A Theory for Worst-Case vs. Average-Case Guarantees for LLMs

Noga Amit* UC Berkeley
Shafi Goldwasser* UC Berkeley
Orr Paradise* UC Berkeley
Guy N. Rothblum* Apple

Abstract

How can we trust the correctness of a learned model on a particular input of interest? Model accuracy is typically measured *on average* over a distribution of inputs, giving no guarantee for any fixed input. This paper proposes a theoretically-founded solution to this problem: to train *Self-Proving models* that prove the correctness of their output to a verification algorithm V via an Interactive Proof. Self-Proving models satisfy that, with high probability over an input sampled from a given distribution, the model generates a correct output *and* successfully proves its correctness to V. The *soundness* property of V guarantees that, for *every* input, no model can convince V of the correctness of an incorrect output. Thus, a Self-Proving model proves correctness of most of its outputs, while *all* incorrect outputs (of any model) are detected by V. We devise and analyze two generic methods for learning Self-Proving models: *Transcript Learning (TL)* which relies on access to transcripts of accepting interactions, and *Reinforcement Learning from Verifier Feedback (RLVF)* which trains a model by emulating interactions with the verifier.

1 Introduction

Bob is studying for his algebra exam and stumbles upon a question Q that he cannot solve. He queries a Large Language Model (LLM) for the answer, and it responds with a number: 42. Bob is aware of recent research showing that the LLM attains a 90% score on algebra benchmarks (cf. Frieder et al. 2023), but should he trust that the answer to his particular question Q is indeed 42?

Bob could ask the LLM to explain its answer in natural language. Though he must proceed with caution, as the LLM might try to convince him of an incorrect answer [Turpin et al., 2023]. Moreover, even if 42 is the correct answer, the LLM may fail to produce a convincing proof [Wang et al., 2023]. If only the LLM could formally prove its answer, Bob would verify the proof and be convinced.

This paper initiates the study of *Self-Proving models* (Fig. 1) that prove the correctness of their answers via an Interactive Proof system [Goldwasser et al., 1985]. Self-Proving models successfully convince a verification algorithm V with *worst-case soundness guarantees*: for any question, V rejects all incorrect answers with high probability over the interaction. This guarantee holds even against provers that have access to V's specification, and unbounded computational power.

Our contributions are as follows.

- We define Self-Proving models (Section 2).
- We propose two methods for learning Self-Proving models (Section 3). The first, *Transcript Learning (TL)*, relies on access to transcripts of accepting interactions. The second method, *Reinforcement Learning from Verifier Feedback (RLVF)*, trains a model by emulating interactions with the verifier.

*Authors listed alphabetically.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

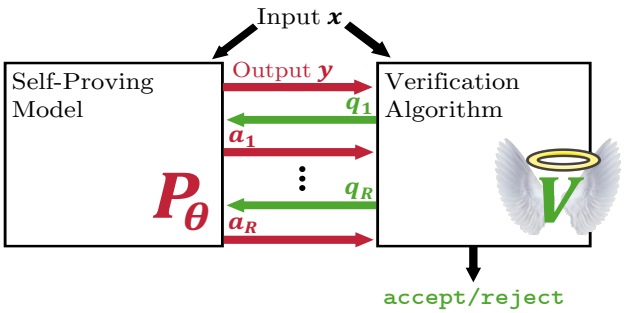

Figure 1: **Self-Proving models.** For input x, Self-Proving model P_θ generates an output y and sends it to a Verification Algorithm V. Then, over $i \in [R]$ rounds, V sends query q_i, and receives an answer a_i from P_θ. Finally, V decides ("accept/reject") whether it is convinced that y is a correct output for x.

- We prove gradient approximation lemmas for both methods (Lemmas 3.2 and 3.3), and a convergence bounds for TL under convexity and Lipschitzness assumptions (Section 4). These are supplemented by empirical validation on a simple arithmetic capability (Appendix F). Code and data are at `https://github.com/orrp/self-proving-models`

This paper develops a theory of learned models that prove their own correctness via an interactive proof system, and thus lies at the intersection of machine learning and Interactive Proof systems. We defer the discussion of relevant prior work from these areas to the related work section in Appendix A. The rich and well-studied question of *which* settings are verifiable within an interactive proof system is beyond our scope. Our theory is general in that it applies to *any* setting which is verifiable within an interactive proof system, e.g., any decision problem solvable in polynomial space [Shamir, 1992]. For a broader introduction to proof systems, see Goldreich [2008].

2 Defining Self-Proving Models

We introduce and formally define our learning framework in which models prove the correctness of their output. We start with preliminaries from the learning theory and proof systems literatures in Section 2.1. We then introduce our main definition in Section 2.2.

2.1 Preliminaries

Let Σ be a finite set of tokens and Σ^* denote the set of finite sequences of such tokens. We consider sequence-to-sequence models $F_\theta \colon \Sigma^* \to \Sigma^*$, which are total functions that produce an output for each possible input sequence. A model is parameterized by a real-valued, finite dimensional vector θ. We consider models as *randomized* functions, meaning that $F_\theta(x)$ is a random variable over Σ^*, of which samples are denoted by $y \sim F_\theta(x)$.

Before we can define models that prove their own correctness, we must first define correctness. Correctness is defined with respect to an input distribution μ over Σ^*, and a ground-truth F^* that defines correct answers. For simplicity of presentation, we focus on the case that each input $x \in \Sigma^*$ has exactly one correct output $F^*(x) \in \Sigma^*$, and a zero-one loss function on outputs (the general case is deferred to Appendix C). The fundamental goal of machine learning can be thought of as learning a model of the ground-truth F^*. Formally,

Definition 2.1 (Correctness)**.** *Let μ be a distribution of input sequences in Σ^* and let $F^* \colon \Sigma^* \to \Sigma^*$ be a fixed (deterministic) ground-truth function. For any $\alpha \in [0,1]$, we say that model F_θ is α-correct (with respect to μ) if*

$$\Pr_{\substack{x\sim\mu \\ y\sim F_\theta(x)}} [y = F^*(x)] \geq \alpha.$$

An *interactive proof system* [Goldwasser et al., 1985] is a protocol carried out between an efficient *verifier* and a computationally unbounded *prover*. The prover attempts to convince the verifier of the

correctness of some assertion, while the verifier accepts only correct claims. The prover is powerful yet untrusted; in spite of this, the verifier must reject false claims with high probability.

In the context of this work, it is important to note that the verifier is *manually-defined* (as opposed to learned). Formally, the verifier is a probabilistic polynomial-time algorithm tailored to a particular ground-truth capability F^*. Informally, the verifier is the anchor of trust: think of the verifier as an efficient and simple algorithm, hosted in a trustworthy environment.

Given an input $x \in \Sigma^*$, the model F_θ "claims" that $y \sim F_\theta(x)$ is correct. We now define what it means to *prove* this claim. We will use P_θ to denote Self-Proving models, noting that they are formally the same object[2] as non-Self-Proving ("vanilla") models F_θ. This notational change is to emphasize that P_θ first outputs $y \sim P_\theta(x)$ *and is then prompted by the verifier*, unlike F_θ who only generates an output $y \sim F_\theta(x)$.

A Self-Proving model proves that $y \sim P_\theta(x)$ is correct to a verifier V over the course of R rounds of interaction (Figure 1). In each round $i \in [R]$, verifier V queries P_θ on a sequence $q_i \in \Sigma^*$ to obtain an answer $a_i \in \Sigma^*$; once the interaction is over, V accepts or rejects. For fixed $x, y \in \Sigma^*$, the decision of V after interacting with P_θ is a random variable over V's decision (accept/reject), determined by the randomness of V and P_θ. The decision random variable is denoted by $\langle V, P_\theta \rangle (x, y)$.

Next, we present a definition of Interactive Proofs restricted to our setting.

Definition 2.2. *Fix a soundness error $s \in (0, 1)$, a finite set of tokens Σ and a ground-truth $F^* \colon \Sigma^* \to \Sigma^*$. A* verifier V (in an Interactive Proof) for F^* *is a probabilistic polynomial-time algorithm that is given explicit inputs $x, y \in \Sigma^*$ and black-box (oracle) query access to a prover P.*[3] *It* interacts *with P over R rounds (see Figure 1) and outputs a decision $\langle V, P \rangle (x, y) \in \{\text{reject}, \text{accept}\}$. Verifier V satisfies the following two guarantees:*

- Completeness: *There exists an* honest prover P^* *such that, for all $x \in \Sigma^*$,*

$$\Pr[\langle V, P^* \rangle (x, F^*(x)) \text{ accepts}] = 1,$$

  where the probability is over the randomness of V.[4]

- Soundness: *For all P and for all $x, y \in \Sigma^*$, if $y \neq F^*(x)$ then*

$$\Pr[\langle V, P \rangle (x, y) \text{ accepts}] \leq s,$$

  where the probability is over the randomness of V and P, and s is the soundness error.

The efficiency of an interactive proof is usually measured with respect to four parameters: the round complexity R, the communication complexity (the overall number of bits transferred during the interaction), P^*'s efficiency and V's efficiency. These complexity measures scale with the computational complexity of computing the ground-truth F^*. For example, an interactive proof for a complex F^* may require multiple rounds of interaction.

Remark 2.3 (Verifier efficiency)**.** *Definition 2.2 requires that V is a polynomial-time algorithm whereas provers are unbounded. This captures a requirement for* efficient verification*. We chose polynomial time as a measure of efficiency because it is common in the Proof systems literature. That said, one could adapt Definition 2.2 to fit alternative efficiency measures, such as space complexity [Condon and Lipton, 1989] or circuit depth [Goldwasser et al., 2007]. Regardless of which measure is taken, to avoid a trivial definition it is crucial that V should be more efficient than the honest prover P^*; else, V can simply execute P^* to perform the computation itself.*

By definition, the soundness error s of a verifier V bounds the probability that it is mistakenly convinced of an incorrect output; in that sense, the smaller s, the "better" the verifier V. In our setting, we think of a manually-defined verifier V who is formally proven (by a human) to have a small soundness error by analysis of V's specification.

