DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models

DeepSeek-AI

research@deepseek.com

Abstract

We introduce DeepSeek-V3.2, a model that harmonizes high computational efficiency with superior reasoning and agent performance. The key technical breakthroughs of DeepSeek-V3.2 are as follows: **(1) DeepSeek Sparse Attention (DSA)**: We introduce DSA, an efficient attention mechanism that substantially reduces computational complexity while preserving model performance in long-context scenarios. **(2) Scalable Reinforcement Learning Framework**: By implementing a robust reinforcement learning protocol and scaling post-training compute, DeepSeek-V3.2 performs comparably to GPT-5. Notably, our high-compute variant, DeepSeek-V3.2-Speciale, surpasses GPT-5 and exhibits reasoning proficiency on par with Gemini-3.0-Pro, achieving gold-medal performance in both the 2025 International Mathematical Olympiad (IMO) and the International Olympiad in Informatics (IOI). **(3) Large-Scale Agentic Task Synthesis Pipeline**: To integrate reasoning into tool-use scenarios, we developed a novel synthesis pipeline that systematically generates training data at scale. This methodology facilitates scalable agentic post-training, yielding substantial improvements in generalization and instruction-following robustness within complex, interactive environments.

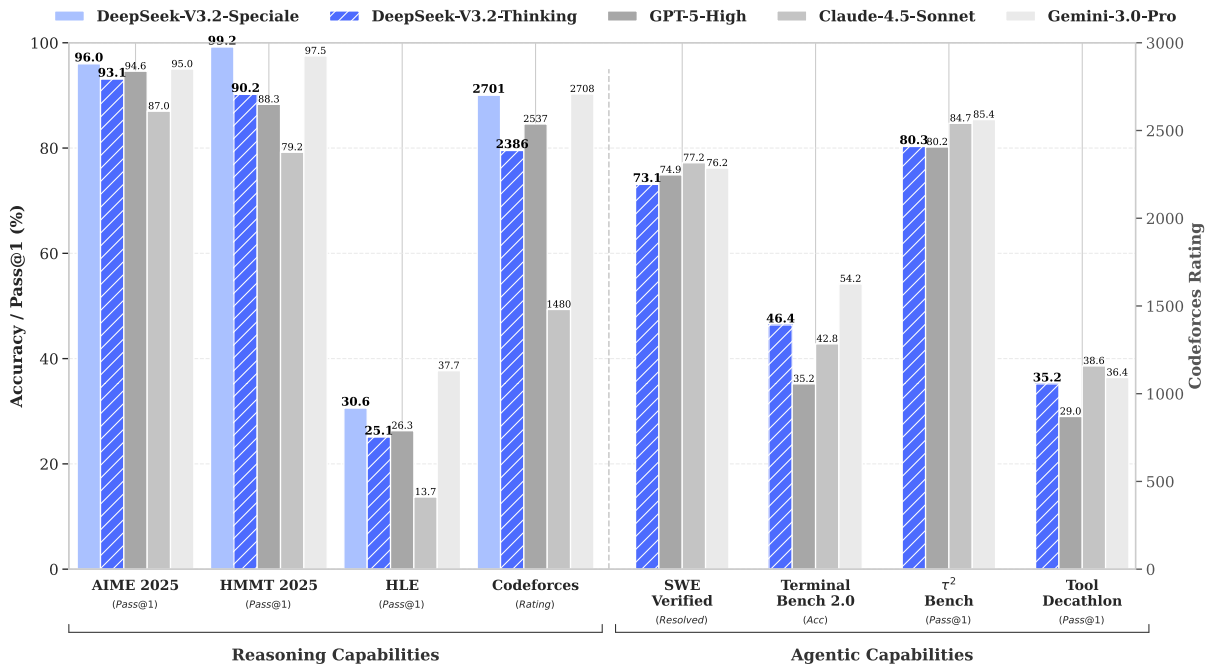


Figure 1 | Benchmark of DeepSeek-V3.2 and its counterparts. For HMMT 2025, we report the February competition, consistent with the baselines. For HLE, we report the text-only subset.

1. Introduction

The release of reasoning models (DeepSeek-AI, 2025; OpenAI, 2024a) marked a pivotal moment in the evolution of Large Language Models (LLMs), catalyzing a substantial leap in overall performance across the verifiable fields. Since this milestone, the capabilities of LLMs have advanced rapidly. However, a distinct divergence has emerged in the past months. While the open-source community (MiniMax, 2025; MoonShot, 2025; Qwen, 2025; ZhiPu-AI, 2025) continues to make strides, the performance trajectory of closed-source proprietary models (Anthropic, 2025b; DeepMind, 2025a; OpenAI, 2025) has accelerated at a significantly steeper rate. Consequently, rather than converging, the performance gap between closed-source and open-source models appears to be widening, with proprietary systems demonstrating increasingly superior capabilities in complex tasks.

Through our analysis, we identify three critical deficiencies that limit the capability of open-source models in complex tasks. First, architecturally, the predominant reliance on vanilla attention (Vaswani et al., 2017) mechanisms severely constrains efficiency for long sequences. This inefficiency poses a substantial obstacle to both scalable deployment and effective post-training. Second, regarding resource allocation, open-source models suffer from insufficient computational investment during the post-training phase, limiting their performance on hard tasks. Finally, in the context of AI agents, open-source models demonstrate a marked lag in generalization and instruction-following capabilities compared to their proprietary counterparts (EvalSys, 2025; Li et al., 2025; Luo et al., 2025), hindering their effectiveness in real deployment.

To address these critical limitations, we first introduce DSA, a highly efficient attention mechanism designed to substantially reduce computational complexity. This architecture effectively addresses the efficiency bottleneck, preserving model performance even in long-context scenarios. Second, we develop a stable and scalable RL protocol that allows for significant computational expansion during the post-training phase. Notably, this framework allocates a post-training computational budget exceeding 10% of the pre-training cost, unlocking advanced capabilities. Thirdly, we propose a novel pipeline to foster generalizable reasoning in tool-use scenarios. First, we implement a cold-start phase utilizing the DeepSeek-V3 (DeepSeek-AI, 2024) methodology to unify reasoning and tool-use within single trajectories. Subsequently, we advance to large-scale agentic task synthesis, where we generate over 1,800 distinct environments and 85,000 complex prompts. This extensive synthesized data drives the RL process, significantly enhancing the model's generalization and instruction-following capability in the agent context.

DeepSeek-V3.2 achieves similar performance with Kimi-k2-thinking and GPT-5 across multiple reasoning benchmarks. Furthermore, DeepSeek-V3.2 significantly advances the agentic capabilities of open models, demonstrating exceptional proficiency on the long-tail agent tasks introduced in EvalSys (2025); Li et al. (2025); Luo et al. (2025). DeepSeek-V3.2 emerges as a highly cost-efficient alternative in agent scenarios, significantly narrowing the performance gap between open and frontier proprietary models while incurring substantially lower costs. Notably, with the aim of pushing the boundaries of open models in the reasoning domain, we relaxed the length constraints to develop DeepSeek-V3.2-Speciale. As a result, DeepSeek-V3.2-Speciale achieves performance parity with the leading closed-source system, Gemini-3.0-Pro (DeepMind, 2025b). It shows gold-medal performance in the IOI 2025, ICPC World Final 2025, IMO 2025, and CMO 2025.

2. DeepSeek-V3.2 Architecture

2.1. DeepSeek Sparse Attention

DeepSeek-V3.2 uses exactly the same architecture as DeepSeek-V3.2-Exp. Compared with DeepSeek-V3.1-Terminus, the last version of DeepSeek-V3.1, the only architectural modification of DeepSeek-V3.2 is the introduction of DeepSeek Sparse Attention (DSA) through continued training.

Prototype of DSA. The prototype of DSA primarily consists of two components: a lightning indexer and a fine-grained token selection mechanism.

The **lightning indexer** computes the index score $I_{t,s}$ between the query token $\mathbf{h}_t \in \mathbb{R}^d$ and a preceding token $\mathbf{h}_s \in \mathbb{R}^d$, determining which tokens to be selected by the query token:

$$I_{t,s} = \sum_{j=1}^{H^I} w_{t,j}^I \cdot \mathrm{ReLU}\left(\mathbf{q}_{t,j}^I \cdot \mathbf{k}_s^I\right), \tag{1}$$

where H^I denotes the number of indexer heads; $\mathbf{q}_{t,j}^I \in \mathbb{R}^{d^I}$ and $w_{t,j}^I \in \mathbb{R}$ are derived from the query token $\mathbf{h}_t$; and $\mathbf{k}_s^I \in \mathbb{R}^{d^I}$ is derived from the preceding token $\mathbf{h}_s$. We choose ReLU as the activation function for throughput consideration. Given that the lightning indexer has a small number of heads and can be implemented in FP8, its computational efficiency is remarkable.

Given the index scores $\{I_{t,s}\}$ for each query token $\mathbf{h}_t$, our **fine-grained token selection mechanism** retrieves only the key-value entries $\{\mathbf{c}_s\}$ corresponding to the top-k index scores. Then, the attention output $\mathbf{u}_t$ is computed by applying the attention mechanism between the query token $\mathbf{h}_t$ and the sparsely selected key-value entries $\{\mathbf{c}_s\}$:

$$\mathbf{u}_t = \mathrm{Attn}\left(\mathbf{h}_t, \left\{\mathbf{c}_s \,\middle|\, I_{t,s} \in \mathrm{Top\text{-}k}(I_{t,:})\right\}\right). \tag{2}$$

Instantiate DSA Under MLA. For the consideration of continued training from DeepSeek-V3.1-Terminus, we instantiate DSA based on MLA (DeepSeek-AI, 2024) for DeepSeek-V3.2. At the kernel level, each key-value entry must be shared across multiple queries for computational efficiency (Yuan et al., 2025). Therefore, we implement DSA based on the MQA (Shazeer, 2019) mode of MLA[1], where each latent vector (the key-value entry of MLA) will be shared across all query heads of the query token. The DSA architecture based on MLA is illustrated in Figure 2. We also provide an open-source implementation of DeepSeek-V3.2[2] to specify the details unambiguously.

