Timely Machine: Awareness of Time Makes Test-Time Scaling Agentic

Yichuan Ma[1,2], **Linyang Li**[1,3†], **Yongkang Chen**[1], **Peiji Li**[1,2], **Xiaozhe Li**[4], **Qipeng Guo**[1], **Dahua Lin**[1,3], **Kai Chen**[1],
[1]Shanghai Artificial Intelligence Laboratory, [2]Fudan University
[3]The Chinese University of Hong Kong, [4]Tongji University
yichuanma24@m.fudan.edu.cn, lilinyang@pjlab.org.cn

Abstract

As large language models (LLMs) increasingly tackle complex reasoning tasks, test-time scaling has become critical for enhancing capabilities. However, in agentic scenarios with frequent tool calls, the traditional generation-length-based definition breaks down: tool latency decouples inference time from generation length. We propose TIMELY MACHINE, redefining test-time as wall-clock time, where models dynamically adjust strategies based on time budgets. We introduce Timely-Eval, a benchmark spanning high-frequency tool calls, low-frequency tool calls, and time-constrained reasoning. By varying tool latency, we find smaller models excel with fast feedback through more interactions, while larger models dominate high-latency settings via superior interaction quality. Moreover, existing models fail to adapt reasoning to time budgets. We propose Timely-RL to address this gap. After cold-start supervised fine-tuning, we use reinforcement learning to enhance temporal planning. Timely-RL improves time budget awareness and consistently boosts performance across Timely-Eval. We hope our work offers a new perspective on test-time scaling for the agentic era.

1 Introduction

As the parameter counts and data volumes of Large Language Models (LLMs) continue to expand, relying solely on scaling these dimensions is becoming increasingly expensive. A new wave of models, represented by OpenAI-o1 and DeepSeek-R1, has pivoted towards exploring test-time scaling, which allocates increased inference time to tackle complex reasoning tasks (Guo et al., 2025; Jaech et al., 2024). Drawing inspiration from Group Relative Policy Optimization (GRPO), methods based on Reinforcement Learning with Verifiable Rewards (RLVR) have emerged as a mainstream paradigm for enhancing LLM reasoning (Shao et al., 2024; Hui et al., 2024; Yu et al., 2025).

Furthermore, a significant portion of research applies RLVR to scenarios involving external tools and environmental interactions. (Qian et al., 2025; Li et al., 2025c). Empowered by the reasoning capabilities of LLMs, agent systems designed for interactive tasks, such as web search and automated machine learning model training, have also demonstrated substantial application potential (Jiang et al., 2025; Jin et al., 2025).

Regarding the reasoning capabilities of LLMs, test-time scaling stands as a pivotal concept. Prior to the widespread adoption of LLMs in tool-use scenarios, test-time scaling in standard reasoning tasks (such as mathematics, coding, and commonsense reasoning) was typically correlated with generation time (Paglieri et al., 2025; Wen et al., 2025; Han et al., 2025). Intuitively, in scenarios involving only model inference, an inherent positive correlation exists between time and generation length.

However, this relationship breaks down when discussing agentic scenarios characterized by complex tool utilization. In such settings, since the latency of tool execution is unpredictable, the total time required to complete a task cannot be estimated solely based on the model's generation length. A concise reasoning trajectory might consume much time if it involves high-latency tools; and conversely, a longer trace could even end faster given rapid environmental feedback. Consequently, the traditional definition of test-time scaling based on generation length proves inapplicable for agentic tasks. Liu et al. (2025b) provides models with remaining tool call budget through an external plugin. While this method is effective, it equivalently treats the budget as the number of tool calls, which still overlooks the latency of the tools themselves.

In this work, we introduce the concept of the TIMELY MACHINE. By decoupling test time from generation tokens, we restore the concept of "test time" to its original meaning as a physical quantity: time itself. We argue that the awareness of task

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 4619–4636
July 2-7, 2026 ©2026 Association for Computational Linguistics

duration and the ability to plan for task costs should be intrinsic capabilities of LLMs.

Specifically, when environmental feedback incurs high latency, the model should adjust its interaction frequency and prioritize the quality of each interaction round to achieve better results; conversely, when the environment responds rapidly, the model should engage in faster interactions to gather more sufficient feedback, thereby exploring for higher payoffs within the same duration.

To validate this capability, we introduce **Timely-Eval**, a benchmark designed to evaluate how well models adapt their strategies based on temporal feedback. Within Timely-Eval, we curate three distinct interactive scenarios: high-frequency tool, low-frequency tool, and general reasoning tasks under strict time constraints. Across these tasks, we aim to assess whether models are aware of the concept of "time", and whether they can utilize this awareness to dynamically adjust their reasoning strategies accordingly.

We then evaluate model performance under different settings of tool latencies by adjusting the feedback delays. We find that, on tasks with shorter tool latencies, relatively smaller models sometimes achieve better performance than larger models due to their ability to perform more rounds of interactions. However, as interaction time increases, the speed advantage of small models gradually diminishes, while large models, benefiting from their higher interaction quality, often yield greater returns.

To further train models with this accurate time awareness, we first synthesize a large Timely Reasoning dataset, where each sample strictly adheres to the time constraints for task completion. We then use this dataset to perform supervised fine-tuning (SFT) on base models, obtaining initial points with time awareness. We then propose **Timely-RL** on these initial points for further enhance. As a variant of GRPO, Timely-RL encourages models to maximally utilize the provided time budget. After Timely-RL training, the model demonstrates the ability to effectively plan and utilize time budget.

To summarize, our contributions are as follows:

- We introduce Timely Machine, redefining test-time in agentic scenarios as wall-clock time.
- We propose Timely-Eval, a benchmark for evaluating reasoning under varying time budgets.
- Through evaluation, we find that: (i) model preference shifts with tool latency, smaller models outperform larger ones under low latency through more interactions, while larger models excel when latency is high (ii) existing models lack the ability to adjust reasoning strategies based on time budgets.
- We develop Timely-RL, a reinforcement learning approach that teaches models time-aware reasoning. Timely-RL enables models to develop accurate time budget awareness and adapt their reasoning strategies accordingly.

2 Related Works

2.1 Test-Time Scaling for LLM Reasoning

In research on enhancing LLM reasoning, several early works improved reasoning accuracy by voting on multiple responses (Wang et al., 2022). Some also explored the possibility of self-correction (Yuan et al., 2024; Ren et al., 2023; Huang et al., 2023). Yao et al. (2023) proposed Tree of Thought, which turns reasoning into a tree structure segmented by steps. Later, some studies investigated applying Monte Carlo Tree Search to LLM sampling to tackle more complex reasoning problems (Guan et al., 2025; Chen et al., 2024).

Shao et al. (2024) proposed Group Reward Policy Optimization (GRPO). GRPO fueled a surge of interest in enhancing model reasoning via Reinforcement Learning with Verifiable Rewards **(RLVR)**. In general reasoning, RLVR-based post-training strategies have brought remarkable performance gains (Li et al., 2025b; Hui et al., 2024; Shen et al., 2025; Team et al., 2025; Yang et al., 2025).

Following RLVR's success on reasoning tasks, subsequent work explored RL practices that involve interacting with tools (Qian et al., 2025; Feng et al., 2025; Song et al., 2025; Li et al., 2025a,c). These efforts show the promise of applying LLMs' general reasoning ability to more complex, agentic interactive environments.

2.2 Budget Control for LLM Reasoning

Before agentic RL became a trend, researchers investigated lengthy reasoning chains in general tasks (Qu et al., 2025; Sui et al., 2025; Zhang et al., 2024). In these settings, budget control is equivalent to controlling output length. Consequently, a line of work focused on regulating generation length (Paglieri et al., 2025; Wen et al., 2025; Han et al., 2025), aiming to make LLMs sensitive to inference budgets for dynamic adjustment. Recently, some studies have further incorporated wall-clock latency into general reasoning via additional designs (Fan et al., 2025; Wang et al., 2025b). Transitioning to agentic scenarios, AgentTTS (Wang et al., 2025a) allocates task budgets based on FLOPs, while Liu et al. (2025b) introduces an external plug-in to modulate tool-use and reasoning budgets. However, these approaches rely on extra designs. We take a more radical stance: time awareness should be an intrinsic capability.

