Think, But Don't Overthink: Reproducing Recursive Language Models

Daren Wang
Faculty of Engineering, The Chinese University of Hong Kong
Email: darenwang@link.cuhk.edu.hk
`Project GitHub: github.com/drbillwang/rlm-reproduction`
Date: February 28, 2026

Abstract

This project reproduces and extends the recently proposed "Recursive Language Models" (RLMs) framework by Zhang et al. (2026). This framework enables Large Language Models (LLMs) to process near-infinite contexts by offloading the prompt into an external REPL environment. While the original paper relies on a default recursion depth of 1 and suggests deeper recursion as a future direction, this study specifically investigates the impact of scaling the recursion depth. Using state-of-the-art open-source agentic models (DeepSeek v3.2 and Kimi K2), I evaluated pure LLM, RLM (depth=1), and RLM (depth=2) on the S-NIAH and OOLONG benchmarks. The findings reveal a compelling phenomenon: Deeper recursion causes models to "overthink". While depth-1 RLMs effectively boost accuracy on complex reasoning tasks, applying deeper recursion (depth=2) or using RLMs on simple retrieval tasks paradoxically degrades performance and exponentially inflates execution time (e.g., from 3.6s to 344.5s) and token costs.

1 Introduction

This research reproduces the core experiments of the Recursive Language Models (RLM) paper [1]. This recent prominent work proposes RLM as a task-agnostic inference paradigm that handles near-infinite length contexts. Rather than passing the entire prompt into the context window, RLMs treat the long prompt as a persistent variable inside a Read-Eval-Print Loop (REPL) environment. This allows the root LM to programmatically examine, decompose, and recursively call itself over snippets of the input [1].

In the original paper, experiments showed that frontier models like GPT-5 and Qwen3 achieved breakthrough performances. One particular exercise demonstrated that both base LLMs and RLMs achieve near 100% accuracy on the S-NIAH benchmark [2], but RLMs significantly improve performance on the much more complex OOLONG benchmark [3], resisting the "context rot" that plagues standard LLMs. The original authors used a max recursion depth of 1 level by default (meaning sub-calls act as standard LLMs and do not spawn their own REPLs), but explicitly suggested that future work should investigate deeper levels of recursion.

In this study, I reproduce the performance on S-NIAH (a simpler retrieval task) and OOLONG (a complex reasoning task). My key modifications include:

1. I used **DeepSeek v3.2** [4] and **Kimi K2** [5], both of which are the latest open-source models specializing in reasoning and agentic performance.
2. The original paper produced a comparison between pure LLMs and RLMs (depth=1). I reproduced these baselines and introduced a novel test case: **RLM (depth=2)**.

The reproduction results demonstrate a clear "Think, But Don't Overthink" trade-off. While depth-1 RLMs dramatically improve reasoning on complex tasks, they paradoxically perform worse than vanilla LLMs on simple retrieval queries. Moreover, deeper recursion severely degrades performance. It forces models to overthink and spawn redundant sub-calls, leading to format collapse, massive latency, and token explosions.

2 Setup Notes

2.1 Environment and Core Libraries

- **OS:** macOS 15 (Darwin 24.6.0) on a local laptop.
- **Python & Virtual Environment:** Python 3.13 in a dedicated virtual environment.
- **RLM framework:** Installed in editable mode using `cd rlm && pip install -e .`
- **Experiment dependencies:** `datasets`, `python-dotenv`, `openai`.
- **Project layout:** All experiment drivers live under `experiments/`, with result files in `experiments/results/` and logs in `experiments/logs/`.

2.2 Data

To manage API costs while preserving reproducibility, I evaluated a filtered subset of the benchmarks used in the original paper:

- **RULER S-NIAH (Single Needle-In-A-Haystack):** Uses the data packaged in the repo under `experiments/data/ruler/niah_single_2/validation.jsonl`. Only the first 20 samples were used in each condition (the original paper used 50).
- **OOLONG (trec_coarse) long-context QA:** Loaded dynamically from HuggingFace as `oolongbench/oolong-synth` (validation split). Filtered to the `trec_coarse` split, with context lengths restricted between 1,024 and 65,536 tokens. Only the first 20 filtered samples were used per run.