2.1.1. *Continued Pre-Training*

Starting from a base checkpoint of DeepSeek-V3.1-Terminus, whose context length has been extended to 128K, we perform continued pre-training followed by post-training to create DeepSeek-V3.2.

The continued pre-training of DeepSeek-V3.2 consists of two training stages. For both stages, the distribution of training data is totally aligned with the 128K long context extension data used for DeepSeek-V3.1-Terminus.

[1] We illustrate the difference between the MQA and MHA modes of MLA in Appendix A.

[2] https://huggingface.co/deepseek-ai/DeepSeek-V3.2-Exp/tree/main/inference

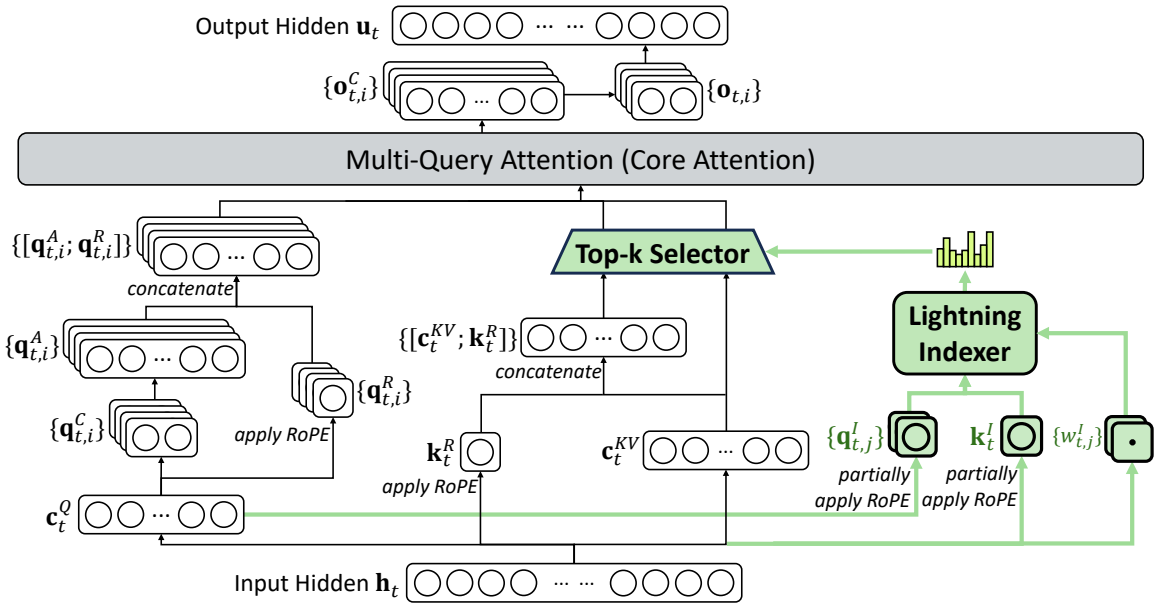


Figure 2 | Attention architecture of DeepSeek-V3.2, where DSA is instantiated under MLA. The green part illustrates how DSA selects the top-k key-value entries according to the indexer.

Dense Warm-up Stage. We first use a short warm-up stage to initialize the lightning indexer. In this stage, we keep dense attention and freeze all model parameters except for the lightning indexer. To align the indexer outputs with the main attention distribution, for the t-th query token, we first aggregate the main attention scores by summing across all attention heads. This sum is then L1-normalized along the sequence dimension to produce a target distribution $p_{t,:} \in \mathbb{R}^t$. Based on $p_{t,:}$, we set a KL-divergence loss as the training objective of the indexer:

$$\mathcal{L}^I = \sum_t \mathbb{D}_{\mathrm{KL}}\left(p_{t,:} \,\middle\|\, \mathrm{Softmax}(I_{t,:})\right). \tag{3}$$

For warm-up, we use a learning rate of 10^{-3}. We train the indexer for only 1000 steps, with each step consisting of 16 sequences of 128K tokens, resulting in a total of 2.1B tokens.

Sparse Training Stage. Following indexer warm-up, we introduce the fine-grained token selection mechanism and optimize all model parameters to adapt the model to the sparse pattern of DSA. In this stage, we also keep aligning the indexer outputs to the main attention distribution, but considering only the selected token set $\mathcal{S}_t = \{s \,|\, I_{t,s} \in \text{Top-k}(I_{t,:})\}$:

$$\mathcal{L}^I = \sum_t \mathbb{D}_{\mathrm{KL}}\left(p_{t,\mathcal{S}_t} \,\middle\|\, \mathrm{Softmax}(I_{t,\mathcal{S}_t})\right). \tag{4}$$

It is worth noting that we detach the indexer input from the computational graph for separate optimization. The training signal of the indexer is from only $\mathcal{L}^I$, while the optimization of the main model is according to only the language modeling loss. In this sparse training stage, we use a learning rate of 7.3×10^{-6}, and select 2048 key-value tokens for each query token. We train both the main model and the indexer for 15000 steps, with each step consisting of 480 sequences of 128K tokens, resulting in a total of 943.7B tokens.

2.2. Parity Evaluation

Standard Benchmark In September 2025, we evaluate DeepSeek-V3.2-Exp on a suite of benchmarks, which focus on diverse capabilities, and compare it with DeepSeek-V3.1-Terminus showing similar performance. While DeepSeek V3.2 Exp significantly improves computational efficiency on long sequences, we do not observe substantial performance degradation compared with DeepSeek-V3.1-Terminus, on both short- and long-context tasks.

Human Preference Given that direct human preference assessments are inherently susceptible to bias, we employ ChatbotArena as an indirect evaluation framework to approximate user preferences for the newly developed base models. Both DeepSeek-V3.1-Terminus and DeepSeek-V3.2-Exp share an identical post-training strategy, and their Elo scores, obtained from evaluations conducted on 10 November 2025, are closely matched. These results suggest that the new base model achieves performance on par with the previous iteration, despite incorporating a sparse attention mechanism.

Long Context Eval Following the release of DeepSeek-V3.2-Exp, several independent long-context evaluations were conducted using previously unseen test sets. A representative benchmark is AA-LCR[3], in which DeepSeek-V3.2-Exp scores four points higher than DeepSeek-V3.1-Terminus in reasoning mode. In the Fiction.liveBench evaluation[4], DeepSeek-V3.2-Exp consistently outperforms DeepSeek-V3.1-Terminus across multiple metrics. This evidence indicates the base checkpoint of DeepSeek-V3.2-Exp does not regress on long context tasks.

2.3. Inference Costs

DSA reduces the core attention complexity of the main model from $O(L^2)$ to $O(Lk)$, where k ($\ll L$) is the number of selected tokens. Although the lightning indexer still has a complexity of $O(L^2)$, it requires much less computation compared with MLA in DeepSeek-V3.1-Terminus. Combined with our optimized implementation, DSA achieves a significant end-to-end speedup in long-context scenarios. Figure 3 presents how token costs of DeepSeek-V3.1-Terminus and DeepSeek-V3.2 vary with the token position in the sequence. These costs are estimated from benchmarking the actual service deployed on H800 GPUs, at a rental price of 2 USD per GPU hour. Note that for short-sequence prefilling, we specially implement a masked MHA mode to simulate DSA, which can achieve higher efficiency under short-context conditions.

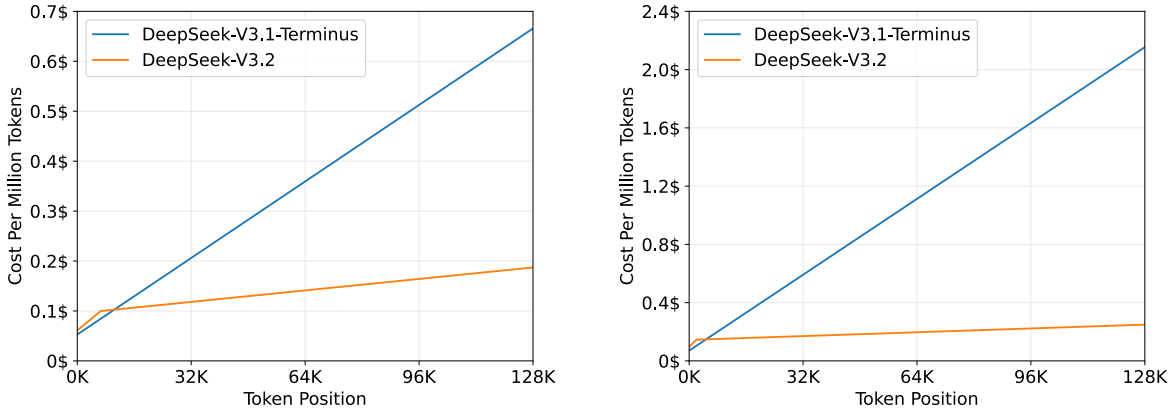
3. Post-Training

After continued pre-training, we perform post-training to create the final DeepSeek-V3.2. The post-training of DeepSeek-V3.2 also employs sparse attention in the same way as the sparse continued pre-training stage. For DeepSeek-V3.2, we maintain the same post-training pipeline as in DeepSeek-V3.2-Exp, which includes specialist distillation and mixed RL training.

