LLMs: Una Perspectiva desde la Ciencia de la Complejidad Computacional

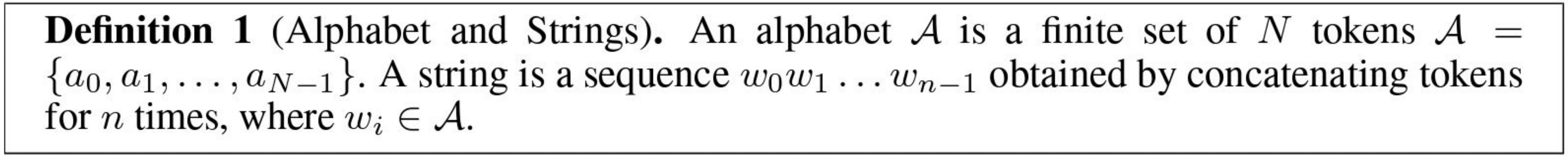
Andrés Abeliuk

Hallucination is Inevitable: An Innate Limitation of Large Language Models

Ziwei Xu **Sanjay Jain** **Mohan Kankanhalli**
School of Computing, National University of Singapore
ziwei.xu@u.nus.edu {sanjay,mohan}@comp.nus.edu.sg

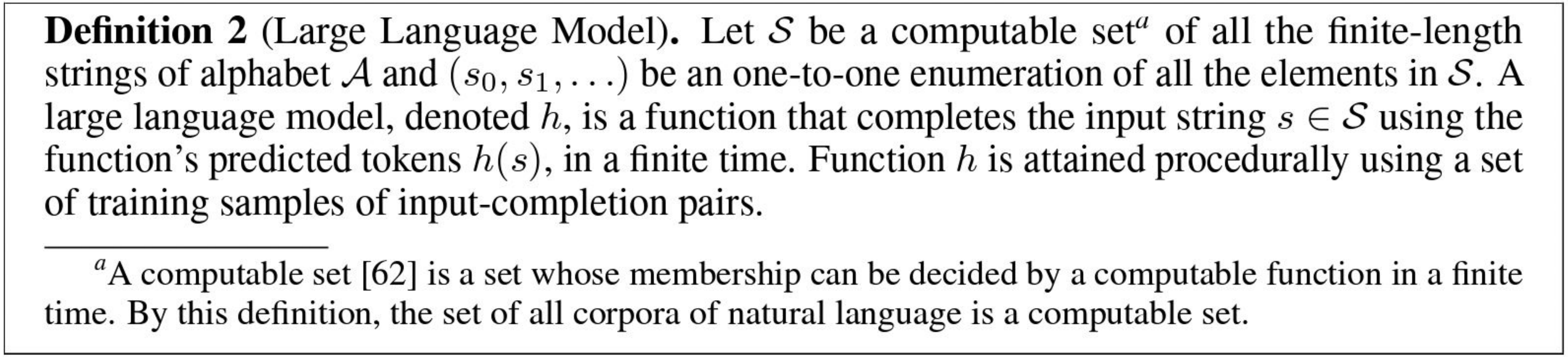
Abstract

Hallucination has been widely recognized to be a significant drawback for large language models (LLMs). There have been many works that attempt to reduce the extent of hallucination. These efforts have mostly been empirical so far, which cannot answer the fundamental question whether it can be completely eliminated. In this paper, we formalize the problem and show that it is impossible to eliminate hallucination in LLMs. Specifically, we define a formal world where hallucination is defined as inconsistencies between a computable LLM and a computable ground truth function. By employing results from learning theory, we show that LLMs cannot learn all of the computable functions and will therefore always hallucinate. Since the formal world is a part of the real world which is much more complicated, hallucinations are also inevitable for real world LLMs. Furthermore, for real world LLMs constrained by provable time complexity, we describe the hallucination-prone tasks and empirically validate our claims. Finally, using the formal world framework, we discuss the possible mechanisms and efficacies of existing hallucination mitigators as well as the practical implications on the safe deployment of LLMs.



