Recursive Language Models

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 paradigm 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 can successfully process inputs more than an order of magnitude beyond model context window limits and, even for shorter prompts, dramatically outperform the quality of vanilla frontier LLMs and common long-context and coding scaffolds (e.g., on GPT-5 by a median across the evaluated benchmarks of 26% against compaction, 130% against CodeAct with sub-calls, and 13% against Claude Code) across four diverse long-context tasks while having comparable cost. At a small scale, we post-train the first model around the RLM. Our model, **RLM-Qwen3-8B**, outperforms the underlying Qwen3-8B model by a median of 28% and even approaches the quality of vanilla GPT-5 on three long-context tasks. Code is available at `https://github.com/alexzhang13/rlm`.

Recursion in Language Models

Less is More: Recursive Reasoning with Tiny Networks

Alexia Jolicoeur-Martineau
Samsung SAIL Montréal
alexia.j@samsung.com

Abstract

Hierarchical Reasoning Model (HRM) is a novel approach using two small neural networks recursing at different frequencies. This biologically inspired method beats Large Language models (LLMs) on hard puzzle tasks such as Sudoku, Maze, and ARC-AGI while trained with small models (27M parameters) on small data (~ 1000 examples). HRM holds great promise for solving hard problems with small networks, but it is not yet well understood and may be suboptimal. We propose Tiny Recursive Model (TRM), a much simpler recursive reasoning approach that achieves significantly higher generalization than HRM, while using a single tiny network with only 2 layers. With only 7M parameters, TRM obtains 45% test-accuracy on ARC-AGI-1 and 8% on ARC-AGI-2, higher than most LLMs (e.g., Deepseek R1, o3-mini, Gemini 2.5 Pro) with less than 0.01% of the parameters.

1. Introduction

While powerful, Large Language models (LLMs) can struggle on hard question-answer problems. Given that they generate their answer auto-regressively, there is a high risk of error since a single incorrect token can render an answer invalid. To improve their reliability, LLMs rely on Chain-of-thoughts (CoT) (Wei et al., 2022) and Test-Time Compute (TTC) (Snell et al., 2024). CoTs seek to emulate human reasoning by having the LLM to sample step-by-step reasoning traces prior to giving their answer. Doing so can improve accuracy, but CoT is expensive, requires high-quality reasoning data (which may not be available), and can be brittle since the generated reasoning may be wrong. To further improve reliability, test-time compute can be used by reporting the most common answer out of K or the highest-reward answer (Snell et al., 2024).

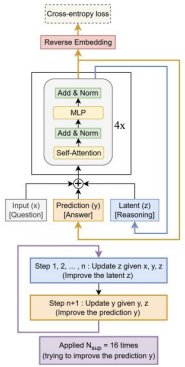


Figure 1. Tiny Recursion Model (TRM) recursively improves its predicted answer y with a tiny network. It starts with the embedded input question x and initial embedded answer y, and latent z. For up to $N_{sup} = 16$ improvements steps, it tries to improve its answer y. It does so by i) recursively updating n times its latent z given the question x, current answer y, and current latent z (recursive reasoning), and then ii) updating its answer y given the current answer y and current latent z. This recursive process allows the model to progressively improve its answer (potentially addressing any errors from its previous answer) in an extremely parameter-efficient manner while minimizing overfitting.

Andy Konwinski About Me

Recursive LLM Prompts

Mar 20, 2023

(The following is copied from the readme at github.com/andyk/recursive_llm); Last updated: April 3, 2023

The idea here is to implement recursion using English as the programming language and an LLM (e.g., GPT-3.5) as the runtime.

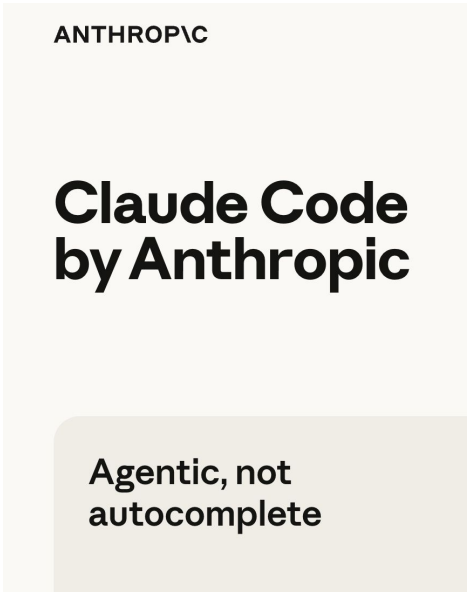
Basically we come up with an LLM prompt which causes the model to return another slightly updated prompt. More specifically, the prompts contain state and each recursively generated prompt updates that state to be closer to an end goal (i.e., a base case).

