

Report: Understanding Limitations of Large Language Models from First Principles: Computational Complexity Circuit Class T_{Ck}

Kris Carlson querying Chat GPT-5

What exactly are the "theoretical ... limitations in the complexity of problems LRMs can solve, despite their structured reasoning approaches" and how exactly do tool augmentations ameliorate them?

The theoretical limitations of Large Reasoning Models (LRMs) in solving complex problems stem from their inherent computational constraints. Specifically, theoretical analyses based on circuit complexity suggest that a Transformer using k Chain-of-Thought (CoT) steps corresponds to the T_{Ck} circuit class. **This implies that even multi-step CoT reasoning is limited in the complexity of problems it can solve, as it cannot exceed the computational power of this class.** Additionally, LRMs often generate lengthy outputs filled with redundant or irrelevant tokens, which increases inference costs without improving task accuracy. These limitations hinder their ability to handle deeply recursive or highly complex reasoning tasks.

Tool augmentations, such as Python interpreters and scratchpads, help ameliorate these limitations by offloading certain computational tasks to external systems. For example:

1. **Python Interpreters:** Tools like Program-of-Thought (PoT) allow LRMs to generate executable Python code, which is then run externally to compute solutions. This bypasses the output length limitations of LRMs and enables them to solve problems requiring extensive computation, such as the Tower of Hanoi with a large number of disks.
2. **Scratchpads:** These provide external memory for recording intermediate reasoning states, allowing LRMs to break down long reasoning chains into smaller, manageable steps. This mitigates the token limit issue and enables the model to handle tasks requiring iterative or recursive reasoning.

By leveraging these tools, LRMs can focus on generating structured reasoning steps while relying on external systems to handle computationally intensive or memory-constrained tasks. This significantly enhances their problem-solving capabilities, as demonstrated in the study, where tool-augmented LRMs consistently outperformed their non-reasoning counterparts on complex tasks.

Tell me about the T_{Ck} circuit class.

The **T_{Ck} circuit class** is a theoretical concept in computational complexity theory that describes a hierarchy of problems solvable by a specific type of Boolean circuits. These

circuits are characterized by their use of **threshold gates**, which compute majority functions, and their depth (number of layers of gates). Here's a breakdown of the key aspects of the TC_k class:

1. Threshold Circuits:

- A **threshold gate** outputs 1 if the weighted sum of its inputs exceeds a certain threshold, and 0 otherwise. This makes threshold gates more powerful than standard Boolean gates like AND, OR, and NOT.
- Threshold circuits are composed of these gates, along with standard Boolean gates, and are used to compute Boolean functions.

2. Definition of TC_k :

- **TC_k** refers to the class of problems solvable by threshold circuits with:
 - **Polynomial size:** The number of gates in the circuit grows polynomially with the size of the input.
 - **Constant depth k :** The circuit has a fixed number of layers (k), independent of the input size.
 - **Unbounded fan-in:** Each gate can take an arbitrary number of inputs.

3. Hierarchy:

- The " k " in TC_k indicates the depth of the circuit. As k increases, the circuits can compute more complex functions because they allow for more layers of computation.
- For example, **TC_0** (constant depth 0) is the simplest class, while **TC_1** , **TC_2** , etc., represent increasingly complex classes.

4. Relation to Other Complexity Classes:

- TC_k is a subclass of **P/poly**, the class of problems solvable by polynomial-size circuits.
- It is more powerful than the **AC_k** class (which uses AND/OR gates instead of threshold gates) because threshold gates can simulate AND/OR gates and perform additional computations like majority voting.

5. Relevance to LRMs:

- In the context of Large Reasoning Models (LRMs), the TC_k class is used to describe the theoretical limits of reasoning steps. A Transformer using **k Chain-of-Thought (CoT) steps** is said to correspond to the TC_k class, meaning its reasoning depth is limited by the number of layers (or steps) it can perform.

- This limitation implies that even with structured reasoning approaches, LRMs cannot solve problems that require computational power beyond the Tck class for a given k .

6. How Tool Augmentations Help:

- Tool augmentations like Python interpreters and scratchpads effectively extend the computational capabilities of LRMs beyond the Tck limits. By offloading complex computations to external systems, LRMs can handle problems that would otherwise exceed their theoretical reasoning depth.

In summary, the Tck class provides a theoretical framework for understanding the computational limits of systems like LRMs, and tool augmentations serve as a practical way to overcome these constraints.

The document does not explicitly classify non-reasoning Large Language Models (LLMs) or Large Reasoning Models (LRMs) within formal complexity classes like **P**, **NP**, or **PSPACE**. However, it provides insights into their practical capabilities and limitations, which can help us infer their relative computational strengths.

Key Observations from the Document:

1. Non-Reasoning LLMs:

- Non-reasoning LLMs (e.g., DeepSeek-V3, Qwen 3) are designed to generate outputs without explicit step-by-step reasoning processes.
- They perform well on simpler tasks but struggle with complex reasoning problems, especially those requiring structured intermediate steps or long outputs.

2. LRMs:

- LRMs (e.g., DeepSeek-R1, Qwen 3 Thinking) incorporate explicit reasoning strategies like Chain-of-Thought (CoT) prompting, self-reflection, and process supervision.
- With tool augmentation (e.g., Python interpreters, scratchpads), LRMs consistently outperform non-reasoning LLMs on complex tasks, suggesting they are better equipped for structured reasoning.

3. Formal Weakness:

- The document highlights that LRMs and LLMs both face limitations in solving certain hard reasoning tasks (e.g., Checker Jumping for $N \geq 3$). This suggests that neither model type is capable of solving problems in higher complexity classes like **NP-Complete** or **PSPACE**.

- LRMs are not necessarily "formally stronger" than non-reasoning LLMs in a theoretical sense, but they are **practically stronger** when equipped with tools that mitigate token limits and enable structured reasoning.

Complexity Class Perspective:

- **Non-Reasoning LLMs:**

- Without explicit reasoning, non-reasoning LLMs may resemble **finite-state machines** or **bounded-depth circuits** in their computational behavior. They are likely weaker than models capable of recursive or iterative reasoning.

- **LRMs:**

- LRMs, with reasoning strategies and tool augmentations, can simulate more structured computations. They may approach the capabilities of **polynomial-time algorithms** for certain tasks, but they are still constrained by token limits and the inherent architecture of transformers.

Conclusion:

While LRMs are practically stronger than non-reasoning LLMs for complex reasoning tasks, there is no formal evidence in the document to suggest that LRMs belong to a higher complexity class. Both types of models are limited by their architecture and token constraints, making them weaker than theoretical models capable of solving problems in **NP-Complete** or **PSPACE**.