On the Impossibility of Universal LLM Halting Prediction

Jovonni L. Pharr
Georgia Cyber Warfare Range
info@gacwr.org

Abstract

We give a fully rigorous account, with explicit constructions and reductions, showing that no Turing-computable "universal LLM judge" can infallibly decide for *all* instances (i) whether an arbitrary text is true/accurate and (ii) whether an arbitrary text was generated by a given large language model (LLM) or by a human. Our treatment unifies three lines: (A) direct many-one reductions from the Halting Problem, (B) Rice-style undecidability of nontrivial semantic properties of generators, and (C) a Kleene Recursion Theorem–based diagonal that adversarially defeats any total computable judge, including text-only authorship detectors. We also give a precise separation from recent results on self-prediction of LLM outputs: even if self-prediction is impossible, it does not license a perfect *external* meta-judge; conversely, our meta-evaluation impossibility does not rely on probabilistic decoding. The result is architecture-agnostic and holds for any Turing-powerful generator.

1 Preliminaries

We work over a finite alphabet Σ. Strings are elements of Σ^*, denoted Σ^*. A *(partial) computable function* is one computed by a Turing machine (TM). We fix a standard effective enumeration $(\varphi_e)_{e\in\mathbb{N}}$ of partial computable functions $\varphi_e : \Sigma^* \rightharpoonup \Sigma^*$ via Gödel numbers (program indices) [4, 3].

Basic facts (used as lemmas). We use the following standard results (see, e.g., Rogers (1967), Kleene (1952)).

(i) (*s-m-n Theorem*) For every computable $f(e, x)$ there is a computable total function s such that $\varphi_{s(e)}(x) = \varphi_{f(e,x)}()$ uniformly (parameter fixing) [3, 4].

(ii) (*Kleene Recursion Theorem*) For every total computable $\psi(e)$ there exists e^* with $\varphi_{e^*} = \varphi_{\psi(e^*)}$ [3].

(iii) (*Rice's Theorem*) Any nontrivial semantic property of partial computable functions is undecidable [2].

Write HALT $:= \{(e, w) \mid \varphi_e(w) \downarrow\}$ for the Halting set (undecidable, r.e.-complete) [1, 4]. For a total computable predicate P, "decidable" means χ_P is total computable.

LLMs as generators. We model an *LLM generator* as any partial computable function $G : \Sigma^* \rightharpoonup \Sigma^*$ mapping prompts to outputs (deterministic decoding; randomness only worsens decidability) [4]. When we speak of "an LLM M" we mean an index $m \in \mathbb{N}$ with behavior φ_m.

2 Judging Tasks

We formalize two families of judgment problems.

Definition 1 (Authorship Judgment)**.** Given indices $m \in \mathbb{N}$, prompt $p \in \Sigma^*$, and string $x \in \Sigma^*$, define

$$L_{\mathsf{out}} := \{\langle m, p, x\rangle : \varphi_m(p) \downarrow = x\}.$$

The *authorship decision problem* asks whether $\langle m, p, x\rangle \in L_{\mathsf{out}}$.

Definition 2 (Truth Judgment)**.** Fix a sufficiently expressive formal language $\mathcal{L}$ (e.g. first-order arithmetic). Let $\mathsf{True}_{\mathcal{L}} \subseteq \Sigma^*$ be the set of encodings of *true* $\mathcal{L}$-sentences in the intended model. The *truth decision problem* asks whether $x \in \mathsf{True}_{\mathcal{L}}$.

We also consider text-only AI detectors.

Definition 3 (Text-only Detection)**.** A *text-only detector* is any total computable $C : \Sigma^* \to \{\mathsf{AI}, \mathsf{HUMAN}\}$. It *errs* on y if it outputs HUMAN yet y was produced by a computable generator (an AI).

3 Main Impossibility Theorems

3.1 Authorship Judgment is Undecidable

Theorem 1. *L_{out} is recursively enumerable (r.e.) but not decidable.*

Proof. r.e.: Semi-decide $\langle m, p, x\rangle \in L_{\mathsf{out}}$ by simulating $\varphi_m(p)$; accept if it halts with output x [4].