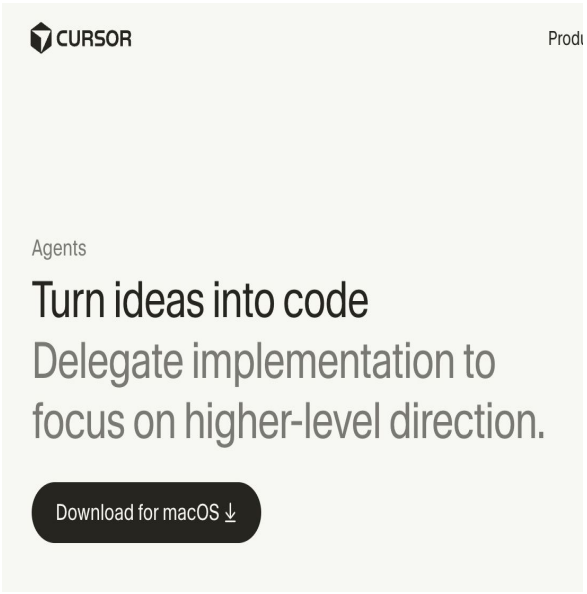
It's kind of like recursion in code, but instead of having a function that calls itself with a different set of arguments, there is a prompt that returns itself with specific parts updated to reflect the new arguments.

For simplicity, let's start without a base case; here is an infinitely recursive fibonacci prompt:

> *You are a recursive function. Instead of being written in a programming language, you are written in English. You have variables FIB_INDEX = 2, MINUS_TWO = 0, MINUS_ONE = 1, CURR_VALUE = 1. Output this paragraph but with updated variables to compute the next step of the Fibbonaci sequence.*

Context Management is Crucial!





LLMs Struggle with Long Context

Prelude: Why is “long-context” research so unsatisfactory?

There is this well-known but difficult to characterize phenomenon in language models (LMs) known as “context rot”. Anthropic defines context rot as “[when] the number of tokens in the context window increases, the model’s ability to accurately recall information from that context decreases”, but many researchers in the community know this definition doesn’t *fully* hit the mark. For example, if we look at popular needle-in-the-haystack benchmarks like RULER, most frontier models actually do extremely well (90%+ on 1-year old models).

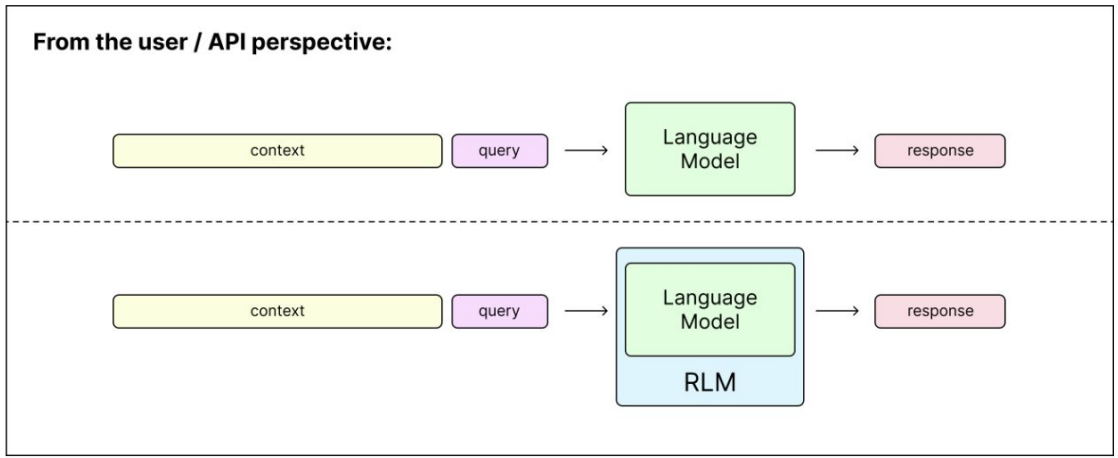
I asked my LM to finish carving the pumpkin joke it started yesterday. It said, “Pumpkin? What pumpkin?” — the context completely rotted.

