PRELIMINARY DRAFT FOR REVIEW/FEEDBACK

Hallucination Stations
On Some Basic Limitations of Transformer-Based Language Models

Varin Sikka
Stanford University

Vishal Sikka
VianAI Systems

Abstract

With widespread adoption of transformer-based language models in AI, there is significant interest in the limits of LLMs' capabilities, specifically so-called "hallucinations", occurrences in which LLMs provide spurious, factually incorrect or nonsensical [1, 2, 3] information when prompted on certain subjects. Furthermore, there is growing interest in "agentic" uses of LLMs - that is, using LLMs to create "agents" that act autonomously or semi-autonomously to carry out various tasks, including tasks with applications in the real world. This makes it important to understand the types of tasks LLMs can and cannot perform. We explore this topic from the perspective of the computational complexity of LLM inference. We show that LLMs are incapable of carrying out computational and agentic tasks beyond a certain complexity, and further that LLMs are incapable of verifying the accuracy of tasks beyond a certain complexity. We present examples of both, then discuss some consequences of this work and ways to expand upon it.

Computational Complexity of LLMs, and its Implications

As described in numerous introductory documents, LLMs have a vocabulary of a size n set of tokens $t_1, t_2, \ldots, t_n$ (A token is a separable component of a word, such as "walk" and "ed" in "walked".) Their basic operation is to take a string of tokens from this vocabulary, and produce a string of tokens as output. The output string is produced one token at a time, and the input string and the generated tokens are both used to produce the next token.

The core operation of producing one token in a large language model (LLM) has a computational complexity of $O(N^2.d)$, where N is the length of the input string and d is the dimensionality of the model [Kalas]. This means that for an input string of length N, generating a single token requires approximately $N^2.d$ floating-point operations, regardless of the specific input. In [Varin's GitHub] we walk through these steps in more detail.

Our intuition in this paper is that, if there is an input string that expresses a task whose computational complexity is higher than $O(N^2.d)$, then the LLM as defined above cannot correctly carry out that task. We first elaborate on this intuition with a few examples.

Example 1: Token Composition

Consider the following task: "Given a size n set of tokens $\{t_1, t_2, \dots, t_n\}$, list every string of length k tokens." Computing this task takes $O(n^k)$ time. For example, for $n = 2$ this task takes $O(2^k)$ time. So this task, or any variant of it (e.g. "How many sentences of length k are possible by combining the tokens in the set tokens $\{t_1, t_2, \dots, t_n\}$?") cannot correctly be carried out by an LLM that executes in $O(N^2.d)$ time[1].

This is because a task requiring a certain minimum number of steps cannot, obviously, be successfully completed in a smaller number of steps. An LLM takes the input string "Given a set of tokens … strings of length k tokens" and proceeds to enumerate a string of tokens in succession, each token obtained as a result of a $\sim O(N^2)$ calculation that picks the token with the highest likelihood of being next. However, this is not the same as carrying out the task at hand, which is to compute and enumerate the exponentially large number of possible sequences from the provided set of tokens. Although N must always be larger than n , for larger values of n and k, n^k will significantly exceed $O(N^2.d)$.

Example 2: Matrix Multiplication

The naive matrix multiplication algorithm involves the multiplication of two matrices by performing the dot product of the rows from the first matrix and the columns of the second matrix. Specifically, given two matrices A and B, where A is of size $m \cdot n$ and B is of size $n \cdot p$, a matrix C that is the product of A and B will be of size $m \cdot p$, where each element C_{ij} is calculated as $C_{ij} = (\Sigma_{k=0}^{n-1})\, A_{ik} \cdot B_{kj}$

We can write the algorithm for this as follows:

Initialize matrix C with dimensions $m * p$

```
For each row i in A:
        For each column j in B:
                Compute C_ij as C_ij = (Σ_{k=0}^{n-1}) A_ik · B_kj
```

This algorithm takes $O(n^3)$ time, or more precisely, $O(m \cdot n \cdot p)$ time, as the first loop is run m times, the second loop is run p times, and within this inner loop the dot product operation has n steps. For LLMs where m, n, and p exceed its vocabulary size, the LLM will not be able to correctly carry out such a matrix multiplication algorithm.