As depicted in Figure 1, each of the model's answers depends on all previous queries and answers in the interaction. This captures the setting of *stateful models*, e.g. a session with a chatbot.

[2] Both are randomized mappings from Σ^* to Σ^*.

[3] We intentionally write P rather than P_θ: Interactive Proofs are defined with respect to all possible provers, not just parameterized ones.

[4] WLOG, the honest prover is deterministic by fixing the optimal randomness of a randomized prover.

Table 1: **Formal guarantees.** Completeness and soundness are fundamental guarantees of a verification algorithm V. Verifiability (novel in this work) is a feature of a model P_θ with respect to a verifier V and input distribution μ. Importantly, V's soundness holds for any input x and output y.

	Guarantee	Type	Def.
V	Completeness & Soundness	Worst-case: $\forall x, y$	2.2
P_θ	Verifiability	Average-case: $x \sim \mu,\ y \sim P_\theta(x)$	2.4

Towards defining Self-Proving models (Section 2.2), let us observe the following. Completeness and soundness are *worst-case guarantees*, meaning that they hold for all possible inputs $x \in \Sigma^*$. In particular, completeness implies that for all $x \in \Sigma^*$, the honest prover P^* convinces V of the correctness of $F^*(x)$; in classical proof systems there is no guarantee that an "almost honest" prover can convince the verifier (cf. Paradise 2021). Yet, if we are to *learn* a prover P_θ, we cannot expect it to agree with P^* perfectly, nor can we expect it to always output $F^*(x)$. Indeed, Self-Proving models will have a *distributional guarantee* with respect to inputs $x \sim \mu$. This distinction is summarized in Table 1.

2.2 Self-Proving Models

We define the *Verifiability* of a model P_θ with respect to an input distribution μ and a verifier V. Intuitively, Verifiability captures the ability of the model to prove the correctness of its answer $y \sim P_\theta(x)$, when the input x is sampled from μ. We refer to models capable of proving their own correctness as *Self-Proving models*. Notice that, as in Definition 2.2, the verifier is fixed and agnostic to the choice of the Self-Proving model.

Definition 2.4 (Self-Proving model)**.** *Fix a verifier V for a ground-truth $F^*\colon \Sigma^* \to \Sigma^*$ as in Definition 2.2, and a distribution μ over inputs Σ^*. The* Verifiability *of a model $P_\theta\colon \Sigma^* \to \Sigma^*$ is defined as*

$$\mathrm{ver}_{V,\mu}(\theta) \coloneqq \Pr_{\substack{x\sim\mu \\ y\sim P_\theta(x)}} \left[\langle V, P_\theta\rangle\,(x, y) \text{ accepts}\right]. \tag{1}$$

*We say that model P_θ is β-*Self-Proving *with respect to V and μ if $\mathrm{ver}_{V,\mu}(\theta) \geq \beta$.*

Remark 2.5 (Verifiability $\implies$ correctness)**.** *Notice that the ground-truth F^* does not appear in Definition 2.4 except for the first sentence. Indeed, once it is established that V is a verifier for F^* (as per Definition 2.2), then Verifiability w.r.t V implies correctness w.r.t F^*: Consider any input distribution μ, ground-truth F^*, and a verifier V for F^* with soundness error s. By a union bound, if a model P_θ is β-Verifiable, then it is $(\beta - s)$-correct. That is to say, Verifiability is formally a stronger guarantee than correctness when V has small soundness error s.*

As depicted in Figure 1, a Self-Proving model P_θ plays a dual role: first, it generates an output $y \sim P_\theta(x)$, and then it proves the correctness of this output to V. Note also that Verifiability is a feature of a *model*, unlike completeness and soundness which are features of a *verifier* (see Table 1).

The benefit of Verifiability over correctness is captured by the following scenario. Alice wishes to use a model P_θ to compute some functionality F^* on an input x_0 in a high risk setting. Alice generates $y_0 \sim P_\theta(x_0)$. Should Alice trust that y_0 is correct? If Alice has a held-out set of labeled samples, she can estimate P_θ's average correctness on μ. Unfortunately, (average) correctness provides no guarantee regarding the correctness of the particular (x_0, y_0) that Alice has in hand. If, however, Alice has access to a verifier V for which P_θ is Self-Proving, then she can trust the model on an input-by-input (rather than average-case) basis: Alice can execute V on (x_0, y_0) and black-box access to P_θ. Soundness of V guarantees that if y_0 is incorrect, then V rejects with high probability, in which case Alice should either generate $P_\theta(x_0)$ again—or find a better model.

3 Algorithms for Learning Self-Proving Models

With a sound verifier V at hand, obtaining Self-Proving models with respect to V holds great promise: a user that prompts the model with input x does not need to take it on good faith that $P_\theta(x)$ is correct; she may simply verify this herself by executing the verification protocol. How, then, can we learn models that are not just approximately-correct, but Self-Proving as well?

We focus on differentiable autoregressive models, and assume that the learner has access to input samples $x \sim \mu$ and correct outputs $F^*(x)$, as well as the verifier's specification (code). Additionally, the learner can emulate the verifier, as the latter is computationally efficient (Remark 2.3).

Importantly, we may *not* assume that the verifier V is differentiable—it is an arbitrary (efficient oracle) Turing machine—and so we cannot directly compute gradients of its decision with respect to model parameters. The challenge is to align the model with a verifier. Algorithms 1 and 2 address this challenge by (essentially) computing unbiased estimators for the Verifiability $\mathrm{ver}_V(\theta)$ or a surrogate (lower-bound) thereof. We formally prove these properties in Lemmas 3.2 and 3.3.

Our approach is inspired by Reinforcement Learning from Human Feedback [Christiano et al., 2017], a method for aligning models with human preferences, which has recently been used to align sequence-to-sequence models [Ouyang et al., 2022]. However, there are two important differences between humans and algorithmic verifiers: (1) Verifiers are efficient algorithms which may be emulated by the learner. This is unlike humans, whose preferences are costly to obtain. On the other hand, (2) verifiers make a single-bit decision at the end of an interaction, but cannot guide the prover (model) in intermediate rounds. In RL terms, this is known as the *exploration problem* for sparse reward signals (e.g. Ladosz et al. 2022).

The full specification of the learning model can be found in Appendix D.1. We will refer to the *transcript* of an interaction between a verifier and a prover (see Figure 1), denoted by $\pi = (y, q_1, a_1, \ldots, q_R, a_R)$. Let $\pi_{<s} \in \Sigma^{s-1}$ denote the s-token long prefix of π.

3.1 Transcript Learning

We first present an algorithm for learning Self-Proving models which relies on access to a distribution of accepting transcripts. We focus on the algorithm first, and then discuss how the learner may obtain accepting transcripts in Section 3.1.1. The idea is to learn a model not just of $x \mapsto y^*$ for a correct output y^*, but of $x \mapsto y^*\pi^*$, where π^* is a transcript of an interaction in which the verifier accepted. Formally, Transcript Learning assumes access to a *transcript generator*—a random variable over transcripts that faithfully represents the interaction of the verifier with some prover for a given input. An *honest transcript generator* is one which is fully supported on transcripts accepted by the verifier. These are defined next.

Definition 3.1 (Transcript generator)**.** *Fix a verifier V in a proof system of $R \in \mathbb{N}$ rounds. A transcript generator $\mathcal{T}_V$ for V is a randomized mapping from inputs $x \in \Sigma^*$ to transcripts $\pi = (y, q_1, a_1, \ldots, q_R, a_R) \in \Sigma^*$. For any input x, $\mathcal{T}_V(x)$ satisfies that for each $r \leq R$, the marginal of $\mathcal{T}_V(x)$ on the r^{th} query q_r agrees with the corresponding marginal of the query generator $(V_q)_r$.*[5] *A transcript generator $\mathcal{T}_V^* := \mathcal{T}_V$ is* honest *if it is fully supported on transcripts π^* for which the verifier accepts.*

Notice that for any verifier V, there is a one-to-one correspondence between transcript generators and (possibly randomized) provers. We intentionally chose *not* to specify a prover in Definition 3.1 to emphasize that transcripts can be "collected" independently of the honest prover (see completeness in Definition 2.2), and in fact can be collected "in advance" prior to learning (see Figure 2). As long as the generator is fully supported on honest transcripts, it can be used for Transcript Learning as depicted in Algorithm 1 and Figure 2.

TL trains a Self-Proving model by autoregressively optimizing towards generating accepting transcripts. At a high level, it works by repeatedly sampling $x \sim \mu$ and $y^*\pi^* \sim \mathcal{T}^*(x)$, and updating the logits $\log p_\theta$ towards agreeing with $y^*\pi^*$ via Gradient Ascent. While TL does not directly estimate the Verifiability gradient $\nabla_\theta \mathrm{ver}_V(\theta)$, we are able to show that it estimates a gradient of a lower-bounding function $A(\theta) \leq \mathrm{ver}_V(\theta)$. Therefore, as it ascends the gradient, it optimizes $\mathrm{ver}_V(\theta)$ via the surrogate. Formally, the lower-bounding function $A(\theta)$ is the agreement of the transcripts generated by the current model P_θ, with the transcripts generated by the honest transcript generator. That is, $A(\theta) := \Pr[\pi = \pi^*]$ where the probability is over $x \sim \mu$, $\pi_\theta \sim \mathcal{T}_V^\theta(x)$, and $\pi^* \sim \mathcal{T}_V^*(x)$.

[5] A *query generator* V_q corresponding to V takes as input a partial interaction and samples from the distribution over next queries by V. Formally, for any $r \leq R$, given input x, output y, and partial interaction $(q_i, a_i)_{i=1}^r$, $V_q(x, y, q_1, a_1, \ldots, q_r, a_r)$ is a random variable over Σ^{L_q}. For completeness' sake, we can say that when prompted with any sequence z that does not encode an interaction, $V_q(z)$ is fully supported on a dummy sequence $\perp \cdots \perp \in \Sigma^{L_q}$.