Intuition on Recursive Language Models

- Handling long context is long-tailed (doesn’t appear often on the training set)
- Natively supporting long context in a base LM is very expensive (requires retraining, scaling the model, and curating datasets)
- We want to give the illusion that LMs can handle infinite context even with the current capabilities of LMs

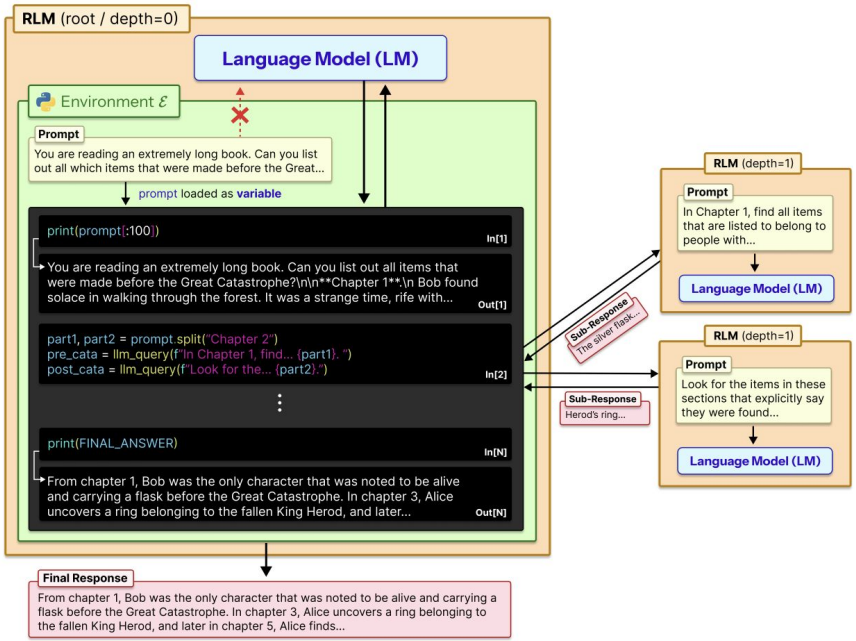
“Bitter Lesson” around Recursive Language Models

- Current approach: we view LLMs as a black-box and engineer scaffolds around it
- Q: Why use engineered scaffoldings instead of letting the model handle its input, subcall, output and train it to do that?

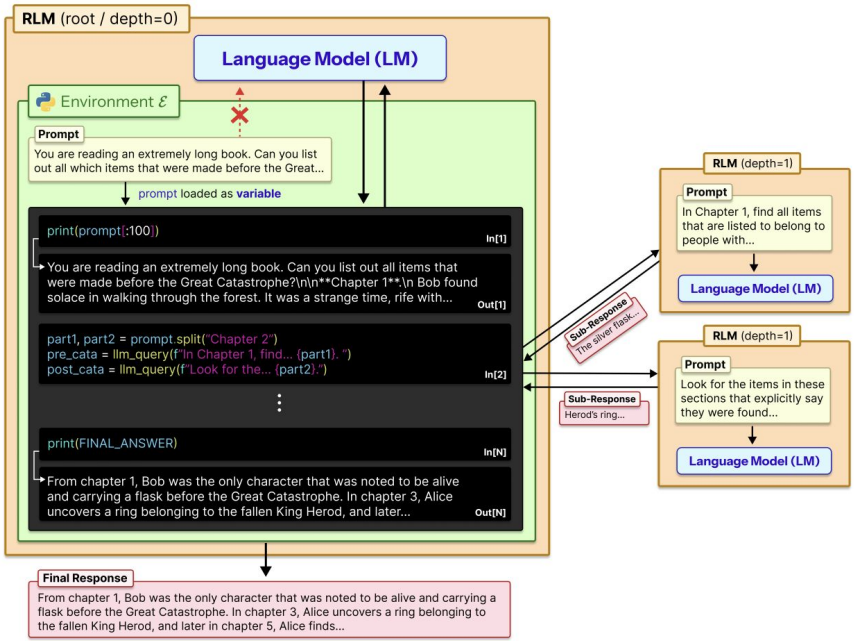


Recursive Language Models

- Recursive Language Models are a general purpose interface paradigm for dramatically scaling the effective input and output length of current LMs
- Main insight: prompts shouldn't be fed into the LM directly but they should be part of an environment the model symbolically interacts with



