey on long context language modeling. [arXiv preprint arXiv:2503.17407](#), 2025.
- [14] Jiaheng Liu, Ken Deng, Congnan Liu, Jian Yang, Shukai Liu, He Zhu, Peng Zhao, Linzheng Chai, Yanan Wu, Ke Jin, et al. M2rc-eval: Massively multilingual repository-level code completion evaluation. [arXiv preprint arXiv:2410.21157](#), 2024.
- [15] Ken Deng, Jiaheng Liu, He Zhu, Congnan Liu, Jingxin Li, Jiakai Wang, Peng Zhao, Chenchen Zhang, Yanan Wu, Xueqiao Yin, et al. R2c2-coder: Enhancing and benchmarking real-world repository-level code completion abilities of code large language models. [arXiv preprint arXiv:2406.01359](#), 2024.
- [16] Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, et al. Do not think that much for 2+3=? on the overthinking of o1-like llms. [arXiv preprint arXiv:2412.21187](#), 2024.
- [17] Mehdi Fatemi, Banafsheh Rafiee, Mingjie Tang, and Kartik Talamadupula. Concise reasoning via reinforcement learning. [arXiv preprint arXiv:2504.05185](#), 2025.
- [18] Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Hanjie Chen, et al. Stop overthinking: A survey on efficient reasoning for large language models. [arXiv preprint arXiv:2503.16419](#), 2025.
- [19] Chenwei Lou, Zewei Sun, Xinnian Liang, Meng Qu, Wei Shen, Wenqi Wang, Yuntao Li, Qingping Yang, and Shuangzhi Wu. Adacot: Pareto-optimal adaptive chain-of-thought triggering via reinforcement learning. [arXiv preprint arXiv:2505.11896](#), 2025.
- [20] 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. [arXiv preprint arXiv:2505.14631](#), 2025.
- [21] Gongfan Fang, Xinyin Ma, and Xinchao Wang. Thinkless: Llm learns when to think. [arXiv preprint arXiv:2505.13379](#), 2025.
- [22] Songjun Tu, Jiahao Lin, Qichao Zhang, Xiangyu Tian, Linjing Li, Xiangyuan Lan, and Dongbin Zhao. Learning when to think: Shaping adaptive reasoning in r1-style models via multi-stage rl. [arXiv preprint arXiv:2505.10832](#), 2025.
- [23] Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. Adaptthink: Reasoning models can learn when to think. [arXiv preprint arXiv:2505.13417](#), 2025.
- [24] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, M. Zhang, Y. Li, Y. Wu, and D. Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. [arXiv preprint arXiv:2402.03300](#), 2024.

- [25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. [arXiv preprint arXiv:1707.06347](https://arxiv.org/abs/1707.06347), 2017.
- [26] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. [International Journal of Computer Vision](https://doi.org/10.1007/s10915-021-01407-0), 129(6):1789–1819, 2021.
- [27] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. [arXiv preprint arXiv:1503.02531](https://arxiv.org/abs/1503.02531), 2015.
- [28] Jiaheng Liu, Chenchen Zhang, Jinyang Guo, Yuanxing Zhang, Haoran Que, Ken Deng, Jie Liu, Ge Zhang, Yanan Wu, Congnan Liu, et al. Ddk: Distilling domain knowledge for efficient large language models. [Advances in Neural Information Processing Systems](https://proceedings.neurips.cc/paper/2024/hash/98297-98319_0.pdf), 37:98297–98319, 2024.
- [29] 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](https://arxiv.org/abs/2412.19437), 2024.
- [30] Xiaojiang Zhang, Jinghui Wang, Zifei Cheng, Wenhao Zhuang, Zheng Lin, Minglei Zhang, Shaojie Wang, Yinghan Cui, Chao Wang, Junyi Peng, et al. Srpo: A cross-domain implementation of large-scale reinforcement learning on llm. [arXiv preprint arXiv:2504.14286](https://arxiv.org/abs/2504.14286), 2025.
- [31] Zihan Zheng, Zerui Cheng, Zeyu Shen, Shang Zhou, Kaiyuan Liu, Hansen He, Dongruixuan Li, Stanley Wei, Hangyi Hao, Jianzhu Yao, Peiyao Sheng, Zixuan Wang, Wenhao Chai, Aleksandra Korolova, Peter Henderson, Sanjeev Arora, Pramod Viswanath, Jingbo Shang, and Saining Xie. Livecodebench pro: How do olympiad medalists judge llms in competitive programming?, 2025.
- [32] ByteDance Seed. Introduction to techniques used in seed1.6. https://seed.bytedance.com/en/seed1_6, 2025. Accessed: 2025-07-21.
- [33] OpenAI. Introducing openai o3-mini. <https://openai.com/index/openai-o3-mini/>, 2025. Published: January 31, 2025; Accessed: 2025-07-21.
- [34] Alex Young, Bei Chen, Chao Li, Chengan Huang, Ge Zhang, Guanwei Zhang, Guoyin Wang, Heng Li, Jiangcheng Zhu, Jianqun Chen, et al. Yi: Open foundation models by 01. ai. [arXiv preprint arXiv:2403.04652](https://arxiv.org/abs/2403.04652), 2024.
- [35] Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, Ido Shahaf, Oren Tropp, Ehud Karpas, Ran Zilberstein, Jiaqi Zeng, Soumye Singhal, Alexander Bukharin, Yian Zhang, Tugrul Konuk, Gerald Shen, Ameya Sunil Mahabaleshwarkar, Bilal Kartal, Yoshi Suhara, Olivier Delalleau, Zijia Chen, Zhilin Wang, David Mosallanezhad, Adi Renduchintala, Haifeng Qian, Dima Rekesh, Fei Jia, Somshubra Majumdar, Vahid Noroozi, Wasi Uddin Ahmad, Sean Narenthiran, Aleksander Ficek, Mehrzad Samadi, Jocelyn Huang, Siddhartha Jain, Igor Gitman, Ivan Moshkov, Wei Du, Shubham Toshniwal, George Armstrong, Branislav Kisacanin, Matvei Novikov, Daria Gitman, Evelina Bakhturina, Jane Polak Scowcroft, John Kamalu, Dan Su, Kezhi Kong, Markus Kliegl, Rabeeh Karimi, Ying Lin, Sanjeev Satheesh, Jupinder Parmar, Pritam Gundecha, Brandon Norick, Joseph Jennings, Shrimai Prabhmoye, Syeda Nahida Akter, Mostofa Patwary, Abhinav Khattar, Deepak Narayanan, Roger Waleffe, Jimmy Zhang, Bor-Yiing Su, Guyue Huang, Terry Kong, Parth Chadha, Sahil Jain, Christine Harvey, Elad Segal, Jining Huang, Sergey Kashirsky, Robert McQueen,