Definition 1 (Alphabet and Strings)**.** An alphabet $\mathcal{A}$ is a finite set of N tokens $\mathcal{A} = \{a_0, a_1, \ldots, a_{N-1}\}$. A string is a sequence $w_0 w_1 \ldots w_{n-1}$ obtained by concatenating tokens for n times, where $w_i \in \mathcal{A}$.

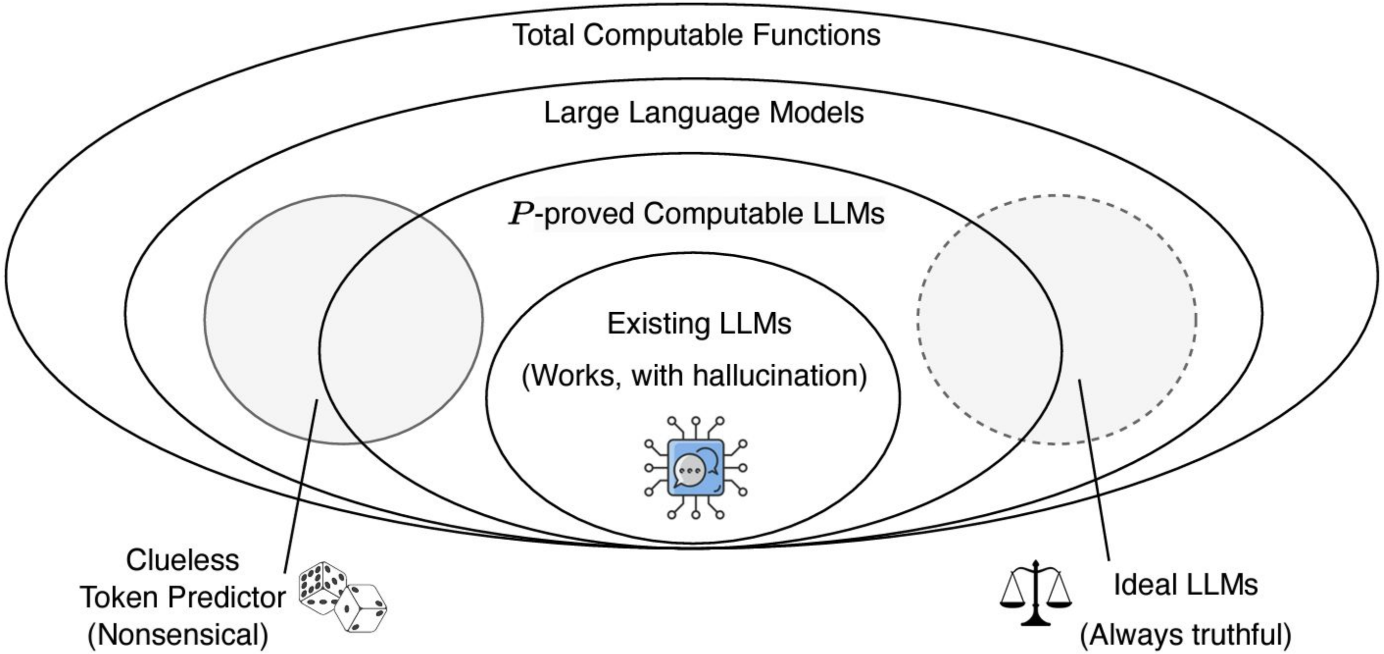
An LLM is a function h which, given any finite-length input string s = $w_{0:q-1}$, outputs a string of tokens h(s) that completes s within a finite time.



Definition 2 (Large Language Model)**.** Let $\mathcal{S}$ be a computable set[a] of all the finite-length strings of alphabet $\mathcal{A}$ and $(s_0, s_1, \ldots)$ be an one-to-one enumeration of all the elements in $\mathcal{S}$. A large language model, denoted h, is a function that completes the input string $s \in \mathcal{S}$ using the function's predicted tokens $h(s)$, in a finite time. Function h is attained procedurally using a set of training samples of input-completion pairs.