RLMS Reason and Delegate through a REPL



Algorithm 1 Recursive Language Model (RLM) Call

Input: Input prompt P, maximum turns $T_{\max}$
Output: Final output O

LOAD PROMPT AND SETUP REPL.
turns ← [ ]
state ← {"prompt": P}
tools ← [sub_RLM$_{\mathcal{M}}$(), ...]

for $t = 1$ **to** $T_{\max}$ **do**
  (reasoning, code) ← LLM$_{\mathcal{M}}$(turns)
  state ← EXEC_REPL$_{\mathcal{E}}$(state, tools, code)
  stdout ← format_str(state)
  turns ← turns.append({reasoning, code, stdout})
  if *HasFinalOutput(state)* **then**
    return ExtractFinalAnswer(state)
  end
end
return LLM$_{\mathcal{M}}$(*turns*)

Recursive Language Models

What are the essential components of RLMs? What was missing in previous approaches?

- **Symbolic handle of prompt P**, which offloads inputs from directly being ingested by the model

- **Persistent symbolic programming** which allows the LM with the prompt, create intermediate variables and intermediate answers

- The ability to perform **symbolic recursion,** which allows the model to invoke itself using programmatic patterns to decompose and solve problems

Recursive Language Models

“The ability to perform **symbolic recursion,** which allows the model to invoke itself using programmatic patterns to decompose and solve problems”

Does the subagent live inside or outside the REPL? Are these representations equal?

- Most agentic implementations treat sub-agents as a tool outside the REPL.

RLM vs. Sub-agents

Algorithm 1: A recursive language model, around LLM $\mathcal{M}$, which itself acts as a "language model".

Input: prompt P
Output: response Y

```
state ← InitREPL(prompt=P)
state ← AddFunction(state, sub_RLM_M)
hist ← [Metadata(state)]
while True do
    code ← LLM_M(hist)
    (state, stdout) ← REPL(state, code)
    hist ← hist || code || Metadata(stdout)
    if state[Final] is set then
        return state[Final]
```

Algorithm 2: Alternate scaffold with standard (poor) design choices.

Input: prompt P
Output: response Y

```
actions ← {Finish, Exec, Search, sub_LLM_M}
hist ← [Metadata(actions), P]        Flaw #1
while True do
    (action, val) ← LLM_M(hist)
    if action is Finish then
        return val                   Flaw #2
    out ← RUN(action, val)           Flaw #3
    hist ← hist || (action, val, out)
    if Tok(hist) > K then
        hist ← Compact(hist)
```

Experiments (takeaway)

- RLMs avoid context rot by not calling base LMs over huge context
- RLMs handle natively near infinite context
- RLMs scale with strong performance on long tasks with dense information

Experiments

- S-NIAH: Find a hidden message in a very long sequence of text (needle in a haystack)
- BrowseComp+: Multi-hop DeepResearch benchmark to find answer to complex queries given a corpus of 1K docs
- LongBench v2: Multiple choice questions over large code repos
- OOLONG and OOLONG-pairs bench: tests a model's ability to analyze individual chunks of text at an atomic level and aggregate them to answer complex questions

Experiments (Takeaways)

- Observation 1: RLMs can scale to 10M token regime
- Observation 2: REPL is necessary and recursive sub-calling helps for inputs with dense information
- Observation 3: RLMs don’t suffer from context rot and can reason better
- Observation 4: Inference cost is comparable to other methods