Specialist Distillation For each task, we initially develop a specialized model dedicated exclusively to that particular domain, with all specialist models being fine-tuned from the same

[3] https://artificialanalysis.ai/evaluations/artificial-analysis-long-context-reasoning

[4] https://fiction.live/stories/Fiction-liveBench-April-6-2025/oQdzQvKHw8JyXbN87



(a) Prefilling (b) Decoding

Figure 3 | Inference costs of DeepSeek-V3.1-Terminus and DeepSeek-V3.2 on H800 clusters.

pre-trained DeepSeek-V3.2 base checkpoint. In addition to writing tasks and general question-answering, our framework encompasses six specialized domains: mathematics, programming, general logical reasoning, general agentic tasks, agentic coding, and agentic search, with all the domains supporting both thinking and non-thinking modes. Each specialist is trained with large-scale Reinforcement Learning (RL) computing. Furthermore, we employ different models to generate training data for long chain-of-thought reasoning (thinking mode) and direct response generation (non-thinking mode). Once the specialist models are prepared, they are used to produce the domain-specific data for the final checkpoint. Experimental results demonstrate that models trained on the distilled data achieve performance levels only marginally below those of domain-specific specialists, with the performance gap being effectively eliminated through subsequent RL training.

Mixed RL Training For DeepSeek-V3.2, we still adopt Group Relative Policy Optimization (GRPO) (DeepSeek-AI, 2025; Shao et al., 2024) as the RL training algorithm. As DeepSeek-V3.2-Exp, we merge reasoning, agent, and human alignment training into one RL stage. This approach effectively balances performance across diverse domains while circumventing the catastrophic forgetting issues commonly associated with multi-stage training paradigms. For reasoning and agent tasks, we employ rule-based outcome reward, length penalty, and language consistency reward. For general tasks, we employ a generative reward model where each prompt has its own rubrics for evaluation.

DeepSeek-V3.2 and DeepSeek-V3.2-Speciale DeepSeek-V3.2 integrates reasoning, agent, and human alignment data distilled from specialists, undergoing thousands of steps of continued RL training to reach the final checkpoints. To investigate the potential of extended thinking, we also developed an experimental variant, DeepSeek-V3.2-Speciale. This model was trained exclusively on reasoning data with a reduced length penalty during RL. Additionally, we incorporated the dataset and reward method from DeepSeekMath-V2 (Shao et al., 2025) to enhance capabilities in mathematical proofs.

We would like to highlight our efforts in how to create a stable recipe to scale up RL compute in Section 3.1, and how to integrate thinking into agentic tasks in Section 3.2.

3.1. Scaling GRPO

We first review the objective of GRPO. GRPO optimizes the policy model π_θ by maximizing the following objective on a group of responses $\{o_1, \cdots, o_G\}$ sampled from the old policy π_{old} given each question q:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{q\sim P(Q),\{o_i\}_{i=1}^{G}\sim\pi_{\text{old}}(\cdot|q)} \Bigg[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|} \min\left(r_{i,t}(\theta)\hat{A}_{i,t}, \text{clip}\left(r_{i,t}(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_{i,t}\right) - \beta\mathbb{D}_{\text{KL}}\left(\pi_\theta(o_{i,t}) \,\|\, \pi_{\text{ref}}(o_{i,t})\right)\Bigg], \tag{5}$$

where

$$r_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t}|q, o_{i,<t})}{\pi_{\text{old}}(o_{i,t}|q, o_{i,<t})} \tag{6}$$

is the importance sampling ratio between the current and old policy. ε and β are hyper-parameters controlling the clipping range and KL penalty strength, respectively. $\hat{A}_{i,t}$ is the advantage of $o_{i,t}$ which is estimated by normalizing the outcome reward within each group. Specifically, a set of reward models are used to score an outcome reward R_i for each output o_i in the group, yielding G rewards $\boldsymbol{R} = \{R_1, \cdots, R_G\}$ respectively. The advantage of $o_{i,t}$ is calculated by subtracting the average reward of the group from the reward of output o_i, i.e., $\hat{A}_{i,t} = R_i - \text{mean}(\boldsymbol{R})$.

In the following, we outline additional strategies that stabilize RL scaling, directly building on the GRPO algorithm.

Unbiased KL Estimate Given $o_{i,t}$ is sampled from the old policy $\pi_{\text{old}}(\cdot|q, o_{i,<t})$, we correct the K3 estimator (Schulman, 2020) to obtain an unbiased KL estimate using the importance-sampling ratio between the current policy π_θ and the old policy π_{old} .

$$\mathbb{D}_{\text{KL}}\left(\pi_\theta(o_{i,t}) \,\|\, \pi_{\text{ref}}(o_{i,t})\right) = \frac{\pi_\theta(o_{i,t}|q, o_{i,<t})}{\pi_{\text{old}}(o_{i,t}|q, o_{i,<t})}\left(\frac{\pi_{\text{ref}}(o_{i,t}|q, o_{i,<t})}{\pi_\theta(o_{i,t}|q, o_{i,<t})} - \log\frac{\pi_{\text{ref}}(o_{i,t}|q, o_{i,<t})}{\pi_\theta(o_{i,t}|q, o_{i,<t})} - 1\right). \tag{7}$$

As a direct result of this adjustment, the gradient of this KL estimator becomes unbiased, which eliminates systematic estimation errors, thereby facilitating stable convergence. This contrasts sharply with the original K3 estimator, particularly when the sampled tokens have substantially lower probabilities under the current policy than the reference policy, i.e., $\pi_\theta \ll \pi_{\text{ref}}$. In such cases, the gradient of the K3 estimator assigns disproportionately large, unbounded weights to maximize the likelihood of these tokens, resulting in noisy gradient updates that accumulate to degrade sample quality in subsequent iterations and lead to unstable training dynamics. In practice, we find that different domains benefit from varying strengths of KL regularization. For certain domains, such as mathematics, applying a relatively weak KL penalty or even omitting it entirely can yield improved performance.

Off-Policy Sequence Masking To improve the efficiency of RL systems, we typically generate a large batch of rollout data, which is subsequently split into multiple mini-batches for several gradient update steps. This practice inherently introduces off-policy behavior. Additionally, inference frameworks used for efficient data generation are often highly optimized, which may differ in implementation details from training frameworks. Such training-inference inconsistency

further exacerbates the degree of off-policyness. To stabilize training and improve tolerance for off-policy updates, we mask negative sequences that introduce significant policy divergence, as measured by the KL divergence between the data-sampling policy π_{old} and the current policy π_θ. More specifically, we introduce a binary mask M into the GRPO loss:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{q\sim P(Q),\{o_i\}_{i=1}^{G}\sim\pi_{\text{old}}(\cdot|q)}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\right.$$
$$\left.\min\left(r_{i,t}(\theta)\hat{A}_{i,t}, \text{clip}\left(r_{i,t}(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_{i,t}\right) M_{i,t} - \beta\mathbb{D}_{\text{KL}}\left(\pi_\theta(o_{i,t})\,\|\,\pi_{\text{ref}}(o_{i,t})\right)\right], \quad (8)$$

where

$$M_{i,t} = \begin{cases} 0 & \hat{A}_{i,t} < 0, \frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\log\frac{\pi_{\text{old}}(o_{i,t}|q,o_{i,<t})}{\pi_\theta(o_{i,t}|q,o_{i,<t})} > \delta \\ 1 & \text{otherwise,} \end{cases} \quad (9)$$

and δ is a hyper-parameter that controls the threshold of policy divergence. Note that π_{old} here denotes the sampling probability directly returned by the inference framework, thus the KL divergence between the old and current policy accounts for both sources of off-policyness mentioned above. It is also worth noting that we only mask sequences with negative advantages.

Intuitively, the model benefits the most by learning from its own mistakes, whereas highly off-policy negative samples can be detrimental, potentially misleading or destabilizing the optimization process. We empirically observe that this Off-Policy Sequence Masking operation improves stability in certain training scenarios that would otherwise exhibit instability.

Keep Routing Mixture-of-Experts (MoE) models improve computational efficiency by activating only a subset of expert modules during inference. However, discrepancies between inference and training frameworks, compounded by policy updates, can result in inconsistent expert routing during inference and training even for identical inputs. Such inconsistency induces abrupt shifts in the active parameter subspace, which destabilizes optimization and exacerbates off-policy issues. To mitigate this, we preserve the expert routing paths used during sampling in the inference framework and enforce the same routing paths during training, ensuring that identical expert parameters are optimized. This Keep Routing operation was found crucial for RL training stability of MoE models, and has been adopted in our RL training pipeline since DeepSeek-V3-0324.

Keep Sampling Mask Top-p and top-k sampling are widely used sampling strategies to enhance the quality of responses generated by LLMs. Employing these strategies in RL training is also advantageous, as it avoids sampling extremely low-probability tokens that would be used as optimization targets. While such truncation preserves sample quality, it introduces a mismatch between the action spaces of π_{old} and π_θ, which violates the principles of importance sampling and destabilizes training. To address this, we preserve the truncation masks during sampling from π_{old} and apply them to π_θ during training, ensuring both policies share identical action subspaces. Empirically, we find that combining top-p sampling with the Keep Sampling Mask strategy effectively preserves language consistency during RL training.