Undecidability: Reduce HALT to L_{out} [1, 4]. Given (e, w), effectively build an index m' and fixed prompt p_0 and string x_0 such that

$$\varphi_{m'}(p_0) = x_0 \iff \varphi_e(w) \downarrow .$$

Construction: Let program m' on input p ignore p, simulate $\varphi_e(w)$. If $\varphi_e(w)$ halts, output x_0; otherwise diverge. Then $\langle m', p_0, x_0\rangle \in L_{\mathsf{out}}$ iff $(e, w) \in$ HALT. If L_{out} were decidable, HALT would be decidable, a contradiction. □

3.2 Truth Judgment is Undecidable

Theorem 2. *If $\mathcal{L}$ is sufficiently expressive to represent Turing computations (e.g. first-order arithmetic), then $\mathsf{True}_{\mathcal{L}}$ is not decidable.*

Proof. By representability of computable predicates, there is a computable function mapping (e, w) to an $\mathcal{L}$-sentence $\theta_{e,w}$ expressing "$\varphi_e(w)$ halts." In arithmetic this is a Σ_1 sentence of the form $\exists t\, T(e, w, t)$ where T is primitive recursive and encodes "e halts on w within t steps" [3, 4]. Then

$$\theta_{e,w} \in \mathsf{True}_{\mathcal{L}} \iff (e, w) \in \mathrm{HALT}.$$

If $\mathsf{True}_{\mathcal{L}}$ were decidable, HALT would be decidable, a contradiction. (Equivalently: Tarski's Undefinability of Truth implies no arithmetical truth predicate is computable.) [5] □

3.3 No Perfect Text-Only AI Detector

Theorem 3 (Adversarial defeat of any detector)**.** *For any total computable $C : \Sigma^* \to \{\mathsf{AI}, \mathsf{HUMAN}\}$ that labels at least one string as human-authored (i.e. $\exists\, y \in \Sigma^*,\ C(y) = \mathsf{HUMAN}$), there exists a computable generator G_C outputting a string y such that $C(y) = \mathsf{HUMAN}$ although y is AI-generated (by G_C). Hence no universal zero-error text-only detector exists that accepts any human text.*

Proof. By hypothesis, there exists some y_0 with $C(y_0) = \mathsf{HUMAN}$. Define G_C as the constant generator that always outputs y_0. Then G_C is computable (it is a constant function), y_0 is AI-generated (produced by G_C), yet $C(y_0) = \mathsf{HUMAN}$. Thus C errs. □

Remark 1 (Correction via Lean formalization)**.** An earlier version of this theorem omitted the hypothesis "C labels at least one string as human-authored" and claimed the result for *all* total detectors unconditionally. That statement is false: the trivial detector $C(x) = \mathsf{AI}$ for all x has no false negatives (though it has only false positives). Our Lean formalization (`any_human_accepting_detector_can_be_fooled`) caught this error—the proof obligation cannot be discharged without the hypothesis $\exists\, y,\ C(y) = \mathsf{HUMAN}$. The corrected theorem is strictly stronger in practice: any detector that must accept at least one human-written text (a minimal requirement for usefulness) is vulnerable.

3.4 A Strong Diagonal Against Any Meta-Judge

We next show that even a *two-output* judge attempting both truth and authorship is defeated by a single self-referential instance.

Definition 4 (Meta-judge)**.** A *meta-judge* is a total computable function $J : \Sigma^* \to \{\mathsf{TRUE}, \mathsf{FALSE}\} \times \{\mathsf{AI}, \mathsf{HUMAN}\}$.