Experiments

Model	CodeQA	BrowseComp+ (1K)	OOLONG	OOLONG-Pairs
Task Length N (tokens)	23K-4.2M	6M-11M	131K	32K
GPT-5 (with RLM sub-calls to GPT-5-mini)				
Base Model	24.0* ($0.13 ± $0.07)	0.0* (N/A) ± (N/A)	44.0 ($0.14 ± $0.02)	0.1 ($0.16 ± $0.10)
CodeAct (+ BM25)	22.0* ($0.06 ± $0.08)	51.0 ($0.71 ± $1.20)	38.0 ($0.61 ± $1.06)	24.7 ($0.75 ± $0.43)
CodeAct (+ sub-calls)	24.0* ($0.06 ± $0.08)	0.0* (N/A) ± (N/A)	40.0 ($0.85 ± $1.27)	28.4 ($1.11 ± $0.62)
Compaction agent	58.0 ($1.31 ± $1.46)	70.5 ($0.57 ± $0.10)	46.0 ($0.13 ± $0.01)	0.1 ($0.13 ± $0.09)
OpenCode	18.0* (N/A) ± (N/A)	0.0* (N/A) ± (N/A)	32.0 (N/A) ± (N/A)	3.1 (N/A) ± (N/A)
OpenCode (+ context offloading)	64.0 (N/A) ± (N/A)	**94.0** (N/A) ± (N/A)	52.0 (N/A) ± (N/A)	4.8 (N/A) ± (N/A)
RLM (recursion depth=0)	58.0 ($0.18 ± $0.56)	88.0 ($0.44 ± $0.90)	36.0 ($0.37 ± $0.42)	43.9 ($0.69 ± $1.16)
RLM (recursion depth=1)	62.0 ($0.11 ± $0.10)	91.3 ($0.99 ± $1.22)	56.0 ($0.43 ± $0.85)	58.0 ($0.33 ± $0.20)
RLM (recursion depth=2)	**66.0** ($0.15 ± $0.30)	92.0 ($0.55 ± $0.69)	56.5 ($1.10 ± $3.25)	65.5 ($0.33 ± $0.44)
RLM (recursion depth=3)	58.0 ($0.15 ± $0.27)	92.0 ($0.51 ± $0.54)	**58.0** ($0.51 ± $0.54)	**76.0** ($0.39 ± $0.32)
Qwen3-Coder-480B-A35B				
Base Model	20.0* ($0.13 ± $0.08)	0.0* (N/A) ± (N/A)	36.0 ($0.06 ± $0.00)	0.1 ($0.05 ± $0.01)
CodeAct (+ BM25)	24.0* ($0.17 ± $0.08)	12.7 ($0.39 ± $0.50)	38.0 ($1.51 ± $1.09)	0.3 ($1.54 ± $0.35)
CodeAct (+ sub-calls)	26.0* ($0.28 ± $0.30)	0.0* (N/A) ± (N/A)	32.0 ($1.83 ± $1.14)	0.1 ($1.49 ± $0.46)
Compaction agent	50.0 ($1.26 ± $1.50)	38.0 ($8.98 ± $2.12)	44.1 ($0.15 ± $0.01)	0.31 ($0.05 ± $0.00)
OpenCode	12.0* (N/A) ± (N/A)	0.0* (N/A) ± (N/A)	36.0 (N/A) ± (N/A)	0.0 (N/A) ± (N/A)
OpenCode (+ context offloading)	40.0 (N/A) ± (N/A)	58.0 (N/A) ± (N/A)	24.0 (N/A) ± (N/A)	2.1 (N/A) ± (N/A)
RLM (recursion depth=0)	**66.0** ($0.18 ± $0.58)	46.0 ($0.82 ± $0.69)	43.5 ($0.32 ± $0.13)	17.3 ($1.77 ± $1.23)
RLM (recursion depth=1)	56.0 ($0.92 ± $1.23)	44.7 ($0.84 ± $0.63)	**48.0** ($0.61 ± $0.49)	**23.1** ($1.02 ± $0.52)
RLM (recursion depth=2)	54.0 ($1.88 ± $3.30)	68.0 ($1.05 ± $0.67)	26.0 ($1.03 ± $1.65)	19.0 ($1.61 ± $0.99)
RLM (recursion depth=3)	44.0 ($1.65 ± $1.63)	**68.7** ($1.10 ± $0.80)	32.0 ($0.80 ± $1.03)	21.1 ($1.67 ± $1.21)
Claude Opus 4.1				
Claude Code	12.0* ($2.03 ± $0.57)	0.0* (N/A) ± (N/A)	40.2 ($3.43 ± $1.60)	0.1 ($6.75 ± $3.57)
Claude Code (+ context offloading)	62.0 ($1.25 ± $0.54)	84.0 ($2.03 ± $1.49)	48.0 ($0.98 ± $0.55)	6.5 ($2.99 ± $1.16)

Recursive Models for Long-Horizon Reasoning

Chenxiao Yang [1] **Nathan Srebro** [1] **Zhiyuan Li** [1]