3.2. Thinking in Tool-Use

3.2.1. Thinking Context Management

DeepSeek-R1 has demonstrated that incorporating a thinking process can significantly enhance a model's ability to solve complex problems. Building on this insight, we aim to integrate thinking capabilities into tool-calling scenarios.

We observed that replicating DeepSeek-R1's strategy—discarding reasoning content upon the arrival of the second round of messages—results in significant token inefficiency. This approach forces the model to redundantly re-reason through the entire problem for each subsequent tool call. To mitigate this, we developed a context management strictly tailored for tool-calling scenarios as shown in Fig 4:

- Historical reasoning content is discarded only when a new **user message** is introduced to the conversation. If only tool-related messages (e.g., tool outputs) are appended, the reasoning content is **retained** throughout the interaction.
- When reasoning traces are removed, the history of **tool calls and their results** remains preserved in the context.

Notably, certain agent frameworks, such as Roo Code or Terminus, simulate tool interactions via user messages. These frameworks may not fully benefit from our enhanced reasoning persistence due to the context management rules outlined above. Therefore, we recommend utilizing non-thinking models for optimal performance with such architectures.

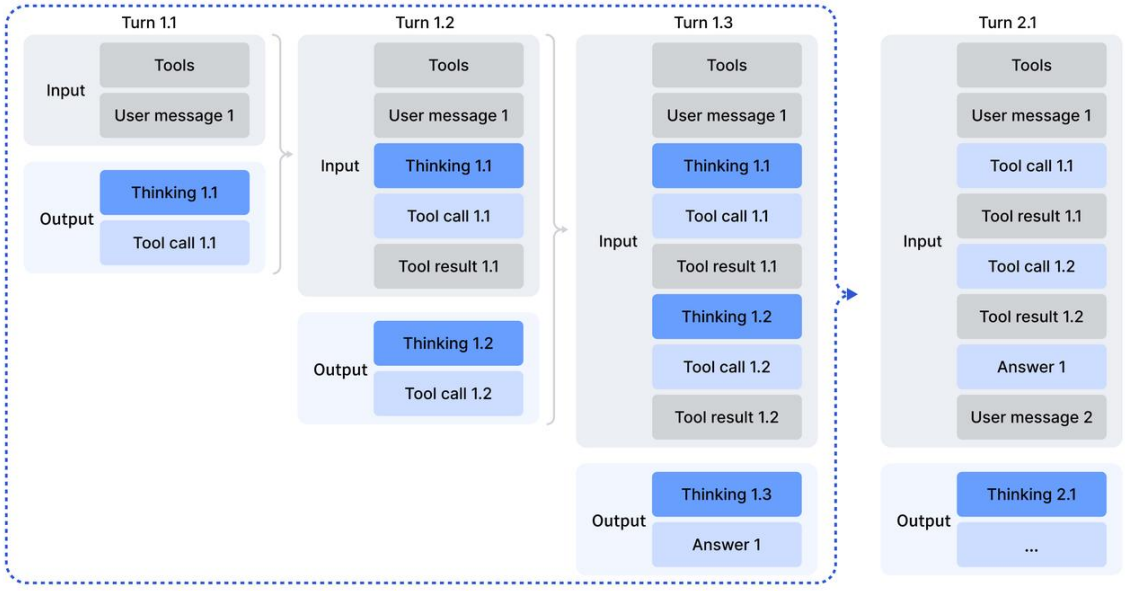


Figure 4 | Thinking retention mechanism in tool-calling scenarios.

3.2.2. Cold-Start

Given the availability of reasoning data (non-agentic) and non-reasoning agentic data, a straightforward strategy for integrating these two capabilities is through carefully designed prompting. We posit that the model possesses sufficient ability to accurately follow explicit instructions, thereby enabling the seamless incorporation of tool execution within the reasoning process.

To demonstrate the operation of the cold-start mechanism, we selectively sample the training data as shown in Appendix Tables 6–8. It is important to note that distinct task prompts are associated with different system prompts. Tables 6–8 present an illustrative example corresponding to a competitive programming prompt. Table 6 presents an example of our reasoning data, which uses a system prompt to explicitly asks the model to do reasoning before the final answer and uses a special tag <think></think> to label the reasoning path. Table 7 shows the prompt of non-reasoning agentic data, where the system prompt contains the guidance of toolcall. Table 8 presents the system prompt we designed to instruct the model to incorporate multiple tool calls within its reasoning process.

In this manner, although the reasoning in tool-use patterns may lack robustness, the model is occasionally able to generate the desired trajectories, thereby providing a basis for subsequent reinforcement learning stages.

3.2.3. Large-Scale Agentic Tasks

A diverse set of RL tasks is crucial for enhancing model robustness. For tasks such as search, code engineering, and code interpretation, we employ real-world tools, including actual web search APIs, coding tools, and Jupyter Notebooks. While these RL environments are real, the prompts employed are either extracted from Internet sources or synthetically generated, rather than obtained from actual user interactions. For other tasks, the environment and prompts are both synthetically constructed. The agent tasks we used are described in Table 1.

Table 1 | The description of different agent tasks, including the number of tasks, environment type (real or synthesized), and prompt source (extracted or synthesized).

	number of tasks	environment	prompt
code agent	24667	real	extracted
search agent	50275	real	synthesized
general agent	4417	synthesized	synthesized
code interpreter	5908	real	extracted

Search Agent We employ a multi-agent pipeline based on DeepSeek-V3.2 to generate diverse, high-quality training data. We first sample informative long-tail entities across diverse domains from large-scale web corpora. A question-construction agent then explores each entity using search tools with configurable depth and breadth parameters, consolidating the discovered information into question-answer pairs. Multiple answer-generation agents with heterogeneous configurations (different checkpoints, system prompts, etc.) produce diverse candidate responses for each proposed QA pair. A verification agent with search capabilities validates all answers through multiple passes, retaining only samples where the ground-truth is correct and all candidates are verifiably incorrect. These data spans multiple languages, domains, and difficulty levels. To complement these verifiable samples and better reflect real-world usage, we also augment the dataset with filtered instances from our existing helpful RL datasets, for which the search tool provides measurable benefits. We then develop detailed evaluation rubrics across multiple quality dimensions and employ a generative reward model to score responses based on these rubrics. This hybrid approach enables optimization for both factual reliability and practical helpfulness.

Code Agent We constructed large-scale, executable environments for software issue resolution by mining millions of issue-Pull Request (PR) pairs from GitHub. This dataset was rigorously filtered using heuristic rules and LLM-based judgments to ensure high quality, requiring that each entry contain a reasonable issue description, a correlated gold patch, and a test patch for validation. An automated environment-setup agent, powered by DeepSeek-V3.2, was employed to build executable environments for these pairs. This agent handles package installation, dependency resolution, and test execution. Test results are output in the standard JUnit format, ensuring consistent parsing across programming languages and test frameworks. An environment is deemed successfully built only when applying the gold patch results in a non-zero count of false-to-positive (F2P) test cases (indicating the issue is fixed) and a zero count of pass-to-fail (P2F) test cases (indicating no regressions). Using this pipeline, we successfully built tens of thousands of reproducible issue resolution environments spanning multiple programming languages, including Python, Java, JavaScript, TypeScript, C, C++, Go, and PHP.

Code Interpreter Agent We utilize Jupyter Notebook as a code interpreter to address complex reasoning tasks. To facilitate this, we curate a diverse set of problems spanning mathematics, logic, and data science, each requiring the model to leverage code execution capabilities to arrive at a solution.

General Agent To scale up agent environments and tasks in RL, we employ an automatic environment-synthesis agent that synthesizes 1,827 task-oriented environments. These tasks are hard to solve but easy to verify. The synthesis workflow primarily consists of environment and toolset construction, task synthesis, and solution generation. Specifically, the workflow proceeds as follows.

1. Given a task category (e.g., planning a travel itinerary) and a sandbox equipped with a bash and a search tool, the agent first uses these tools to generate or retrieve relevant data from the Internet and store them in the sandbox database.
2. The agent then synthesizes a set of task-specific tools, each implemented as a function.
3. To create tasks that are both challenging and automatically verifiable, the agent initially proposes a simple task based on the current database, along with its solution and verification functions implemented in Python. The solution function is restricted to invoking tool functions or performing logical computations, and cannot call other functions or directly access the database, ensuring the task can only be solved through the tool interface. Additionally, the results produced by the solution function must be validated by the verification function. If the solution is not validated, the agent will modify the solution or verification functions until the solution's output passes the verification. The agent then iteratively increases the difficulty of the task and updates the corresponding solution and verification functions. During this iterative process, if the current toolset is not sufficient to solve the task, the agent will augment the toolset.

Following this workflow, we obtain thousands of <environment, tools, task, verifier> tuples. We then perform RL on this dataset using DeepSeek-V3.2 and retain only instances with non-zero pass@100, resulting in 1,827 environments and their corresponding tasks (4,417 in total). A synthetic trip-planning example is illustrated below. This example highlights that, while searching the large combinatorial space for a trip plan that satisfies all constraints is challenging, checking whether a given candidate solution satisfies these constraints is relatively straightforward.

