Position: The Turing-Completeness of Autoregressive Transformers Relies Heavily on Context Management

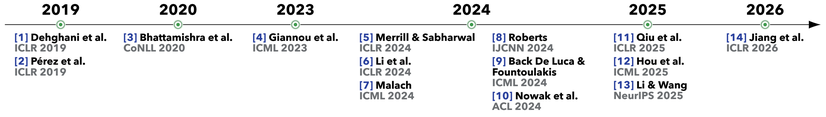
Guanyu Cui[1] Zhewei Wei[1] Kun He[2]

[1]Gaoling School of Artificial Intelligence, Renmin University of China [2]DEKE Lab, Renmin University of China

ICML International Conference On Machine Learning

Research on Turing-Completeness of Transformers

A growing literature argues that Transformers are Turing-complete or can simulate Turing machines.



[1] *Universal Transformers*
[2] *On the Turing Completeness of Modern Neural Network Architectures*
[3] *On the Computational Power of Transformers and its Implications in Sequence Modeling*
[4] *Looped Transformers as Programmable Computers*
[5] *The Expressive Power of Transformers with Chain of Thought*
[6] *Chain of Thought Empowers Transformers to Solve Inherently Serial Problems*
[7] *Auto-Regressive Next-Token Predictors are Universal Learners*
[8] *How Powerful are Decoder-Only Transformer Neural Models?*
[9] *Simulation of Graph Algorithms with Looped Transformers*
[10] *On the Representational Capacity of Neural Language Models with Chain-of-Thought Reasoning*
[11] *Ask, and it shall be given: On the Turing completeness of prompting*
[12] *Universal Length Generalization with Turing Programs*
[13] *Constant Bit-size Transformers Are Turing Complete*
[14] *Softmax Transformers are Turing-Complete*

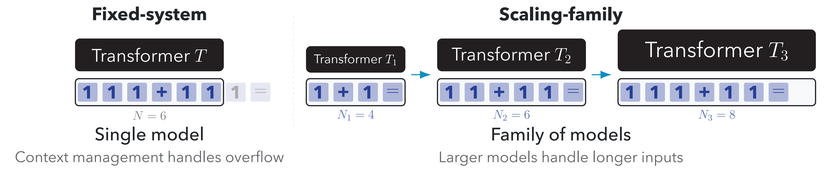
This is tempting because it suggests that deployed LLMs are already computationally universal. A closer look shows that "Transformer" does not always denote the same mathematical object.

1. Two Settings of "Transformers Are Turing-Complete"

Turing-completeness requires an object to perform **arbitrary-length computation**. A deployed LLM has a fixed context window and fixed numerical precision, so one forward pass can handle only fixed-length sequences. The key question is **where the unboundedness comes from**.

Existing work uses two settings:

- **Fixed-system**: one fixed pretrained Transformer plus a context manager that controls input and intermediate reasoning tokens.
- **Scaling-family**: a family whose window†, precision, width, or depth grows with input length.



fixed-system setting → Conclusion: given a TM, ∃ a **fixed LLM system** that simulates it.

scaling-family setting → Conclusion: given a TM, ∃ a **Transformer family** that simulates it.

Scaling-family "universality" is a resource statement about model families. Whether one real LLM system is Turing-complete is a fixed-system question.

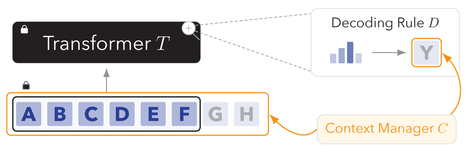
2. The Fixed-System Regime as a (T, D, C) Triple

A fixed-system LLM can be modeled as a triple (T, D, C):

- T: a Transformer with **fixed context-window length, weights, and precision**; $T : \Sigma^N \to \Delta(\Sigma)$ maps bounded token sequences to distributions over tokens;
- D: a **fixed decoding rule**, e.g., greedy; $D : \Delta(\Sigma) \to \Sigma$ maps token distributions to tokens;
- C: a **context manager** that maintains the token history, selects tokens for the context window, and implements harness operations such as compression, retrieval, or tool calls.

System loop:

- Initialize $r^{(1)} = x$.
- At step t, the context manager uses C_w to select the window $w^{(t)} = C_w\left(r^{(t)}\right)$;
- T, D produce the next token $\hat{x}_{t+1} = D\left(T\left(w^{(t)}\right)\right)$;
- The context manager uses C_r to update $r^{(t+1)} = C_r\left(\hat{x}_{t+1}, r^{(t)}\right)$.



Once T is fixed, capability depends on the context manager C and the decoding rule/interface D.

† Although the context length usually does not change the total number of parameters, except in special cases such as learned absolute positional embeddings, reliable extension from a small context window to a larger one is not guaranteed by parameter transfer alone. We therefore treat the fixed context-window length as a model property.

3. Looking Back: What Do Existing Proofs Assume?

Many works make the claim that *Autoregressive Transformers are Turing-complete.*

Existing works split into two regimes:

- **fixed-system**, which **better reflects deployed LLM** systems; and
- **scaling-family**, where model size grows with input length and **no single deployable model realizes the whole family**.

We classify representative works by these assumptions. Most works fall into the scaling-family setting.

