RECURSIVE LANGUAGE MODELS

Alex L. Zhang
MIT CSAIL
altzhang@mit.edu

Tim Kraska
MIT CSAIL
kraska@mit.edu

Omar Khattab
MIT CSAIL
okhattab@mit.edu

ABSTRACT

We study allowing large language models (LLMs) to process arbitrarily long prompts through the lens of inference-time scaling. We propose **Recursive Language Models** (**RLM**s), a general inference strategy that treats long prompts as part of an external *environment* and allows the LLM to *programmatically* examine, decompose, and recursively call itself over snippets of the prompt. We find that RLMs successfully handle inputs up to two orders of magnitude beyond model context windows and, even for shorter prompts, dramatically outperform the quality of base LLMs and common long-context scaffolds across four diverse long-context tasks, while having comparable (or cheaper) cost per query.

1 INTRODUCTION

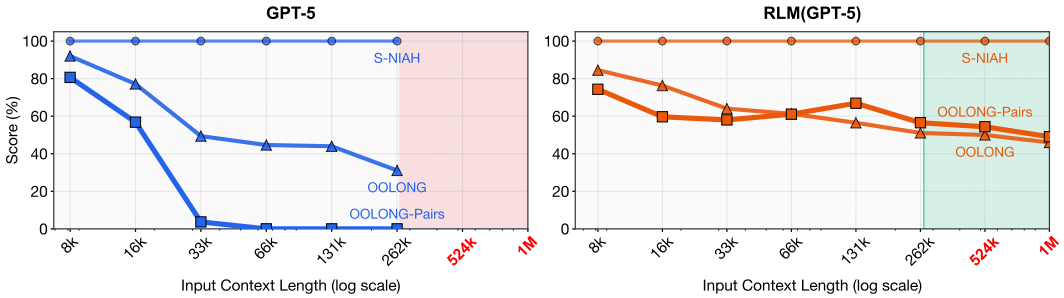

Figure 1: A comparison of GPT-5 and a corresponding RLM on three long-context tasks of increasing complexity: **S-NIAH**, **OOLONG**, and **OOLONG-Pairs**. For each task, we scale the input length from 2^{13} to 2^{18}. GPT-5 performance degrades significantly as a function of both input length and task complexity, while the RLM maintains strong performance. Inputs beyond the red region do not fit in GPT-5's context window of 272K tokens, but the RLM handles them effectively. Additional experiments across other models, methods, and benchmarks are in §2.

Despite rapid progress in reasoning and tool use, modern language models still have limited context lengths and, even within these limits, appear to inevitably exhibit *context rot* (Hong et al., 2025), the phenomenon illustrated in the left-hand side of Figure 1 where the quality of even frontier models like GPT-5 degrades quickly as context gets longer. Though we expect context lengths to steadily rise through improvements to training, architecture, and infrastructure, we are interested in *whether it possible to dramatically scale the context size of general-purpose LLMs by orders of magnitude*. This is increasingly urgent as LLMs begin to be widely adopted for long-horizon tasks, in which they must routinely process tens if not hundreds of millions of tokens.

We study this question through the lens of scaling inference-time compute. We draw broad inspiration from *out-of-core* algorithms, in which data-processing systems with a small but fast main memory can process far larger datasets by cleverly managing how data is fetched into memory. Inference-time methods for dealing with what are in essence long-context problems are very common, though typically task-specific. One general and increasingly popular inference-time approach in this space is context condensation or compaction (Khattab et al., 2021; Smith, 2025; OpenAI, 2025; Wu et al., 2025), in which the context is repeatedly summarized once it exceeds a length threshold. Unfortunately, compaction is rarely expressive enough for tasks that require dense access