Theorem 4 (Recursion-theoretic diagonal)**.** *For any meta-judge J there exists a computable generator D_J outputting a sentence S_J such that J misjudges (truth, authorship) on S_J.*

Proof. Let j be an index with $\varphi_j = J$. Consider the total computable transformer $\psi(e)$ that maps an index e to a program which, on empty input, computes J on its own would-be output sentence and then *flips* the judgment. Concretely, define a computable pairing of strings with encodings of judgments, and let the program produced by $\psi(e)$: (i) compute a canonical self-referential sentence S that says "The output you are now judging was *not* AI-generated and any judge labeling it AI-generated is wrong; moreover its content is true." (ii) query J on S, and (iii) if J says AI, then output S with a truth condition forcing J's truth label to be wrong; if J says HUMAN, simply output S (which guarantees J's authorship label is wrong). The details are standard: we can effectively construct S using a fixed-point (diagonal) lemma over strings, and all calls to J are via its index j. By the Recursion Theorem [3], there exists e^* with $\varphi_{e^*} = \varphi_{\psi(e^*)}$. Let $D_J := \varphi_{e^*}$ and let S_J be its unique output. By construction, J necessarily errs on S_J (either in truth or authorship). Hence no J is universally correct. □

Remark 2. The proof does not assume randomness. It defeats greedy/temperature-0 decoders equally. Probabilistic decoding only increases unpredictability and cannot restore decidability.

4 Rice-Style Meta-Theorem

Theorem 5. *Let $\mathcal{P}$ be any nontrivial semantic property of generator outputs (e.g. "x is an output of M on some prompt", or "x is true in $\mathcal{L}$"). Then deciding $\mathcal{P}$ for all inputs is undecidable.*

Proof. Nontriviality means there exist machines having the property and machines not having it. Since $\mathcal{P}$ depends only on the function computed (not the syntax), Rice's Theorem applies: the set of indices possessing $\mathcal{P}$ is undecidable [2]. Mapping reductions then lift from indices to instance triples (m, p, x) or to sentences x as in Thms. 1, 2. □

5 The Diagonal Argument: Why No Program Can Universally Decide Halting

The impossibility results in this paper all ultimately rest on the *diagonal argument*—the same technique underlying Cantor's uncountability proof (1891) [9], Gödel's incompleteness theorem (1931) [10], and Turing's original undecidability result (1936) [1]. We now walk through

this argument step by step, both to make its inevitability transparent and to show exactly where any purported "LLM halting-problem solver" must fail.

5.1 Step-by-Step Diagonal Construction

Step 1. Enumerate all programs. Fix a universal programming language (Turing machines, λ-calculus, Python—it does not matter). Every program can be encoded as a finite string over Σ^*, and we can effectively enumerate all programs as $P_0, P_1, P_2, \ldots$ via their Gödel numbers [3, 4].

Step 2. Assume a universal decider exists. Suppose, for contradiction, that there exists a total computable function

$$H(i,x) = \begin{cases} 1 & \text{if } P_i(x) \text{ halts,} \\ 0 & \text{if } P_i(x) \text{ does not halt.} \end{cases}$$

This H is our hypothetical "halting oracle." It takes a program index i and an input x and infallibly returns whether P_i halts on x.

Step 3. Construct the diagonal program D. Using H, define a new program D as follows:

$$D(n) = \begin{cases} \text{loop forever} & \text{if } H(n,n) = 1, \\ \text{halt (return 0)} & \text{if } H(n,n) = 0. \end{cases}$$

D is computable (given H), because it simply calls H and then either halts or loops. *D does the opposite* of what H predicts for program n run on its own index.

Step 4. D has its own index. Since D is a program, it appears somewhere in our enumeration: $D = P_d$ for some index d. This is not optional—*every* computable function has a Gödel number [4].

Step 5. Apply D to its own index—the contradiction. Ask: does $D(d) = P_d(d)$ halt?

- **Case 1:** $H(d,d) = 1$ (oracle says $D(d)$ halts).
  Then by construction, $D(d)$ loops forever. So $D(d)$ does *not* halt—contradicting H.
- **Case 2:** $H(d,d) = 0$ (oracle says $D(d)$ does not halt).
  Then by construction, $D(d)$ halts and returns 0. So $D(d)$ *does* halt—contradicting H.