Abstract

Modern language models reason within bounded context, an inherent constraint that poses a fundamental barrier to long-horizon reasoning. We identify recursion as a core principle for overcoming this barrier, and propose recursive models as a minimal realization, where the model can recursively invoke itself to solve subtasks in isolated contexts. We prove that any computable problem admits a recursive decomposition in which each subtask requires only exponentially smaller active context than standard autoregressive models; this strictly surpasses any context management approach confined to a single sequence, such as summarization. We further generalize our framework to modern agentic systems with arbitrary context processing and control flows, and prove that recursive models can achieve optimal power within this broader class. Experimentally, we train a 3B model to reason recursively and evaluate on Boolean satisfiability, a task requiring long-horizon combinatorial search, where it significantly outperforms frontier LLMs.

1. Introduction

et al., 2025; Zhou et al., 2025; Yan et al., 2025); memory-augmented approaches write and retrieve relevant information in external storage (Packer et al., 2024; Chhikara et al., 2025; Suzgun et al., 2025; Xu et al., 2025); and in agentic systems, subtasks are distributed across agents, each operating in its own context while collaborating toward a shared goal (Hong et al., 2024; Wu et al., 2023; Li et al., 2023).

Yet questions remain: how do these different systems formally compare in their reasoning power? What core mechanisms, as scaffolding that wraps around the base generator, can enable models to handle long-horizon tasks that are otherwise impossible because of context constraints? And are these mechanisms optimal? Despite the importance of these questions, existing work lacks a formalization for these questions to be answered systematically. Notable related works are Yang et al. (2025b;c), which, however, focus on summarization-based context management and self-correction in diffusion language models respectively.

In this work, we identify recursion as a core principle for overcoming context constraints, and a source of power exclusive to modern agentic systems. In a broad sense, recursion refers to the application of a finite, static set of rules to a target problem, that dynamically produces a potentially infinite depth of behaviors that, though contextually isolated from each other, contribute to the final solution.