Algorithm 1: Transcript Learning (TL)

Hyperparameters: Learning rate $\lambda \in (0,1)$ and number of samples $N \in \mathbb{N}$.
Input: An autoregressive model family $\{P_\theta\}_{\theta \in \mathbb{R}^d}$, verifier specification (code) V, and sample access to an input distribution μ and an accepting transcript generator $\mathcal{T}^*_V(\cdot)$.
Output: A vector of parameters $\bar{\theta} \in \mathbb{R}^d$.

Initialize $\theta_0 := \vec{0}$.
for $i = 0, \ldots, N-1$ **do**
  Sample $x \sim \mu$ and $\pi^* = (y^*, q_1^*, a_1^*, \ldots, q_R^*, a_R^*) \sim \mathcal{T}^*_V(x)$. Denote $a_0 := y^*$.
  foreach *Round of interaction* $r = 0, \ldots, R$ **do**
    Let $S(r)$ denote the indices of the r^{th} answer a_r in π^*, and let $\pi_{<s}$ denote the prefix of the partial transcript $(y, q_1^*, a_1^*, \ldots, q_r^*)$.
    for $s \in S(r)$ **do**
      Compute `# Forwards and backwards pass`

$$\alpha_s(\theta_i) := \Pr_{\sigma \sim p_{\theta_i}(x\pi_{<s})}[\sigma = \pi^*_s]$$

$$\vec{d}_s(\theta_i) := \nabla_\theta \log \alpha_s(\theta_i) = \nabla_\theta \log \Pr_{\sigma \sim p_{\theta_i}(x\pi_{<s})}[\sigma = \pi^*_s].$$

  Update

$$\theta_{i+1} := \theta_i + \lambda \cdot \prod_{\substack{r \in [R] \cup \{0\} \\ s \in S(r)}} \alpha_s(\theta_i) \cdot \sum_{\substack{r \in [R] \cup \{0\} \\ s \in S(r)}} \vec{d}_s(\theta_i).$$

Output $\bar{\theta} := \frac{1}{N} \sum_{i \in [N]} \theta_i$.

Lemma 3.2 (TL gradient estimation)**.** *Fix an input distribution μ over Σ^* and a verifier V with round complexity R and answer length L_a. Fix an honest transcript generator $\mathcal{T}^*_V$. Let θ denote the parameters of the model P_θ, let $A(\theta)$ be as defined above and let the terms $S(r)$, $\alpha_s(\theta)$, and $\vec{d}_s(\theta)$ be as defined in Algorithm 1. Then,*

$$\nabla A(\theta) = \mathop{\mathbb{E}}_{\substack{x \sim \mu \\ \pi^* \sim \mathcal{T}^*_V}} \left[\prod_{\substack{r \in [R] \cup \{0\} \\ s \in S(r)}} \alpha_s(\theta) \cdot \sum_{\substack{r \in [R] \cup \{0\} \\ s \in S(r)}} \vec{d}_s(\theta) \right].$$

The proof is deferred to Appendix D.2. Note that Lemma 3.2 is true for *any* model P_θ. Moreover, the random vector over which the expectation is taken (in the right hand side) is precisely the direction of the update performed in Algorithm 1. In Section 4, we will use Lemma 3.2 to prove convergence bounds for TL under certain conditions.

3.1.1 Access to Accepting Transcripts

As mentioned, Transcript Learning relies on access to accepting transcripts. In this section we discuss how such access can be realized (grounded in the theory of Interactive Proofs).

Doubly-Efficient Interactive Proofs. *When the honest prover P^* is efficient (polynomial time)*, the learner (who has the code for P^* and V) can execute P^* on input x to collect accepting transcripts—asuming no distribution shift at inference time. This setting is formalized by the notion of *Doubly-Efficient Interactive Proofs* (DEIPs), introduced in the foundational work of Goldwasser et al. [2015], who construct DEIPs for all problems computable by log-space uniform families of polynomial-size circuits with polylogarithmic depth. Their protocols ensure that the verifier runs in nearly linear time, while the prover operates in polynomial time—or more generally, in time proportional to the circuit size. Later, Reingold et al. [2016] showed that any problem computable by a Turing machine running in polynomial time and sublinear space admits a constant-round DEIP. Subsequent theoretical works (e.g., Goldreich and Rothblum 2018a,b) have constructed DEIPs for specific problems, while applied works (e.g., Zhang et al. 2021, Thaler 2013) have improved the time and space complexity of such protocols in practice.

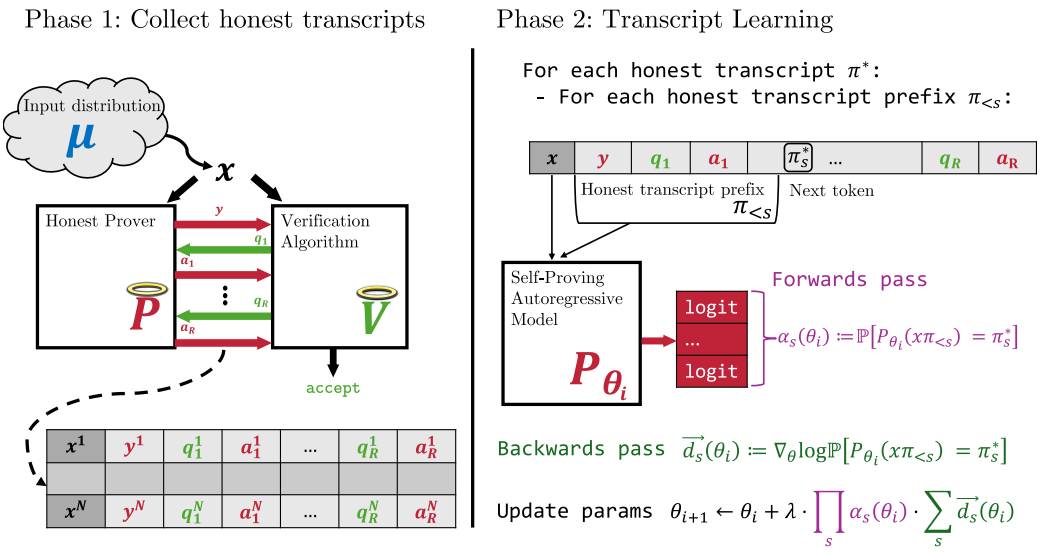

Figure 2: **Transcript Learning, visualized.** To understand Algorithm 1, consider the above visualization. In Phase 1, N honest transcripts are collected by interacting an honest prover with the Verifier; these serve as samples from the transcript generator $\mathcal{T}_V^*(x)$. Phase 2 runs Algorithm 1: for each transcript π^* (lines 2–3) and each prefix π_s (lines 4–6), the values $\alpha_s(\theta_i)$ and $\vec{d_s}(\theta_i)$ are computed via forward and backward passes (line 7), followed by a parameter update (line 8).

Backward Data Generation. Recent works on "AI for advanced mathematical problems" [Charton et al., 2021, Alfarano et al., 2024] propose to reverse the generation process of problem-solution pairs as follows: rather than sampling problems and searching for solutions, one first samples solutions and then constructs the corresponding problems. In our setting, this inspires the following approach. Suppose it is computationally hard in the worst case to start with the input x and generate an accepting transcript $\pi = (y, q_1, a_1, \ldots, q_R, a_R)$ for x. Could we instead jointly sample (x, π), or first sample a transcript π and then extract the input x for which π is an accepting transcript? As long as the resulting x's are distributed the same as inference time, this would enable Transcript Learning of such instances.

We explain the idea further with a cryptographic example. Setting up a Diffie–Hellman key-exchange scheme for security parameter 1^k requires producing as a public parameter a prime p in a factored form, namely $p - 1 = \Pi {q_i}^{\alpha_i}$ where q_i are primes. However, factoring $(p-1)$ is generally hard. Instead, one could go "backwards:" first generate $(p-1)$ in factored form by choosing primes q_i and exponents α_i [Kalai, 2003] and then testing if $p = 1 + \Pi {q_i}^{\alpha_i}$ is prime. By the prime number theorem, p is likely to be prime after a few attempts. In the context of our paper, one may ask a Self-Proving model to produce certified primes, training it on tuples $(x := p, \pi := (q_i, \alpha_i)_i)$.

3.2 Reinforcement Learning from Verifier Feedback (RLVF)

As mentioned in Section 3.1, Transcript Learning uses access to an honest transcript generator to estimate gradients of (a lower bound on) the Verifiability of a model P_θ. Next we present *Reinforcement Learning from Verifier Feedback* (RLVF, Algorithm 2), which estimates this gradient without access to a transcript generator.

Note that the parameters are updated (line 11) only when an accepting transcript was generated. This means that the learner can first fully generate the transcript (lines 6-7), and then take backwards passes (line 9) only if the transcript was accepted by V. This is useful in practice (e.g. when using neural models) as backwards passes are more computationally expensive than forwards passes.

On the other hand, this means that RLVF requires the parameter initialization θ_0 to have Verifiability bounded away from 0, so that accepting transcripts are sampled with sufficient probability. Fortunately, such a Self-Proving base model can be learned using TL. This gives a learning paradigm in which a somewhat-Self-Proving base model is learned with TL (with Verifiability $\delta \gg 0$), and

Algorithm 2: Reinforcement Learning from Verifier Feedback (RLVF)

Hyperparameters: Learning rate $\lambda \in (0,1)$ and number of samples $N \in \mathbb{N}$.
Input: An autoregressive model family $\{P_\theta\}_{\theta \in \mathbb{R}^d}$, initial parameters $\theta_0 \in \mathbb{R}^d$, verifier specification (code) V, and sample access to an input distribution μ.
Output: A vector of parameters $\bar{\theta} \in \mathbb{R}^d$.