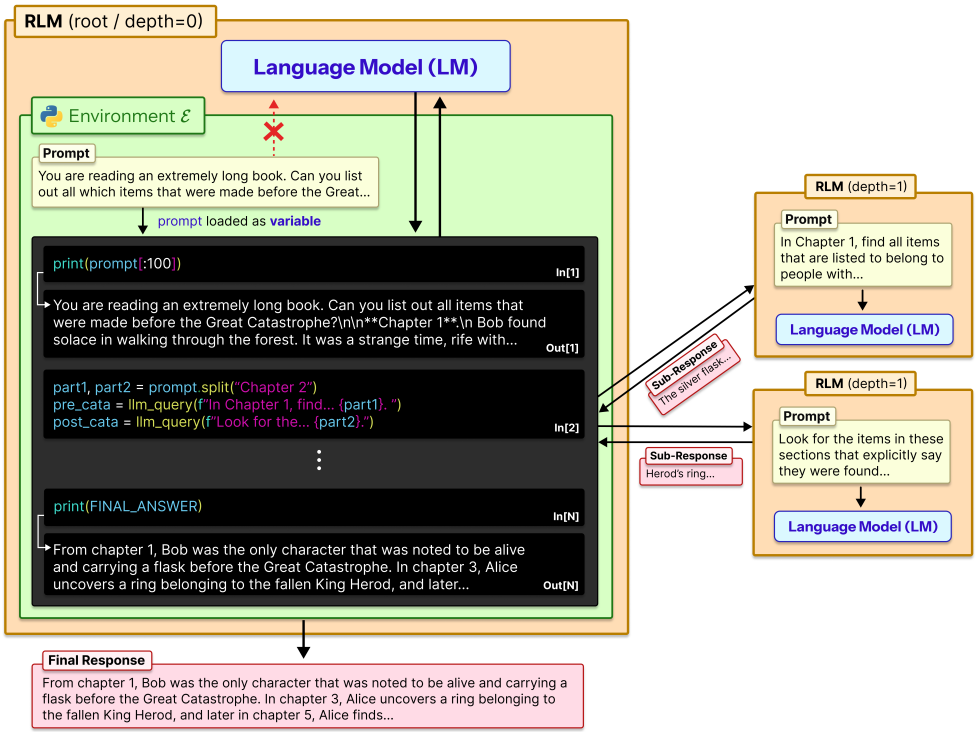

Figure 2: A Recursive Language Model (RLM) treats prompts as part of the environment. It loads the input prompt as a variable inside a Python REPL environment $\mathcal{E}$ and writes code to peek into, decompose, and invoke itself recursively over programmatic snippets of the variable.

to many parts of the prompt, as it presumes in effect that *some* details that appear early in the prompt can safely be forgotten to make room for new content.

We introduce **Recursive Language Models** (**RLM**s), a general-purpose inference paradigm for dramatically scaling the effective input and output lengths of modern LLMs. The key insight is that long prompts should not be fed into the neural network (e.g., Transformer) directly but should instead be treated as *part of the environment that the LLM can symbolically interact with*.

As Figure 2 illustrates, an RLM exposes the same external interface as an LLM: it accepts a string prompt of arbitrary structure and produces a string response. Given a prompt P, the RLM initializes a Read-Eval-Print Loop (REPL) programming environment in which P is set as the value of a variable. It then offers the LLM general context about the REPL environment (e.g., the length of the string P), and permits it to write code that peeks into and decomposes P, and to iteratively observe any side effects from execution. Crucially, RLMs encourage the LLM, in the code it produces, to programmatically construct sub-tasks on which they can invoke themselves recursively.

By treating the prompt as an object in the external environment, this simple design of RLMs tackles a foundational limitation in the many prior approaches (Anthropic, 2025; Sentient, 2025; Schroeder et al., 2025; Sun et al., 2025), which focus on recursive decomposition of the *tasks* but cannot allow their input to scale beyond the context window of the underlying LLM.

We evaluate RLMs using a frontier closed model (GPT-5; OpenAI 2025) and a frontier open model (Qwen3-Coder-480B-A35B; Team 2025) across four diverse tasks with varying levels of complexity for deep research (Chen et al., 2025), information aggregation (Bertsch et al., 2025), code repository understanding (Bai et al., 2025), and a synthetic pairwise reasoning task where even frontier models fail catastrophically. We compare RLMs against direct LLM calls as well as context compaction, retrieval tool-use agents, and code-generation agents. We find that RLMs demonstrate extremely strong performance even at the 10M+ token scale, and dramatically outperform all other approaches at long-context processing, in most cases by double-digit percentage gains while maintaining a comparable or lower cost. In particular, as demonstrated in Figure 1 exhibit far less severe degradation for longer contexts and more sophisticated tasks.