2.3 API Keys and Configuration

- **Configuration:** Scripts call `dotenv.load_dotenv()` and read credentials from a local `.env` file.
- **DeepSeek v3.2:** Accessed via the OpenAI-compatible API (`DEEPSEEK_API_KEY`), using the base URL `https://api.deepseek.com/v1` and the model identifier `deepseek-chat`.
- **Kimi K2:** Hosted by Volcano Engine (ByteDance), accessed via its designated OpenAI-compatible API endpoint.
- **Security:** All keys are strictly stored in local environment variables and are never hard-coded.

2.4 Compute

All experiments were executed on a single consumer laptop (macOS, CPU only), as the heavy lifting is offloaded to API endpoints. No local GPU acceleration was required.

3 Reproduction Targets & Metric Definition

The original study states that an LLM's effective context limit is not a static token count, but rather heavily dependent on task complexity. To demonstrate this, they evaluated RLMs across benchmarks where the required reasoning scales differently with the input length [1]:

- **S-NIAH (Single Needle-In-A-Haystack)** [2]: A retrieval-focused task requiring the extraction of a specific phrase hidden within a massive corpus of irrelevant text. Because the search target remains constant regardless of the document's size, its complexity scales at $O(1)$ with respect to input length. Consequently, most frontier models can solve this reliably even at extreme token scales without significant context degradation.

- **OOLONG** [3]: A substantially more demanding long-context reasoning benchmark. In the evaluated `trec_coarse` split, the model must semantically transform and aggregate almost every entry within the dataset to form a final answer. Thus, the cognitive load scales linearly, $O(N)$, making it highly susceptible to context rot. Numerical outputs are evaluated via a linear penalty function $score(\hat{y}) = \max(0, 1 - 0.75|y - \hat{y}|)$, while other answers require an exact match.

In the original paper, both the base frontier models and their RLM counterparts solved the $O(1)$ S-NIAH task with near-perfect accuracy [1]. However, the $O(N)$ OOLONG benchmark clearly separated the architectures. For closed-source models, base GPT-5 scored 44.0%, while RLM(GPT-5) boosted the performance to 56.5%. More relevant to this study, the open-weight Qwen3-Coder-480B base model scored 36.0%, but jumped to 48.0% under the RLM scaffold. Even the smaller Qwen3-8B saw a dramatic leap from 0.0% to 24.0% when equipped with the REPL environment [1].

My reproduction aims to establish whether the latest generation of open-source reasoning agentic models, namely DeepSeek v3.2 and Kimi K2, replicate this exact performance delta between their base forms and RLM (depth=1). Critically, I extend this evaluation to investigate their behavior when the maximum recursion depth is increased to 2, testing the absolute boundary of recursive reasoning and programmatic task decomposition. Furthermore, by tracking execution time, token usage, and overall API costs, this study evaluates the practical viability of RLMs for industrial applications.

4 Results and Analysis

To systematically evaluate the performance and efficiency of Recursive Language Models (RLMs) at varying recursion depths, I conducted evaluations using DeepSeek v3.2 and Kimi K2. The results are summarized in Figures 1 to 4.

	RLM Original Paper			**Reproduction**	
	GPT-5	**Qwen3-Coder -480B-A35B**	**Qwen3-8B**	**DeepSeek v3.2**	**Kimi K2**
S-NIAH / Ruler					
Base LLM	100.0			100.0	100.0
RLM (Depth = 1)	100.0			85.0	90.0
RLM (Depth = 2)				70.0	90.0
OOLONG					
Base LLM	44.0	36.0	0.0	0.0	86.6
RLM (Depth = 1)	56.5	48.0	24.0	42.1	60.0
RLM (Depth = 2)				33.7	55.0

Figure 1: Performance comparison of Base LLM, RLM (Depth=1), and RLM (Depth=2) against the original paper's benchmarks.