1 **for** $i = 0, \ldots, N-1$ **do**
2 Sample $x \sim \mu$.
3 Initialize $a_0 := y \sim P_{\theta_i}(x)$.
4 **foreach** *Round of interaction* $r = 1, \ldots R$ **do**
5 Sample the r^{th} query `# Emulate the verifier`

$$q_r \sim V_q(x, a_0, q_1, a_1, \ldots, q_{r-1}, a_{r-1}).$$

Sample the r^{th} answer `# Forwards pass`

$$a_r \sim P_{\theta_i}(x, a_0, q_1, a_1, \ldots, q_r).$$

Let $\tau_r := (a_0, q_1, \ldots, a_{r-1}, q_r)$.
6 **for** $s \in [L_a]$ **do**
7 Let $a_{r,s}$ denote the s^{th} token in a_r. Compute `# Backwards pass`

$$\vec{d}_s(\theta_i) := \nabla_\theta \log \Pr_{\sigma \sim p_{\theta_i}(x\tau_r)}[\sigma = a_{r,s}].$$

8 **if** $V(x, y, q_1, a_1, \ldots, q_R, a_R)$ accepts **then**
9 Update

$$\theta_{i+1} := \theta_i + \lambda \cdot \sum_{\substack{r \in [R] \cup \{0\} \\ s \in [L_a]}} \vec{d}_s(\theta_i).$$

10 Output $\bar{\theta} := \frac{1}{N} \sum_{i \in [N]} \theta_i$.

then "amplified" to a fully Self-Proving model using RLVF. This can be seen as an adaptation of the method of Nair et al. [2018] to the setting of Self-Proving models.

When comparing Algorithms 1 and 2, we see that the latter (RLVF) does not keep track of the probabilities α_s. This is because, in RL terms, RLVF is an *on-policy* algorithm; it generates transcripts using the current learned model, unlike TL that samples them from a distribution whose parameterization is unknown to the learner. Hence, the update step in RLVF is simpler than TL.

We show that the update step in RLVF maximizes the Verifiability of P_θ.

Lemma 3.3 (RLVF gradient estimation)**.** *Fix an input distribution μ over Σ^* and a verifier V with round complexity R and answer length L_a. For any transcript $(x, y, q_1, \ldots, a_R)$ we let $\mathrm{Acc}_V(x, y, q_1, \ldots, a_R)$ denote the indicator random variable which equals 1 if and only if V accepts the transcript. For any model P_θ, denote by $\mathrm{ver}(\theta)$ the verifiability of P_θ with respect to V and μ (Definition 2.4). Then, for any θ,*

$$\nabla_\theta \mathrm{ver}(\theta) = \mathop{\mathbb{E}}_{\substack{x \sim \mu \\ y \sim P_\theta(x) \\ (q_r, a_r)_{r=1}^R}} \left[\mathrm{Acc}_V(x, y, q_1, \ldots, a_R) \cdot \sum_{\substack{r \in [R] \cup \{0\} \\ s \in [L_a]}} \vec{d}_s(\theta) \right]$$

where $(q_r, a_r)_{r=1}^R$ are as in lines 5-6 of Algorithm 2, and $\vec{d}_s(\theta)$ is as defined in line 8 therein.

Note that, because $\mathrm{Acc}_V(\cdot)$ is a 0-1 indicator of whether a transcript was accepted, then the right hand side of the above equation is precisely the direction of the step taken in RLVF (line 9). The proof of Lemma 3.3 can be found in Appendix D.3.

4 Convergence of Transcript Learning

As an application of Lemma 3.2, we prove that, under certain conditions, Transcript Learning (TL, Algorithm 1) is expected to output a Self-Proving model. While the theorem relies on simplifying assumptions such as convexity and Lipschitzness, it offers clean mathematical guarantees that illuminate the core dynamics of Transcript Learning. As we discuss following the theorem statement, such conditions are common in theoretical machine learning literature, allowing us to build intuition even when the assumptions may not hold in practice.

Theorem 4.1 (informal)**.** *Fix a verifier V, an input distribution μ, and an autoregressive model family $\{P_\theta\}_{\theta\in\mathbb{R}^d}$. Fix an honest transcript generator $\mathcal{T}_V^*$. Assume the following:*

1. *The* agreement $A(\theta)$*, informally defined as the probability that P_θ generates transcripts agreeing with $\mathcal{T}_V^*$, is concave in θ. Additionally, the logits of P_θ are B_{Lip}-Lipschitz in θ.*
2. *There exist parameters θ^* with $||\theta^*|| \leq B_{\mathrm{Norm}}$ such that P_{θ^*} is $(1-\varepsilon/2)$-Self Proving.*
3. *The number of tokens sent by the prover in the proof system is at most C.*

Then, in expectation, TL run on $O(C^2 B_{\mathrm{Norm}}^2 B_{\mathrm{Lip}}^2/\varepsilon^2)$ samples outputs a $(1-\varepsilon)$-Self Proving model.

The full statement and proof of Theorem 4.1 are deferred to Appendix D.4. Its conditions can be split into two parts. First (item 1), convexity and Lipschitzness, which are simplifying assumptions commonly needed to prove SGD convergence. While convexity does not hold in general for DNNs, analyzing convex settings provides clean mathematical tools for establishing foundational results—an approach commonly used in ML theory, particularly for DNNs. Indeed, several works have addressed the problem of proving convergence without convexity [Du et al., 2019, Bartlett et al., 2006, Khaled and Richtárik, 2023].

Norm-boundedness (item 2), on the other hand, is a *(necessary) realizability assumption*: if the architecture $\{P_\theta\}_\theta$ cannot be instantiated with parameters θ^*, then it cannot be trained to be Self-Proving. This assumption is well-grounded for transformer architectures, as recent theoretical work has established their Turing-completeness [Bhattamishra et al., 2020, Dehghani et al., 2019].

Finally (item 3), the bound C on the communication complexity of the prover in the Interactive Proof system. This parameter directly affects the efficiency of TL, as reflected in the *number of iterations* (and sampled transcripts): it depends on both the *optimization landscape complexity* $B_{\mathrm{Norm}}^2 B_{\mathrm{Lip}}^2/\varepsilon^2$ and the *communication complexity* C^2. Reducing communication has long been a central objective in the study of proof systems (e.g., Goldreich and Håstad 1998, Goldreich et al. 2002, Reingold et al. 2016). Theorem 4.1 formalizes how communication-efficient proof systems improve the performance of Self-Proving models.

Remark 4.2 (Towards a convergence theorem for RLVF)**.** *RLVF can be derived by viewing Self-Proving as a reinforcement learning problem in which the agent (prover) is rewarded when the verifier accepts. Indeed, RLVF is the Policy Gradient method [Sutton et al., 1999] for a verifier-induced reward. Convergence bounds for Policy Gradient methods are a challenging and active area of research (e.g. Agarwal et al. 2021), and so we leave the full analysis to future work.*

5 Conclusions

Trust between a learned model and its user is fundamental. Interactive Proofs [Goldwasser et al., 1985] provide a general framework for establishing trust via verification algorithms. This work shows that models can be trained to formally prove their outputs within such systems—we call these *Self-Proving* models. Self-Proving models connect the theory of Interactive Proofs with the goal of Trustworthy ML: they offer formal *worst-case soundness guarantees*; enabling users to be confident when their models generate correct answers—and detect incorrect answers with high probability.

We support our definition with two general-purpose training methods: Transcript Learning (TL) and Reinforcement Learning from Verifier Feedback (RLVF), whose analyses draw on learning theory, RL, and computational complexity. This work can be extended in several directions: finding conditions for the convergence of RLVF, improving sample complexity bounds for TL, or designing altogether different learning algorithms (e.g., by taking advantage of properties of the verifier).

Acknowledgments and Disclosure of Funding

We are grateful to Micah Carroll, Mark Rofin, Avishay Tal, Mojtaba Yaghoobzadeh and anonymous reviewers for their helpful comments. This research was supported by DARPA-TA1 under grant no. HR001119S0076, and by the Simons Collaboration on the Theory of Algorithmic Fairness.

References

Alekh Agarwal, Sham M. Kakade, Jason D. Lee, and Gaurav Mahajan. On the theory of policy gradient methods: Optimality, approximation, and distribution shift. *J. Mach. Learn. Res.*, 22: 98:1–98:76, 2021. URL http://jmlr.org/papers/v22/19-736.html.

Alberto Alfarano, François Charton, and Amaury Hayat. Global lyapunov functions: a long-standing open problem in mathematics, with symbolic transformers. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/aa280e73c4e23e765fde232571116d3b-Abstract-Conference.html.

Cem Anil, Guodong Zhang, Yuhuai Wu, and Roger B. Grosse. Learning to give checkable answers with prover-verifier games. *CoRR*, abs/2108.12099, 2021. URL https://arxiv.org/abs/2108.12099.

Peter L Bartlett, Michael I Jordan, and Jon D McAuliffe. Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156, 2006.

E. Bezout. *Theorie Generale Des Equations Algebriques*. Kessinger Publishing, 1779. ISBN 9781162056128. URL https://books.google.co.il/books?id=wQZvSwAACAAJ.

Satwik Bhattamishra, Arkil Patel, and Navin Goyal. On the computational power of transformers and its implications in sequence modeling. In Raquel Fernández and Tal Linzen, editors, *Proceedings of the 24th Conference on Computational Natural Language Learning, CoNLL 2020, Online, November 19-20, 2020*, pages 455–475. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.CONLL-1.37. URL https://doi.org/10.18653/v1/2020.conll-1.37.

Jonah Brown-Cohen, Geoffrey Irving, and Georgios Piliouras. Scalable AI safety via doubly-efficient debate. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=6jmdOTRMIO.

François Charton. Can transformers learn the greatest common divisor? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 6-11, 2024*. OpenReview.net, 2024.