2 SCALING LONG CONTEXT TASKS

Recent work (Hsieh et al., 2024; Goldman et al., 2025; Hong et al., 2025) has successfully argued that the *effective* context window of LLMs can often be much shorter than a model's physical maximum number of tokens. Going further, we hypothesize that the effective context window of an LLM cannot be understood independently of the *specific task*. That is, more "complex" problems will exhibit degradation at even *shorter* lengths than simpler ones. Because of this, we must characterize tasks in terms of how their complexity *scales with prompt length*.

For example, needle-in-a-haystack (NIAH) problems generally keep 'needles' constant as prompt length is scaled. As a result, while previous generations of models struggled with NIAH tasks, frontier models can reliably solve these tasks in RULER (Hsieh et al., 2024) even in the 1M+ token settings. Nonetheless, the same models struggle even at shorter lengths on OOLONG (Bertsch et al., 2025), which is a task where the answer depends explicitly on almost every line in the prompt.[1]

2.1 TASKS

Grounded in this intuition, we design our empirical evaluation around tasks where we are able to vary not just the lengths of the prompts, but also consider different scaling patterns for problem complexity. We loosely characterize each task by *information density*, i.e. how much information an agent is required to process to answer the task, and how this scales with different input sizes.

S-NIAH. Following the single needle-in-the-haystack task in RULER (Hsieh et al., 2024), we consider a set of 50 single needle-in-the-haystack tasks that require finding a specific phrase or number in a large set of unrelated text. These tasks require finding a single answer regardless of input size, and as a result scale roughly constant in processing costs with respect to input length.

BrowseComp-Plus (1K documents) (Chen et al., 2025). A multi-hop question-answering benchmark for DeepResearch (OpenAI, 2025) questions that requires reasoning over multiple different documents. The benchmark provides a verified offline corpus of 100K documents that is guaranteed to contain gold, evidence, and hard negative documents for each task. Following Sun et al. (2025), we use 150 randomly sampled tasks as our evaluation set; we provide 1000 randomly chosen documents to the model or agent, in which the gold and evidence documents are guaranteed to exist. We report the percentage of correct answers. The answer to each task requires piecing together information from several documents, making these tasks more complicated than **S-NIAH** despite also requiring a constant number of documents to answer.

OOLONG (Bertsch et al., 2025). A long reasoning benchmark that requires examining and transforming chunks of the input semantically, then aggregating these chunks to form a final answer. We report scoring based on the original paper, which scores numerical answers as $\texttt{score}(\hat{y}) = 0.75^{|y-\hat{y}|}$ and other answers as exact match. We focus specifically on the `trec_coarse` split, which is a set of 50 tasks over a dataset of questions with semantic labels. Each task requires using nearly all entries of the dataset, and therefore scales linearly in processing costs relative to the input length.

OOLONG-Pairs. We manually modify the `trec_coarse` split of OOLONG to include 20 new queries that specifically require aggregating *pairs* of chunks to construct the final answer. In Appendix E.1, we explicitly provide all queries in this benchmark. We report F1 scores over the answer. Each task requires using nearly all *pairs* of entries of the dataset, and therefore scales quadratically in processing costs relative to the input length.

LongBench-v2 CodeQA (Bai et al., 2025). A multi-choice code repository understanding split from LongBench-v2 that is challenging for modern frontier models. We report the score as the percentage of correct answers. Each task requires reasoning over a fixed number of files in a codebase to find the right answer.

[1] This intuition helps explain the patterns seen in Figure 1 earlier: GPT-5 scales effectively on the S-NIAH task, where the needle size is constant despite longer prompts, but shows faster degradation at increasingly *shorter* context lengths on the *linear* complexity OOLONG and the *quadratic* complexity OOLONG-Pairs.