---

[1] N here represents the number of tokens in the input string, whereas n represents the size of the given set of tokens, which is always smaller than N since the string expressing the n tokens is a substring of the input string, which includes additional tokens to express the task.

Similarly there are countless other tasks whose minimum computational complexity is $O(N^3)$ or higher. Fast-growing functions of cubic time or higher computational complexity frequently arise in real world scenarios. Examples include super-exponential functions like Ackermann Function (and its relevance to problems such as Petri Net reachability or Vector addition [refwiki, refvector], to Floyd-Warshall's algorithm for finding shortest paths between all pairs of nodes in a network (which has a computational complexity that is cubic in the number of nodes), to subset enumeration (which is generally exponential in the size of the set), several join operations in relational databases, computational fluid dynamics (e.g. Navier-Stokes [refNS]), and more.

Example 3: Agentic AI

For our third example, we look into agentic uses of LLMs. Agentic AI refers to artificial intelligence systems capable of autonomous decision-making and goal-oriented behavior, where agents that use LLMs, carry out tasks (sometimes with none to minimal human intervention) [refNVDAAA, others]. Recently, interest in agentic AI has increased significantly due to its role in automating various tasks, in particular software related tasks [OpenAI and Anthropic Agents references]. All major technology companies have recently introduced products centered around agentic AI, including [see refINVAA for two recent overviews]. Such tasks can range from purely informational [refINFAA] to tasks that have side-effects in the real world, from making financial transactions [refVISA] to purchasing products or services [refAMZN], from booking travel and managing hotel and restaurant reservations [refOPENAI], and filing taxes or legal documents [refKPMG] to managing industrial equipment [refAMZNWH]).

Given that agentic use of LLMs can go beyond informational activities to taking actions that affect lives and work in unprecedented ways, it is particularly important to examine their limitations. Our two earlier examples also apply to agentic use-cases, where agents are instructed to autonomously or semi-autonomously carry out tasks. Indeed each task given to an LLM can easily be cast as an agentic task. Tasks that AI agents are instructed to perform, can clearly have computational complexity beyond $O(N^2.d)$. Consequently LLM-based agents cannot correctly carry out those tasks either. In this vein, we will now explore if agents can be used to verify the correctness of another agent's solution to a given task. In the following, we will show that this is not possible either, because verification of a task is often harder than the task itself.

Let's first consider the case of the Traveling Salesperson Problem (TSP), a computational task where solution verification can require exponential time. Given n cities and a symmetric distance matrix, the problem is to verify whether a claimed route R with total distance D_R is the shortest among all possible routes. If the verification must be carried out

without leveraging precomputed bounds, heuristics, or other approximations, then this requires R to be compared against all $\frac{(n-1)!}{2}$ possible routes. This brute-force approach grows factorially with n, making the verification process significantly greater than our correctness threshold of $O(N^2.d)$. For instance, verifying a claimed solution for a 20-city TSP instance involves evaluating approximately 10^{17} candidate routes [ref].

This phenomenon is not unique to TSP but appears in other real-world problems where exhaustive verification is required. Problems such as vehicle routing in logistics, bin packing in warehouses, and scheduling tasks involving crew/staff or resources e.g. in airlines or services companies, often also require exponential time for exact solutions. Applications that involve solving the quadratic assignment problem [QAP] are in this category too. In formal verification of hardware and software systems, model checking often incurs state explosion, where the number of system states grows exponentially with the number of components or variables, making exhaustive verification computationally intractable for large designs [refs]. We believe, this case to be especially pertinent since one of the most prevalent applications of LLMs is to write and verify software [ref].