3 Preliminary Study

3.1 Test-Time Scaling in Agentic Scenarios

Before presenting our approach, we first clarify what test-time means. In tasks like math and code, test-time was identical to the model's inference time. Therefore, for a given reasoning task, suppose the model performs N reasoning steps. The total time cost t_{all} can be written as:

$$t_{\text{all}} = \sum_{i=1}^{N} t_{\text{gen}}^{(i)}, \tag{1}$$

where $t_{\text{gen}}^{(i)}$ denotes the generation time at the i-th step. Here, total time equals inference time. Therefore, allocating a larger time budget is equivalent to allocating a larger token budget.

However, in an LLM-powered agentic task with tool calls, the total time cost t_{all} becomes:

$$t_{\text{all}} = \sum_{i=1}^{N} t_{\text{gen}}^{(i)} + \sum_{i=1}^{N} t_{\text{tool}}^{(i)}, \tag{2}$$

where $t_{\text{tool}}^{(i)}$ denotes the time consumed by the tool call at the i-th step. Since tool latency is now part of the overall cost, test-time is no longer directly proportional to the number of generated tokens. For clarity, we discuss the following three cases:

Tools Take the Lead We first consider the case where the tool latency in each round is much larger than generation latency:

$$t_{\text{tool}}^{(i)} \gg t_{\text{gen}}^{(i)}. \tag{3}$$

In this case, generation time has little impact on test-time. Therefore, under a fixed time budget t_{budget}, the number of interaction rounds N is largely determined by tool latency:

$$N \approx \frac{t_{\text{budget}}}{t_{\text{tool}}}. \tag{4}$$

Consequently, the per-round interaction quality becomes the key factor that determines task performance. For such tasks, the model should plan more carefully in each round.

Models Take the Lead In some scenarios, tool latency is negligible, or the model can even solve the task without any tool use:

$$t_{\text{gen}}^{(i)} \gg t_{\text{tool}}^{(i)}. \tag{5}$$

In this case, test-time collapses back to the conventional setting, where the total cost is dominated by the number of generated tokens.

Meeting in the Middle In more general settings, the model's generation latency and the tool-call latency are often of a similar scale, i.e.,

$$t_{\text{tool}}^{(i)} \approx t_{\text{gen}}^{(i)}. \tag{6}$$

In this regime, test-time compute is jointly determined by tool latency and generation latency. We define a per-step latency ratio

$$m_i = \frac{t_{\text{tool}}^{(i)}}{t_{\text{gen}}^{(i)}}. \tag{7}$$

Then $t_{\text{tool}}(i) = m_i \, t_{\text{gen}}(i)$, the total cost becomes

$$t_{\text{all}} = \sum_{i=1}^{N} (m_i + 1) \, t_{\text{gen}}^{(i)}. \tag{8}$$

Since neither $t_{\text{gen}}^{(i)}$ nor $t_{\text{tool}}^{(i)}$ is known beforehand, m_i can vary substantially across steps. When $m_i > 1$, tool latency dominates, and the model should spend more effort to improve the quality of each tool call; When $m_i < 1$, generation dominates, and the model may shorten its reasoning to enable more interaction rounds. Overall, because m_i is unpredictable, token-based budget control is insufficient. Therefore, agent systems must condition their strategies on *time* and adapt dynamically to the observed latency feedback.

4 Method

In this section, we describe TIMELY-EVAL and TIMELY-RL introduced in the Introduction.

4.1 Timely-Eval

To measure an LLM's awareness of time and its ability to regulate behavior in agentic tasks, we introduce TIMELY-EVAL, which covers three task families: (i) Interactive Text Games, (ii) Machine Learning Tasks, and (iii) General Reasoning Tasks.

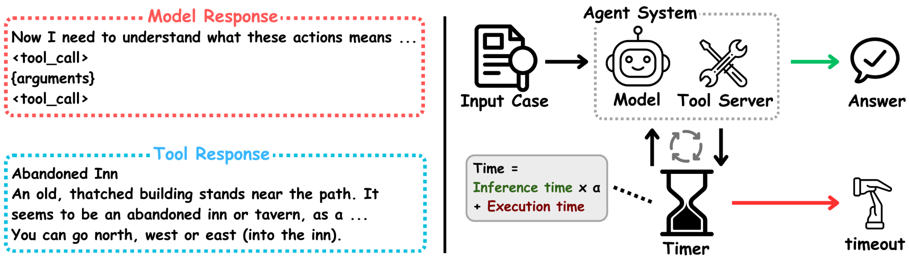

Figure 1: Pipeline for Timely-RL

Interactive Games Interactive fiction games present a natural testbed for agentic scaling. We build upon **Jericho**[1], an interface connecting agents to classic text-based games like *Zork*. In these environments, models navigate through rounds of interaction, selecting actions to maximize rewards.

Since these games are lightweight, we can easily assign different arbitrary interaction delays to each action without worrying about actual tool execution time. This allows us to simulate both high-latency and low-latency scenarios, enabling exploration of how models adapt their strategies across varying time constraints. We benchmark each model to obtain its average per-step latency τ, then set the time budget as an integer multiple of τ.

Machine Learning In machine learning tasks, tool execution dominates the time budget. We adapt several tasks from MLEBench-Lite (Chan et al., 2024), where LLMs should train ML models to solve given problems. Models should interact with code interpreter, submitting solutions and observing execution outputs, error messages, and performance scores across multiple iterations.

After estimating the average per-round latency τ (including both LLM reasoning and model training), we allocate multiples of τ as the time budget. Because the training of ML models is typically time-consuming, this suite evaluates whether the LLM can make effective use of additional time budget when tool cost dominates.

General Reasoning Finally, we return to standard reasoning tasks, where test-time is equivalent to generation time. To ensure the model conditions on *time* (rather than only token length), we provide a timer tool that reports elapsed time. Models must query it during reasoning to avoid timing out. As before, we run speed test to obtain τ and set per-instance budgets as scaled multiples. This suite tests whether time awareness emerges naturally.

[1]https://github.com/microsoft/jericho

4.2 Timely Cold Start

Before applying reinforcement learning, we first distill supervised data to provide a cold start and establish baselines.

On general reasoning tasks, We distill 1M training examples from Qwen3-235B-2507-Instruct (Yang et al., 2025) on queries sampled from AM-DeepSeek-R1-Distilled-1.4M (Zhao et al., 2025). Specifically, we equip the teacher model with a timer tool, enabling it to produce solutions under varying time budgets.

Similarly, we distill interaction traces from DeepSeek-V3.2 (Liu et al., 2025a) on interactive games. To prevent memorization, we limit distillation to the first **50** steps per game, ensuring that subsequent exploration must be learned rather than merely recalled. We do not perform task-specific distillation for ML tasks.

After distillation, we combine the synthetic dataset with DataScience-Instruct-500K (Zhang et al., 2025) to form the final cold-start mixture. Through SFT on various base models, we provide the initialization point for subsequent RL training.

4.3 Timely-RL

Tool latency creates dynamic, real-time feedback signals that naturally lend themselves to reinforcement learning. We therefore propose Timely-RL, a time-aware RL approach that trains models to optimize their behavior under time constraints. The whole pipeline is shown in Fig. 1.

In each interaction round i, the time feedback $t^{(i)}$ is computed as the sum of inference time $t^{(i)}_{\text{gen}}$ and tool execution time $t^{(i)}_{\text{tool}}$. To adapt the model to different conditions, we multiply $t^{(i)}_{\text{gen}}$ by a coefficient α as the feedback received by the model, simulating scenarios on faster or slower devices.

Next, we design reward functions for the three tasks during RL training. For each response, let r denote the accuracy reward, t denote the time cost for task completion, and $T_{\max}$ denote the time limit.