2.2 METHODS AND BASELINES

We compare RLMs against other commonly used task-agnostic methods. For each of the following methods, we use two contemporary LMs, GPT-5 with medium reasoning (OpenAI, 2025) and default sampling parameters and Qwen3-Coder-480B-A35B (Yang et al., 2025) using the sampling parameters described in Team (2025), chosen to provide results for a commercial and open frontier model respectively. For Qwen3-Coder, we compute costs based on the Fireworks provider (Fireworks, 2025). In addition to evaluating the base model on all tasks, we also evaluate the following methods and baselines:

RLM with REPL. We implement an RLM that loads its context as a string in the memory of a Python REPL environment. The REPL environment also loads in a module that allows it to query a sub-LM inside the environment. The system prompt is fixed across all experiments (see Appendix D). For the GPT-5 experiments, we use GPT-5-mini for the recursive LMs and GPT-5 for the root LM, as we found this choice to strike a powerful tradeoff between the capabilities of RLMs and the cost of the recursive calls.

RLM with REPL, no sub-calls. We provide an ablation of our method. In it, the REPL environment loads in the context, but is not able to use sub-LM calls. In this setting, the LM can still interact with its context in a REPL environment before providing a final answer.

Summary agent. Following Sun et al. (2025); Wu et al. (2025); Yu et al. (2025), we consider an iterative agent that invokes a summary of the context as it is filled. For example, given a corpus of documents, it will iteratively view the documents and summarize when full. In cases where the provided context exceeds the model window, the agent will chunks the input to fit within the model context window and invoke the same strategy over these chunks. For GPT-5, due to the extremely high cost of handling large token inputs, we use GPT-5-nano for compaction and GPT-5 to provide the final answer.

CodeAct (+ BM25). We compare directly to a CodeAct (Wang et al., 2024) agent that can execute code inside of a ReAct (Yao et al., 2023) loop. Unlike an RLM, it does not offload its prompt to the code environment, and instead provides it directly to the LM. Furthermore, following Jimenez et al. (2024); Chen et al. (2025), we equip this agent with a BM25 (Robertson & Zaragoza, 2009) retriever that indexes the input context for tasks where this is appropriate.

3 RESULTS AND DISCUSSION

We focus our main experiments in Table 1 on the benchmarks described in §2.1. Furthermore, we explore how frontier model and RLM performance degrades as input contexts grow in Figure 1.

Table 1: Performance comparison of different methods across long-context benchmarks of varying complexity. In gray is the average API cost ± the standard deviation of each method on each task. * indicates runs where the method ran into input context limits.

Model	**CodeQA**	**BrowseComp+ (1K)**	**OOLONG**	**OOLONG-Pairs**
Task Length N (tokens)	23K-4.2M	6M-11M	131K	32K
Qwen3-Coder-480B				
Base Model	20.00* ($0.13 ± $0.08)	0.00* (N/A) ± (N/A)	36.00 ($0.06 ± $0.00)	0.06 ($0.05 ± $0.01)
CodeAct (+ BM25)	24.00* ($0.17 ± $0.08)	12.66 ($0.39 ± $0.50)	38.00 ($1.51 ± $1.09)	0.28 ($1.54 ± $0.35)
Summary agent	50.00 ($1.26 ± $1.50)	38.00 ($8.98 ± $2.12)	44.06 ($0.15 ± $0.01)	0.31 ($0.05 ± $0.00)
RLM	56.00 ($0.92 ± $1.23)	44.66 ($0.84 ± $0.63)	**48.00** ($0.61 ± $0.49)	**23.11** ($1.02 ± $0.52)
RLM (no sub-calls)	**66.00** ($0.18 ± $0.58)	**46.00** ($0.82 ± $0.69)	43.50 ($0.32 ± $0.13)	17.34 ($1.77 ± $1.23)
GPT-5				
Base Model	24.00* ($0.13 ± $0.07)	0.00* (N/A) ± (N/A)	44.00 ($0.14 ± $0.02)	0.04 ($0.16 ± $0.10)
CodeAct (+ BM25)	22.00* ($0.06 ± $0.08)	51.00 ($0.71 ± $1.20)	38.00 ($0.61 ± $1.06)	24.67 ($0.75 ± $0.43)
Summary agent	58.00 ($1.31 ± $1.46)	70.47 ($0.57 ± $0.10)	46.00 ($0.13 ± $0.01)	0.01 ($0.13 ± $0.09)
RLM	**62.00** ($0.11 ± $0.10)	**91.33** ($0.99 ± $1.22)	**56.50** ($0.43 ± $0.85)	**58.00** ($0.33 ± $0.20)
RLM (no sub-calls)	58.00 ($0.18 ± $0.56)	88.00 ($0.44 ± $0.90)	36.00 ($0.37 ± $0.42)	43.93 ($0.69 ± $1.16)