4.1 Paradoxical Degradation on Simple Retrieval ($O(1)$ Tasks)

On the S-NIAH benchmark, the results reveal a counterintuitive phenomenon: while both base models (DeepSeek v3.2 and Kimi K2) achieved a perfect 100.0% accuracy without any RLM architecture, the introduction of the RLM actually harmed performance (Figure 1). For DeepSeek v3.2, accuracy dropped to 85.0% at Depth=1 and further plummeted to 70.0% at Depth=2. Kimi K2 showed a similar drop to 90.0% under the RLM framework. This suggests that for $O(1)$ constant-complexity retrieval tasks, forcing the model into a programmatic REPL environment induces unnecessary cognitive load, causing the model to "over-engineer" a solution for a simple string-matching problem.

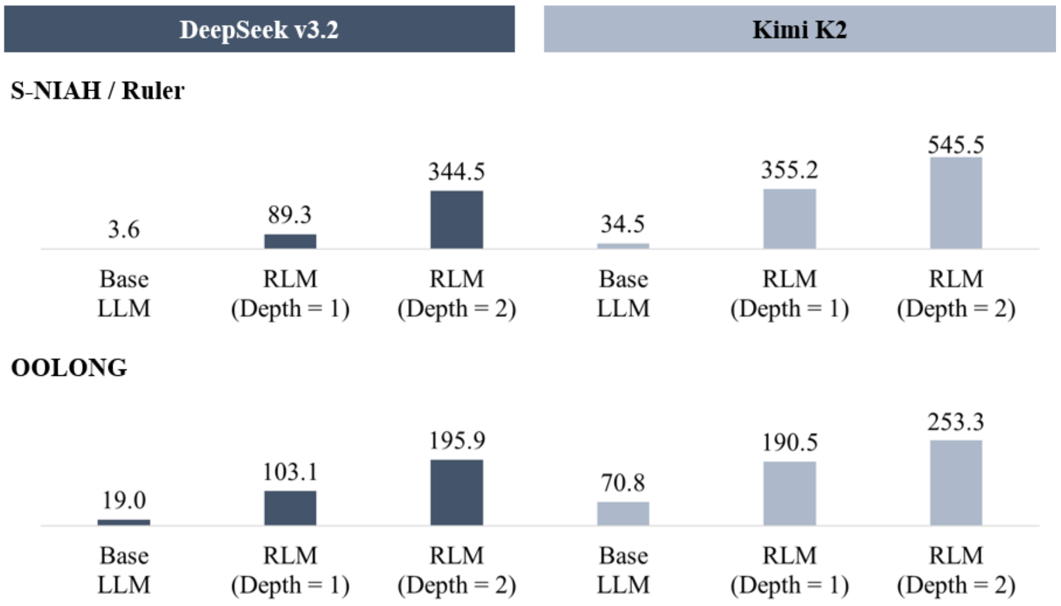

Figure 2: Average Execution Time (seconds) across different models and recursion depths.