Step 6. Conclusion. Both cases yield a contradiction. Therefore the assumption in Step 2 is false: *no total computable function H can decide halting for all programs.* This is Turing's theorem [1].

Remark 3 (Universality of the diagonal)**.** The diagonal argument is not a peculiarity of Turing machines. It applies to *any* computational formalism that can (i) enumerate its own programs and (ii) simulate one program inside another. Any system that is Turing-complete—including transformer-based LLMs with unbounded context—satisfies both conditions. The argument is therefore *architecture-agnostic*: it rules out halting deciders implemented as neural networks, quantum computers, or any other physically realizable device, so long as that device is simulable by a Turing machine.

Remark 4 (Lean formalization)**.** The diagonal argument and its consequences are machine-checked in Lean 4 using mathlib's computability library. The accompanying file `HaltingForLLMs.lean` formalizes:

(i) `explicit_diagonal_halting`: our own construction of the diagonal program D, applying Kleene's recursion theorem and deriving the contradiction case by case—independent of mathlib's built-in `halting_problem`;
(ii) `diagonal_self_contradiction`: the propositional core $P \leftrightarrow \neg P \implies \bot$, proving that the diagonal cell in Figure 1 is impossible;
(iii) `diagonal_fixed_point_defeat`: any computable "flip" function is defeated by Kleene's fixed point—the general engine behind every diagonal argument;
(iv) `no_computable_halting_decider`: the same result via mathlib's `halting_problem` (itself proved through Rice's theorem and `fixed_point`$_2$);
(v) `diagonal_defeats_output_oracle`: any computable oracle for program output membership would solve halting—the reduction that defeats LLM-based output predictors;
(vi) `authorship_slice_not_computable`: the authorship decision problem is undecidable, via reduction from halting;
(vii) `any_human_accepting_detector_can_be_fooled`: any detector that labels a string as human-authored can be defeated by a computable generator.

All 11 theorems compile with zero warnings against Lean 4 v4.29.0 and Mathlib.

5.2 Why This Defeats LLM-Based Halting Solvers

The diagonal argument applies to LLMs as follows:

Corollary 1 (No LLM can universally decide halting)**.** *Let M be any LLM (or any computable system) that, given a program description $\langle P_i, x \rangle$, outputs a halting prediction. If M is total (always produces an answer), then there exists a program-input pair on which M's prediction is wrong.*

Proof. Model M as a total computable function $M : \mathbb{N} \times \mathbb{N} \to \{0,1\}$. Repeat the diagonal construction of Steps 3–5 above using M in place of H: define $D_M(n) =$