An Example of Synthesized Task: Trip Planning

I'm planning a three-day trip starting from Hangzhou, and I need help creating an itinerary from October 1st to October 3rd, 2025. A few important requirements: I don't want to repeat any cities, hotels, attractions, or restaurants during the entire trip. Also, please make sure that every hotel, restaurant, and attraction you recommend is actually located in the city where I'll be staying that day. One more thing about the second day - I'm trying to be smart about my budget. If I end up booking a luxury hotel that costs 800 CNY or more per night, then I need to be more careful with other expenses: my total spending on both restaurants (lunch and dinner) should stay under 350 CNY, both restaurants should be rated at least 4.0 stars, and the afternoon attraction ticket needs to be less than 120 CNY. If the hotel on day 2 is in the mid-to-high range (500-800 CNY), then I have a bit more flexibility - I just need to make sure at least one of my restaurant choices is rated 4.0 or higher, and the attraction ticket should be below 180 CNY. For more affordable hotels (200-500 CNY range), I only need to ensure that at least one restaurant has a rating of 3.2 or above. Can you help me put together this itinerary?

Submit Result Format

[
{ "time": "2025-10-01", "city": "cite_name", "hotel": "hotel_name", "afternoon_restaurant": "restaurant_name", "afternoon_attraction": "attraction_name", "evening_restaurant": "restaurant_name" },
{ "time": "2025-10-02", "city": "cite_name", "hotel": "hotel_name", "afternoon_restaurant": "restaurant_name", "afternoon_attraction": "attraction_name", "evening_restaurant": "restaurant_name" },
{ "time": "2025-10-03", "city": "cite_name", "hotel": "hotel_name", "afternoon_restaurant": "restaurant_name", "afternoon_attraction": "attraction_name", "evening_restaurant": "restaurant_name" }
]

Tool Set for Trip Planning

Function Name	Description
`get_all_attractions_by_city(city)`	Get all attractions for given city.
`get_all_cities()`	Get all cities from the database.
`get_all_hotels_by_city(city)`	Get all hotels for given city.
`get_all_restaurants_by_city(city)`	Get all restaurants for given city.
`get_city_by_attraction(attraction)`	Get city for given attraction name.
`get_city_by_hotel(hotel)`	Get city for given hotel name.
`get_city_by_restaurant(restaurant)`	Get city for given restaurant name.
`get_city_transport(city)`	Get all intra-city transport options for given city.
`get_infos_by_attraction(info_keywords, attraction)`	Get specified infos for given attraction.
`get_infos_by_city(info_keywords, city)`	Get specified infos for given city.
`get_infos_by_hotel(info_keywords, hotel)`	Get specified infos for given hotel.
`get_infos_by_restaurant(info_keywords, restaurant)`	Get specified infos for given restaurant.
`get_inter_city_transport(from_city, to_city)`	Get all transports between given city pair.
`get_weather_by_city_date(city, date)`	Get weather for given city-date pair.
`submit_result(answer_text)`	Submit the final answer content.

4. Evaluation

4.1. Main Results

We evaluate models on MMLU-Pro (Wang et al., 2024), GPQA Diamond (Rein et al., 2023), Human Last Exam (HLE) Text-only (Phan et al., 2025), LiveCodeBench (2024.08-2025.04), Code-

forces, Aider-Polyglot, AIME 2025, HMMT Feb 2025, HMMT Nov 2025 (Balunović et al., 2025), IMOAnswerBench (Luong et al., 2025), Terminal Bench 2.0, SWE-Verified (OpenAI, 2024b), SWE Multilingual (Yang et al., 2025), BrowseComp (Wei et al., 2025), BrowseCompZh (Zhou et al., 2025), τ^2-bench (Barres et al., 2025), MCP-Universe (Luo et al., 2025), MCP-Mark (EvalSys, 2025), and Tool-Decathlon (Li et al., 2025). Tool-use benchmarks are evaluated using the standard function call format, wherein models are configured to thinking mode. For MCP-Universe (Luo et al., 2025) and MCP-Mark (EvalSys, 2025), we evaluate all models with our internal environment, because the search and playwright environment might be slightly different from the official setting. We set the temperature to 1.0, and the context window to 128K tokens. For math-related tasks such as AIME, HMMT, IMOAnswerBench, and HLE, we eval with the following template: `"{question}\nPlease reason step by step, and put your final answer within \boxed{}."` In the case of HLE, we additionally assessed DeepSeek-V3.2-Thinking using the official template, resulting in a score of 23.9.

Table 2 | Comparison between DeepSeek-V3.2 and closed/open models. For open models, we just compare with models supports thinking in tooluse. Numbers in bold represent the best scores within each model class (open-source and closed-source). The τ^2-Bench result is computed by the average of each category. Regarding BrowseComp, the performance with the context management technique is noted with *.

	Benchmark (Metric)	**Claude-4.5-Sonnet**	**GPT-5 High**	**Gemini-3.0 Pro**	**Kimi-K2 Thinking**	**MiniMax M2**	**DeepSeek-V3.2 Thinking**
English	MMLU-Pro (EM)	88.2	87.5	**90.1**	84.6	82.0	**85.0**
	GPQA Diamond (Pass@1)	83.4	85.7	**91.9**	**84.5**	77.7	82.4
	HLE (Pass@1)	13.7	26.3	**37.7**	23.9	12.5	**25.1**
Code	LiveCodeBench (Pass@1-COT)	64.0	84.5	**90.7**	82.6	83.0	**83.3**
	Codeforces (Rating)	1480	2537	**2708**	-	-	2386
Math	AIME 2025 (Pass@1)	87.0	94.6	**95.0**	**94.5**	78.3	93.1
	HMMT Feb 2025 (Pass@1)	79.2	88.3	**97.5**	89.4	-	**92.5**
	HMMT Nov 2025 (Pass@1)	81.7	89.2	**93.3**	89.2	-	**90.2**
	IMOAnswerBench (Pass@1)	-	76.0	**83.3**	**78.6**	-	78.3
Code Agent	Terminal Bench 2.0 (Acc)	42.8	35.2	**54.2**	35.7	30.0	**46.4**
	SWE Verified (Resolved)	**77.2**	74.9	76.2	71.3	69.4	**73.1**
	SWE Multilingual (Resolved)	**68.0**	55.3	-	61.1	56.5	**70.2**
Search Agent	BrowseComp (Pass@1)	24.1	**54.9**	-	-/60.2*	44.0	**51.4/67.6***
	BrowseCompZh (Pass@1)	42.4	63.0	-	62.3	48.5	**65.0**
	HLE (Pass@1)	32.0	35.2	**45.8**	**44.9**	31.8	40.8
ToolUse	τ^2-Bench(Pass@1)	84.7	80.2	**85.4**	74.3	76.9	**80.3**
	MCP-Universe (Success Rate)	46.5	47.9	**50.7**	35.6	29.4	**45.9**
	MCP-Mark (Pass@1)	33.3	**50.9**	43.1	20.4	24.4	**38.0**
	Tool-Decathlon (Pass@1)	**38.6**	29.0	36.4	17.6	16.0	**35.2**

DeepSeek-V3.2 achieves similar performance with GPT-5-high on reasoning tasks, but is slightly worse than Gemini-3.0-Pro. Compared to K2-Thinking, DeepSeek-V3.2 achieves comparable scores with substantially fewer output tokens, as shown in Table 3. These performance gains can be attributed to the increased computational resources allocated to RL training. Over recent months, we have observed consistent performance improvements correlating with extended RL training budget, which already exceeds 10% of the pre-training cost. We hypothesize that reasoning capabilities could be further enhanced with additional computational budget allocation. Notably, the performance of DeepSeek-V3.2 presented herein is constrained by a length constraint reward model; upon removal of the restriction, we observe further improvement in

model performance, as detailed in Section 4.2.

In code agent evaluations, DeepSeek-V3.2 significantly outperforms open-source LLMs on both SWE-bench Verified and Terminal Bench 2.0, demonstrating its potential within real-world coding workflows. Regarding Terminal Bench 2.0, as previously noted, our context management strategy for the 'thinking mode' is currently incompatible with Terminus; consequently, the reported score of 46.4 was achieved using the Claude Code framework. We also evaluated DeepSeek-V3.2 with Terminus in non-thinking mode, yielding a score of 39.3. For SWE-bench Verified, the primary score was obtained using our internal framework. Robustness tests across other settings—including the Claude Code and RooCode frameworks, as well as non-thinking mode—produced consistent results, ranging from 72 to 74.

For the search agent evaluation, we assess our models using a standard commercial search API. Since DeepSeek-V3.2 supports a maximum context length of only 128K, approximately 20%+ of the test cases exceed this limit. To address this, we employ a context management method to derive the final score. For reference, the score is 51.4 without context management. Further details are provided in Section 4.4.

On tool-use benchmarks, DeepSeek-V3.2 substantially narrows the performance gap between open-source and closed-source LLMs, though it remains below frontier models. For τ^2-bench, we employ the model itself as the user agent, achieving final category scores of 63.8 (Airline), 81.1 (Retail), and 96.2 (Telecom). For the MCP benchmarks, we employ the function calling format and place tool outputs within messages designated with the 'tool' role, rather than the 'user' role. During our testing, we observed that DeepSeek-V3.2 frequently engages in redundant self-verification, generating excessively long trajectories. This tendency often causes the context length to exceed the 128K limit, particularly in tasks such as MCP-Mark GitHub and Playwright evaluation. Consequently, this phenomenon hinders the final performance of DeepSeek-V3.2. However, integrating context management strategies can further enhance performance. We identify this as a direction for future work and a practical consideration for users. Even if DeepSeek-V3.2 suffers from the issue, it still significantly outperforms existing open models. Notably, since the environments and toolsets employed in these benchmarks were not encountered during RL training, the observed improvements demonstrate DeepSeek-V3.2's capacity to generalize its reasoning strategies to out-of-domain agentic scenarios. The evaluation of non-thinking model in the agent scenario is shown in Appendix Table 9.