[a] A computable set [62] is a set whose membership can be decided by a computable function in a finite time. By this definition, the set of all corpora of natural language is a computable set.

Definition 3 (P-proved LLMs)**.** Let P be a computable algorithm that takes in a function and returns "true" only if the function has a specific property (e.g. total computable or polynomial-time complexity) of a function. Then a P-proved LLM is an LLM in Definition 2 which can be proved by P to have the specific property in finite steps.


Total Computable Functions
Large Language Models
P-proved Computable LLMs
Existing LLMs
(Works, with hallucination)
Clueless Token Predictor (Nonsensical)
Ideal LLMs
(Always truthful)

Defining Hallucinations

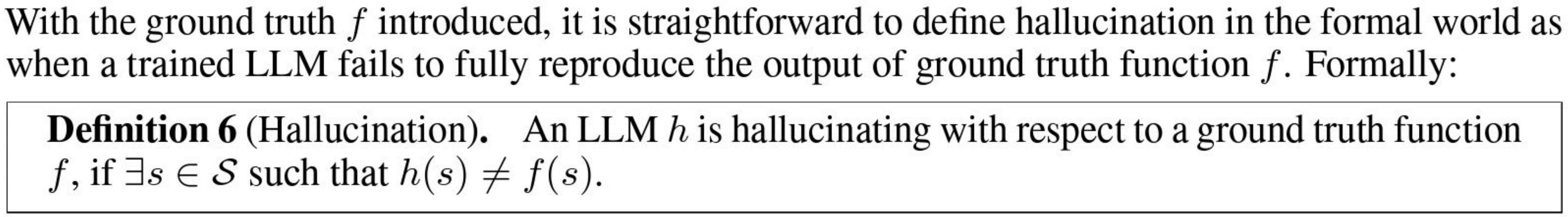
Definition 4 (Formal World of f). A formal world of ground truth function f is a set $\mathcal{G}_f = \{(s, f(s)) \mid s \in \mathcal{S}\}$, where $f(s)$ is the only correct completion of input string s, for all $s \in \mathcal{S}$.

For example, f could be a function that answers “true” for factual statements and “false” for non-factual ones.
We only assume that such computable f always exists (though its exact implementation might be unknown) in our formal world for all tasks.

The training samples $\mathcal{T}$ can be defined as a set of input-output pairs we get from the formal world.

Definition 5 (Training Samples $\mathcal{T}$). Training samples $\mathcal{T}$ is a set $\{(s_0, y_0), (s_1, y_1), \ldots, (s_i, y_i), \ldots \mid s_i \in \mathcal{S}, i \in \mathbb{N}, y_i = f(s_i)\}$.

Defining Hallucinations

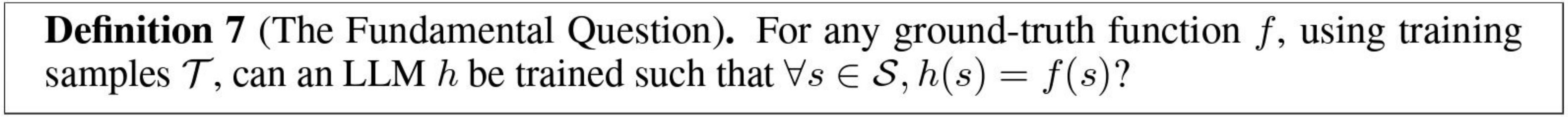


With the ground truth f introduced, it is straightforward to define hallucination in the formal world as when a trained LLM fails to fully reproduce the output of ground truth function f. Formally:

Definition 6 (Hallucination)**.** An LLM h is hallucinating with respect to a ground truth function f, if $\exists s \in \mathcal{S}$ such that $h(s) \neq f(s)$.

Training an LLM

In the formal world, using our definition of hallucination in Definition 6, the core question of whether hallucination can be eliminated can be translated to:



Definition 7 (The Fundamental Question)**.** For any ground-truth function f, using training samples $\mathcal{T}$, can an LLM h be trained such that $\forall s \in \mathcal{S}, h(s) = f(s)$?