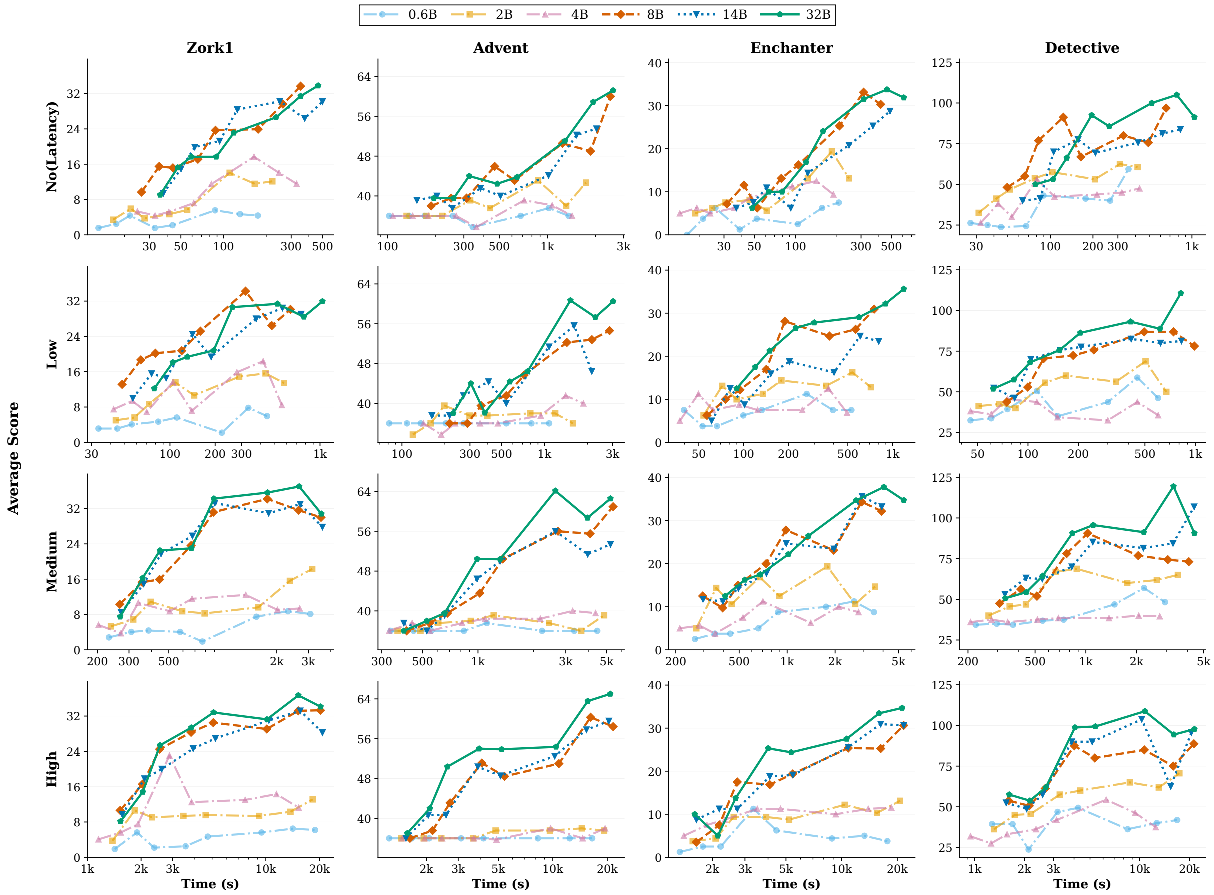

Figure 2: Scaling experiments across four text-based interactive games (Zork1, Advent, Enchanter, Detective) using Qwen3 models (0.6B-32B parameters). All models are cold-started with the same dataset. Each row represents different tool call latency settings: No latency, Low (∼2s), Medium (∼10s), and High (∼50s).

The reward function can be expressed as:

$$R(t, r) = \begin{cases} 0, & t > T_{\max} \\ r_{\mathrm{f}}, & t \le T_{\max}, r = 0 \\ r_{\mathrm{f}} + r + \lambda U(t), & t \le T_{\max}, r > 0 \end{cases} \tag{9}$$

where r_{f} is a fixed reward for correct formatting and timely completion, and $U(t)$ represents the model's utilization rate of time cost, calculated as:

$$U(t) = \sin\left(\frac{\pi}{2} \cdot \min\left(\frac{t}{T_{\max}}, 1\right)\right) \tag{10}$$

This ensures that as the time ratio nears 1.0, the reward increases more slowly, preventing minor time differences near the deadline from causing large reward fluctuations.

5 Experiments

5.1 Tool Latency Flips the Scaling Winner

8B Beats 32B To compare the scaling performance of models with different sizes under various tool call scenarios, we select four well-known interactive text games as evaluation benchmarks: Enchanter, Zork1, Advent, and Detective. We perform cold start on six models from the Qwen3 series, ranging from 0.6B to 32B parameters, and conduct evaluations under four settings with different tool call latencies, as shown in Fig. 2. The deployment resources are fixed for all models.

The results show that when tool latency is low, smaller models can outperform larger ones within the same time constraint due to faster inference. For instance, in the zero latency setting on **Detective**, with a 50s-120s time budget, the 8B model achieves top performance among all models, outperforming both 14B and 32B. On other games, the largest 32B model still shows no significant performance advantage under the same budget.

However, as tool latency increases, the task transitions to a tool-dominated regime. Under the same time budget, different-sized models perform a similar number of interaction rounds. At this point, larger models demonstrate higher interaction qual-

Table 1: Evaluation results on Timely Eval across three task categories. For general reasoning tasks, we report the average accuracy under various time budgets. For interactive games, we show the average game scores achieved under different budgets (averaged over 8 trials per game). For ML tasks, we present the average test set accuracy of generated code (averaged over 8 trials per task). The best score is in **bold**, while the second best is underlined.

Model	General Reasoning			Interactive Reasoning				Machine Learning			
	MATH	AIME	GPQA Diam.	Zork1	Advent	Enchan -ter	Detect -ive	Leaf Clf.	Space -ship	Pizza	Insult Detect
GPT-5.1(medium)	71.5	**46.7**	**71.0**	<u>34.1</u>	**57.6**	24.4	**105.0**	67.5	63.5	58.0	79.0
DeepSeek-V3.2	**82.9**	43.3	58.7	24.9	48.7	15.9	63.2	72.0	68.1	68.6	78.0
Gemini2.5-pro	63.0	37.5	<u>59.0</u>	**34.9**	<u>50.7</u>	**34.1**	71.9	71.4	65.4	74.6	75.1
Qwen3-8B	71.2	40.0	37.5	2.3	36.0	5.2	54.1	32.0	17.8	28.6	51.9
Qwen3-14B	70.8	41.7	21.0	9.8	34.9	9.5	50.5	46.4	55.9	60.7	70.6
Qwen3-32B	75.0	<u>45.7</u>	35.5	14.4	38.2	11.7	70.0	51.8	50.1	59.0	57.2
TimelyLM-8B	<u>78.0</u>	42.5	49.5	27.5	48.5	<u>29.5</u>	<u>88.1</u>	**93.9**	**74.4**	**76.1**	**84.0**

ity. In the high latency setting, 32B consistently achieves the best performance across all tasks. Additionally, when model size is too small, exemplified by the 0.6B, 1.7B, and 4B models, despite their fast generation speed, their performance remains inferior to larger models regardless of tool setting due to capability limitations. Therefore, for an agentic task, beyond tool latency, the model's inherent capability must also be considered. An appropriate model selection should balance inference speed and performance.

5.2 Experimental Setups

Timely-Eval Details We first introduce the components of Timely-Eval. For interactive games, We collect and wrap 57 text-based interactive games from the Jericho environment. Models are provided with tools to execute actions, query valid moves, check scores, and terminate games.

General benchmarks comprise three reasoning benchmarks: AIME, MATH, and GPQA-diamond (Hendrycks et al., 2021; Rein et al., 2024). Models are provided with a tool to track elapsed inference time, and must adjust their reasoning strategies based on time costs and budget consumption.

ML benchmarks contain four ML tasks: Leaf Classification, Spaceship Titanic, Random Acts of Pizza, and Detecting Insults in Social Commentary. Models are allowed to use tools to execute code, obtain results, and monitor elapsed time.

Evaluation Setups We then introduce the experimental settings in Timely-Eval. For General Benchmarks, we set time budgets at $0.75\times$, $1.0\times$, $2.0\times$, and $3.0\times$ of this baseline, treating responses exceeding the budget as incorrect. We report accuracy averaged across budgets.

For Machine Learning Benchmarks, we measure the average time per reasoning step (including tool calls) and allocate budgets ranging from $1\times$ to $5\times$ this duration.