Francois Charton, Amaury Hayat, and Guillaume Lample. Learning advanced mathematical computations from examples. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=-gfhS00XfKj.

Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4299–4307, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/d5e2c0adad503c91f91df240d0cd4e49-Abstract.html.

Anne Condon and Richard J. Lipton. On the complexity of space bounded interactive proofs (extended abstract). In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 462–467. IEEE Computer Society, 1989. doi: 10.1109/SFCS.1989.63519. URL https://doi.org/10.1109/SFCS.1989.63519.

Anne Condon, Joan Feigenbaum, Carsten Lund, and Peter W. Shor. Probabilistically checkable debate systems and nonapproximability of pspace-hard functions. *Chic. J. Theor. Comput. Sci.*, 1995, 1995. URL http://cjtcs.cs.uchicago.edu/articles/1995/4/contents.html.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

Simon S. Du, Jason D. Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 1675–1685. PMLR, 2019. URL http://proceedings.mlr.press/v97/du19c.html.

Simon Frieder, Luca Pinchetti, Alexis Chevalier, Ryan-Rhys Griffiths, Tommaso Salvatori, Thomas Lukasiewicz, Philipp Petersen, and Julius Berner. Mathematical capabilities of chatgpt. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/58168e8a92994655d6da3939e7cc0918-Abstract-Datasets_and_Benchmarks.html.

Oded Goldreich. Probabilistic proof systems: A primer. *Found. Trends Theor. Comput. Sci.*, 3(1): 1–91, 2008. doi: 10.1561/0400000023. URL https://doi.org/10.1561/0400000023.

Oded Goldreich and Johan Håstad. On the complexity of interactive proofs with bounded communication. *Inf. Process. Lett.*, 67(4):205–214, 1998. doi: 10.1016/S0020-0190(98)00116-1. URL https://doi.org/10.1016/S0020-0190(98)00116-1.

Oded Goldreich and Guy Rothblum. Counting t-cliques: Worst-case to average-case reductions and direct interactive proof systems. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 77–88, 2018a. doi: 10.1109/FOCS.2018.00017.

Oded Goldreich and Guy N. Rothblum. Simple doubly-efficient interactive proof systems for locally-characterizable sets. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pages 18:1–18:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018b. doi: 10.4230/LIPICS.ITCS.2018.18. URL https://doi.org/10.4230/LIPIcs.ITCS.2018.18.

Oded Goldreich, Salil P. Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. *Comput. Complex.*, 11(1-2):1–53, 2002. doi: 10.1007/S00037-002-0169-0. URL https://doi.org/10.1007/s00037-002-0169-0.

Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304. ACM, 1985. doi: 10.1145/22145.22178. URL https://doi.org/10.1145/22145.22178.

Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. Verifying and decoding in constant depth. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 440–449. ACM, 2007. doi: 10.1145/1250790.1250855. URL https://doi.org/10.1145/1250790.1250855.

Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4):27:1–27:64, 2015. doi: 10.1145/2699436. URL https://doi.org/10.1145/2699436.

Shafi Goldwasser, Guy N. Rothblum, Jonathan Shafer, and Amir Yehudayoff. Interactive proofs for verifying machine learning. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185

of *LIPIcs*, pages 41:1–41:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi: 10.4230/LIPICS.ITCS.2021.41. URL https://doi.org/10.4230/LIPIcs.ITCS.2021.41.

Thomas Gransden, Neil Walkinshaw, and Rajeev Raman. SEPIA: search for proofs using inferred automata. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 246–255. Springer, 2015. doi: 10.1007/978-3-319-21401-6_16. URL https://doi.org/10.1007/978-3-319-21401-6_16.

Lewis Hammond and Sam Adam-Day. Neural interactive proofs. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=R2834dhBlo.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In Joaquin Vanschoren and Sai-Kit Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Geoffrey Irving, Paul F. Christiano, and Dario Amodei. AI safety via debate. *CoRR*, abs/1805.00899, 2018. URL http://arxiv.org/abs/1805.00899.

Adam Tauman Kalai. Generating random factored numbers, easily. *Journal of Cryptology*, 16(4): 287–289, 2003.

Ahmed Khaled and Peter Richtárik. Better theory for SGD in the nonconvex world. *Trans. Mach. Learn. Res.*, 2023, 2023. URL https://openreview.net/forum?id=AU4qHN2VkS.

Jan Hendrik Kirchner, Yining Chen, Harri Edwards, Jan Leike, Nat McAleese, and Yuri Burda. Prover-verifier games improve legibility of LLM outputs. *CoRR*, abs/2407.13692, 2024. doi: 10.48550/ARXIV.2407.13692. URL https://doi.org/10.48550/arXiv.2407.13692.

Donald E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms*. Addison-Wesley, 1969. ISBN 0201038021. URL https://www.worldcat.org/oclc/310551264.

Pawel Ladosz, Lilian Weng, Minwoo Kim, and Hyondong Oh. Exploration in deep reinforcement learning: A survey. *Inf. Fusion*, 85:1–22, 2022. doi: 10.1016/J.INFFUS.2022.03.003. URL https://doi.org/10.1016/j.inffus.2022.03.003.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tülu 3: Pushing frontiers in open language model post-training. *CoRR*, abs/2411.15124, 2024. doi: 10.48550/ARXIV.2411.15124. URL https://doi.org/10.48550/arXiv.2411.15124.

Nayoung Lee, Kartik Sreenivasan, Jason D. Lee, Kangwook Lee, and Dimitris Papailiopoulos. Teaching arithmetic to small transformers. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 6-11, 2024*. OpenReview.net, 2024.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 6-11, 2024*. OpenReview.net, 2024.

Shikhar Murty, Orr Paradise, and Pratyusha Sharma. Pseudointelligence: A unifying lens on language model evaluation. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 7284–7290. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-EMNLP.485. URL https://doi.org/10.18653/v1/2023.findings-emnlp.485.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 6292–6299. IEEE, 2018. doi: 10.1109/ICRA.2018.8463162. URL https://doi.org/10.1109/ICRA.2018.8463162.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html.

Orr Paradise. Smooth and strong pcps. *Comput. Complex.*, 30(1):1, 2021. doi: 10.1007/S00037-020-00199-3. URL https://doi.org/10.1007/s00037-020-00199-3.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *CoRR*, abs/2009.03393, 2020. URL https://arxiv.org/abs/2009.03393.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following. *CoRR*, abs/2507.02833, 2025. doi: 10.48550/ARXIV.2507.02833. URL https://doi.org/10.48550/arXiv.2507.02833.

Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 49–62, 2016. doi: 10.1145/2897518.2897652.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Guy N. Rothblum, Salil P. Vadhan, and Avi Wigderson. Interactive proofs of proximity: delegating computation in sublinear time. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 793–802. ACM, 2013. doi: 10.1145/2488608.2488709. URL https://doi.org/10.1145/2488608.2488709.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press, 2014. ISBN 978-1-10-705713-5. URL http://www.cambridge.org/de/academic/subjects/computer-science/pattern-recognition-and-machine-learning/understanding-machine-learning-theory-algorithms.

Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992. doi: 10.1145/146585.146609. URL https://doi.org/10.1145/146585.146609.

Zirui Song, Guangxian Ouyang, Mingzhe Li, Yuheng Ji, Chenxi Wang, Zixiang Xu, Zeyu Zhang, Xiaoqing Zhang, Qian Jiang, Zhenhao Chen, Zhongzhi Li, Rui Yan, and Xiuying Chen. Maniplvm-r1: Reinforcement learning for reasoning in embodied manipulation with large vision-language models. *CoRR*, abs/2505.16517, 2025. doi: 10.48550/ARXIV.2505.16517. URL https://doi.org/10.48550/arXiv.2505.16517.

Richard S. Sutton, David A. McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 1057–1063. The MIT Press, 1999. URL http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. Proofwriter: Generating implications, proofs, and abductive statements over natural language. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Findings of the Association for Computational Linguistics: ACL/IJCNLP 2021, Online Event, August 1-6, 2021*, volume ACL/IJCNLP 2021 of *Findings of ACL*, pages 3621–3634. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.FINDINGS-ACL.317. URL https://doi.org/10.18653/v1/2021.findings-acl.317.

Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 71–89, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nat.*, 625(7995):476–482, 2024. doi: 10.1038/S41586-023-06747-5. URL https://doi.org/10.1038/s41586-023-06747-5.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/ed3fea9033a80fea1376299fa7863f4a-Abstract-Conference.html.

Jonathan Uesato, Nate Kushman, Ramana Kumar, H. Francis Song, Noah Y. Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. *CoRR*, abs/2211.14275, 2022. doi: 10.48550/ARXIV.2211.14275. URL https://doi.org/10.48550/arXiv.2211.14275.

Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984. doi: 10.1145/1968.1972. URL https://doi.org/10.1145/1968.1972.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Stephan Wäldchen, Kartikey Sharma, Berkant Turan, Max Zimmer, and Sebastian Pokutta. Interpretability guarantees with Merlin-Arthur classifiers. In Sanjoy Dasgupta, Stephan Mandt, and Yingzhen Li, editors, *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, pages 1963–1971. PMLR, 02–04 May 2024. URL https://proceedings.mlr.press/v238/waldchen24a.html.

Boshi Wang, Xiang Yue, and Huan Sun. Can chatgpt defend its belief in truth? evaluating LLM reasoning via debate. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 11865–11881. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-EMNLP.795. URL https://doi.org/10.18653/v1/2023.findings-emnlp.795.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Sean Welleck, Jiacheng Liu, Ximing Lu, Hannaneh Hajishirzi, and Yejin Choi. Naturalprover: Grounded mathematical proof generation with language models. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/1fc548a8243ad06616eee731e0572927-Abstract-Conference.html.

Fang Wu, Weihao Xuan, Ximing Lu, Zaïd Harchaoui, and Yejin Choi. The invisible leash: Why RLVR may not escape its origin. *CoRR*, abs/2507.14843, 2025a. doi: 10.48550/ARXIV.2507.14843. URL https://doi.org/10.48550/arXiv.2507.14843.