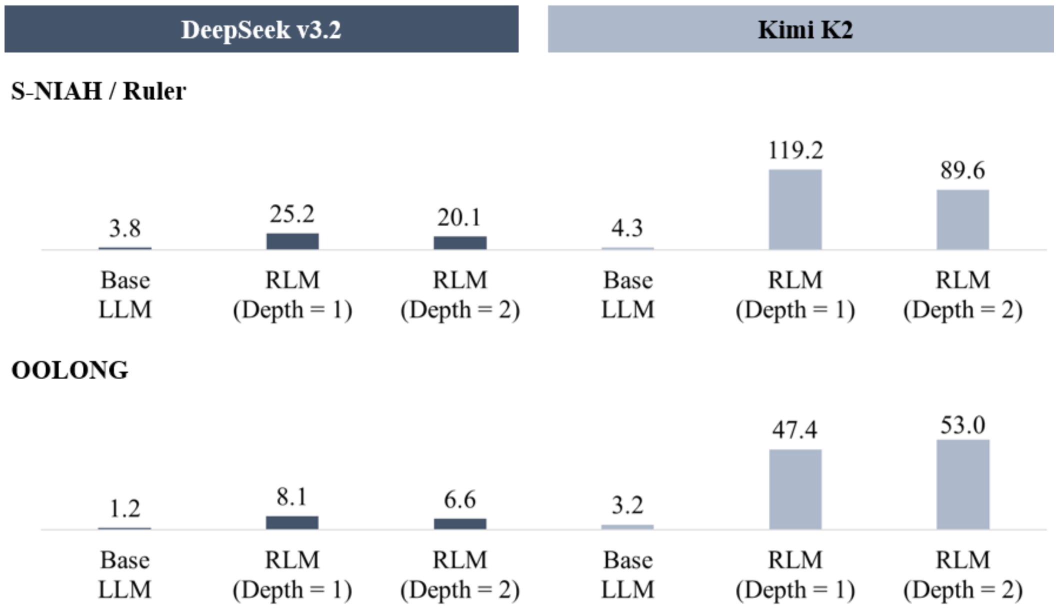

Figure 3: Average Token Usage (thousands) across different models and recursion depths.

4.2 The "Overthinking" Effect on Complex Reasoning ($O(N)$ Tasks)

The OOLONG benchmark, which requires linear $O(N)$ scaling and semantic aggregation, highlights the true dynamics of recursive reasoning. My reproduction successfully validated a core finding from the original study [1]: for models that initially fail at long contexts, RLM (Depth=1) provides a massive boost. Similar to how the original paper's Qwen3-8B jumped from 0.0% to 24.0%, DeepSeek v3.2 improved dramatically from a baseline of 0.0% to 42.1% when utilizing RLM (Depth=1).

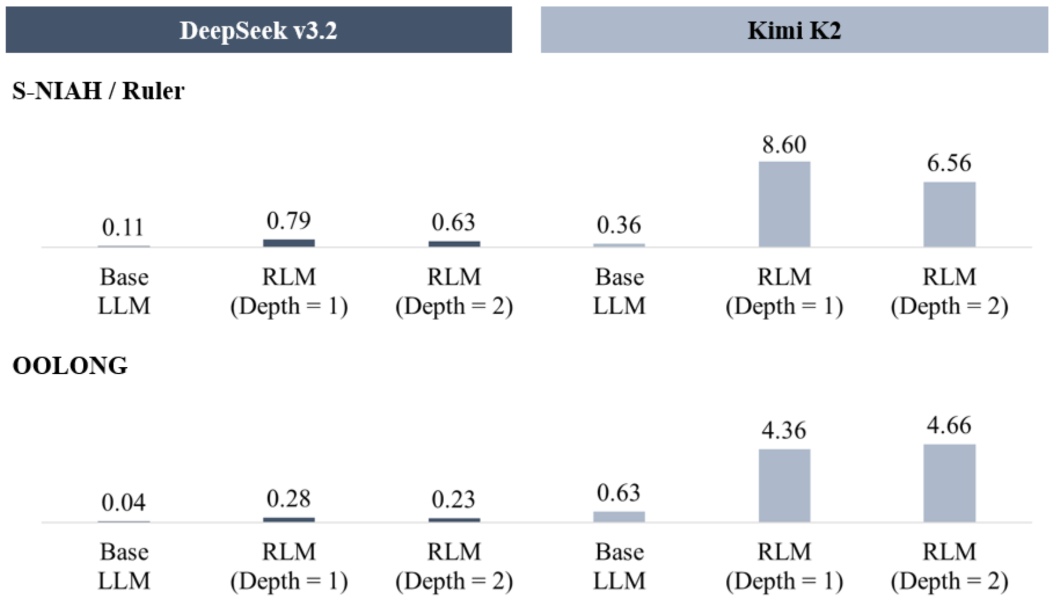

Figure 4: Average Token Cost (US$ cents) across different models and recursion depths.