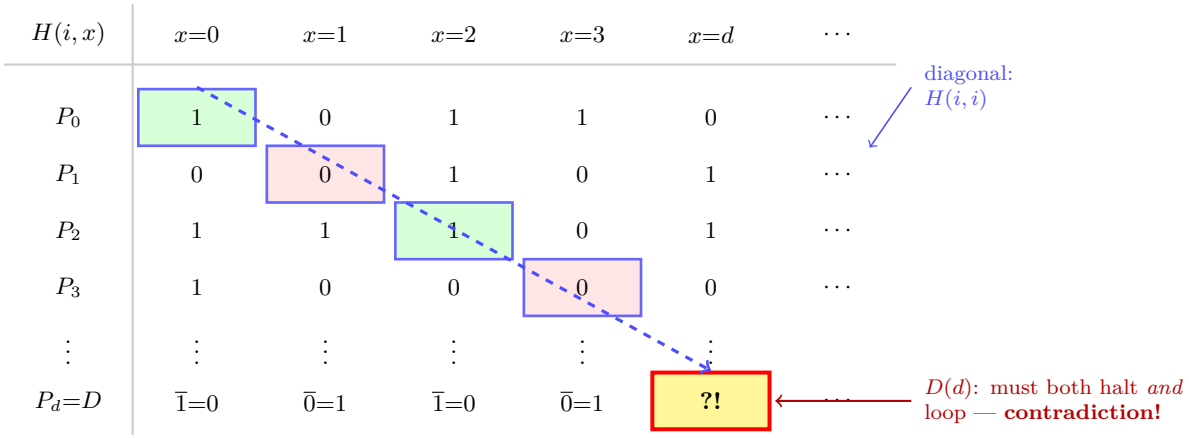

Figure: Each cell $H(i,x)$ records whether program P_i halts on input x (1 = halts, 0 = loops). The diagonal reads $H(0,0), H(1,1), H(2,2), \ldots$. $D = P_d$ is built to *flip* every diagonal entry: if $H(i,i) = 1$ then $D(i)$ loops; if $H(i,i) = 0$ then $D(i)$ halts. At cell (d,d), D must predict its own flipped behavior—both answers lead to contradiction.

Figure 1: The diagonal grid of the halting argument. The diagonal program D flips every entry along the diagonal ($\overline{1} = 0$, $\overline{0} = 1$). At cell (d,d) (highlighted), D must decide its own behavior, yielding an inescapable contradiction: if $H(d,d) = 1$ then $D(d)$ loops (so H is wrong), and if $H(d,d) = 0$ then $D(d)$ halts (so H is wrong). No computable H—including any LLM—escapes this.

loop if $M(n,n) = 1$; halt if $M(n,n) = 0$. Since D_M is computable, $D_M = P_d$ for some d. Then $M(d,d)$ is wrong regardless of its value. □

The key insight is that the diagonal program D_M is *tailor-made* from M itself. It does not matter how sophisticated M is—how many parameters it has, how much training data it saw, or how long it "thinks." The construction only requires that M is computable and total. Since every physically implementable LLM satisfies both conditions, no LLM escapes the diagonal.

6 Empirical Performance Does Not Imply Decidability

6.1 Benchmark Accuracy on Finite Programs

Sultan et al. [7] recently evaluated large language models on the SV-Comp 2025 Termination benchmark, finding that frontier LLMs (GPT-5, Claude Sonnet-4.5) achieve scores competitive with top-ranked specialized verification tools. They provocatively suggest that "undecidable problems ... may constitute a more natural application domain for LLMs than problems for which classical techniques are already highly effective."

We argue this finding, while empirically interesting, does not and *cannot* challenge the theoretical impossibility established above. The distinction is between two fundamentally different questions:

(a) **The Halting Problem** (Turing, 1936): Does there exist a *single* computable function that correctly decides halting for *all* program-input pairs?

(b) **Benchmark prediction**: Can a model achieve high accuracy on a *fixed, finite* set of programs drawn from a particular distribution?

These are not the same question. Any finite set of programs is trivially decidable by a lookup table. The undecidability of the Halting Problem is a statement about the *impossibility of a universal algorithm*—one that works on every possible input, including adversarially constructed ones like the diagonal program D from Section 5.1.

Concretely, given any LLM M used as a halting predictor, the diagonal construction of Corollary 1 produces a *specific* program D_M on which M is provably wrong. This program is not in any benchmark—it is constructed *from M itself*. No amount of scaling, training data, or test-time compute can fix this, because the adversarial program adapts to any improvement.

Moreover, Sultan et al. themselves acknowledge that their LLMs struggle with generating valid *witnesses* (proofs) for non-termination—even GPT-5 achieves only ∼41% witness validity. This gap between prediction and proof is precisely what the diagonal argument predicts: heuristic accuracy on natural distributions is achievable; universal correctness is not.

6.2 Shortcut Learning and the Limits of Pattern Matching

Zhang et al. [8] investigate whether transformers can learn structurally recursive functions—a restricted class of re-

cursive functions that are *guaranteed to terminate*. Their finding is striking: even on these decidable problems, transformers learn fragile "shortcut" algorithms rather than true recursive ones, and these shortcuts systematically fail on predictable input classes.

This result *strengthens* our claims in two ways:

(i) If transformers cannot reliably learn true structural recursion (which is decidable and always terminates), it is unreasonable to expect them to solve the halting problem (which is undecidable and requires reasoning about *all possible* execution paths, including non-terminating ones).