For Interactive Games, we assign latencies to different tools to simulate real execution. We evaluate on four games (Enchanter, Zork1, Advent, Detective) with budgets equivalent to 30, 50, 100, and 200 steps, reporting scores averaged across settings. In Section 5.1, budgets range from 30 to over 400 steps.

Training Setups We first perform SFT as cold-start, followed by Timely-RL. SFT runs for 2 epochs using Xtuner (Contributors, 2023). For RL, each sample is rolled out 8 times with batch size 16, with maximum generation length of 16,384 tokens. We set learning rate to 1e-6 and clip ratio to [0.2, 0.28]. We use VeRL for RL (Sheng et al., 2024).

5.3 Timely-Eval Results

Starting from the cold-started 8B, we apply Timely-RL across three tasks to obtain TimelyLM-8B. For evaluation on Timely-Eval, we select several mainstream general models as performance references, and also evaluate models from the Qwen3 series for direct comparison. The evaluation results on Timely-Eval are shown in Tab. 1.

First, TimelyLM-8B outperforms Qwen3-8B on all three general benchmarks. Notably, all models show significant performance drops on Timely-Eval compared to raw benchmarks. For reasoning models(such as DeepSeek-V3.2), this is because they cannot control the generation length, and thus frequently fail to complete the task within the time

Table 2: Performance under a 0.75× time budget. **On Time**: percentage of samples finished within the time limit (%). **ACC**: accuracy (%). **Tool-use rate**: percentage of samples that invoked at least one tool call (%).

	AIME (0.75×)			**GPQA-Diamond** (0.75×)			**MATH** (0.75×)		
Model	**On Time**	**ACC**	**Tool-use rate**	**On Time**	**ACC**	**Tool-use rate**	**On Time**	**ACC**	**Tool-use rate**
DeepSeek-V3.2	33.3	33.3	100.0	43.4	33.3	100.0	**75.0**	**74.0**	100.0
GPT5.1	33.3	30.0	40.0	48.0	**44.0**	8.0	34.0	34.0	8.0
Qwen3-8B	30.0	26.7	53.3	36.0	20.0	94.0	48.0	42.0	66.0
Qwen3-14B	30.0	20.0	10.0	18.0	12.0	8.0	38.0	32.0	6.0
Qwen-32B	33.3	26.7	26.7	34.0	28.0	52.0	46.0	42.0	26.0
TimelyLM-8B	**53.3**	**33.3**	**100.0**	**52.0**	36.0	**100.0**	56.0	52.0	**100.0**

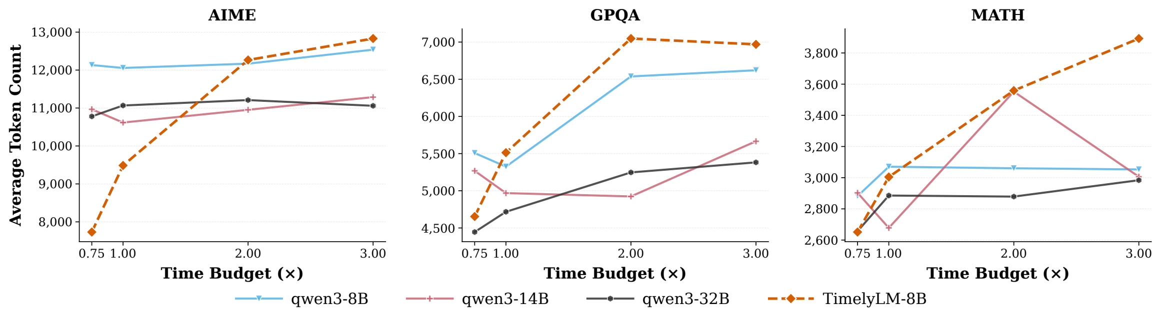

Figure 3: Changes in reasoning length across different time budgets. The time budget is the model's original reasoning time scaled by factors such as 0.75× and 1.0×.

budget at 0.75× or even larger time constraints. While GPT-5.1 does not perform lengthy reasoning, the time constraint forces it into extremely short reasoning, leading to performance degradation. The Qwen series models consider fewer tool calls during reasoning. Due to their faster inference speed, they occasionally complete reasoning within the time limit. More detailed discussion of this phenomenon is provided in Section 5.4.

On Interactive Games and Machine Learning tasks, TimelyLM-8B achieves strong performance. On ML tasks, larger models sometimes choose to implement more complex solutions, causing the code execution to exceed the time limit. TimelyLM, however, produces more stable code implementations. On Interactive Games, although we only use the first 50 steps as a cold start, TimelyLM still achieves performance comparable to the strongest general models, demonstrating that the model has effectively learned exploration strategies for these interactive games.

5.4 Training LLMs to be ON TIME

Furthermore, we design additional experiments to verify whether Timely-RL truly equips the model with better time awareness.

5.4.1 Behavior under reduced budgets

To demonstrate this, we first select general benchmarks and compare the on-time completion rates of different models under a 0.75× time budget. As shown in Tab. 2, models exemplified by Qwen3-8B cannot adapt to shorter time constraints and thus fail to complete reasoning within limit. However, TimelyLM-8B can leverage the timing tool to obtain the remaining budget in real-time and adjust its reasoning process across the tasks, thereby enabling more tasks to be completed on time.

Additionally, we observe two interesting phenomena. First, the performance variation caused by budget reduction differs across benchmarks. On more challenging tasks like AIME and GPQA-diamond, higher completion rates do not lead TimelyLM to significantly outperform other models. However, on simpler problems like MATH, the model's on-time completion rate more closely aligns with its accuracy rate, meaning that most tasks completed on time are solved correctly. On such tasks, the benefit of **completing tasks on time** outweighs that of performing more extensive reasoning.

Second, comparing more powerful models, DeepSeek-V3.2 better utilizes tool calls to obtain remaining time, while GPT-5.1 prefers single-

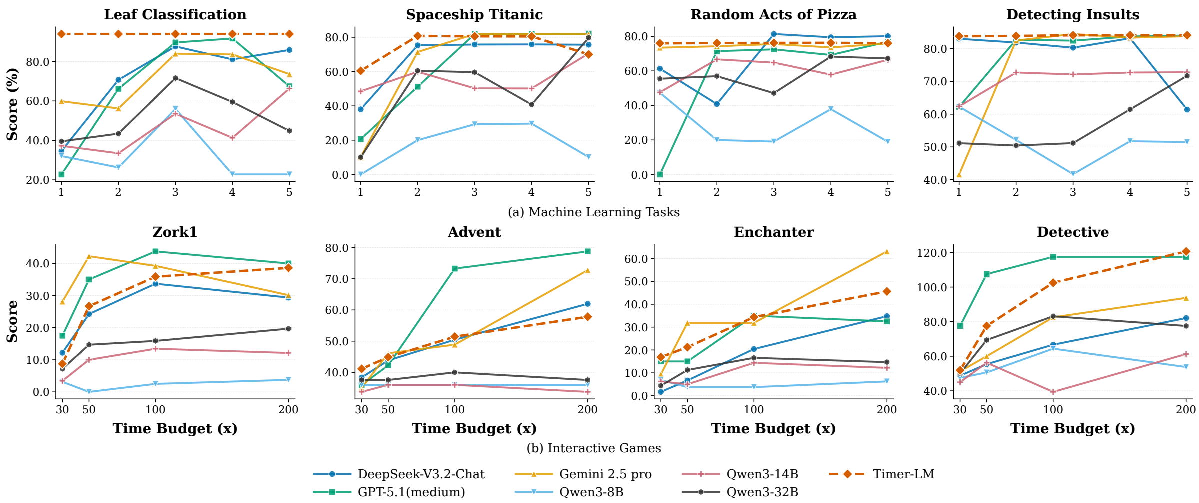

Figure 4: Scaling Experiments on ML tasks and Interactive Games.

round reasoning to meet time constraints. A similar pattern appears between Qwen3-8B and Qwen3-32B, 32B prefers single-round reasoning followed by direct summarization, while 8B is more willing to invoke tool calls before summarizing. The 14B model almost completely ignores the instruction requirements for tool calls, instead performing complete single-round reasoning directly.

5.4.2 Behavior under increased budgets

Moreover, Fig 3 shows the changes in reasoning length under different time budgets. We observe that TimelyLM exhibits a significant increasing trend in reasoning length as the time budget increases. In contrast, the Qwen series models do not show this trend, and their reasoning length increases marginally under different time budgets. This demonstrates that Timely-RL indeed enables the model to learn to adjust reasoning strategies based on changing time budgets.