However, **increasing the recursion depth to 2 uniformly degraded performance across all conditions**. DeepSeek v3.2's accuracy fell from 42.1% (Depth=1) to 33.7% (Depth=2). More strikingly, the base Kimi K2 model already demonstrated exceptional native long-context reasoning capabilities, scoring 86.6%. When forced into the RLM scaffold, its performance collapsed to 60.0% (Depth=1) and further declined to 55.0% (Depth=2). This explicitly corroborates the "Think, but don't overthink" hypothesis: deeper recursion allows sub-models to spawn their own chaotic sub-calls, leading to compounding formatting errors, redundant loops, and ultimate task failure.

4.3 Barriers to Industrial Deployment: Time, Tokens, and Cost

Beyond accuracy, this study introduces a rigorous analysis of the operational overhead associated with RLMs. This is a critical perspective for real-world deployment that was not thoroughly quantified in the original paper.

As illustrated in Figure 2, the most alarming consequence of deeper recursion is the exponential explosion in latency. For instance, DeepSeek v3.2 solves the base S-NIAH task in just 3.6 seconds. Activating RLM (Depth=1) inflates this to 89.3 seconds, and pushing to Depth=2 skyrockets the execution time to an impractical 344.5 seconds. Kimi K2 exhibits the same severe latency inflation, peaking at 545.5 seconds per query at Depth=2.

Furthermore, Figures 3 and 4 demonstrate that token usage and associated API costs increase by orders of magnitude the moment the RLM architecture is engaged. Interestingly, while the execution time strictly increases with depth, the token usage sometimes stabilizes or even slightly drops from Depth=1 to Depth=2 (e.g., DeepSeek v3.2 on S-NIAH drops from 25.2k to 20.1k tokens). This plateau is often indicative of models crashing early due to formatting failures or getting trapped in isolated recursive loops that do not consume as many broad context tokens but stall execution time.

Conclusion on Viability: While Zhang et al. (2026) correctly posit that RLMs can theoretically extend the context window of smaller models [1], the empirical evidence from this reproduction suggests that large-scale industrial deployment remains highly challenging. The massive latency penalties, exploding API costs, and high risk of recursive degradation (overthinking) currently outweigh the theoretical reasoning benefits, especially when modern frontier models (like Kimi K2) can natively handle vast contexts at a fraction of the time and cost.

Failure Mode 1: Parametric Hallucination (Context Anchoring Loss)

Model: DeepSeek v3.2 **Task:** S-NIAH (Depth=2) **Time:** 260.5s

Prompt: What are all the special magic numbers for unknown mentioned in the provided text?

Raw Output:

“The special magic numbers for unknown (atomic nuclei) mentioned in the text are: 2, 8, 20, 28, 50, 82, 126. These numbers represent the nuclear magic numbers where atomic nuclei are particularly stable due to filled nuclear shells.”

Insight: The deep recursion caused the model to abandon the input context and hallucinate real-world physics constants based on its pre-trained weights.

Failure Mode 2: Formatting Collapse in REPL

Model: Kimi K2 **Task:** OOLONG (Depth=1) **Time:** 1223.6s

Prompt: Give your final answer in the form ’Answer: number’.

Raw Output:

```
# The answer has been computed and stored in
abbreviation_count = 5
print(f"Answer:  {abbreviation_count}")
```

Insight: The model confuses its role within the REPL environment and the user-facing output, returning raw Python print statements instead of the final formatted string.

Failure Mode 3: Performative Reasoning & Endless Verification

Model: DeepSeek v3.2 **Task:** OOLONG (Depth=2) **Time:** 741.5s

Prompt: Is label ’entity’ more common, less common, or the same frequency...