Observation 1: RLMs can scale to the 10M+ token regime and can outperform base LMs and existing task-agnostic agent scaffolds on long context tasks. Across all tasks, RLMs demonstrate strong performance on input tasks well beyond the effective context window of a frontier LM, outperforming base models and common long-context scaffolds by up to $2\times$ the performance while maintaining comparable or cheaper average token costs. Notably, RLMs scale well to the theoretical costs of extending a base model's context window – on BrowseComp-Plus (1K), the cost of GPT-5-mini ingesting 6-11M input tokens is $\$1.50 - \$2.75$, while RLM(GPT-5) has an average cost of $\$0.99$ and outperforms both the summarization and retrieval baselines by over 29%.

Furthermore, on tasks where processing costs scale with the input context, RLMs make significant improvements over the base model on tasks that fit well within the model's context window. On OOLONG, the RLM with GPT-5 and Qwen3-Coder outperform the base model by 28.4% and 33.3% respectively. On OOLONG-Pairs, both GPT-5 and Qwen3-Coder make little progress with F1 scores of $<0.1\%$, while the RLM using these models achieve F1 scores of 58.00% and 23.11% respectively, highlighting the emergent capability of RLMs to handle extremely information-dense tasks.

Observation 2: The REPL environment is necessary for handling long inputs, while the recursive sub-calling of RLMs provides strong benefits on information-dense inputs. A key characteristic of RLMs is offloading the context as a variable in an environment $\mathcal{E}$ that the model can interact with. Even without sub-calling capabilities, our ablation of the RLM is able to scale beyond the context limit of the model, and outperform the base model and other task-agnostic baselines on most long context settings. On the CodeQA and BrowseComp+ tasks with Qwen3-Coder, this ablation is able to outperform the RLM by 17.9% and 3% respectively.

On information-dense tasks like OOLONG or OOLONG-Pairs, we observed several cases where recursive LM sub-calling is necessary. In §3.1, we see RLM(Qwen3-Coder) perform the necessary semantic transformation line-by-line through recursive sub-calls, while the ablation without sub-calls is forced to use keyword heuristics to solve these tasks. Across all information-dense tasks, RLMs outperform the ablation without sub-calling by 10%-59%.

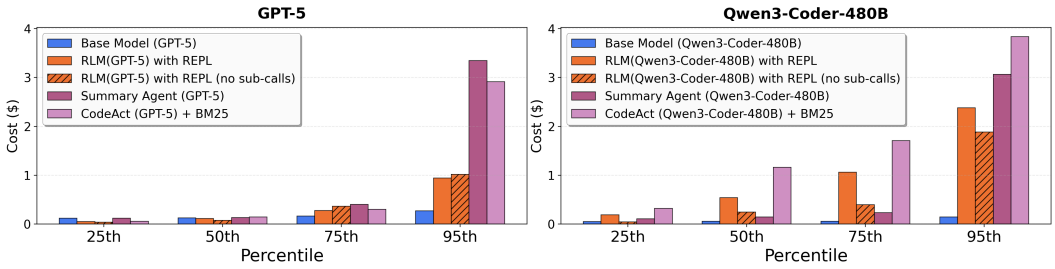

Figure 3: Cost of RLM and baselines described in §2.2 plotted at the 25th, 50th, 75th, and 95th percentile of total API cost. We observe comparable or even lower costs for RLMs at the 50th percentile, but sharp increases at the tail end due to potentially long RLM trajectories.