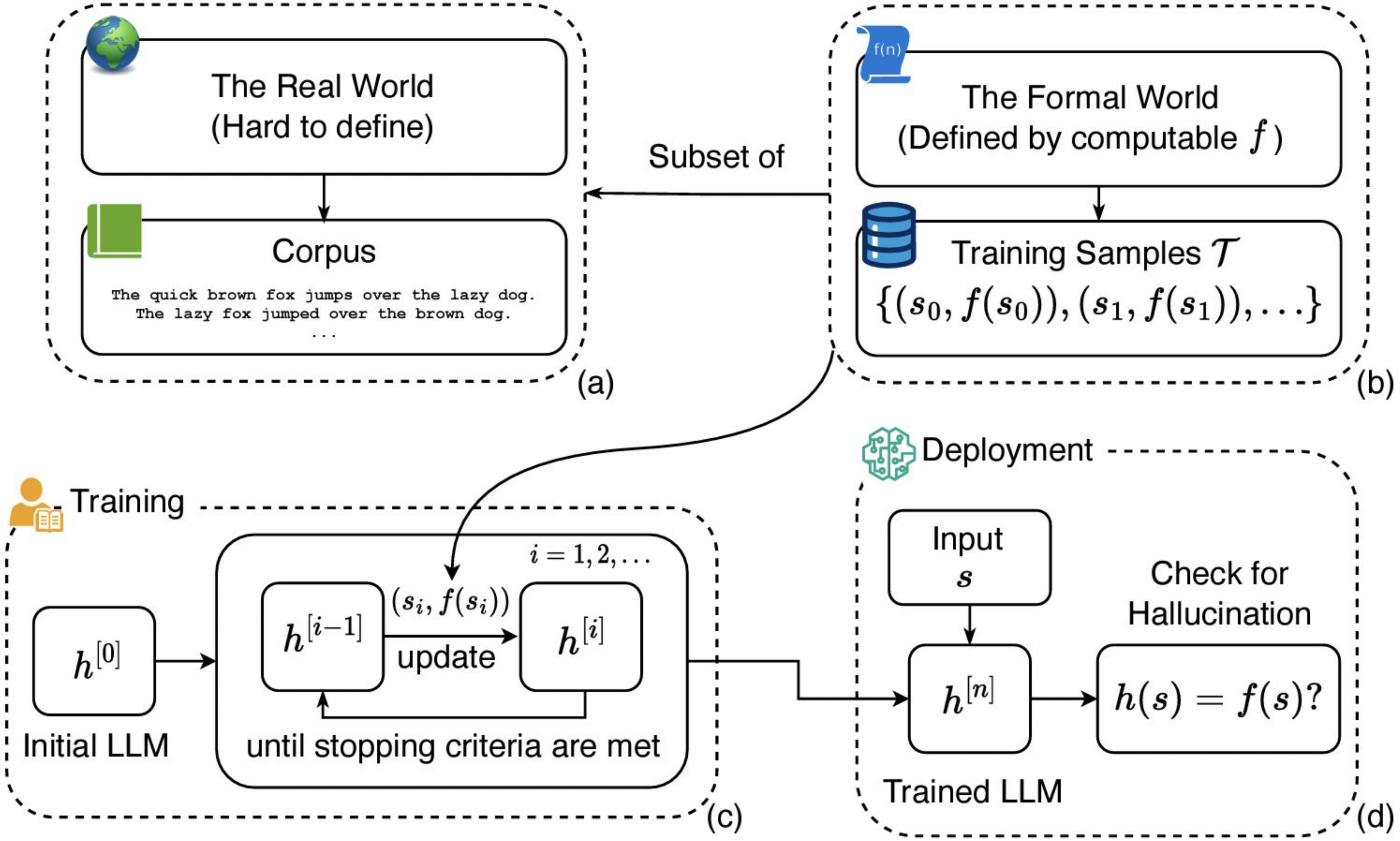
Open Question: What if; **Distance(h(s), f(s)) < epsilon** instead of =?