Next, we consider the case of matrix multiplication verification. As demonstrated by Pan, verifying the correctness of a matrix multiplication task inherently requires $O(n^2)$ time, under the assumption that accessing and processing the entries of the matrices dominates the computational cost. This lower bound applies even when using efficient randomized verification techniques, such as Freivalds' algorithm, which confirm correctness with high probability in $O(n^2)$ time for a single matrix multiplication. Consequently, when tasked with verifying n independent pairs of matrices, where each pair requires separate verification, the total computational effort scales linearly with the number of pairs. Therefore, verifying n such pairs will require at least $O(n^3)$ time, as each independent verification contributes an $O(n^2)$ cost. Similarly all-pairs shortest path verification, to naive matmul and others. [Needed?]

These examples and their variants show that attempts by LLM-based agents to verify the correctness of tasks performed by other agents, will in general not work. Suppose A_1 and A_2 are two agents in the agentic AI sense — that is, agents that carry out tasks using an LLM. Let A_1 be tasked with executing a problem P with a computational complexity of $O(n^3)$ or higher, where n is included in the input prompt provided to A_1. Let A_2 be tasked with validating, i.e. verifying the correctness, of A_1's solution for P. Since all of A_2's operations are limited to $O(N^2 \cdot d)$ complexity (note once more the difference between N and n), given that the inherent complexity of P, i.e. $O(n^3)$ or higher, exceeds A_2's maximum computational complexity, it follows that, in general, A_2 cannot accurately verify the

correctness of A_1's solution to P. This is because any such verification procedure for P would itself in general require at least $O(n^3)$ time complexity in order to function reliably.

With these examples in mind, we can now state the following theorem:

Theorem 1:

Given a prompt of length N, which includes a computational task within it of complexity $O(n^3)$ or higher, where $n < N$, an LLM, or an agent based on LLMs, will unavoidably hallucinate in response to that task.

Proof:

Hartmanis and Stearns, in their seminal time-hierarchy theorem [ref], showed that if $t_2(n)$ is an asymptotically larger function than $t_1(n)$ (e.g., $t_2(n) = n^2$ and $t_1(n) = n$), then there are decision problems solvable in $O(t_2(n))$ but not in $O(t_1(n))$. Consequently, any task that requires time greater than $O(N^2.d)$, such as the ones in our examples above, but indeed infinitely many such tasks, will not be correctly solved by LLMs.

A corollary to this theorem is: there are tasks or sequences of tasks that can be given to LLM agents to perform, whose verification or whose check for accuracy or semantic properties, cannot be correctly performed by LLMs. To show this, it suffices to show that many verification tasks require computational complexity beyond $O(N^2.d)$. In more practical situations, as discussed earlier, infinitely many tasks of both polynomial and non-polynomial time complexity exist whose verification is worse than $O(N^2.d)$.

Furthermore, throughout this paper we have shown multiple examples of real-world prompts such that the computational complexity of the task described within exceeds that of the LLM's output generation. Although this was done to show practical applications of the theorem outlined above, it suffices to show that we can generate a prompt that simply instructs the LLM to perform any task it chooses involving X floating-point operations, where X is engineered to exceed the number of floating-point operations performed by that particular LLM in response, given the length of the prompt and the LLM's dimensionality and other properties.

Discussion:

In essence the argument we have presented is: if the prompt to an LLM involves a computation (or a computational task) whose complexity is higher than that of the LLM's core operation, then the LLM will hallucinate in its responses. Further, any LLM-based agent (i.e. in the Agentic AI sense) cannot correctly carry out tasks beyond the $O(N^2.d)$

complexity of LLMs. In addition, any LLM or LLM-based agent cannot correctly verify the correctness of tasks beyond this complexity.

This leads us to conclude that, despite their obvious power and applicability in various domains, extreme care must be used before applying LLMs to problems or use-cases that require accuracy or solving problems of non-trivial complexity. Mitigating these limitations is an area of significant ongoing work, and various approaches are being developed, from building composite systems [Potts] to augmenting or constraining LLMs with rigorous approaches [vianai, alpha geometry, Lenat others].