Observation 3: LM performance degrades as a function of input length and problem complexity, while RLM performance scales better. The benchmarks S-NIAH, OOLONG, and OOLONG-Pairs contain a fixed number of tasks over a context with lengths ranging from 2^{13} to 2^{18}. Furthermore, each benchmark can be loosely categorized by different processing costs of the input context with respect to length (roughly constant, linear, and quadratic respectively). In Figure 1, we directly compare an RLM using GPT-5 to base GPT-5 on each task – we find that GPT-5 performance degrades significantly faster for more complex tasks, while RLM performance degrades but at a much slower rate, which aligns with the findings of Goldman et al. (2025). For context lengths beyond 2^{14}, the RLM consistently outperforms GPT-5.

Furthermore, RLM costs scale proportionally to the the complexity of the task, while still remaining in the same order of magnitude of cost as GPT-5 (see Figure 9 in Appendix C). In §3.1, we explore what choices the RLM makes in these settings that causes these differences in cost. Lastly, in this setting, we also observe that the base LM outperforms RLM in the small input context regime. By construction, an RLM has strictly more representation capacity than an LM: the choice of an environment that calls the root LM is equivalent to the base LM; in practice, however, we observe that

RLM performance is slightly worse on smaller input lengths, suggesting a tradeoff point between when to use a base LM and when to use an RLM.

Observation 4: The inference cost of RLMs remain comparable to a base model call but are high variance due to differences in trajectory lengths. RLMs iteratively interact with their context until they find a suitable answer, leading to large differences in iteration length depending on task complexity. In Figure 3, we plot the quartile costs for each method across all experiments in Table 1 excluding BrowseComp-Plus (1K), as the base models cannot fit any of these tasks in context. For GPT-5, the median RLM run is cheaper than the median base model run, but many outlier RLM runs are significantly more expensive than any base model query. However, compared to the summarization baseline which ingests the entire input context, RLMs are up to $3\times$ cheaper while maintaining stronger performance across all tasks because the model is able to selectively view context.

We additionally report runtime numbers of each method in Figures 5, 6 in Appendix C, but we note several important caveats. Unlike API costs, these numbers are heavily dependent on implementation details such as the machine used, API request latency, and the asynchrony of LM calls. In our implementation of the baselines and RLMs, all LM calls are blocking / sequential. Nevertheless, similar to costs, we observe a wide range of runtimes, especially for RLMs.

Observation 5: RLMs are a model-agnostic inference strategy, but different models exhibit different overall decisions on context management and sub-calling. While GPT-5 and Qwen3-Coder-480B both exhibit strong performance as RLMs relative to their base model and other baselines, they also exhibit different performance and behavior across all tasks. On BrowseComp-Plus in particular, RLM(GPT-5) nearly solves all tasks while RLM(Qwen3-Coder) struggles to solve half.

We note that the RLM system prompt is fixed for each model across all experiments and is not tuned for any particular benchmark. Between GPT-5 and Qwen3-Coder, the only difference in the prompt is an extra line in the RLM(Qwen3-Coder) prompt warning against using too many sub-calls (see Appendix D). We provide an explicit example of this difference in example B.3, where RLM(Qwen3-Coder) performs the semantic transformation in OOLONG as a separate sub-LM call per line while GPT-5 is conservative about sub-querying LMs.

3.1 Emergent Patterns in RLM Trajectories

Even without explicit training, RLMs exhibit interesting context management and problem decomposition behavior. We select several examples of snippets from RLM trajectories to understand how they solve long context problems and where they can improve. We discuss particular examples of interesting behavior here, with additional examples in Appendix B.

Filtering input information using code execution based on model priors. A key intuition for why the RLM abstraction can maintain strong performance on huge inputs without exploding costs is the LM's ability to filter input context without explicitly seeing it. Furthermore, model priors enable the RLM to narrow the search space and process fewer input tokens. As an example, in Figure 4a, we observed RLM(GPT-5) using `regex` queries search for chunks containing keywords in the original prompt (e.g. "festival") and phrases it has a prior about (e.g. "La Union"). Across most trajectories, a common strategy we observed was probing the context by printing a few lines back to the root LM, then filtering based on its observations.

Chunking and recursively sub-calling LMs. RLMs defer essentially unbounded-length reasoning chains to sub-(R)LM calls. The choice of decomposition can greatly affect task performance, especially for information-dense problems. In our experiments, we did not observe complicated partitioning strategies beyond uniform chunking or keyword searches. In Figure 4b, RLM(Qwen3-Coder) chunks by newline in a 1000+ line context from OOLONG.