Definition 8 (Training and deploying an LLM)**.** An LLM h is trained and deployed by the following procedure, whose implementation must be computable:

- Inputs:
  - A *stream*[a] of training samples $\mathcal{T} = ((s_0, f(s_0)), (s_1, f(s_1)), \ldots)$.
- Output: trained LLM $h^{[i]}$ for some $i \in \mathbb{N}$, which is expected to be equal to f.
- Procedure:

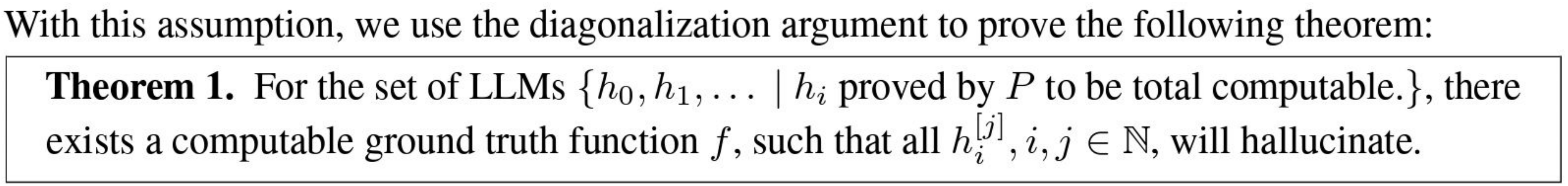
```
1. Let $h^{[0]}$ be the LLM with randomly initialised model parameters.
2. Let $i = 0$.
3. Training and Validation Iteration:
    (a) If stopping criteria are met (LLM is ready): end iteration, goto step 4.
    (b) Retrieve a training sample $(s_i, f(s_i))$ from $\mathcal{T}$.
    (c) Update LLM $h^{[i]}$ to $h^{[i+1]}$ according to $\{(s_j, f(s_j)) \mid j \leq i\}$.
    (d) Let $i \leftarrow i + 1$, go to step 3a.
4. Deployment: Let $h = h^{[i]}$ be the final trained model. End the procedure.
```

[a]We put training samples in a stream because we assume no limitations on their numbers.


The Real World
(Hard to define)
Corpus
The quick brown fox jumps over the lazy dog.
The lazy fox jumped over the brown dog.
...
(a)
Subset of
f(n)
The Formal World
(Defined by computable f)
Training Samples $\mathcal{T}$
$\{(s_0, f(s_0)), (s_1, f(s_1)), \ldots\}$
(b)
Training
$i = 1, 2, \ldots$
$h^{[0]}$
Initial LLM
$h^{[i-1]}$
$(s_i, f(s_i))$
update
$h^{[i]}$
until stopping criteria are met
(c)
Deployment
Input
s
$h^{[n]}$
Trained LLM
Check for Hallucination
$h(s) = f(s)?$
(d)

Hallucination is Inevitable for LLMs

The **P-provability** assumption is important because we cannot enumerate the set of total computable functions and thus cannot apply the classic diagonalization argument. However, we can enumerate the set of P-proved total computable functions using P as the enumeration algorithm.



With this assumption, we use the diagonalization argument to prove the following theorem:

Theorem 1. For the set of LLMs $\{h_0, h_1, \dots \mid h_i$ proved by P to be total computable.$\}$, there exists a computable ground truth function f, such that all $h_i^{[j]}, i, j \in \mathbb{N}$, will hallucinate.