5.5 Task-dependent budget sensitivity

Fig. 4 illustrates the impact of providing additional time budgets on performance for interactive games and ML tasks. On ML tasks, we observe little direct relationship between increased budgets and performance improvements. The first few rounds of generated code often determine the final performance. In contrast, on Interactive Games, additional time budgets lead to clear score improvements.

Therefore, tasks exemplified by Machine Learning rely more heavily on generation quality. Once a solution is good enough, further test-time scaling yields diminishing returns. Conversely, Interactive Games require multiple rounds of interaction to obtain rewards. For such exploratory tasks, the benefits of continued test-time scaling remain evident.

Method	AIME	GPQA	MATH
Qwen3-8B	30.0	36.0	48.0
+ SFT	46.7	48.0	56.0
+ GRPO	33.3	46.0	48.0
+ Timely-RL	**53.3**	**52.0**	**56.0**

Table 3: On-time completion rates under $0.75\times$ time budget. All methods start from Qwen3-8B.

5.6 Ablation Study on Timely-RL

Training Method Finally, Table 3 presents our ablation study on different stages of Timely-RL training. The results demonstrate that Timely-RL achieves notably higher on-time completion rates compared to SFT alone. Moreover, when applying standard RLVR training without time constraints on the SFT model, the model fails to learn comparable reasoning strategies. Therefore, we conclude that Timely-RL training effectively enhances the model's awareness of task time budgets.

Reward Design In addition to our sinusoidal reward design, we compare it against two alternative reward formulations, namely, a step-function reward and a linearly increasing reward, to justify the necessity of our design choice.

Specifically, for the linear reward function, given the task completion time t and the maximum allowed time $t_{\max}$, the model's utilization rate of time cost $U(t)$ is defined as:

$$U(t) = \min\left(\frac{t}{t_{\max}}, 1\right). \tag{11}$$

Table 4: Performance of different reward designs under a reduced inference budget(0.75x). **On Time**: percentage of samples finished within the time limit (%). **ACC**: accuracy (%). **Tool-use rate**: percentage of samples that invoked at least one tool call (%).

	AIME			GPQA-Diamond			MATH		
Method	**On Time**	**ACC**	**Tool-use rate**	**On Time**	**ACC**	**Tool-use rate**	**On Time**	**ACC**	**Tool-use rate**
Linear Reward	33.3	20.0	100.0	46.0	34.0	98.0	60.0	52.0	100.0
Step Reward	50.0	16.7	100.0	44.0	32.0	98.0	60.0	50.0	100.0
Ours	**53.3**	**33.3**	**100.0**	**52.0**	**36.0**	**100.0**	56.0	**52.0**	**100.0**

Table 5: Comparison of different reward function designs under the standard evaluation setting.

Method	GPQA-diamond	MATH	AIME
Linear Reward	**50.5**	74.5	35.9
Step Reward	42.0	69.0	28.4
Ours	49.5	**78.0**	**42.5**

For the step-function reward, we define $U(t)$ as the following piecewise function:

$$U(t) = \begin{cases} 0.25 & \text{if } r < 0.25 \\ 0.5 & \text{if } 0.25 \le r < 0.5 \\ 0.75 & \text{if } 0.5 \le r < 0.75 \\ 0.1 & \text{if } r \ge 0.75 \end{cases}, \quad r = \min\left(\frac{t}{t_{\max}}, 1\right). \tag{12}$$

Table 5 presents the main comparison results across different reward designs.

To further understand the behavior of different training strategies under a reduced inference budget, we report additional metrics, including on-time rate, accuracy, and tool-use rate, in Table 4.

As shown in Table 4, among the three methods, the sinusoidal-curve-based reward still achieves the best overall performance. For the linear reward, because it does not impose any additional preference toward making fuller use of the available time, the reward differences among solutions that finish relatively early are not substantial. As a result, the model is rewarded primarily for solving the problem correctly, while it is not sufficiently incentivized to make fuller and more efficient use of the available time. Consequently, on AIME, when the model is given a shorter inference budget, its on-time rate drops substantially.

The step reward suffers from a different issue. Since the reward is heuristically divided into several intervals, there is little distinction among trajectories that fall within the same interval. This lack of granularity weakens the incentive for the model to perform more thorough reasoning. As a result, its performance degrades even when sufficient reasoning budget is available.

In contrast, the sinusoidal reward design can both encourage more adequate reasoning and provide an appropriate incentive for the model to make greater use of the available time. This leads to the best overall performance among the compared methods.

6 Conclusion

In this work, we propose the concept of Timely Machine, arguing that test time in agentic scenarios should directly correspond to physical time itself. We advocate that LLMs should possess awareness of time budgets. To this end, we introduce Timely-RL to train models to better perceive task execution time and dynamically adjust their reasoning accordingly. We hope our work provides a new perspective for test-time scaling in agentic scenarios.

Limitations

Finally, we discuss several limitations of this work. First, our evaluation is currently conducted on text-based tasks. In multimodal interactive tasks, processing image or video streams typically introduces more complex and variable latencies, presenting a more challenging scenario. Second, our framework does not fully address cases where LLMs are deployed in multi-agent systems. In these multi-agent settings, latency may arise from another LLM's generation time, and training LLMs for effective collaboration in such contexts remains an open topic. Finally, an interesting observation is that while models become more timely under time constraints, accuracy improvements vary with benchmark difficulty. Future work could explore how to better balance the trade-off between time constraints and model performance.

References

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, and 1 others. 2024. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. 2024. Alphamath almost zero: process supervision without process. *Advances in Neural Information Processing Systems*, 37:27689–27724.

XTuner Contributors. 2023. Xtuner: A toolkit for efficiently fine-tuning llm. https://github.com/InternLM/xtuner.

Qi Fan, An Zou, and Yehan Ma. 2025. Timebill: Time-budgeted inference for large language models. *arXiv preprint arXiv:2512.21859*.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. 2025. rstar-math: Small llms can master math reasoning with self-evolved deep thinking. *arXiv preprint arXiv:2501.04519*.

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

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

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

Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2023. Large language models can self-improve. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 1051–1068.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. 2025. Aide: Ai-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*.

Peiji Li, Jiasheng Ye, Yongkang Chen, Yichuan Ma, Zijie Yu, Kedi Chen, Ganqu Cui, Haozhan Li, Jiacheng Chen, Chengqi Lyu, and 1 others. 2025a. Internbootcamp technical report: Boosting llm reasoning with verifiable task scaling. *arXiv preprint arXiv:2508.08636*.

Xiaozhe Li, Xinyu Fang, Shengyuan Ding, Linyang Li, Haodong Duan, Qingwen Liu, and Kai Chen. 2025b. Np-engine: Empowering optimization reasoning in large language models with verifiable synthetic np problems. *Preprint*, arXiv:2510.16476.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. 2025c. Torl: Scaling tool-integrated rl. *arXiv preprint arXiv:2503.23383*.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, and 1 others. 2025a. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

Tengxiao Liu, Zifeng Wang, Jin Miao, I Hsu, Jun Yan, Jiefeng Chen, Rujun Han, Fangyuan Xu, Yanfei Chen, Ke Jiang, and 1 others. 2025b. Budget-aware tool-use enables effective agent scaling. *arXiv preprint arXiv:2511.17006*.

Davide Paglieri, Bartłomiej Cupiał, Jonathan Cook, Ulyana Piterbarg, Jens Tuyls, Edward Grefenstette, Jakob Nicolaus Foerster, Jack Parker-Holder, and Tim Rocktäschel. 2025. Learning when to plan: Efficiently allocating test-time compute for llm agents. *arXiv preprint arXiv:2509.03581*.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*.

Yuxiao Qu, Matthew YR Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. 2025. Optimizing test-time compute via meta reinforcement fine-tuning. *arXiv preprint arXiv:2503.07572*.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

Jie Ren, Yao Zhao, Tu Vu, Peter J Liu, and Balaji Lakshminarayanan. 2023. Self-evaluation improves selective generation in large language models. In *Proceedings on*, pages 49–64. PMLR.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Haozhan Shen, Peng Liu, Jingcheng Li, Chunxin Fang, Yibo Ma, Jiajia Liao, Qiaoli Shen, Zilun Zhang, Kangjia Zhao, Qianqian Zhang, and 1 others. 2025. Vlm-r1: A stable and generalizable r1-style large vision-language model. *arXiv preprint arXiv:2504.07615*.