Answer verification through sub-LM calls with small contexts. We observed several instances of answer verification made by RLMs through sub-LM calls. Some of these strategies implicitly avoid context rot by using sub-LMs to perform verification (see example B.1), while others solely use code execution to programmatically verify answers are correct. In some instances, however, the answer verification is redundant and significantly increases the cost per task — in example B.3, we observed a trajectory on OOLONG where the model tries to reproduce its correct answer more than five times before choosing the incorrect answer in the end.

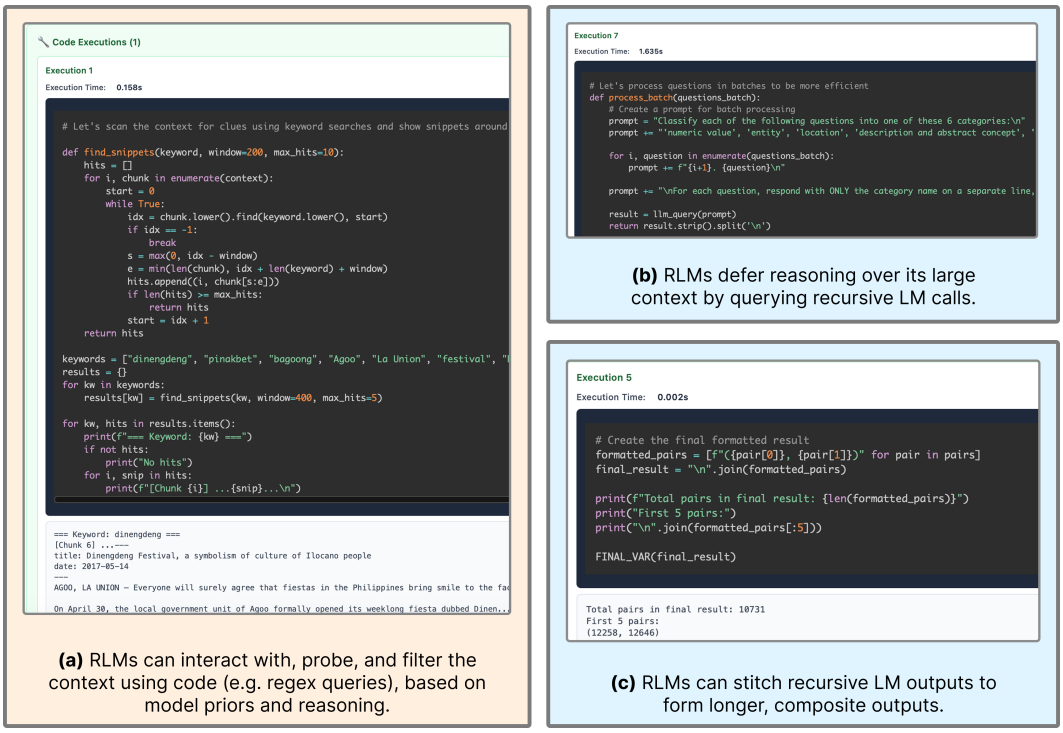

Figure 4: RLMs have common patterns in their trajectories when solving tasks. (a) We frequently observed RLMs filtering and interacting with their context through code like `regex` queries. (b) We found that RLMs can effectively decompose their context through recursive sub-calls (c) On long-output tasks, RLMs are able to solve sub-problems using recursive sub-LM calls and stitch their outputs to form a final output.

Passing recursive LM outputs through variables for long output tasks. RLMs are able to produce essentially unbounded tokens well beyond the limit of the base LM by returning variables in the REPL as output. Through the REPL, the RLM can iteratively construct these variables as a mixture of programmatic and sub-(R)LM output calls. We observed this strategy used heavily in OOLONG-Pairs trajectories, where the RLM stored the output of sub-LM calls over the input in variables and stitched them together to form a final answer (see Figure 4c).