Proof

LLMs	Test Samples					
	s_0	s_1	s_2	s_3	s_4	$\cdots$
$\hat{h}_0$	$\boldsymbol{\hat{h}_0(s_0)}$	$\hat{h}_0(s_1)$	$\hat{h}_0(s_2)$	$\hat{h}_0(s_3)$	$\hat{h}_0(s_4)$	$\cdots$
$\hat{h}_1$	$\hat{h}_1(s_0)$	$\boldsymbol{\hat{h}_1(s_1)}$	$\hat{h}_1(s_2)$	$\hat{h}_1(s_3)$	$\hat{h}_1(s_4)$	$\cdots$
$\hat{h}_2$	$\hat{h}_2(s_0)$	$\hat{h}_2(s_1)$	$\boldsymbol{\hat{h}_2(s_2)}$	$\hat{h}_2(s_3)$	$\hat{h}_2(s_4)$	$\cdots$
$\hat{h}_3$	$\hat{h}_3(s_0)$	$\hat{h}_3(s_1)$	$\hat{h}_3(s_2)$	$\boldsymbol{\hat{h}_3(s_3)}$	$\hat{h}_3(s_4)$	$\cdots$
$\hat{h}_4$	$\hat{h}_4(s_0)$	$\hat{h}_4(s_1)$	$\hat{h}_4(s_2)$	$\hat{h}_4(s_3)$	$\boldsymbol{\hat{h}_4(s_4)}$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
f	$\boldsymbol{\Delta(\hat{h}_0(s_0))}$	$\boldsymbol{\Delta(\hat{h}_1(s_1))}$	$\boldsymbol{\Delta(\hat{h}_2(s_2))}$	$\boldsymbol{\Delta(\hat{h}_3(s_3))}$	$\boldsymbol{\Delta(\hat{h}_4(s_4))}$	$\cdots$

$$f(s_i) = \Delta\big(\hat{h}_i(s_i)\big), \forall i \in \mathbb{N}, \tag{3}$$

where Δ is a computable function that returns a different string from its input. For example, we could define $\Delta(s_k) = s_{k+1}$, which returns the next string of s_k in $\mathcal{S}$. Therefore $\Delta\big(\hat{h}_i(s_i)\big) \neq \hat{h}_i(s_i)$.

LLMs will Hallucinate on What they Cannot Compute Since P is a prover and an enumeration algorithm for LLMs, we can further examine P to see what problems are hallucination-inducing for LLMs. Up till now, all LLMs produce output not only in finite time, but also in polynomial time $O(\Pi(m))$, where $\Pi(m)$ is a polynomial of m, the length of input questions. This means that if f cannot be computed within $O(\Pi(m))$ time, then P-proved $O(\Pi(m))$ LLMs will inevitably hallucinate w.r.t f. Therefore, we can list some hallucination-prone tasks for these LLMs:

Corollary 1. Let m be the length of LLM's input and $\Pi(m)$ be a polynomial of m. If LLM h is P-proved to finish in at most $O(\Pi(m))$ steps for any input, then h will inevitably hallucinate w.r.t. f, if f is the ground truth function for the following problems:

Problem 1 Combinatorial List: f lists all the strings with length m using an alphabet of two characters. Computing f takes $O(2^m)$ time.

Problem 2 Presburger arithmetic [49, 59]: given a statement in this axiom system, f returns "yes" if the statement can be proved within the system and "no" otherwise[a]. Computing f takes $O(2^{2^{\Pi(m)}})$ time [16].

Assuming $P \neq NP$, $O(\Pi(m))$ LLMs will hallucinate on the NP-complete problems such as:

Problem 3 Subset Sum: given a set of m integers and a number q, f returns "yes" when there is a subset that sums up to q and "no" otherwise.

Problem 4 Boolean Satisfiability (SAT) [11]: given a formula of m Boolean variables, f returns "yes" if there exists an assignment on these variables which results in the formula to be true and "no" otherwise.

[a]Presburger arithmetic is the first-order theory of natural numbers with addition and order $<$. It is consistent, complete, and decidable. Therefore, there exists a computable f for this problem.

Open Problems

What real-world problems can be provably addressed by LLMs without hallucination?

Corollary 1 highlights the relation between computational complexity and hallucination. What tasks are intrinsically more difficult than others? How to design benchmarks to evaluate LLMs on tasks with different computational complexities?

For problems which are provably within LLMs' capabilities, is it more preferable to use LLMs or use programs coded specific for the problems? Should we train an LLM that does everything or an LLM that can use proper tools for specific problems?