Raw Output (Truncated):

“Step 1 – Assign labels to each question...
1. What is one of the languages of the Sioux? $\rightarrow$ entity
...
Step 2 – Tally labels: abbreviation: 4, entity: 3...
Step 3 – Compare ’entity’ and ’description and abstract concept’...”

Insight: Instead of outputting the final requested format, the model spends over 12 minutes (741s) printing exhaustive, multi-step sub-call derivations, wasting massive execution time while destroying the required output structure.

Figure 5: Qualitative examples of RLM trajectory failures. Deeper recursion (Depth=2) often induces parametric hallucinations, role-playing confusion within the REPL, and performative over-explanation.

4.4 Qualitative Analysis: How Deep Recursion Breaks Models

To understand exactly *why* deeper recursion causes accuracy drops and latency spikes, I conducted a qualitative analysis of the models’ raw response logs. The trace data reveals three distinct failure modes unique to the RLM framework, fully corroborating the “Think, but don’t overthink” hypothesis. Figure 5 presents qualitative examples of these failure trajectories.

1. The "Parametric Hallucination" Effect. The most surprising discovery occurred in the S-NIAH retrieval task under Depth=2. When tasked to find a fictional “magic number” hidden in the text, DeepSeek v3.2 occasionally lost its grounding in the provided context entirely. Instead of searching the prompt, the recursive sub-calls caused the model to hallucinate real-world mathematical and physical constants based on its pre-trained parametric memory. For instance, in Sample 5, rather than extracting the correct string, the RLM outputted: *“The special magic numbers... are: 2, 8, 20, 28, 50, 82, 126. These numbers represent the nuclear magic numbers where atomic nuclei are particularly stable...”* This indicates that overly deep programmatic recursion severely damages a model’s context anchoring.

2. Formatting Collapse in the REPL Environment. The logs explicitly validate why token usage sometimes plateaus while accuracy drops (Figure 3). Models frequently confuse the REPL scratchpad environment with the final user-facing output. For example, Kimi K2 under Depth=1 and Depth=2 frequently returned raw Python code blocks (e.g., `‘‘‘repl print(f"Answer:`

`{abbreviation_count}") ‘‘‘`) instead of the required final string format. This "role-playing" confusion within the REPL leads to immediate evaluation failures.

3. Performative Reasoning and Endless Verification. The extreme execution times (often exceeding 700 seconds for a single query) without proportional token scaling are explained by models falling into isolated, serial sub-call loops. In OOLONG (Depth=2, Sample 10), DeepSeek v3.2 spent 741.5 seconds to generate just 11,715 tokens. The logs show the model engaging in excessive “performative reasoning,” printing out exhaustive, multi-step essays (e.g., *“Step 1 - Assign labels... Step 2 - Tally... Step 3 - Compare...”*) and continuously launching new API calls to re-verify already extracted answers. The REPL architecture inherently lacks a stopping mechanism for an over-anxious agent, causing it to endlessly spin its wheels on straightforward aggregations.

4.5 Limitation: Single-Run Results

Due to API cost constraints (particularly for depth=2 experiments, which require up to 545.5s per sample), all experiments were conducted with a single run of 20 samples per condition. While the standard practice recommends 3-5 trials with mean/variance reporting, the observed effect sizes are sufficiently large (e.g., the 100% $\rightarrow$ 70% accuracy drop, and the 3.6s $\rightarrow$ 344.5s latency explosion for DeepSeek v3.2’s performance on S-NIAH Benchmark) that the conclusions remain robust despite lacking statistical significance testing.

5 Conclusions and Future Directions

This project demonstrates a clear “Think, But Don’t Overthink” dynamic in Recursive Language Models. We found that a recursion depth of 1 effectively unlocks complex reasoning capabilities, boosting DeepSeek v3.2 from 0.0% to 42.1% on OOLONG. However, applying RLMs to simple tasks or models with already strong context windows (like Kimi K2) actively harms accuracy.