Jialong Wu, Shaofeng Yin, Ningya Feng, and Mingsheng Long. Rlvr-world: Training world models with reinforcement learning. *CoRR*, abs/2505.13934, 2025b. doi: 10.48550/ARXIV.2505.13934. URL https://doi.org/10.48550/arXiv.2505.13934.

Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J. Prenger, and Animashree Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/4441469427094f8873d0fecb0c4e1cee-Abstract-Datasets_and_Benchmarks.html.

Mengjiao Yang, Dale Schuurmans, Pieter Abbeel, and Ofir Nachum. Chain of thought imitation with procedure cloning. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/ebdb990471f653dffb425eff03c7c980-Abstract-Conference.html.

Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 159–177. ACM, 2021. doi: 10.1145/3460120.3484767. URL https://doi.org/10.1145/3460120.3484767.

Sheng Zhang, Qianchu Liu, Guanghui Qin, Tristan Naumann, and Hoifung Poon. Med-rlvr: Emerging medical reasoning from a 3b base model via reinforcement learning. *CoRR*, abs/2502.19655, 2025. doi: 10.48550/ARXIV.2502.19655. URL https://doi.org/10.48550/arXiv.2502.19655.

Jiecheng Zhou, Qinghao Hu, Yuyang Jin, Zerui Wang, Peng Sun, Yuzhe Gu, Wenwei Zhang, Mingshu Zhai, Xingcheng Zhang, and Weiming Zhang. Rl in the wild: Characterizing rlvr training in llm deployment. *arXiv preprint arXiv:2509.25279*, 2025.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Definitions are formally stated in the body of the paper. Claims are stated and proven in the paper.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.

- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are discussed throughout the body and in Appendix B.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Definitions and theorems are formally stated and proved. Formal proofs appear in the appendix, with an overview and discussion of the theorems in the body of the paper.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.

- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Experiments are presented in Appendix F with full details needed for reproducibility in Appendix G.2.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Model, data and code are released in `https://github.com/orrp/self-proving-models`

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.

- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Yes, in Appendix G.2.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Standard errors are reported.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).

- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Yes, in Appendix G.2

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms with the code of ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper discusses societal impacts throughout the body. As this is a theoretical paper, immediate impact is limited.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.

- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The experiments on a simple arithmetic capability pose no such risk.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Yes, codebases used in the experiments are cited and used under the appropriate license.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

Justification: The details of the model and data released as a part of this paper are detailed in `https://github.com/orrp/self-proving-models`

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: No human subjects were used.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: NO IRB approval was required.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: The method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Related Work

We overview related work from machine learning (ML) and interactive proof systems (IPs) literature.

ML and IPs. IPs have found numerous applications in ML towards a diverse set of goals. Anil et al. [2021] introduce Prover–Verifier Games (PVGs), a game-theoretic framework for learned provers and learned verifiers. Since our paper initially appeared, PVGs were further investigated in at least two subsequent works: Hammond and Adam-Day [2025] study multi-prover and Zero Knowledge variants of PVGs. Additionally, Kirchner et al. [2024] successfully utilize PVGs towards obtaining human-legible outputs from LLMs. Notably, they require a relaxed completeness guarantee of their learned proof system—this requirement is the same as our Definition 2.4 of Self-Proving models.

Beyond PVGs, Wäldchen et al. [2024] cast the problem of model interpretability as a Prover–Verifier interaction between a learned feature selector and a learned feature classifier. Debate systems [Condon et al., 1995], a multiprover variant of IPs, were considered for aligning models with human values [Irving et al., 2018, Brown-Cohen et al., 2024]. In such Debate systems, two competing models are each given an alleged answer $y \neq y'$, and attempt to prove the correctness of their answer to a (human or learned) judge. Lastly, Murty et al. [2023] define Pseudointelligence: a model learner L_M and an evaluator learner L_E are each given samples from a ground-truth; L_M learns a model of the ground-truth, while L_E learns an evaluator of such models; the learned evaluator then attempts to distinguish between the learned model and the ground-truth in a Turing Test-like interaction.

All of these works consider *learned verifiers*, whereas our work focuses on training models that interact with a manually-defined verifier. More related in this regard is IP-PAC [Goldwasser et al., 2021], in which a learner proves that she learned a model that is Probably Approximately Correct [Valiant, 1984]. We, however, consider *models* that prove their own correctness on a *per-input basis*, rather than *learners* that prove *average-case correctness* of a model.

Models that generate formal proofs. Self-Proving models are verified by an algorithm with formal completeness and soundness guarantees (see Definition 2.2). In this sense, Self-Proving models generate a formal proof of the correctness of their output. Several works propose specialized models that generate formal proofs.

AlphaGeometry [Trinh et al., 2024] is capable of formally proving olympiad-level geometry problems; Others have trained models to produce proofs in Gransden et al. [2015], Polu and Sutskever [2020] and others train models to produce proofs in Coq [Gransden et al., 2015], Metamath [Polu and Sutskever, 2020], Lean [Yang et al., 2023], or manually-defined deduction rules [Tafjord et al., 2021]; FunSearch [Romera-Paredes et al., 2024] evolves LLM-generated programs by systematically evaluating their correctness. Indeed, all of these can be cast as Self-Proving models developed for *specific proof systems*. Meanwhile, this work defines and studies the class of such models *in general*. Several works (e.g. Welleck et al. 2022) consider models that generate natural language proofs or explanations, which are fundamentally different from formal proofs (or provers) verified by an algorithm.

Training on intermediate steps. Chain-of-Though (CoT, Wei et al. 2022) refers to additional supervision on a model in the form of intermediate reasoning steps. CoT is known to improve model performance whether included in-context [Wei et al., 2022] or in the training phase itself [Yang et al., 2022]. Transcript Learning (TL, Section 3.1) can be viewed as training the model on a Chain-of-Thought induced by the interaction of a verifier and an honest prover (Definition 2.2).

To complete the analogy, let us adopt the terminology of Uesato et al. [2022], who consider *outcome supervision* and *process supervision*. In our case, the *outcome* is the decision of the verifier, and the *process* is the interaction between the verifier and the model. Thus, Reinforcement Learning from Verifier Feedback (RLVF, Section 3.2) is outcome-supervised while TL is process-supervised. In a recent work, Lightman et al. [2024] find that process-supervised transformers outperform outcome-supervised ones on the MATH dataset [Hendrycks et al., 2021].

Subsequent work on RLVF. Following the preprint version of this paper on May 2024, Reinforcement Learning from Verifier Feedback (RLVF, Section 3.2) has been implemented and widely adopted. Of particular note is RLVR (Reinforcement Learning from Verifiable Reward, Lambert et al. 2024), which implements RLVF by adding KL regularization to the RL objective of Algorithm 2. This implementation provided valuable empirical validation and state-of-the-art performance on full-scale LLMs, sparked additional empirical analysis [Wu et al., 2025a, Zhou et al., 2025], and led to widespread adoption across numerous domains including medical reasoning, computer vision, and robotics [Zhang et al., 2025, Wu et al., 2025b, Song et al., 2025]. Most recently, Pyatkin et al. [2025] presented IFBench which measures performance in interactive (3-round) proof systems. These empirical contributions complement our theoretical foundations, demonstrating the surprising power and applicability of RLVF as a practical post-training method.

B Limitations

First, in our current learning methods, each individual ground-truth capability requires training a separate Self-Proving model. A natural generalization of this approach is to adapt our definition and methods to deal with a single *generalist* Self-Proving model that proves its correctness to multiple verifiers of different ground-truths.

Moreover, the second strategy presented in Section 3.1.1—namely, generating accepting transcripts first and then constructing matching inputs—has an inherent limitation: the resulting training distribution is biased by the design of the generator. As a consequence, models may end up learning patterns of the construction process, rather than acquiring generalizable reasoning capabilities. This highlights the importance of using sufficiently diverse generators, and of evaluating model performance on out-of-distribution inputs.

C A Definition for General Loss Functions and One-To-Many Relations

We present variants of Self-Proving models (Definition 2.4) generalized to one-to-many relations, and general bounded loss functions. While these generalizations provide a richer framework that may accommodate a wider range of applications, the theorems in this paper are based on the forgoing Definition 2.4, which captures the essential properties while remaining mathematically manageable.

General (bounded) loss functions. In Definition 2.1 we implicitly use the 0-1 loss when measuring the correctness of a model: For any $x \in X$, we measure only whether the model generated the correct output $y = F^*(x)$, but not how "far" the generated y was from $F^*(x)$. It is often the case in machine learning that we would be satisfied with models that generate a "nearly-correct" output. This is formalized by specifying a loss function $\ell\colon \Sigma^* \times \Sigma^* \to [0,1]$ and measuring the probability that $\ell(x,y)$ is smaller than some threshold $\lambda \in [0,1)$, where x is drawn from the input distribution μ, and y is generated by the model when given input x.

In the context of language modeling, different loss function allow for a more fine-grained treatment of the *semantics* of a given task. As an example, consider the *prime-counting task*:

- Given an integer $x < 10^9$, output the number of primes less than or equal to x.

In the notation of Section 2, the prime-counting task would be captured by the ground-truth function

$$F^*(x) \coloneqq \left|\{p \in \mathbb{N} \mid p \leq x,\ p \text{ is prime}\}\right|.^6$$

Per Definition 2.1, any output other than $F^*(x)$ is "just as incorrect" as any other. Yet, we might prefer outputs that are closer to the correct answer, say, in L_1 norm. This preference can be captured

[6] Formally, the input and output are strings in Σ^* representing integers (e.g. in decimal representation).

by the following bounded loss function

$$\ell_1(x, y) \coloneqq \begin{cases} |y - F^*(x)| \cdot 10^{-9} & \text{if } y \leq 10^9 \\ 1 & \text{else.} \end{cases}$$

In particular, if we are interested in knowing the answer only up to some additive constant C, we could say that an output y is "correct-enough" if $\ell_1(x, y) \leq C \cdot 10^{-9}$.