1. Assumptions in Scaling-Family Works

Most scaling-family works assume either

- **scaling window**: a **context window large enough** to hold the input length (n), or the required working space ($s(n) \geq n$), or the input together with the full CoT sequence ($n + t$); or
- **scaling precision**: the **numerical precision** of internal representations is unbounded, polynomial, or logarithmic.

Table 1. Assumptions used in representative scaling-family works.

Window	Precision	Work
$n + t$	unbounded	(Dehghani et al., 2019)*, (Pérez et al., 2019; Bhattamishra et al., 2020; Roberts, 2024; Nowak et al., 2024; Jiang et al., 2026)
	poly(n)	(Li et al., 2024)†
	$O(\log(n + t))$	(Merrill & Sabharwal, 2024; Li et al., 2024)†, (Qiu et al., 2025), (Hou et al., 2025)*
	$O(1)$	(Malach, 2024)*,†
$s(n)$	$O(1)$	(Li & Wang, 2025)
n	unbounded	(Back De Luca & Fountoulakis, 2024)
	$O(\log n)$	(Giannou et al., 2023)

* The original paper does not explicitly state an upper bound on the numerical precision, to our knowledge.
† The original paper does not explicitly claim that Turing-completeness has been proved; instead, it claims that one can construct a Transformer that simulates the execution of a Turing machine on a given input instance.

Observation. All surveyed works use a non-constant context window, and only two assume constant precision.

2. Results in the Fixed-System Setting

Fewer papers study the power of Transformers with **fixed context-window length and precision**.

Table 2. Computational power under different context managers.

Work	Context Management	Power
(Schuurmans, 2023)	read / write external memory	≡ TM
(Schuurmans et al., 2024)	decode at most 2 tokens and append	≡ TM
(Schuurmans et al., 2024)	decode at most 1 token and append	≡ $O(n)$-space TM

Although some prior works study fixed-system settings, they **do not clearly distinguish this regime from the scaling-family setting**.

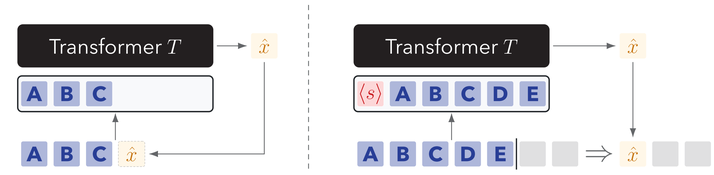
In this more deployment-aligned regime, **system power can arise from context management rather than from scaling the Transformer** itself.

We follow this direction by comparing several context-management methods and showing that they lead to different system power.

4. Context Management Matters in the Fixed-System Setting

1. Summarization-style

- When the context window has **spare capacity**, decoding proceeds normally and each **decoded token is appended** to the maintained sequence.
- When the window is **nearly full**, or when the input cannot fit in the window, the context manager **prepends a compression instruction** to the current prefix, asking the Transformer to summarize prior history before decoding continues.

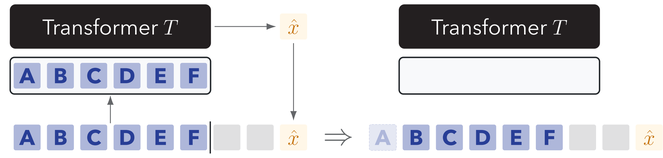


Effect: The system can be simulated by a **constant space Turing machine**, i.e., can only compute some functions in FDSPACE(1) even with sufficiently long CoT sequences.

Implication: When viewed as a decider, the system can only **decide some regular languages (REG)**, so it cannot reliably recognize repetition $\{x\#x : x \in \Sigma^*\}$, palindromes $\{x\#x^{\mathcal{R}} : x \in \Sigma^*\}$, or binary addition $\{\text{bin}(x)\#\text{bin}(y)\#\text{bin}(z) : x + y = z\}$.

2. Appending-style

- When the context window has **spare capacity**, decoding proceeds normally and each **decoded token is appended** to the maintained sequence.
- When the window is **nearly full**, or when the input cannot fit in the window, the context manager **appends the decoded token** to the maintained sequence for later processing.



Effect: The system is equivalent to a linear space Turing machine, i.e., it can compute functions in FDSPACE(n).

Implication: When viewed as a decider, the system can **decide deterministic context-sensitive languages (DCSL)**, which are far more expressive than regular languages.

3. Other Context Management

- Schuurmans (2023) shows that when **external memory can be read and written**, the whole system can be Turing-complete.
- If **tool calls** are allowed and the tools are sufficiently powerful to be Turing-complete (e.g., executing arbitrary Python code), the whole system becomes trivially Turing-complete.
- Schuurmans et al. (2024) show that an appending-style system can also be Turing-complete if **one forward-pass decoding step may generate and append at most two tokens**.

5. Call to Action

We conclude with three calls to action.

1. **State the regime and assumptions.** Scaling-family results help us understand resource needs, but they do not imply that one fixed LLM is Turing-complete.
2. **Analyze the whole system.** Under fixed windows and fixed precision, study the interaction of T, D, and C, not the Transformer alone.
3. **Go beyond qualitative universality.** Report resource budgets, learnability assumptions, and robustness requirements.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Nos. L2524018, U2241212, 92470128, and 62472430).

Contact: cuiguanyu@ruc.edu.cn; zhewei@ruc.edu.cn; hekun2023@ruc.edu.cn