4 Related Works

Long Context LM Systems. There have primarily been two orthogonal directions for long context management in language model systems: 1) directly changing the architecture of and retraining the base LM to handle longer contexts (Press et al., 2022; Gu et al., 2022; Munkhdalai et al., 2024), and 2) building a scaffold around the LM that implicitly handles the context – RLMs focus on the latter. One popular class of such strategies is *lossy* context management, which uses summarization or truncation to compress the input context at the cost of potentially losing fine-grained information. For example, MemWalker (Chen et al., 2023) constructs a tree-like data structure of the input that the LM can navigate when answering long context questions. ReSum (Wu et al., 2025) is another work that adds a summarization tool to periodically compress the context of a multi-turn agent. Another class of strategies implement an explicit memory hierarchy in the agent scaffold (Packer et al., 2024; Chhikara et al., 2025; Zhang et al., 2025). RLMs are different from prior work in that all context window management is implicitly handled by the LM itself.

Task Decomposition through sub-LM calls. Many LM-based agents (Guo et al., 2024; Anthropic, 2025) use multiple, well-placed LM calls to solve a problem, however many of these calls are placed based on human-engineered workflows. Several methods like ViperGPT Surís et al. (2023), THREAD (Schroeder et al., 2025), DisCIPL (Grand et al., 2025), ReDel Zhu et al. (2024), Context

Folding (Sun et al., 2025), and AgentFold (Ye et al., 2025) have explored deferring the choice of sub-LM calls to the LM. These techniques emphasize *task* decomposition through recursive LM calls, but are unable to handle long context inputs beyond the length of the base LM. RLMs, on the other hand, are enabled by an extremely simple intuition (i.e., placing the prompt as part of the external environment) to *symbolically* manipulate arbitrarily long strings and to iteratively refine their recursion via execution feedback from the persistent REPL environment.

5 Limitations and Future Work

While RLMs show strong performance on tasks beyond the context window limitations of existing LMs at reasonable inference costs, the optimal mechanism for implementing RLMs remains under-explored. We focused on synchronous sub-calls inside of a Python REPL environment, but we note that alternative strategies involving asynchronous sub-calls and sandboxed REPLs can potentially significantly reduce the runtime and inference cost of RLMs. Furthermore, we chose to use a max recursion depth of one (i.e. sub-calls are LMs); while we found strong performance on existing long-context benchmarks, we believe that future work should investigate deeper layers of recursion.

Lastly, we focused our experiments on evaluating RLMs using *existing* frontier models. Explicitly training models to be used as RLMs (e.g. as root or sub-LMs) could provide additional performance improvements – as we found in §3.1, current models are inefficient decision makers over their context. We hypothesize that RLM trajectories can be viewed as a form of reasoning (OpenAI et al., 2024; DeepSeek-AI et al., 2025), which can be trained by bootstrapping existing frontier models (Zelikman et al., 2022; 2024).

6 Conclusion

We introduced Recursive Language Models (RLMs), a general inference framework for language models that offloads the input context and enables language models to recursively sub-query language models before providing an output. We explored an instantiation of this framework that offloads the context into a Python REPL environment as a variable in memory, enabling the LM to reason over its context in code and recursive LM calls, rather than purely in token space. Our results across multiple settings and models demonstrated that RLMs are an effective task-agnostic paradigm for both long-context problems and general reasoning. We are excited to see future work that explicitly trains models to reason as RLMs, which could result in another axis of scale for the next generation of language model systems.

Acknowledgments

This research is partially supported by the Laude Institute. We thank Noah Ziems, Jacob Li, James Moore, and the MIT OASYS and MIT DSG labs for insightful discussions throughout this project. We also thank Matej Sirovatka, Ofir Press, Sebastian Müller, Simon Guo, and Zed Li for helpful feedback.

References

Anthropic. Claude code: Subagents — modular ai workflows with isolated agent contexts, 2025. URL https://docs.anthropic.com/en/docs/claude-code/sub-agents.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks, 2025. URL https://arxiv.org/abs/2412.15204.

Amanda Bertsch, Adithya Pratapa, Teruko Mitamura, Graham Neubig, and Matthew R. Gormley. Oolong: Evaluating long context reasoning and aggregation capabilities, 2025. URL https://arxiv.org/abs/2511.02817.