More generally, we relax Definition 2.1 to capture approximate correctness as follows.

Definition C.1 (Approximate correctness)**.** *Let μ be a distribution over input sequences in Σ^* and let $\ell \colon \Sigma^* \times \Sigma^* \to [0, 1]$ be a loss function. For any $\alpha, \lambda \in [0, 1]$, we say that model F_θ is (α, λ)-correct with respect to μ if*

$$\Pr_{\substack{x \sim \mu \\ y \sim F_\theta(x)}} [\ell(x, y) \leq \lambda] \geq \alpha.$$

One-to-many-relations. In Section 2, we focused on the setting of models of a ground-truth function $F^* \colon \Sigma^* \to \Sigma^*$. That is, when each input x has exactly one correct output, namely $F^*(x)$. A more general setting would be to consider a ground-truth *relation* $L \subseteq \Sigma^* \times \Sigma^*$. Then, we say that y is a correct output for x if $(x, y) \in L$. Importantly, this allows a single x to have many possible correct outputs, or none at all.

Note that we must take care to choose a loss function ℓ that captures correctness with respect to the relation L, i.e., $\ell(x, y) = 0$ if and only if $(x, y) \in L$. Equivalently, any loss function ℓ induces a relation $L \coloneqq \{(x, y) \mid \ell(x, y) = 0\}$. Therefore, our relaxation to approximate-correctness Definition C.1 already captures the setting of one-to-many relations, since an input x may have multiple y^* such that $\ell(x, y^*) = 0$.

C.1 The General Definition

We first present a relaxed definition of Interactive Proof systems for verifying approximate-correctness.

Definition C.2 (Definition 2.2, generalized)**.** *Fix a soundness error $s \in (0, 1)$, a threshold $\lambda \in [0, 1)$, a finite set of tokens Σ, and a loss function $\ell \colon \Sigma^* \times \Sigma^* \to [0, 1]$. A* verifier V for ℓ with threshold λ *is a probabilistic polynomial-time algorithm that is given explicit inputs $x, y \in \Sigma^*$ and black-box (oracle) query access to a prover P. It* interacts *with P over R rounds (see Figure 1) and outputs a decision $\langle V, P \rangle (x, y) \in \{\text{reject}, \text{accept}\}$. Verifier V satisfies the following two guarantees:*

- Completeness: *There exists an* honest prover P^* *such that, for all $x, y \in \Sigma^*$, if $\ell(x, y) = 0$ then*

  $$\Pr[\langle V, P^* \rangle (x, y) \text{ accepts}] = 1,$$

  where the probability is over the randomness of V.

- Soundness: *For all P and for all $x, y \in \Sigma^*$, if $\ell(x, y) > \lambda$ then*

  $$\Pr[\langle V, P \rangle (x, y) \text{ accepts}] \leq s,$$

  where the probability is over the randomness of V and P, and s is the soundness error.

Indeed, for a given ground-truth function $F^* \colon \Sigma^* \to \Sigma^*$, Definition 2.2 can be recovered by choosing the 0-1 loss

$$\ell_{F^*}(x, y) \coloneqq \begin{cases} 1 & \text{if } x \neq F^*(y) \\ 0 & \text{else} \end{cases}$$

and any threshold $\lambda \in [0, 1)$.

Remark C.3 (Connection to Interactive Proofs of Proximity)**.** *Definition C.2 can be seen as a slight generalization of (perfect completeness) Interactive Proofs of Proximity (IPPs, Rothblum et al. 2013). An IPP for a relation $L \subseteq \Sigma^* \times \Sigma^*$ with proximity parameter λ is obtained by instantiating Definition C.2 with the loss function ℓ_{Hamming} defined by*

$$\ell_{\text{Hamming}}(x, y) \coloneqq \min \left\{ \frac{\#\{i \mid y_i \neq y^*_i\}}{|y|} \;\middle|\; (x, y^*) \in L,\ |y^*| = |y| \right\},$$

that is, $\ell_{\mathrm{Hamming}}(x, y)$ is the fraction of tokens in y that must be changed to obtain an output y^* with $(x, y^*) \in L$. However, the motivation of Rothblum et al. [2013] was studying sublinear time verification, whereas ours is to relax the requirements of traditional Interactive Proofs towards meeting common desiderata in machine learning.

With this relaxed notion of Interactive Proofs in hand, we are now ready to define Self-Proving models for general (bounded) loss functions.

Definition C.4 (Definition 2.4, generalized)**.** *Fix a loss function $\ell\colon \Sigma^* \times \Sigma^* \to [0, 1]$, a verifier V for ℓ with threshold $\lambda \in [0, 1)$ as in Definition C.2, and a distribution μ over inputs Σ^*. The* Verifiability *of a model $P_\theta \coloneqq \Sigma^* \to \Sigma^*$ is defined as*

$$\mathrm{ver}_{V,\mu}(\theta) \coloneqq \Pr_{\substack{x\sim\mu \\ y\sim P_\theta(x)}} \left[\langle V, P_\theta\rangle\,(x, y) \text{ accepts}\right].$$

*We say that model P_θ is β-*Self-Proving *with respect to V and μ if $\mathrm{ver}_{V,\mu}(\theta) \geq \beta$.*

Analogously to Remark 2.5, we observe that Verifiability as per Definition C.4 implies approximate-correctness: Suppose P_θ is β-Self-Proving model with respect to a verifier V that has soundness error s and threshold parameter λ for loss function ℓ. Then, by a union bound,

$$\Pr_{\substack{x\sim\mu \\ y\sim P_\theta(x)}} [\ell(x, y) \leq \lambda] \geq \beta - s.$$

Importantly, as emphasized throughout this paper, soundness of V implies that for *all* inputs x, any output y such that $\ell(x, y) > \lambda$ is rejected with high probability $(1 - s)$.

D Formal proofs

Appendix D.1 formally specifies the setup in which our results reside. We then prove Lemma 3.2 in Appendix D.2, Lemma 3.3 in Appendix D.3, and Theorem 4.1 in Appendix D.4.

D.1 Specification of the Learning Model

In this section, we fully specify the theoretical framework in which our results reside. We define a *learner* as an algorithm Λ with access to a family of autoregressive models $\{P_\theta\}_\theta$ and samples from the input distribution $x \sim \mu$. In our setting of Self-Proving models (and in accordance with the Interactive Proofs literature), we give the learner the full specification of the verifier V. More formally,

Definition D.1 (Self-Proving model learner)**.** *A* (Self-Proving model) learner *is a probabilistic oracle Turing Machine Λ with the following access:*

- *A family of* autoregressive models $\{P_\theta\}_{\theta\in\mathbb{R}^d}$ *where $d \in \mathbb{N}$ is the number of parameters in the family. For each θ and $z \in \Sigma^*$, the random variable $P_\theta(z)$ is determined by the logits $\log p_\theta(z) \in \mathbb{R}^{|\Sigma|}$. For any $z \in \Sigma^*$ and $\sigma \in \Sigma$, the learner Λ can compute the gradient of the σ^{th} logit, that is, $\nabla_\theta \log \Pr_{\sigma'\sim p_\theta(z)}[\sigma = \sigma']$. In particular, $\log \Pr_{\sigma'\sim p_\theta(z)}[\sigma = \sigma']$ is always differentiable in θ.*
- *Sample access to the* input distribution *μ. That is, Λ can sample $x \sim \mu$.*
- *The full specification of the verifier V, i.e., the ability to emulate the verification algorithm V. More specifically, Λ is able to compute V's decision after any given interaction; that is, given input x, output y, and a sequence of queries and answers $(q_i, a_i)_{i=1}^R$, the learner Λ can compute the decision of V after this interaction.*

D.2 Proof of Lemma 3.2

We let $\mathcal{T}_V^\theta$ denote the transcript generator induced by the model P_θ when interacting with V: for each x, $\mathcal{T}_V^\theta(x)$ is the distribution over transcripts of interactions between V and P_θ on input x. We stress that $\pi^* \sim \mathcal{T}_V^*(x)$ and $\pi \sim \mathcal{T}_V^\theta(x)$ are transcripts produced when interacting *with the same verifier queries*; we can think of the verifier as simultaneously interacting with the honest prover

and with the model P_θ.[7] In what follows, we use $\pi^* \sim \mathcal{T}_V^*(x)$ and $\pi \sim \mathcal{T}_V^\theta(x)$ to denote two transcripts that share the same queries. That is, taking $\pi^* = (y^*, q_1^*, a_1^*, \ldots, q_R^*, a_R^*)$ to denote an accepting transcript sampled from $\mathcal{T}_V^*(x)$, and $\pi = (y, q_1^*, a_1, \ldots, q_R^*, a_R)$ to denote a random transcript sampled from $\mathcal{T}_V^\theta(x)$, we say that π and π^* *agree* if they agree on the prover answers, namely if:

$$(y, a_1, \ldots, a_R) = (y^*, a_1^*, \ldots, a_R^*).$$

This definition implicitly uses the independence of the verifier and model's randomness. We now prove that TL correctly estimates the gradient of $A(\theta)$ in its update step.