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

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. 2025. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, and 1 others. 2025. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, and 1 others. 2025. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*.

Fali Wang, Hui Liu, Zhenwei Dai, Jingying Zeng, Zhiwei Zhang, Zongyu Wu, Chen Luo, Zhen Li, Xianfeng Tang, Qi He, and 1 others. 2025a. Agentttts: Large language model agent for test-time compute-optimal scaling strategy in complex tasks. *arXiv preprint arXiv:2508.00890*.

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

Zili Wang, Tianyu Zhang, Haoli Bai, Lu Hou, Xianzhi Yu, Wulong Liu, Shiming Xiang, and Lei Zhu. 2025b. Faster and better llms via latency-aware test-time scaling. *arXiv preprint arXiv:2505.19634*.

Hao Wen, Xinrui Wu, Yi Sun, Feifei Zhang, Liye Chen, Jie Wang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. 2025. Budgetthinker: Empowering budget-aware llm reasoning with control tokens. *arXiv preprint arXiv:2508.17196*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason E Weston. 2024. Self-rewarding language models. In *Forty-first International Conference on Machine Learning*.

Shaolei Zhang, Ju Fan, Meihao Fan, Guoliang Li, and Xiaoyong Du. 2025. Deepanalyze: Agentic large language models for autonomous data science. *arXiv preprint arXiv:2510.16872*.

Xuechen Zhang, Zijian Huang, Ege Onur Taga, Carlee Joe-Wong, Samet Oymak, and Jiasi Chen. 2024. Efficient contextual llm cascades through budget-constrained policy learning. *Advances in Neural Information Processing Systems*, 37:91691–91722.

Han Zhao, Haotian Wang, Yiping Peng, Sitong Zhao, Xiaoyu Tian, Shuaiting Chen, Yunjie Ji, and Xiangang Li. 2025. 1.4 million open-source distilled reasoning dataset to empower large language model training. *Preprint*, arXiv:2503.19633.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, and 1 others. 2024. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583.

A Experimental Details

A.1 Details for Timely-Eval

In this section, we first introduce the implementation details of TIMELY-EVAL. Specifically, we elaborate on the evaluation protocols for three distinct benchmarks.

A.1.1 General Benchmarks

For general benchmarks, we provide the following system prompt to the model to enable time-aware strategic reasoning:

System Prompt for Time-Aware Strategic Reasoner

Act as a Time-Aware Strategic Reasoner. Your objective is to conquer complex challenges within a strictly enforced time window.
You must treat Time as your most critical resource. 1. **Initial Assessment**: Immediately gauge the problem's complexity against the remaining time. 2. **Cognitive Budgeting**: Allocate specific time slots for analyzing, utilizing tools, and synthesizing the final answer. Do not over-invest in low-value details. 3. **Dynamic Adjustment**: If a reasoning path proves too time-consuming, decisively pivot to a more efficient heuristic or alternative method. 4. **Convergence**: You must guarantee a complete conclusion before the deadline. A partial perfect derivation is a failure; a complete, logically sound conclusion is success.
Output Requirements: - Wrap your condensed final insights in `<summary>...</summary>` tags. - Record your actual elapsed duration in `<conclusion>total duration: {time} seconds</conclusion>` tags. Your final answer should be wrapped in `<answer>` and `</answer>` tags, for example, `<answer>\boxed{100}</answer>`.
Tools: You may call functions to assist with the user query. Available tool: `get_duration()` - Gets the total elapsed time (in seconds).

In our evaluation of the Qwen3 series models, all experiments were consistently conducted on hardware equipped with 8×NVIDIA H200 GPUs. To determine the time constraints for each benchmark, we first performed a baseline inference pass for every case without any time limits. We then established the target time limit by multiplying this baseline duration by a designated coefficient.

A.1.2 Interactive Games

For the interactive games task, our environment is built upon the Jericho framework, which supports 57 interactive fiction games. Models are given the initial observation of the environment, and asked to gain as much score as possible.

The training split specifically includes the following 16 games: *acorncourt, advent, adventureland, anchor, awaken, balances, ballyhoo, curses, cutthroat, deephome, detective, dragon, enchanter, hhgg, hollywood, and zork1*. For these games, the system prompt provided to the model is as follows:

Table 6: Full list of 57 games in the Jericho environment.

905	acorncourt	advent	adventureland
afflicted	anchor	awaken	balances
ballyhoo	curses	cutthroat	deephome
detective	dragon	enchanter	enter
gold	hhgg	hollywood	huntdark
infidel	inhumane	jewel	karn
lgop	library	loose	lostpig
ludicorp	lurking	moonlit	murdac
night	omniquest	partyfoul	pentari
planetfall	plundered	reverb	seastalker
sherlock	snacktime	sorcerer	spellbrkr
spirit	temple	theatre	trinity
tryst205	weapon	wishbringer	yomomma
zenon	zork1	zork2	zork3
ztuu			

System Prompt for Interactive Games

Act as a Time-Aware Strategic Reasoner. Your objective is to conquer complex challenges within a strictly enforced time window.
You must treat Time as your most critical resource. 1. **Initial Assessment**: Gauge complexity vs remaining time. 2. **Cognitive Budgeting**: Allocate specific time slots. 3. **Dynamic Adjustment**: Pivot to efficient heuristics if needed. 4. **Convergence**: Guarantee a conclusion before the deadline.
Output Requirements: - Record duration: `<conclusion>total duration: {time} seconds</conclusion>` - Summarize score: `<score>{your_score}</score>`

The model uses the following tools to interact with the engine:

- `step(action)`: Execute game action.
- `get_available_actions()`: Get valid moves.
- `get_score()`: Check current score.
- `get_max_score()`: Check goal score.
- `end_game()`: Terminate session.

Table 8 demonstrates the initial strategic reasoning of TIMELYLM-8B on *Zork1*.

A.1.3 Machine Learning Tasks

As the third component of TIMELY-EVAL, we select four representative machine learning (ML) tasks: *leaf_classification*, *spaceship_titanic*, *random_acts_of_pizza*, and *detecting_insults_in_social_commentary*. In these tasks, the model is required to perform end-to-end data science reasoning, including data preprocessing, model selection, training, and evaluation.

The system prompt for ML tasks is designed to emphasize time efficiency and rapid execution:

Interaction Example: TimelyLM-8B on MATH Dataset

User: The volume of the cylinder shown is 45π cubic cm. What is the height in centimeters of the cylinder? [asy] size(120); draw(shift(2.2,0)*yscale(0.3)*Circle((0,0), 1.2)); ... label("$r=3$",midpoint((2.2,0)–(3.4,0)),N); [/asy] Please answer the question within **5.31 seconds**.

Assistant: <think> I'll start by assessing the problem... The volume is $V = \pi r^2 h$. We have $V = 45\pi$ and $r = 3$. </think>
<tool_call> {"name": "get_duration", "arguments": {}} </tool_call>

User: <tool_response> 0.54 seconds. </tool_response>

Assistant: <think> Good, I have about 4.77 seconds left. $45\pi = \pi \cdot 3^2 \cdot h \Rightarrow 45 = 9h \Rightarrow h = 5$. </think>
<tool_call> {"name": "get_duration", "arguments": {}} </tool_call>

User: <tool_response> 1.36 seconds. </tool_response>

Assistant: <think> Still plenty of time. The calculation is correct. </think>
<summary> The volume is $V = \pi r^2 h$. With $V = 45\pi$ and $r = 3$, we have $45\pi = 9\pi h$, yielding $h = 5$ cm. </summary>
<conclusion> total duration: 1.36 seconds </conclusion>
<answer> $\boxed{5}$ </answer>

Table 7: A full reasoning trace of TimelyLM-8B. The model strategically monitors its remaining time using the `get_duration` tool to ensure completion within the 5.31s limit.