4.2. Results of DeepSeek-V3.2-Speciale

Table 3 demonstrates that DeepSeek-V3.2-Speciale achieves superior performance by leveraging increased reasoning tokens, surpassing the state-of-the-art Gemini-3.0-Pro across multiple benchmarks. Remarkably, as shown in Table 4, this general-purpose model attains gold-medal level performance in the 2025 International Olympiad in Informatics (IOI) and the ICPC World Finals (ICPC WF) without targeted training. Furthermore, by incorporating techniques from Shao et al. (2025), the model excels in complex proof tasks, reaching gold-medal thresholds in the 2025 International Mathematical Olympiad (IMO) and China Mathematical Olympiad (CMO)[5]. Detailed evaluation protocols are provided in Appendix D.

However, the token efficiency of DeepSeek-V3.2-Speciale remains significantly inferior to that of Gemini-3.0-Pro. To mitigate deployment costs and latency, we imposed stricter token constraints during the training of the official DeepSeek-V3.2, aiming to optimize the trade-off

[5]We evaluated the English version of CMO 2025. The IMO 2025 and CMO 2025 problems, together with the inference code, can be found at: https://github.com/deepseek-ai/DeepSeek-Math-V2.

Table 3 | Benchmark performance and efficiency of reasoning models. For each benchmark, cells show accuracy and output token count (in thousands). The highest accuracy per benchmark is in bold; the second-highest is underlined.

Benchmark	GPT-5 High	Gemini-3.0 Pro	Kimi-K2 Thinking	DeepSeek-V3.2 Thinking	DeepSeek-V3.2 Speciale
AIME 2025 (Pass@1)	94.6 (13k)	<u>95.0</u> (15k)	94.5 (24k)	93.1 (16k)	**96.0** (23k)
HMMT Feb 2025 (Pass@1)	88.3 (16k)	<u>97.5</u> (16k)	89.4 (31k)	92.5 (19k)	**99.2** (27k)
HMMT Nov 2025 (Pass@1)	89.2 (20k)	<u>93.3</u> (15k)	89.2 (29k)	90.2 (18k)	**94.4** (25k)
IMOAnswerBench (Pass@1)	76.0 (31k)	<u>83.3</u> (18k)	78.6 (37k)	78.3 (27k)	**84.5** (45k)
LiveCodeBench (Pass@1-COT)	84.5 (13k)	**90.7** (13k)	82.6 (29k)	83.3 (16k)	<u>88.7</u> (27k)
CodeForces (Rating)	2537 (29k)	**2708** (22k)	-	2386 (42k)	<u>2701</u> (77k)
GPQA Diamond (Pass@1)	<u>85.7</u> (8k)	**91.9** (8k)	84.5 (12k)	82.4 (7k)	<u>85.7</u> (16k)
HLE (Pass@1)	26.3 (15k)	**37.7** (15k)	23.9 (24k)	25.1 (21k)	<u>30.6</u> (35k)

between performance and cost. We believe that token efficiency remains a critical area for future investigation.

Table 4 | Performance of DeepSeek-V3.2-Speciale in top-tier mathematics and coding competitions. For ICPC WF 2025, we report the number of submissions for each successfully solved problem. DeepSeek-V3.2-Speciale ranked 2nd in ICPC WF 2025 and 10th in IOI 2025.

Competition	P1	P2	P3	P4	P5	P6	Overall	Medal
IMO 2025	7	7	7	7	7	0	35/42	Gold
CMO 2025	18	18	9	21	18	18	102/126	Gold
IOI 2025	100	82	72	100	55	83	492/600	Gold

Competition	A	B	C	D	E	F	G	H	I	J	K	L	Overall	Medal
ICPC WF 2025	3	-	1	1	2	2	-	1	1	1	1	1	10/12	Gold

4.3. Synthesis Agentic Tasks

In this section, we perform ablation experiments to study the effect of synthetic agentic tasks. We focus on two questions. First, are synthetic tasks sufficiently challenging for reinforcement learning? Second, how well do these synthetic tasks generalize, i.e., can they transfer to different downstream tasks or real-world environments?

To address the first question, we randomly sample 50 instances from the general synthesized agentic tasks and evaluate both the model used for synthesis and frontier closed-source LLMs. As shown in Table 5, DeepSeek-V3.2-Exp attains an accuracy of only 12%, while frontier closed-source models achieve at most 62%. These results indicate that the synthetic data include agentic tasks that are challenging for both DeepSeek-V3.2-Exp and frontier closed-source models.

To investigate whether RL on synthetic data can generalize to different tasks or real-world environments, we apply RL to the SFT checkpoint of DeepSeek-V3.2 (denoted DeepSeek-V3.2-SFT). To exclude the effects of long CoT and other RL data, we conduct RL only on synthetic agentic tasks in non-thinking mode. We then compare the model with DeepSeek-V3.2-SFT and DeepSeek-V3.2-Exp, where DeepSeek-V3.2-Exp is trained with RL only in search and code environments. As shown in Figure 5, large-scale RL on synthetic data yields substantial improve-

Table 5 | Accuracy of general synthesized tasks on different models.

Pass@K	DeepSeek-v3.2-Exp	Sonnet-4.5	Gemini-3.0 Pro	GPT-5-Thinking
1	12%	34%	51%	62%
2	18%	47%	65%	75%
4	26%	62%	74%	82%

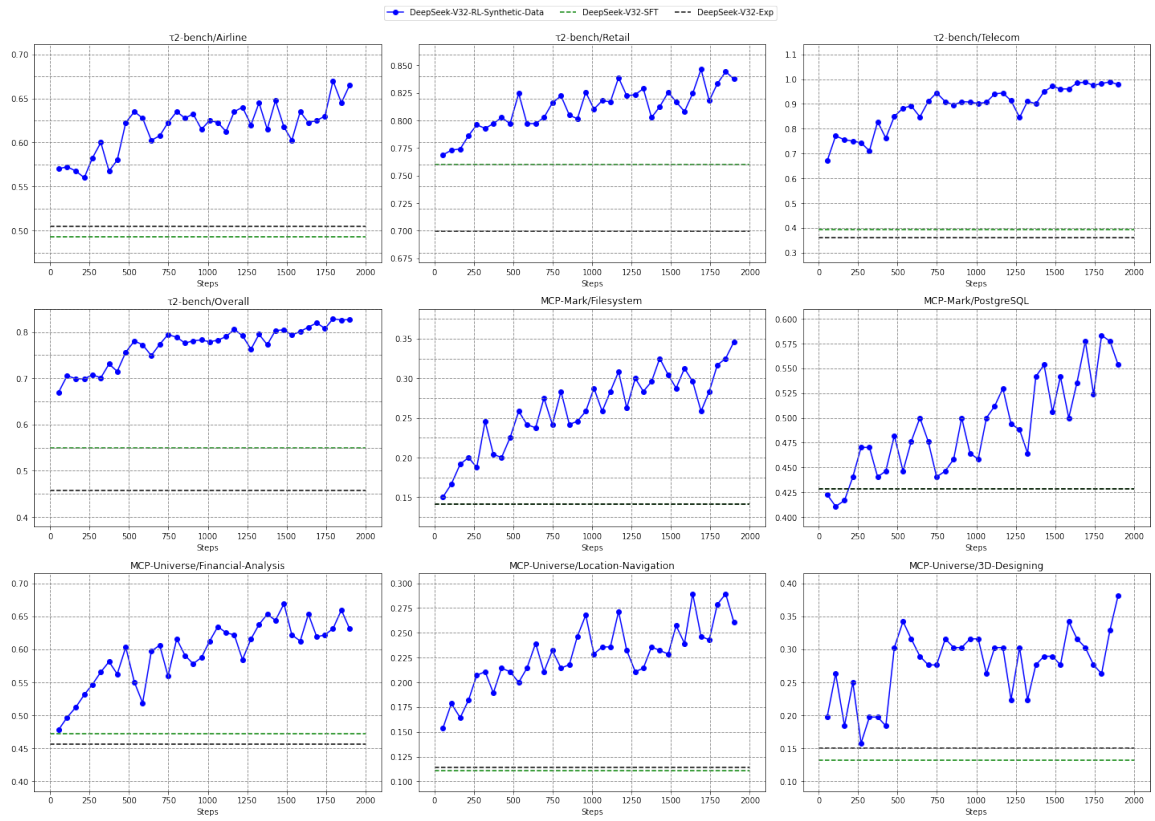


Figure 5 | RL training of DeepSeek-V3.2-SFT using exclusively synthetic general agent data.

ments over DeepSeek-V3.2-SFT on Tau2Bench, MCP-Mark, and MCP-Universe benchmarks. In contrast, restricting RL to code and search scenarios does not improve performance on these benchmarks, further highlighting the potential of synthetic data.