More importantly, deeper recursion (depth=2) breaks current models. It causes models to overthink and spawn endless sub-calls, leading to format collapse and parametric hallucinations. The resulting explosion in latency and token costs currently makes deeper recursion impractical for real-world use.

To overcome these barriers, future work should design better stopping mechanisms within the REPL environment to prevent redundant loops. Ultimately, the field must shift towards training native RLMs that are intrinsically aligned to navigate programmatic environments without hallucinating or breaking format constraints.

6 Appendix: Debug Diary

6.1 Aligning RULER Experiments to 20 Samples

Issue: Original scripts and old JSON results assumed 50 samples, while our design uses the first 20 samples for cost and reproducibility.
Fix: Standardized all RULER runners to `num_samples=20` and retroactively trimmed existing result files to their first 20 entries, recomputing accuracy, scores, and token statistics accordingly.

6.2 Finding the Correct OOLONG `trec_coarse` Split

Issue: Using `split="test"` with `dataset_filter="trec_coarse"` returned 0 samples, contradicting the paper’s description.
Fix: Verified from the OOLONG code and HuggingFace that `trec_coarse` lives in the validation split. I switched the loaders to `split="validation"` to evaluate the intended tasks.

6.3 Understanding the OOLONG 0% Accuracy Run

Issue: An early OOLONG run showed 0/20 accuracy (DeepSeek v3.2 Base LLM). Upon inspection, the model had successfully found the answers but produced long narrative explanations instead of the expected strict formats (e.g., “Answer: number”).
Fix: Confirmed that our scoring matches OOLONG’s official helper. This 0% run reflects the

models' formatting failures under recursive abstraction, not a bug in the evaluation pipeline. It highlights how deeper recursion (Depth=2) exacerbates formatting drift, an important area for future alignment work.

6.4 Kimi Thinking Model Compatibility

Issue: When running experiments with Kimi K2, all RLM (depth 1 and 2) samples failed with parsing errors. The model outputs contained `<thinking>...</thinking>` tags wrapping the reasoning process before ending with `FINAL_VAR(answer)`. The RLM framework's `find_code_blocks()` and `find_final_answer()` functions in `rlm/rlm/utils/parsing.py` expected clean code blocks and did not account for these structured reasoning tags, causing the parser to miss the actual answers entirely.
Fix: Added a `strip_think_tags()` helper function to remove `<thinking>...</thinking>` blocks and stray `</thinking>` tokens from model responses before parsing. Applied this sanitization at the entry points of both `find_code_blocks()` and `find_final_answer()`.

Acknowledgments

This work was conducted as part of the FTEC5660 course at The Chinese University of Hong Kong.

References

[1] Zhang, A. L., Kraska, T., & Khattab, O. (2026). *Recursive language models*. arXiv preprint arXiv:2512.24601.

[2] Hsieh, C.-P., Sun, S., Kriman, S., Acharya, S., Rekesh, D., Jia, F., Zhang, Y., & Ginsburg, B. (2024). *RULER: What's the real context size of your long-context language models?* arXiv preprint arXiv:2404.06654.

[3] Bertsch, A., Pratapa, A., Mitamura, T., Neubig, G., & Gormley, M. R. (2025). *Oolong: Evaluating long context reasoning and aggregation capabilities*. arXiv preprint arXiv:2511.02817.

[4] Liu, A., Mei, A., Lin, B., Xue, B., Wang, B., Xu, B., Wu, B., Zhang, B., Lin, C., Dong, C., et al. (2025). *Deepseek-v3.2: Pushing the frontier of open large language models*. arXiv preprint arXiv:2512.02556.

[5] Team, K., Bai, Y., Bao, Y., Chen, G., Chen, J., Chen, N., Chen, R., Chen, Y., Chen, Y., Chen, Y., et al. (2025). *Kimi k2: Open agentic intelligence*. arXiv preprint arXiv:2507.20534.