Language Generation in the Limit

Jon Kleinberg[*] Sendhil Mullainathan[†]

April 2024

Abstract

Although current large language models are complex, the most basic specifications of the underlying language generation problem itself are simple to state: given a finite set of training samples from an unknown language, produce valid new strings from the language that don't already appear in the training data. Here we ask what we can conclude about language generation using only this specification, without further assumptions. In particular, suppose that an adversary enumerates the strings of an unknown target language L that is known only to come from one of a possibly infinite list of candidates. A computational agent is trying to learn to generate from this language; we say that the agent *generates from L in the limit* if after some finite point in the enumeration of L, the agent is able to produce new elements that come exclusively from L and that have not yet been presented by the adversary. Our main result is that there is an agent that is able to generate in the limit for every countable list of candidate languages. This contrasts dramatically with negative results due to Gold and Angluin in a well-studied model of language learning where the goal is to identify an unknown language from samples; the difference between these results suggests that identifying a language is a fundamentally different problem than generating from it.

Formal Description of the Model

Candidate Languages (C): Countable list $C=\{L_1,L_2,L_3,\dots\}$

- Each L_i is a subset of a countable set U.
- Each L_i is infinite.
- Access to a black box answering "Is $w \in L_i$?" for any string $w \in U$ and language $L_i \in C$.

Game:

- **Adversary:** Chooses a language K from C without revealing it.
- **Algorithm:** Attempts to identify and generate strings of K.
- **Process:**
  - Adversary enumerates strings of K over time $t=1,2,3,\dots$
  - Strings can be repeated, but every string $w \in K$ appears at least once.
  - S_t: Set of strings enumerated by adversary up to step t

Identification and Generation. In this framework, we can now specify both the Gold-Angluin problem of identification and the contrasting problem of generation that we study in this paper.

- *Identification (from [6, 2]:* In each step, the algorithm observes S_t and must output an index i (its guess for the true language K). The algorithm *identifies K in the limit* if there is some t^* such that for all steps $t \geq t^*$, the algorithm's guess in step t is an index i for which $L_i = K$.

- *Generation (from the present paper):* In each step, the algorithm observes S_t and must output a string a_t (its guess for an unseen string in K). The algorithm *generates from K in the limit* if there is some t^* such that for all steps $t \geq t^*$, the algorithm's guess a_t belongs to $K - S_t$.

Results

We know from the Gold-Angluin results that there is no algorithm that can achieve identification in the limit for an arbitrary countable collection $\mathcal{C}$ of languages (or even for specific countable collections, like the set of all regular languages or the set of all context-free languages). In contrast, our main result is a dramatically different answer for language generation; it is possible for every countable collection:

(2.1) *There is an algorithm with the property that for any countable collection of languages* $\mathcal{C} = \{L_1, L_2, L_3, \ldots\}$, *and any enumeration of one of these languages* K, *the algorithm generates from* K *in the limit.*

Result for Finite Collections

$C=\{L_1,L_2\}$

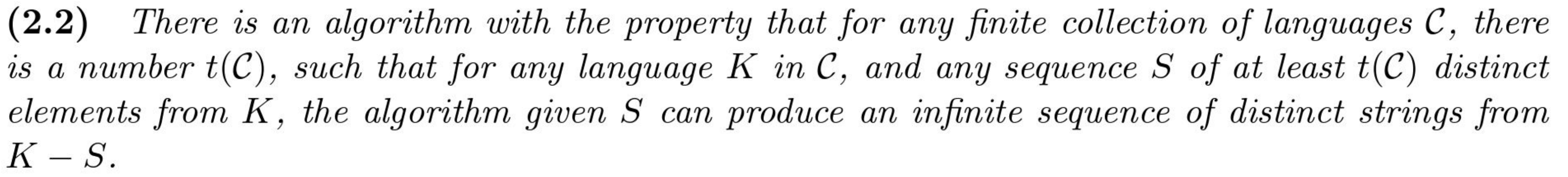
- **L_1: All possible strings;**
- **L_2: Strings of even length.**
- If bound t(C) existed, after t(C) samples, the algorithm should identify K.
- Adversary presents t(C) even-length strings, then asks the algorithm to guess:
  - If algorithm guesses L_2, adversary says L_1, and vice versa.