One question that we have often heard recently is, do reasoning models overcome these limitations? While we will analyze this question more rigorously in subsequent work, intuitively we don't believe they do. Reasoning models, such as OpenAI's o3 and DeepSeek's R1, generate a large number of tokens in a "think" or "reason" step, before providing their response. So one interesting question is, to what extent does the generation of these additional tokens address the underlying complexity gap? In other words can the sum of the think tokens and the response tokens provide the necessary complexity to correctly solve a problem of higher complexity? We don't believe so, for two fundamental reasons: one that the base operation to generate a single token in these reasoning LLMs, still carries the complexity discussed above, and the computation needed to correctly carry out that very step can be one of a higher complexity (ref our examples above), and secondly, the token budget for reasoning steps is far smaller than what would be necessary to carry out many tasks. Recent work by researchers at Apple [refApple] shows that reasoning models suffer from a "reasoning collapse" when faced with problems of higher complexity, and they use the Towers of Hanoi problem as a reference, a problem which (like some of our examples above) inherently requires an exponential time computation.

Summary: [Is this needed]

In summary, an LLM is a system with a vocabulary of a certain number of tokens, and given a prompt of length N tokens from its vocabulary, it performs $O(N^2.d)$ neural network operations to compute the next token. When an LLM's prompt specifies a computational task of complexity $O(n^3)$ or higher, it can naturally not correctly carry out this task. It can also not correctly verify tasks in general whose verification incurs complexity of $O(n^3)$ or higher. Thus, despite the strong interest in LLMs, one must be very careful in the use of LLMs to perform real tasks, in particular involving their Agentic use.

Acknowledgements:

The authors wish to thank: Professors John Etchemendy, Percy Liang, Chris Potts of Stanford University, Dr. Carolyn Talcott of SRI, Dr. RV Guha of Microsoft, Dr. Alan Kay of

Viewpoints Research, Dr. Kai-Fu Lee of Sinovation, Kevin Dunnell of MIT, Harsh Sikka of Manifold Computing, and Dr. Navin Budhiraja, Dr. Sanjay Rajagopalan, Dr. Tao Liu and Xinrui Wang of Vianai Systems for valuable contribution, advice and criticism.

References:

1. Wikipedia -- Hallucinations definition

2. ibm -- Hallucinations definition

3. NUS -- Hallucinations paper

Keles: https://proceedings.mlr.press/v201/duman-keles23a/duman-keles23a.pdf

QAP: https://coral.ise.lehigh.edu/wp-content/uploads/2014/07/QAP_Survey_04.pdf

Refwiki: Wikipedia Petrinet Reachability

Refvector: Jérôme Leroux. Vector Addition System Reachability Problem (A Short Self-Contained Proof). Principles of Programming Languages, Jan 2011, Austin, TX, United States. pp.307–316

refNS: https://par.nsf.gov/servlets/purl/10175474

refNVDAAA: https://blogs.nvidia.com/blog/what-is-agentic-ai/

refWA: https://www.wired.com/story/uncanny-valley-podcast-unpacking-ai-agents/

refINVAA: refWA and https://www.investors.com/news/technology/meta-stock-amazon-stock-ai-agentic-booking-stock/

refINFAA: https://arxiv.org/abs/2501.16150

refVISA: https://apnews.com/article/ai-artificial-intelligence-5dfa1da145689e7951a181e2253ab349

refAMZN: https://www.retaildive.com/news/amazon-buy-for-me-agentic-ai-third-party-sites/745047/

refOPENAI: https://openai.com/index/introducing-operator/

refKPMG: https://kpmg.com/xx/en/media/press-releases/2025/06/kpmg-launches-a-multi-agent-ai-platform-transforming-client-delivery-and-ways-of-working-across-the-global-organization.html

refAMZNWH: https://www.investopedia.com/amazon-launches-agentic-ai-group-to-enhance-its-warehouse-robots-reports-say-11749004