System Prompt for Machine Learning Tasks

Act as a Time-Aware Strategic Reasoner. Your objective is to conquer machine learning reasoning tasks within a strictly enforced time window.
You must treat Time as your most critical resource. 1. **Initial Assessment**: Immediately gauge complexity against remaining time. 2. **Cognitive Budgeting**: Allocate time slots for analysis and coding. 3. **Dynamic Adjustment**: Pivot to efficient solutions if a method is too slow. 4. **Convergence**: Guarantee a complete conclusion before the deadline.
Instructions: Wrap your code in <code>...</code> tags. Avoid long-running code or cross-validation.
Output Requirements: - <conclusion>total duration: {time} seconds</conclusion> - <accuracy>{accuracy_float}</accuracy>

In each round, the model generates an implementation plan and the corresponding Python code. The code is then executed via a specialized tool: `execute_code_and_get_duration(code)`. This tool returns the execution `stdout`, the final evaluation accuracy, and the time consumed.

Table 9 illustrates a complete execution trace of TimelyLM-8B on the *leaf classification* task.

A.1.4 Inference and Evaluation Setup

During the evaluation phase, all models were deployed on a hardware cluster equipped with 8×NVIDIA H200 GPUs. For high-throughput inference, we employed the SGLang framework (Zheng et al., 2024).

To balance efficiency and stability, the concurrency level (number of parallel requests) for model calls was set to 16 across all benchmarks. For tasks that exhibit inherent variance—specifically **Interactive Games** and **Machine Learning Tasks**—we evaluated each individual case across 8 independent runs. The reported results for these categories represent the aggregate performance across these iterations to ensure statistical significance and mitigate the impact of stochastic generation.

A.1.5 Time Budget Allocation Strategy

General Benchmarks For general tasks, the time budget is determined on a **per-case basis**. Let x denote the duration a model requires to complete a specific case during a baseline inference pass (without time constraints). The enforced time budget for that particular case is then defined as $n \times x$, where n is a configurable scaling factor. This approach ensures that the difficulty of each individual problem is intrinsically accounted for.

Interactive Games and ML Tasks In contrast, for Interactive Games and Machine Learning Tasks, we employ a **step-based budgeting strategy**. The total time budget is calculated by multiplying a measured average duration per step (derived from baseline performance) by a fixed constant. This ensures a consistent pressure on the model's strategic planning and tool-calling efficiency throughout the multi-turn interaction or the entire coding pipeline.

Reasoning Trace: TimelyLM-8B on Zork1 (Steps 0–3)

System: [Rules and Tool Definitions]
User: Game started. Observation: *West of House. You are standing in an open field west of a white house... There is a small mailbox here.* The time limit is **47.99 seconds**.

Assistant (Step 0): `<think>` I have 47.99s to maximize my score. Let me get the max score and available actions. `</think>`
`<tool_call>` {"name": "get_max_score"} `</tool_call>`
Tool Response: The max score is 350. You have played for 3.6 seconds.

Assistant (Step 1): `<tool_call>` {"name": "get_available_actions"} `</tool_call>`
Tool Response: Available actions: ['open mailbox', 'north', 'south', 'west']. You have played for 4.1 seconds.

Assistant (Step 2): `<think>` Max score 350. Time ≈ 44s. Let me check my score before starting. `</think>`
`<tool_call>` {"name": "get_score"} `</tool_call>`
Tool Response: Your current score is: 0. You have played for 4.6 seconds.

Assistant (Step 3): `<think>` Time is critical. I'll examine the mailbox first. `</think>`
`<tool_call>` {"name": "step", "arguments": {"action": "open mailbox"}} `</tool_call>`
Tool Response: Opening the small mailbox reveals a leaflet. Played for 6.0 seconds.

Table 8: Initial steps of TimelyLM-8B in Zork1. The model uses the first few seconds to calibrate its goals and state.

ML Reasoning Trace: TimelyLM-8B on Leaf Classification

User: The objective is to identify 99 species of plants using shape, margin, and texture features. You must produce `submission.csv` with predicted probabilities.
Data: `train.csv` (891 rows), `test.csv` (99 rows).
Please finish the task within **5.32 seconds**.

Assistant: `<think>` I'll use a Random Forest classifier. It handles multi-class tabular data effectively and trains quickly. I will: (1) Load data, (2) Encode labels, (3) Train a model with `n_jobs = -1`, (4) Generate probabilities. `</think>`

```python
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder

# Load and Prepare
train_df = pd.read_csv('./data/public/train.csv')
test_df = pd.read_csv('./data/public/test.csv')
X = train_df.drop(['id', 'species'], axis=1)
y = LabelEncoder().fit_transform(train_df['species'])

# Train and Predict
clf = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)
clf.fit(X, y)
X_test = test_df.drop(['id'], axis=1)
probs = clf.predict_proba(X_test)

# Submit
sub = pd.DataFrame(probs, columns=label_encoder.classes_)
sub.insert(0, 'id', test_df['id'])
sub.to_csv('./submission.csv', index=False)
```

`<tool_call>` {"name": "execute_code_and_get_duration", "arguments": {...}} `<tool_call>`

Tool Response: Code execution succeeded. Stdout: Submission file created successfully!
Evaluation accuracy: 0.9394. You have spent **5.22 seconds**.

Assistant: `<conclusion>` total duration: 5.22 seconds `</conclusion>`
`<accuracy>` 0.9394 `</accuracy>`

Table 9: An example of TimelyLM-8B solving an ML task. The model selects a computationally efficient Random Forest model to ensure the entire pipeline (loading, training, and prediction) completes within the strict 5.32s limit.

A.2 Details for Timely-RL

In this section, we provide a comprehensive description of the implementation and training details for TIMELY-RL. Our RL framework is built upon the `VeRL` (Sheng et al., 2024) and `InternBootCamp` frameworks. We designed three categories of tools and corresponding RL pipelines tailored to the three task types. For the sake of clarity, we restate the unified reward function introduced in Section 4.3 here:

$$R(t,r)=\begin{cases}0, & t>T_{\max}\\ r_{\mathrm{f}}, & t\le T_{\max}, r=0\\ r_{\mathrm{f}}+r+\lambda U(t), & t\le T_{\max}, r>0\end{cases} \tag{13}$$

where r_{f} is a fixed reward for correct formatting and timely completion, and $U(t)$ represents the model's utilization rate of time cost, calculated as:

$$U(t)=\sin\left(\frac{\pi}{2}\cdot\min\left(\frac{t}{T_{\max}},1\right)\right) \tag{14}$$

A.2.1 Timer Tool

We implemented a *Timer Tool* that supports a single function call: `get_duration()`. To simulate diverse execution environments, the tool's initialization allows for specific time scaling coefficients and stochastic perturbation factors (jitter). When the model calls this tool, the returned duration is the product of the actual elapsed time and a designated coefficient, plus a random noise component. This design enables TIMELY-RL to generalize across different computational resources and varying inference speeds.

A.2.2 General Reasoning RL

For general reasoning tasks, the *Timer Tool* serves as the primary interface for time-awareness. We deployed a dedicated service to manage timers for different cases. Before each `VeRL` rollout session, every case is registered with this service to ensure that the timing for each instance is strictly independent and isolated.

During the RL process, the model is encouraged to call `get_duration()` to monitor its own reasoning progress. Following the reward formulation in Eq. 13, for General Reasoning tasks, we set the parameters as follows: the task reward $r=0.5$ (for correct answers), the formatting reward $r_{\mathrm{f}}=0.1$, and the time-utilization weight $\lambda=0.4$.

A.2.3 Interactive Games RL

For the interactive games task, we deploy a specialized service to host the Jericho game engine. Specifically, an isolated game instance is initialized prior to each rollout session. The model interacts with this environment by invoking designated tools until the game reaches a terminal state or the model explicitly calls the `end_game` tool. We integrate the *Timer Tool* (described in Section A.2.1) directly into this service, ensuring that the model receives both environmental feedback and the total elapsed time in a single turn.

Critically, we differentiate between **inference time** (the latency of model generation) and **tool call time** (the latency of environment execution, including simulated tool delays). To simulate diverse reasoning speeds, we apply a scaling coefficient to the actual inference time, while the tool call duration remains unadjusted. This separation allows us to better analyze the model's strategic behavior under varied computational constraints. For interactive games, r is calculated as follows:

$$r=\left(\frac{s_{curr}}{s_{max}}\right)^{1/3} \tag{15}$$

where s_{curr} is the score gained in this game, and s_{max} denotes the maximum score of the game.