4.4. Context Management of Search Agent

Even with extended context windows such as 128k, agentic workflows, particularly in search-based scenarios, frequently encounter maximum length limitations that prematurely truncate the reasoning process. This bottleneck inhibits the full realization of test-time compute potential. To address this, we introduce context management employing simple strategies to extend token budgets at test time， when the token usage exceeds 80% of the context window length. These strategies include (1) **Summary**, which summarizes the overflowed trajectory and re-initiates the rollout; (2) **Discard-75%**, which discards the first 75% tool call history in the trajectory to free up spaces; (3) **Discard-all**, which resets the context by discarding all previous tool call history (similar to the new context tool (Anthropic, 2025a)). For comparison, we also implement a parallel scaling baseline, **Parallel-fewest-step**, which samples N independent trajectories and

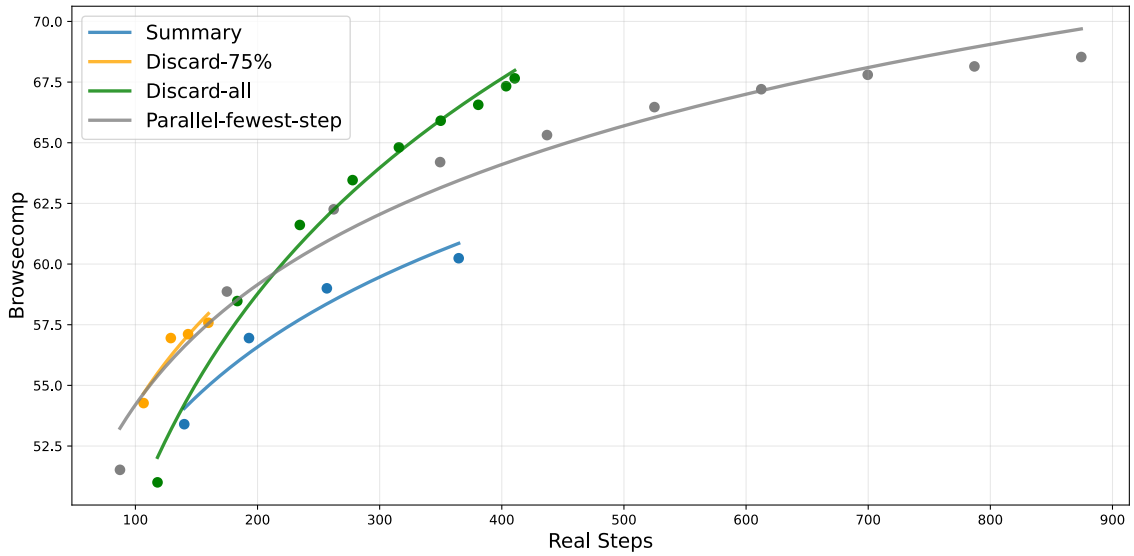


Figure 6 | Accuracy of Browsecomp with different test-time compute expansion strategies.

selects the trajectory with the fewest steps.

We evaluate these strategies on the BrowseComp benchmark (Wei et al., 2025). As illustrated in Figure 6, under varying compute budgets, context management leads to significant performance gains by allowing the model to scale up test-time compute, providing more space to perform additional execution steps. For example, Summary extends the average steps to 364, achieving a performance improvement of up to 60.2. However, its overall efficiency is relatively low. Despite its simplicity, Discard-all performs well in both efficiency and scalability, achieving a score of 67.6, comparable to parallel scaling while using significantly fewer steps.

In summary, test-time compute can be scaled either serially through context management or in parallel, both effectively extending the model's problem-solving capacity. However, different strategies exhibit varying efficiency and scalability. Thus, it is crucial to account for actual compute costs when benchmarking model performance. Meanwhile, finding the optimal combination of serial and parallel scaling to maximize both efficiency and scalability remains a crucial direction for future work.

5. Conclusion, Limitation, and Future Work

In this work, we introduced DeepSeek-V3.2, a framework that effectively bridges the gap between computational efficiency and advanced reasoning capabilities. Using DSA, we addressed critical computation complexity without sacrificing long-context performance. By increasing computational budget, DeepSeek-V3.2 achieves comparable performance with GPT-5 on reasoning benchmarks. Finally, the integration of our large-scale agentic task synthesis pipeline significantly enhances tool-use proficiency, unlocking new possibilities for robust and generalizable AI agents with open LLM. Furthermore, our high-compute variant, DeepSeek-V3.2-Speciale, validated by gold-medal achievements in the IMO and IOI, sets a milestone for open LLMs.

Despite these achievements, we acknowledge certain limitations when compared to frontier closed-source models such as Gemini-3.0-Pro. First, due to fewer total training FLOPs, the breadth of world knowledge in DeepSeek-V3.2 still lags behind that of leading proprietary

models. We plan to address this knowledge gap in future iterations by scaling up the pre-training compute. Second, token efficiency remains a challenge; DeepSeek-V3.2 typically requires longer generation trajectories (i.e., more tokens) to match the output quality of models like Gemini-3.0-Pro. Future work will focus on optimizing the intelligence density of the model's reasoning chains to improve efficiency. Third, solving complex tasks is still inferior to frontier models, motivating us to further refine our foundation model and post-training recipe.

References

Anthropic. System card: Claude opus 4.5, 2025a. URL https://assets.anthropic.com/m/64823ba7485345a7/Claude-Opus-4-5-System-Card.pdf.

Anthropic. Introducing claude sonnet 4.5, 2025b. URL https://www.anthropic.com/news/claude-sonnet-4-5l.

M. Balunović, J. Dekoninck, I. Petrov, N. Jovanović, and M. Vechev. Matharena: Evaluating llms on uncontaminated math competitions. Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmark, 2025.

V. Barres, H. Dong, S. Ray, X. Si, and K. Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

DeepMind. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. arXiv preprint arXiv:2507.06261, 2025a.

G. DeepMind. Gemini 3 pro model card, 2025b. URL https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Pro-Model-Card.pdf.

DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. CoRR, abs/2405.04434, 2024. doi: 10.48550/ARXIV.2405.04434. URL https://doi.org/10.48550/arXiv.2405.04434.

DeepSeek-AI. Deepseek-v3 technical report, 2024. URL https://arxiv.org/abs/2412.19437.

DeepSeek-AI. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. Nature, 645(8081):633–638, 2025.

EvalSys. Mcpmark leaderboard, 2025. URL https://mcpmark.ai/leaderboard.

J. Li, W. Zhao, J. Zhao, W. Zeng, H. Wu, X. Wang, R. Ge, Y. Cao, Y. Huang, W. Liu, et al. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. arXiv preprint arXiv:2510.25726, 2025.

Z. Luo, Z. Shen, W. Yang, Z. Zhao, P. Jwalapuram, A. Saha, D. Sahoo, S. Savarese, C. Xiong, and J. Li. Mcp-universe: Benchmarking large language models with real-world model context protocol servers. arXiv preprint arXiv:2508.14704, 2025.

T. Luong, D. Hwang, H. H. Nguyen, G. Ghiasi, Y. Chervonyi, I. Seo, J. Kim, G. Bingham, J. Lee, S. Mishra, A. Zhai, C. H. Hu, H. Michalewski, J. Kim, J. Ahn, J. Bae, X. Song, T. H. Trinh, Q. V. Le, and J. Jung. Towards robust mathematical reasoning. In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, 2025. URL https://aclanthology.org/2025.emnlp-main.1794/.

MiniMax. https://www.minimax.io/news/minimax-m2, 2025. URL https://www.minimax.io/news/minimax-m2.

MoonShot. Introducing kimi k2 thinking, 2025. URL https://moonshotai.github.io/Kimi-K2/thinking.html.

OpenAI. Learning to reason with llms, 2024a. URL https://openai.com/index/learning-to-reason-with-llms/.

OpenAI. Introducing SWE-bench verified we're releasing a human-validated subset of swe-bench that more, 2024b. URL https://openai.com/index/introducing-swe-bench-verified/.

OpenAI. Introducing gpt-5, 2025. URL https://openai.com/index/introducing-gpt-5/.

L. Phan, A. Gatti, Z. Han, N. Li, J. Hu, H. Zhang, C. B. C. Zhang, M. Shaaban, J. Ling, S. Shi, et al. Humanity's last exam. arXiv preprint arXiv:2501.14249, 2025.

Qwen. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. arXiv preprint arXiv:2311.12022, 2023.

J. Schulman. Approximating KL divergence, 2020. URL http://joschu.net/blog/kl-approx.html.

Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, M. Zhang, Y. K. Li, Y. Wu, and D. Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. CoRR, abs/2402.03300, 2024. doi: 10.48550/ARXIV.2402.03300. URL https://doi.org/10.48550/arXiv.2402.03300.

Z. Shao, Y. Luo, C. Lu, Z. Ren, J. Hu, T. Ye, Z. Gou, S. Ma, and X. Zhang. Deepseekmath-v2: Towards self-verifiable mathematical reasoning, 2025.

N. Shazeer. Fast transformer decoding: One write-head is all you need. CoRR, abs/1911.02150, 2019. URL http://arxiv.org/abs/1911.02150.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. pages 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Y. Wang, X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, W. Ren, A. Arulraj, X. He, Z. Jiang, T. Li, M. Ku, K. Wang, A. Zhuang, R. Fan, X. Yue, and W. Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. CoRR, abs/2406.01574, 2024. URL https://doi.org/10.48550/arXiv.2406.01574.

J. Wei, Z. Sun, S. Papay, S. McKinney, J. Han, I. Fulford, H. W. Chung, A. T. Passos, W. Fedus, and A. Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. arXiv preprint arXiv:2504.12516, 2025.

J. Yang, K. Lieret, C. E. Jimenez, A. Wettig, K. Khandpur, Y. Zhang, B. Hui, O. Press, L. Schmidt, and D. Yang. Swe-smith: Scaling data for software engineering agents, 2025. URL https://arxiv.org/abs/2504.21798.