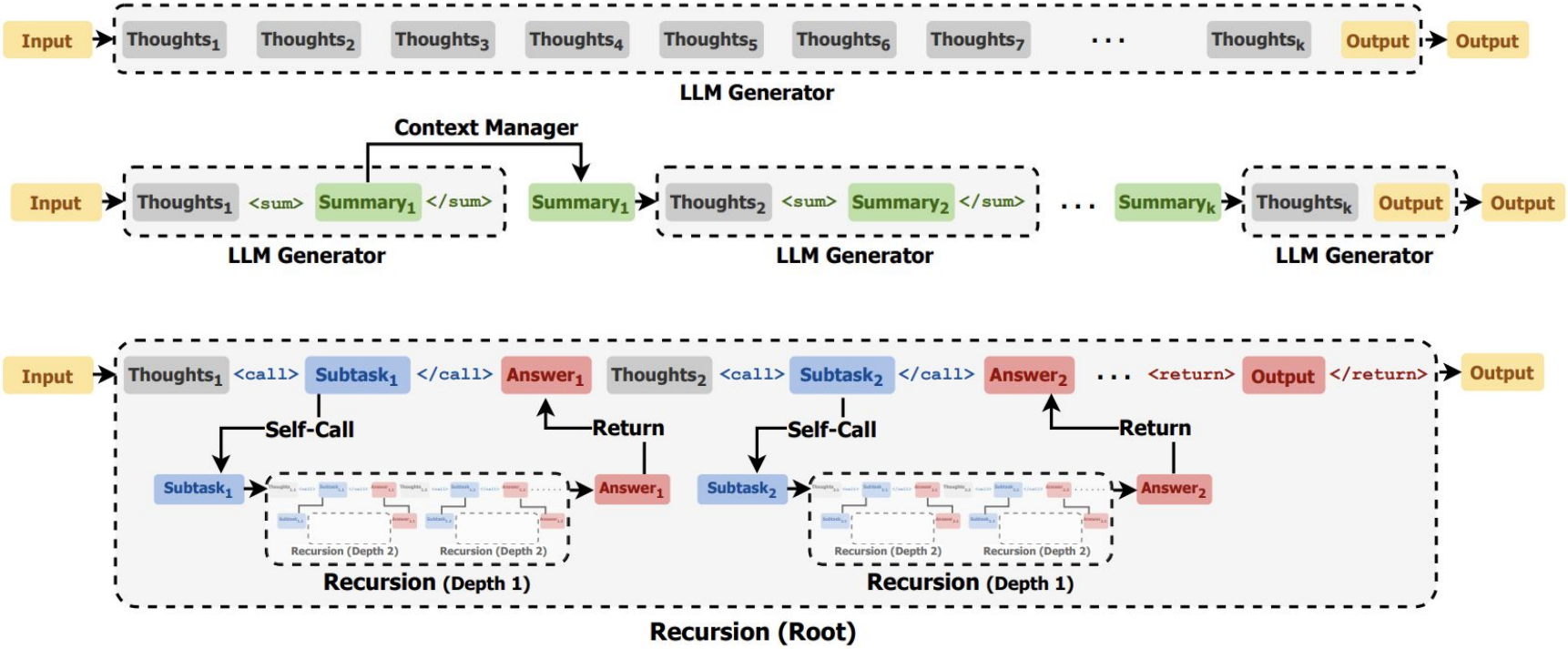
Intuition on Recursive Language Models

- “We identify recursion as a core principle for overcoming context constraints, and a source of power exclusive to modern agentic systems”

- Any computable problem admits a recursive decomposition, and furthermore, by doing so, the required context can be reduced exponentially

- Other approaches for context management:
  - Summarization (strictly less powerful than recursion)
  - Prompt fixing
  - Question preservation to already accomplished tasks

Recursion in LMs



Recursion in LMs

Algorithm 1 Autoregressive Generation, ARM_π

Input: Input sequence $\mathbf{x} \in \Sigma^*$, next-token generator $\pi : \Sigma^* \to \Sigma$, stopping condition $\mathsf{stop} : \Sigma^* \to \{0, 1\}$.

while $\neg\mathsf{stop}(\mathbf{x})$ **do**
  Generate: $y \leftarrow \pi(\mathbf{x})$
  Append: $\mathbf{x} \leftarrow \mathbf{x}.\mathsf{append}(y)$
return $\mathbf{x}$

Algorithm 2 Recursive Model, RCM_f

Input: Sequence $\mathbf{x} \in \Sigma^*$, language model $f : \Sigma^* \to \Sigma^*$.

while true:
  $\mathbf{y} \leftarrow f(\mathbf{x})$
  if $\mathbf{y} = \mathbf{y}' \circ \langle\texttt{return}\rangle\, \mathbf{a}\, \langle/\texttt{return}\rangle$: **return** $\mathbf{a}$
  if $\mathbf{y} = \mathbf{y}' \circ \langle\texttt{call}\rangle\, \mathbf{q}\, \langle/\texttt{call}\rangle$:
    $\mathbf{x} \leftarrow \mathbf{y}'.\mathsf{append}(\mathsf{RCM}_f(\mathbf{q}))$

Definitions

- Recursive language models have *context stack* **S_t**
  - Every element in the stack is a different context

- Two entities: *context manager* **g** and *sequence generator* **f** $\mathbf{S}_{t+1} = g(\mathbf{S}_t, f(\mathbf{S}_t[-1]))$
  - *Call* and *return* are special operations for the context manager

- Global vs. local space (entire stack vs. max element of the stack)

Definitions

- Complexity class of recursive models:
  - Decision problems solvable by recursive models with constant-size O(log(n))-precision transformers such that:
    - Local space at most S(n) (max length of any element in the stack)
    - Recursion depth (max length of the stack) at most D(n),
    - and total number of steps at most T(n)

- Standard complexity classes:
  - TIME(T(n)): set of problems solvable in T(n) timesteps
  - SPACE(S(n)): set of problems solvable in S(n) space
  - TM(T(n),S(n)): intersection of the above

Main Result

Theorem 1 (Deep Recursive Models). *For any $S(n) \geq n$, recursive models can solve any problem in* $\mathsf{TIME}(2^{\mathcal{O}(S(n))})$ *under local space constraint* $\mathcal{O}(S(n))$*:*

$$\mathsf{TIME}(2^{\mathcal{O}(S(n))}) \subseteq \mathsf{RCM}(\mathcal{O}(S(n)), \infty, \infty). \quad (6)$$

- Any computable problem is modularizable!
- If we restrict D=1 only then we can only handle tasks whose time is between S(n), S^2(n)
- Shallow recursion isn’t enough
- Simple recursive system can perform as well as complicated scaffolds