A.2.4 Machine Learning Tasks RL

Lastly, for machine learning tasks, we deploy a dedicated code execution service to process the model-generated scripts. Unlike interactive games, ML tasks do not require maintaining complex environmental state transitions; therefore, we simply instantiate a separate *Timer Tool* for each case to monitor execution duration.

In this category, the performance component r in Eq. 13 corresponds to the evaluation metric (accuracy) achieved by the model's code on the test set. Since all four selected ML benchmarks are classification tasks, r is naturally a floating-point value within the range $[0,1]$. To balance the total reward magnitude, we define the performance contribution to the final reward as $0.5\times r$.

A.3 Prompts for ML Tasks

We designed individual task prompts for the four selected machine learning benchmarks. The detailed prompts for each task are provided below.

Task 1: Detecting Insults in Social Commentary

Please first outline your proposed solution in natural language, followed by a single markdown code block.
The task description is: The challenge is to detect when a comment from a conversation would be considered insulting to another participant. The goal is to create a generalizable classifier for real-time moderation.
Note:
- Datasets `train.csv` and `test.csv` are in `./data/public/`.
- You MUST produce `./submission.csv` on the test split.
- Predict probability (0 to 1). Evaluated using AUC.
- `./submission.csv` format: `Insult,Date,Comment`

Task 2: Leaf Classification

Please first outline your proposed solution in natural language, followed by a single markdown code block.
The task description is: Use binary leaf images and extracted features (shape, margin, texture) to identify 99 species of plants.
Note:
- Datasets: `train.csv`, `test.csv`, and `images/` in `./data/public/`.
- Task: Predict probabilities for 99 species. Evaluated using multi-class logarithmic loss.
- Predicted probabilities should be clipped: $\max(\min(p, 1 - 10^{-15}), 10^{-15})$.
- `./submission.csv` format: `id, Acer_Capillipes, Acer_Circinatum, ...`

Task 3: Random Acts of Pizza (RAOP)

First outline your proposed solution in natural language, followed by a single markdown code block.
The task description is: Predict which textual requests for pizza from the Reddit community will be successful based on request text and meta-data.
Note:
- Datasets: `train.json` and `test.json` in `./data/public/`.
- Task: Predict probability of receiving pizza. Evaluated on ROC AUC.
- `./submission.csv` format: `request_id, requester_received_pizza`

Task 4: Spaceship Titanic

First outline your proposed solution in natural language, followed by a code block.
The task description is: Predict which passengers were transported to an alternate dimension after the *Spaceship Titanic* collided with a spacetime anomaly.
Note:
- Datasets: `train.csv` and `test.csv` in `./data/public/`.
- Task: Achieve best classification accuracy on test split.
- `./submission.csv` format: `PassengerId, Transported`

A.4 Prompts for Interactive Games

In this section, we present the initial observations and prompts received by the model during the first step of several representative games in the Jericho environment. Each task provides the model with a `max_score` target and a textual description of the starting environment.

Game: Zork1 (Target Score: 350)

```
Game started.  Initial observation:
Copyright (c) 1981, 1982, 1983 Infocom,
Inc.  All rights reserved.
ZORK is a registered trademark of
Infocom, Inc.
Revision 88 / Serial number 840726

West of House
You are standing in an open field west
of a white house, with a boarded front
door.  There is a small mailbox here.
```

Game: Detective (Target Score: 360)

```
Game started.  Initial observation:
Type "help" for more information about
this version
Detective.  By Matt Barringer.  Ported
by Stuart Moore.
Release 1 / Serial number 000715 /
Inform v6.21

« Chief's office »
You are standing in the Chief's office.
He is telling you "The Mayor was
murdered yesterday night at 12:03 am.
I want you to solve it before we get
any bad publicity or the FBI has to
come in." "Yessir!" You reply.  He
hands you a sheet of paper.  Once you
have read it, go north or west.

You can see a piece of white paper
here.
Your score has just gone up by ten
points.
```

Game: Advent (Target Score: 350)

```
Game started.  Initial observation:
Welcome to Adventure!  (Please type
HELP for instructions.)
ADVENTURE. The Interactive Original.
By Will Crowther (1976) and Don Woods
(1977).

At End Of Road
You are standing at the end of a road
before a small brick building.  Around
you is a forest.  A small stream flows
out of the building and down a gully.
```

Game: Enchanter (Target Score: 400)

```
Game started.  Initial observation:
"Krill's evil must be unmade," he
begins...  "Only then may his vast evil
be lessened or, with good fortune,
destroyed."

ENCHANTER. Infocom interactive fiction.
Copyright (C) 1983, 1984 by Infocom,
Inc.  All rights reserved.

Fork
You stand at a point of decision on a
road which makes a wide fork to the
northeast and southeast, circling the
base of the Lonely Mountain, which
looms high overhead to the east.  A
very long and winding road starts here
and stretches out of sight to the west
through low, smoky hills.  The sun is
rising over the lands to the east.
```

B Distillation Details

In this section, we provide the implementation details for the distillation process used to generate cold-start data, specifically focusing on the General Reasoning tasks.

To provide a high-quality cold-start foundation for TIMELY-RL, we developed a distillation pipeline using queries sampled from the AM-1.4M dataset. We utilized `Qwen3-235B-Instruct-2507` as the teacher model, equipped with a *Timer Tool* and instructed to complete tasks accurately within strictly enforced deadlines. To ensure the model generalizes across various hardware scenarios and reasoning speeds, we randomly sampled different time limits and timer acceleration coefficients during data synthesis.

For response verification, we employed a two-stage checking process:

1. **Rule-based Verification**: We first used the `math-verify`[2] library to perform symbolic and rule-based correctness checks.
2. **Model-based Verification**: Cases that failed rule-based checks were further evaluated by `Qwen3-235B`. Each case was checked 3 times; if all three evaluations passed, the solution was deemed accurate.

[2]https://github.com/huggingface/Math-Verify

The system prompt provided to the teacher model during data generation is shown below:

System Prompt for Distillation (Teacher Model)

Act as a Time-Aware Strategic Reasoner. Your objective is to conquer complex challenges within a strictly enforced time window.
You must treat Time as your most critical resource. 1. **Initial Assessment**: Immediately gauge complexity against remaining time. 2. **Cognitive Budgeting**: Allocate time slots for analysis and tools. 3. **Dynamic Adjustment**: Pivot to efficient heuristics if needed. 4. **Convergence**: Guarantee a complete conclusion before the deadline.
Output Requirements: - Wrap insights in <summary>...</summary> tags. - Record duration in <conclusion>total duration: {time} seconds</conclusion>. - Wrap final answer in <answer>\boxed{...}</answer>. - *Note*: Keep final conclusions brief. Split reasoning into multiple short steps and call the `get_duration` tool frequently to avoid estimation errors.

For the verification stage, the system prompt for the **judger model** is as follows:

System Prompt for Judger Model

You are an expert agent evaluating the timely reasoning tasks. You will be given the final answer and the correct answer, and you should evaluate the final answer and output the evaluation result.
Please answer only **yes** or **no** in the following format:
<answer>yes or no</answer>

C Broader Impacts

In this section, we discuss the broader implications, ethical considerations, and reproducibility of our work.

Ethics Statement All datasets utilized in this study are obtained from publicly available sources and are free for academic research. Our data synthesis and evaluation processes do not involve the collection or processing of any sensitive personally identifiable information (PII). Consequently, we do not foresee any significant privacy risks associated with the data used in this work.

Reproducibility We are committed to open-source and reproducible research. To facilitate the reproduction of our experimental results, we will release our complete source code, the synthesized distillation datasets, and the TIMELY-EVAL evaluation framework upon publication. Detailed implementation parameters and hardware configurations are also documented in the preceding sections of this appendix.

Contributions and Perspective The primary contribution of this research is the introduction of a novel perspective on *test-time scaling* within agentic scenarios. We posit that test-time in agentic contexts should be directly coupled with physical wall-clock time rather than purely symbolic computation steps. By providing a theoretical framework and designing the TIMELY-RL algorithm, we demonstrate that time-aware strategic reasoning is a distinct capability that can be significantly enhanced through specialized training. We hope this work inspires further research into making autonomous agents more efficient and reliable in time-sensitive real-world environments.