Izzy Putterman, George Lam, Arun Venkatesan, Sherry Wu, Vinh Nguyen, Manoj Kilaru, Andrew Wang, Anna Warno, Abhilash Somasamudramath, Sandip Bhaskar, Maka Dong, Nave Assaf, Shahar Mor, Omer Ullman Argov, Scot Junkin, Oleksandr Romanenko, Pedro Larroy, Monika Katariya, Marco Rovinelli, Viji Balas, Nicholas Edelman, Anahita Bhiwandiwalla, Muthu Subramaniam, Smita Ithape, Karthik Ramamoorthy, Yuting Wu, Suguna Varshini Velury, Omri Almog, Joyjit Daw, Denys Fridman, Erick Galinkin, Michael Evans, Shaona Ghosh, Katherine Luna, Leon Derczynski, Nikki Pope, Eileen Long, Seth Schneider, Guillermo Siman, Tomasz Grzegorzek, Pablo Ribalta, Monika Katariya, Chris Alexiuk, Joey Conway, Trisha Saar, Ann Guan, Krzysztof Pawelec, Shyamala Prayaga, Oleksii Kuchaiev, Boris Ginsburg, Oluwatobi Olabiyi, Kari Briski, Jonathan Cohen, Bryan Catanzaro, Jonah Alben, Yonatan Geifman, and Eric Chung. Llama-nemotron: Efficient reasoning models, 2025.

- [36] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track, 2024.
- [37] Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In Proc. of NAACL, 2019.
- [38] Bill Yuchen Lin, Yuntian Deng, Khyathi Chandu, Faeze Brahman, Abhilasha Ravichander, Valentina Pyatkin, Nouha Dziri, Ronan Le Bras, and Yejin Choi. Wildbench: Benchmarking llms with challenging tasks from real users in the wild, 2024.
- [39] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In First Conference on Language Modeling, 2024.
- [40] HuggingFaceH4. Math-500 dataset. <https://huggingface.co/datasets/HuggingFaceH4/MATH-500>, 2024. Accessed: 2025-07-10.
- [41] Maxwell Jia. Aime_2024 dataset. https://huggingface.co/datasets/Maxwell-Jia/AIME_2024, 2024. Accessed: 2025-07-10.
- [42] OpenCompass. Aime2025 dataset. <https://huggingface.co/datasets/opencompass/AIME2025>, 2025. Accessed: 2025-07-10.
- [43] Qin Zhu, Fei Huang, Runyu Peng, Keming Lu, Bowen Yu, Qinyuan Cheng, Xipeng Qiu, Xuanjing Huang, and Junyang Lin. Autologi: Automated generation of logic puzzles for evaluating reasoning abilities of large language models, 2025.
- [44] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage,

Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

- [45] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. [arXiv preprint arXiv:2108.07732](#), 2021.
- [46] Naman Jain, King Han, 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. [arXiv preprint](#), 2024.
- [47] Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In [Forty-second International Conference on Machine Learning](#), 2025.