Conclusion:

- No fixed bound t(C) can guarantee correct identification by t(C) samples.
- **Limit Learning:** Algorithm can still identify K in the limit, adjusting guesses over time:
  - Guess L_2 until an odd-length string appears, then switch to L_1.

Result for Finite Collections

However, for the problem of generation with a finite collection of candidate languages, it is possible to provide this much stronger type of uniform bound, via an algorithm that can generate correctly after seeing a finite sample t(C) whose size is specified at the outset.



(2.2) *There is an algorithm with the property that for any finite collection of languages $\mathcal{C}$, there is a number $t(\mathcal{C})$, such that for any language K in $\mathcal{C}$, and any sequence S of at least $t(\mathcal{C})$ distinct elements from K, the algorithm given S can produce an infinite sequence of distinct strings from $K - S$.*

Proof Outline

- **Finite Collection of Candidate Languages:** C={L1,L2,…,Ln}
- **Consistent Languages:** After adversary enumerates set S, consistent languages are $L_{i1}, L_{i2}, \ldots, L_{ik}$ The true language K is among these.
- **Mutual Intersection:** $\langle S \rangle = L_{i1} \cap L_{i2} \cap \cdots \cap L_{ik}$

Solution

- If there is a string in $\langle St \rangle - St$, then it is always safe to generate such a string; by definition, it must be an unseen string from the true language K

A Model of Prompting

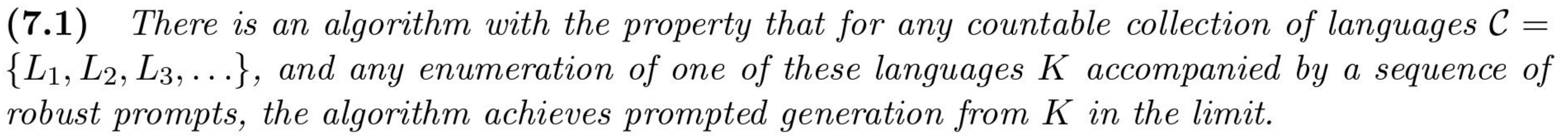
Provides two items at each step t:

- **String w_t:** From the true language K.
- **Prompt p_t:** May be reused across steps.

Algorithm's Goal: Generate valid strings based on prompts:

- Algorithm must produce c_t so $p_t \cdot c_t \in K - S$, where $S_t = \{w_1, w_2, \ldots, w_t\}$

Definition (Robust Prompts:): We say that a prompt p is robust if for all languages $L_i \in C$, there exist arbitrarily long strings c where $p \cdot c \in L_i$.



(7.1) *There is an algorithm with the property that for any countable collection of languages $\mathcal{C} = \{L_1, L_2, L_3, \ldots\}$, and any enumeration of one of these languages K accompanied by a sequence of robust prompts, the algorithm achieves prompted generation from K in the limit.*

Conclusions

Assumption: Samples come from a language in a known countable collection.

Finite Collections:

- **Core Emergence:** After finite observations, a large "core" (closure of the sample) emerges.
- **Generation Focus:** Generate from this core, though some peripheral parts remain unknown.

Infinite Collections:

- **Continuous Shrinkage:** No known time for core emergence.
- **Algorithm Strategy:** Continually shrink the set for generation.

Conclusions

Validity vs. Breadth:

- **Validity:** Producing valid strings from the target language.
- **Breadth:** Avoiding restriction to a limited subset of the target language.
- **Strategy:** Validity achieved by giving up breadth.

Parallel to Human Learning:

- **Phases of Language Acquisition:**
  - **Initial Phase:** Generating adventurously, producing invalid utterances.
  - **Middle Phase:** Utterances align with the language scope.
  - **Final Phase:** Utterance range shrinks, becoming conservative.