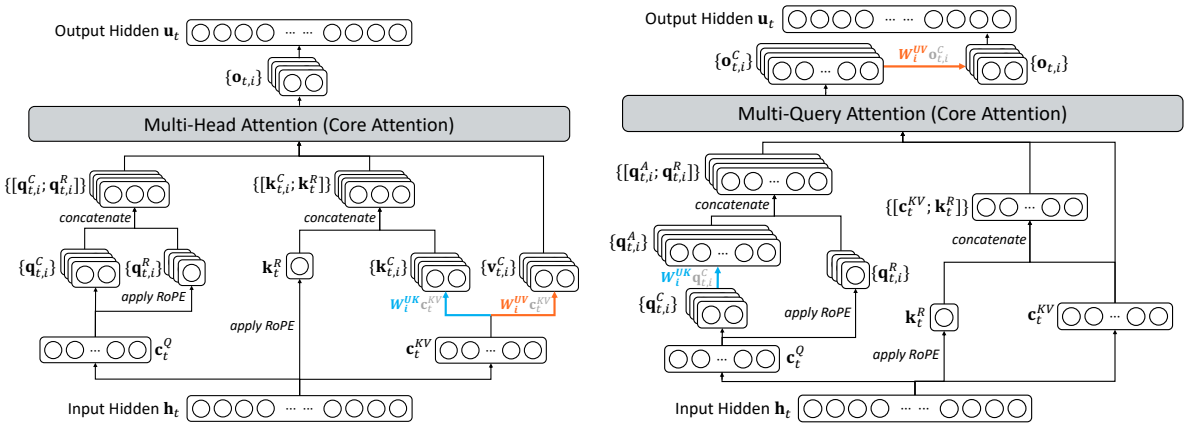
J. Yuan, H. Gao, D. Dai, J. Luo, L. Zhao, Z. Zhang, Z. Xie, Y. Wei, L. Wang, Z. Xiao, Y. Wang, C. Ruan, M. Zhang, W. Liang, and W. Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, editors, Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, pages 23078–23097. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.1126/.

ZhiPu-AI. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. arXiv preprint arXiv:2508.06471, 2025.

P. Zhou, B. Leon, X. Ying, C. Zhang, Y. Shao, Q. Ye, D. Chong, Z. Jin, C. Xie, M. Cao, et al. Browsecomp-zh: Benchmarking web browsing ability of large language models in chinese. arXiv preprint arXiv:2504.19314, 2025.

Appendices

A. MHA and MQA Modes of MLA



(a) MHA mode of MLA.

(b) MQA mode of MLA.

Figure 7 | Illustration of the MHA and MQA modes of MLA. For DeepSeek-V3.1-Terminus, the MHA mode is used for training and prefilling, while the MQA mode is used for decoding.

Figure 7 illustrates two aspects of MLA – the MHA and MQA modes – as well as the transformation between them.

B. Cold Start Template

Table 6 | An example of the reasoning data system prompt. The system prompt requires the model to output the reasoning process in the tag <think></think>.

Reasoning System Prompt	You are an expert Python programmer. You will be given a question (problem specification) and will generate a correct Python program that matches the specification and passes all tests. Please first reason before giving the final answer. The reasoning process enclosed within <think> </think>. The final answer is output after the </think> tag.
Prompt	Given a linked list, swap every two adjacent nodes and return its head ...
Reasoning Response	<think> ... </think> [FINAL ANSWER]

Table 7 | {TOOL-DESCRIPTIONS} and {TOOLCALL-FORMAT} will be replaced with the specific tools and our designed toolcall format.

Agent System Prompt	Use Python interpreter tool to execute Python code. The code will not be shown to the user. This tool should be used for internal reasoning, but not for code that is intended to be visible to the user (e.g. when creating plots, tables, or files). When you send a message containing Python code to python, it will be executed in a stateful Jupyter notebook environment. python will respond with the output of the execution or time out after 120.0 seconds. ## Tools You have access to the following tools: {TOOL-DESCRIPTIONS} Important: ALWAYS adhere to this exact format for tool use: {TOOLCALL-FORMAT}
Prompt	Given a linked list, swap every two adjacent nodes and return its head ...
Agent Response	[MULTI-TURN TOOLCALL] [FINAL ANSWER]

Table 8 | The model executes tool calls in thinking process.

Reasoning Required Agent System Prompt	You are a helpful assistant with access to a Python interpreter. - You may use the Python tool **multiple times** during your reasoning, a.k.a in <think></think>, with a maximum of 20 code executions. - Call the Python tool early in your reasoning to aid in solving the task. Continue reasoning and invoking tools as needed until you reach the final answer. Once you have the answer, stop reasoning and present your solution using Markdown and LaTeX. - Do NOT invoke any tools in your presented final solution steps. - To improve efficiency and accuracy, you should prefer code execution over language-based reasoning whenever possible. Keep your reasoning succinct; let the code do the heavy lifting. ## Tools You have access to the following tools: {TOOL-DESCRIPTIONS} Important: ALWAYS adhere to this exact format for tool use: {TOOLCALL-FORMAT}
Prompt	Given a linked list, swap every two adjacent nodes and return its head ...
Agent Response with Thinking	<think> [MULTI-TURN Thinking-Then-TOOLCALL] </think> [FINAL ANSWER]

C. Non-thinking DeepSeek-V3.2 Agentic Evaluation

Table 9 | Comparison between DeepSeek-V3.2 non-thinking and thinking modes. The terminal bench scores are evaluated with the Claude Code framework in the table. Non-thinking score of Terminal Bench 2.0 with Terminus framework is 39.3.

	Benchmark (Metric)	**non-thinking**	**thinking**
Code Agent	Terminal Bench 2.0 (Acc)	37.1	46.4
	SWE Verified (Resolved)	72.1	73.1
	SWE Multilingual (Resolved)	68.9	70.2
ToolUse	τ^2-bench (Pass@1)	77.2	80.3
	MCP-Universe (Success Rate)	38.6	45.9
	MCP-Mark (Pass@1)	26.5	38.0
	Tool-Decathlon (Pass@1)	25.6	35.2

The performance of non-thinking mode is slightly worse than the thinking mode, but still competitive.

D. Evaluation Method of IOI, ICPC World Final, IMO, and CMO

For all competitions, the model's maximum generation length is set to 128k. No tools or internet access are used, and testing strictly adheres to the contest's time and attempt limits.

For the IOI evaluation, we designed our submission strategy in accordance with the official competition rules, which permit up to 50 submissions per problem and score each submission based on the maximum points achieved across all subtasks. Specifically, we first sampled 500 candidate solutions for each problem, then applied a multi-stage filtering pipeline. In the initial stage, we eliminated invalid submissions that failed to pass the provided sample test cases or exceeded the length constraints. Subsequently, we employed the DeepSeek-V32-Exp model to identify and remove samples in which the model explicitly indicated an inability or refusal to solve the problem. From the remaining valid candidates, we selected the 50 samples with the longest thinking traces for final submission.

For the ICPC evaluation, we adapted the same filtering methodology but with a smaller initial sample size. We generated 32 candidate solutions per problem and applied the identical filtering criteria to select submissions.

In the IMO and CMO tasks, we employ a generate-verify-refine loop. The model iteratively improves its solution until it achieves a perfect self-evaluation or hits the maximum revision cap, identical to the process in Shao et al. (2025).

E. Author List

Research & Engineering: Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan*, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Erhang Li, Fangqi Zhou*, Fangyun Lin, Fucong Dai, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoran Wei, Haowei Zhang, Haowen Luo, Haozhe Ji, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, Jialiang Huang, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jingchang Chen, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jinhua Zhu, Jun Ran*, Junguang Jiang, Junjie Qiu, Junlong Li*, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexin Huang*, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Liang Zhao, Liangsheng Yin*, Lihua Guo, Lingxiao Luo, Linwang Ma, Litong Wang, Liyue Zhang, M.S. Di, M.Y Xu, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Panpan Huang, Peixin Cong, Peiyi Wang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, S.H. Liu, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Songyang Zhou, Tao Ni, Tao Yun, Tian Pei, Tian Ye, Tianyuan Yue, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjun Gao, Wentao Zhang, Xi Gao, Xiangwen Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyuan Li*, Xu Chen, Xuecheng Su, Xuehai Pan, Xuheng Lin, Xuwei Fu, Y.Q. Wang, Yang Zhang, Yanhong Xu, Yanru Ma, Yao Li, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yiliang Xiong, Ying He, Ying Zhou, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuduan Wang, Yue Gong*, Yuhan Wu, Yuheng Zou, Yukun Li, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z.F. Wu, Z.Z. Ren, Zehua Zhao, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhiyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Zizheng Pan, Zongqing Yao

Data Annotation: Bei Feng, Hui Li, J.L. Cai, Jiaqi Ni, Lei Xu, Meng Li, Ning Tian, R.J. Chen, R.L. Jin, S.S. Li, Shuang Zhou, Tianyu Sun, X.Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xinnan Song, Xinyi Zhou, Y.X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Zhen Huang, Zhipeng Xu, Zhongyu Zhang

Business & Compliance: Dongjie Ji, Jian Liang, Jianzhong Guo, Jin Chen, Leyi Xia, Miaojun Wang, Mingming Li, Peng Zhang, Ruyi Chen, Shangmian Sun, Shaoqing Wu, Shengfeng Ye, T.Wang, W.L. Xiao, Wei An, Xianzu Wang, Xiaowen Sun, Xiaoxiang Wang, Ying Tang, Yukun Zha, Zekai Zhang, Zhe Ju, Zhen Zhang, Zihua Qu

Authors are listed alphabetically by their first name. Names marked with * denote individuals who have departed from our team.