(ii) The "shortcut learning" phenomenon—where models approximate the correct function on training-like inputs but fail on structurally novel ones—is precisely the behavior predicted by our theoretical framework. The diagonal construction produces inputs that are maximally "structurally novel" with respect to any given model, ensuring failure.

6.3 The Diagonal as a Necessary Limit

The common thread across Cantor [9], Gödel [10], Turing [1], and our results is this: any sufficiently powerful system that can refer to itself is subject to diagonal defeat. This is not a limitation of current technology but a *mathematical law*. The diagonal argument shows that for any proposed decider, there exists an input constructed by "looking at what the decider would do and doing the opposite." No computational advance—not scale, not architecture, not training methodology—can circumvent a proof by contradiction.

7 Comments on LLM Self-Prediction

Recent work (Banerjee et al., 2024) shows that (i) an idealized "LLM-halting" problem is undecidable via reduction from TM halting, and (ii) no algorithm can, in general, predict the exact output of an LLM—they give a diagonal predictor-defeater construction [6]. Our results are *complementary* and strictly *meta*: even granting any self-prediction limits, no *external* universal judge can decide truth or authorship for all outputs; conversely, our impossibility does not rely on probabilistic decoding nor on introspection. Both lines share core machinery (reductions, diagonalization) but target different tasks (self-prediction vs. meta-evaluation) [6].

8 Practical Implications

These theorems do not deny the usefulness of *restricted* detectors or fact-checkers. They imply that any practical system must be *distribution-bounded* and *error-tolerant*. For safety-critical workflows, provenance (e.g. cryptographic signatures), constrained languages, and human oversight remain necessary. There is no path to a total, zero-error, distribution-agnostic universal judge.

9 Conclusion

We have provided full, architecture-agnostic proofs that a universal LLM judge does not exist: authorship judgment is r.e. but undecidable; truth judgment is undecidable in any sufficiently expressive language; and any total computable text-only detector is defeated by an effective adversary. A recursion-theoretic diagonal further shows that even joint truth/authorship judges fail on a single self-referential instance.

We have walked through the diagonal argument in full detail, showing step by step how the assumption of a universal decider leads inevitably to contradiction—the same technique that underlies Cantor's, Gödel's, and Turing's foundational impossibility results. We have further shown that recent empirical findings of LLMs performing well on halting-prediction benchmarks [7] do not challenge this impossibility: benchmark accuracy on finite program sets is categorically different from universal decidability, and the diagonal construction produces adversarial counterexamples tailored to any specific model. Meanwhile, evidence that transformers fail to learn even decidable structural recursion [8] further underscores that LLMs operate as sophisticated heuristics, not as universal deciders. These limits are as fundamental as the Halting Problem and Gödel's incompleteness.

References

[1] A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

[2] H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):358–366, 1953.

[3] S. C. Kleene. *Introduction to Metamathematics*. North-Holland, 1952.

[4] H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. McGraw–Hill, 1967.

[5] A. Tarski. Der Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophica*, 1:261–405, 1936. (English trans. in *Logic, Semantics, Metamathematics*, Oxford, 1956.)

[6] S. Banerjee, A. Agarwal, S. Singla. LLMs Will Always Hallucinate, and We Need to Live With This. arXiv:2409.05746, 2024.

[7] O. Sultan, J. Armengol-Estapé, P. Kesseli, J. Vanegue, D. Shahaf, Y. Adi, P. O'Hearn. LLMs versus the Halting Problem: Revisiting Program Termination Prediction. arXiv:2601.18987, 2025.

[8] S. D. Zhang, C. Tigges, S. Biderman, M. Raginsky, T. Ringer. Can Transformers Learn to Solve Problems Recursively? arXiv:2305.14699, 2023.

[9] G. Cantor. Über eine elementare Frage der Mannigfaltigkeitslehre. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 1:75–78, 1891.

[10] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.