Proof of Lemma 3.2. Throughout this proof, expectations and probabilities will be over the same distributions as in the lemma statement. First, we use the law of total probability together with the autoregressive property of P_θ (Section 3) to switch from probabilities on transcripts, to products of next-token probabilities. Formally, consider a fixed input x, an honest transcript $\pi^* = (y^*, q_1^*, a_1^*, \ldots, q_R^*, a_R^*)$, and denote a random transcript sampled from $\mathcal{T}_V^\theta(x)$ when using the same verifier queries by $\pi = (y, q_1^*, a_1, \ldots, q_R^*, a_R)$. For any $r \in [R]$ denote the random variable $\mathcal{T}_V^{\theta,<r} \coloneqq \mathcal{T}_V^\theta(yq_1^* a_1 \cdots a_{r-1} q_r^*)$. Then,

$$\begin{aligned}
\Pr_\pi[\pi = \pi^*] &= \Pr_\pi[(y, a_1, \ldots, a_R) = (y^*, a_1^*, \ldots, a_R^*)] && (2)\\
&= \Pr_{y \sim P_\theta(x)}[y = y^*] \cdot \prod_{r \in [R]} \Pr_{a \sim \mathcal{T}_V^{\theta,<r}}[a = a_r^*] \\
&= \Pr_{y \sim P_\theta(x)}[y = y^*] \cdot \prod_{\substack{r \in [R] \\ s \in S(r)}} \Pr_{\sigma \sim p_\theta(\pi^*_{<s})}[\sigma = \pi_s^*] && (3)\\
&= \prod_{\substack{r \in [R] \cup \{0\} \\ s \in S(r)}} \alpha_s(\theta), && (4)
\end{aligned}$$

where, as noted above, Equation (2) uses the independence of the verifier and model's randomness, Equation (3) uses the autoregressive property of P_θ (Definition D.1), and Equation (4) is by definition of α_s and of a_0. Next, a basic calculus identity gives

$$\nabla_\theta \left(\Pr_\pi[\pi = \pi^*] \right) = \Pr_\pi[\pi = \pi^*] \cdot \nabla_\theta \log \left(\Pr_\pi[\pi = \pi^*] \right). \tag{5}$$

This implicitly assumes that $\Pr_\pi[\pi = \pi^*]$ is differentiable in θ; indeed, this follows from Definition D.1, where the logits of the model were assumed to by differentiable. Let us focus on the rightmost factor. By Equation (4),

$$\nabla_\theta \log \left(\Pr_\pi[\pi = \pi^*] \right) = \nabla_\theta \log \left(\prod_{\substack{r \in [R] \cup \{0\} \\ s \in S(r)}} \alpha_s(\theta) \right) = \sum_{\substack{r \in [R] \cup \{0\} \\ s \in S(r)}} \nabla_\theta \log \alpha_s(\theta) = \sum_{\substack{r \in [R] \cup \{0\} \\ s \in S(r)}} \vec{d}_s(\theta) \tag{6}$$

where the last equality is by definition of $\vec{d}_s(\theta)$. Combining Equation (4) and Equation (5) gives

$$\nabla_\theta \left(\Pr_\pi[\pi = \pi^*] \right) = \prod_{\substack{r \in [R] \cup \{0\} \\ s \in S(r)}} \alpha_s(\theta) \cdot \sum_{\substack{r \in [R] \cup \{0\} \\ s \in S(r)}} \vec{d}_s(\theta).$$

By the law of total probability and the linearity of the gradient,

$$\mathop{\mathbb{E}}_{x,\pi^*} \left[\nabla_\theta \left(\Pr_\pi[\pi = \pi^*] \right) \right] = \nabla_\theta \left(\mathop{\mathbb{E}}_{x,\pi^*} \left[\Pr_\pi[\pi = \pi^*] \right] \right) = \nabla_\theta \left(\Pr_{x,\pi^*,\pi}[\pi = \pi^*] \right) = \nabla_\theta A(\theta).$$

which concludes the proof. □

[7] The way it is presented in the algorithm, first the verifier is called by $\mathcal{T}_V^*$ and outputs queries $(q_1^*, \ldots q_R^*)$, and then the model is prompted with the verifier queries one a time. This maintains soundness, since a proof system is sound as long as the prover does not know the verifier's queries in advance.

D.3 Proof of Lemma 3.3

Recall the *transcript generator of* P_θ, denoted by $\mathcal{T}_V^\theta$ (see Lemma 3.2). By the definitions of Verifiability in Definition 2.4 and $V(x, y, q_1, \ldots, a_R)$ in the lemma statement,

$$\begin{aligned}
\mathrm{ver}(\theta) &\coloneqq \Pr_{\substack{x\sim\mu \\ y\sim P_\theta(x)}} \left[\langle V, P_\theta\rangle\,(x,y) \text{ accepts}\right] \\
&= \mathop{\mathbb{E}}_{\substack{x\sim\mu \\ y\sim P_\theta(x) \\ (q_r,a_r)_{r=1}^R}} \left[\mathrm{Acc}_V(x, y, q_1, \ldots, a_R)\right] \\
&= \mathop{\mathbb{E}}_{x\sim\mu} \left[\Pr_{\pi\sim\mathcal{T}_V^\theta(x)} \left[\mathrm{Acc}_V(x,\pi)\,\right]\right] \qquad (7)
\end{aligned}$$

Now, for every input x, let $\Pi^*(x) \subset \Sigma^*$ denote the set of accepting transcripts:

$$\Pi^*(x) \coloneqq \left\{\pi^* \in \Sigma^* : \mathrm{Acc}_V(x, \pi^*) = 1\right\}.$$

We can assume that $\Pi^*(x)$ has finite cardinality, since V's running time is bounded and hence the number of different transcripts that it can read (and accept) is finite. For any fixed input x, we can express its acceptance probability by the finite sum:

$$\Pr_{\pi\sim\mathcal{T}_V^\theta(x)} [\mathrm{Acc}_V(x,\pi)] = \sum_{\pi^*\in\Pi^*(x)} \Pr_{\pi\sim\mathcal{T}_V^\theta(x)} [\pi = \pi^*]. \qquad (8)$$

We will use Equations (2) through (6) from the proof of Lemma 3.2. Up to a change in index notation, these show that, for any π^*,

$$\nabla_\theta \Pr_{\pi\sim\mathcal{T}^\theta(x)} [\pi = \pi^*] = \Pr_{\pi\sim\mathcal{T}^\theta(x)} [\pi = \pi^*] \cdot \sum_{\substack{r\in R\cup\{0\} \\ s\in[L_a]}} \nabla_\theta \vec{d}_s(\theta).$$

Combining Equations (7) and (8), by linearity of expectation we have that

$$\begin{aligned}
\nabla_\theta \mathrm{ver}(\theta) &= \mathop{\mathbb{E}}_{x\sim\mu} \left[\sum_{\pi^*\in\Pi^*(x)} \nabla_\theta \Pr_{\pi\sim\mathcal{T}^\theta(x)} [\pi = \pi^*]\right] \\
&= \mathop{\mathbb{E}}_{x\sim\mu} \left[\sum_{\pi^*\in\Pi^*(x)} \Pr_{\pi\sim\mathcal{T}^\theta(x)} [\pi = \pi^*] \cdot \sum_{\substack{r\in R\cup\{0\} \\ s\in[L_a]}} \nabla_\theta \vec{d}_s(\theta)\right] \\
&= \mathop{\mathbb{E}}_{x\sim\mu} \left[\mathop{\mathbb{E}}_{\pi\sim\mathcal{T}^\theta(x)} \left[\mathrm{Acc}_V(x,\pi) \cdot \sum_{\substack{r\in R\cup\{0\} \\ s\in[L_a]}} \nabla_\theta \vec{d}_s(\theta)\right]\right] \\
&= \mathop{\mathbb{E}}_{\substack{x\sim\mu \\ \pi\sim\mathcal{T}^\theta(x)}} \left[\mathrm{Acc}_V(x,\pi) \cdot \sum_{\substack{r\in R\cup\{0\} \\ s\in[L_a]}} \nabla_\theta \vec{d}_s(\theta)\right] \\
&= \mathop{\mathbb{E}}_{\substack{x\sim\mu \\ y\sim P_\theta(x) \\ (q_r,a_r)_{r=1}^R}} \left[\mathrm{Acc}_V(x, y, q_1, \ldots, a_R) \cdot \sum_{\substack{r\in R\cup\{0\} \\ s\in[L_a]}} \nabla_\theta \vec{d}_s(\theta)\right],
\end{aligned}$$

where in the last equality, the probability is over (q_r, a_r) sampled as in Algorithm 2, and it follows from the definition of the transcript generator $\mathcal{T}^\theta(x)$. □

D.4 Proof of Theorem 4.1

We first restate Theorem 4.1 in full formality.

Theorem D.2. *Fix a verifier V, an input distribution μ, an autoregressive model family $\{P_\theta\}_{\theta\in\mathbb{R}^d}$, and a norm $||\cdot||$ on $\mathbb{R}^d$. Fix an honest transcript generator $\mathcal{T}_V^*$, and assume that the* agreement function $A(\theta) \coloneqq \Pr[\pi = \pi^*]$ *is concave in θ, where the verifier queries are the same in π^* and π, and the probability is over $x \sim \mu$, $\pi_\theta \sim \mathcal{T}_V^\theta(x)$, and $\pi^* \sim \mathcal{T}_V^*(x)$. For any $\varepsilon > 0$, let B_{Norm}, B_{Lip} and C be upper-bounds such that the following conditions hold:*

- *There exists $\theta^* \in \mathbb{R}^d$ with $||\theta^*|| < B_{\mathrm{Norm}}$ such that $A(\theta^*) \geq 1 - \varepsilon/2$.*
- *For all θ, the logits of P_θ are B_{Lip}-Lipschitz in θ. That is, $\sup_{\substack{\theta\in\mathbb{R}^d \\ z\in\Sigma^*}} ||\nabla_\theta \log p_\theta(z)|| \leq B_{\mathrm{Lip}}$.*
- *In the proof system defined by V, the total number of tokens (over all rounds) is at most C.*

Denote by $\bar{\theta}$ the output of TL running for $N \geq \left(4 \cdot C^2 \cdot B_{\mathrm{Norm}}^2 \cdot B_{\mathrm{Lip}}^2\right)/\varepsilon^2$ iterations and learning rate $\lambda = B_{\mathrm{Norm}}/CB_{\mathrm{Lip}}\sqrt{N}$. Then the expected Verifiability (over the randomness of the samples collected by TL) of $\bar{\theta}$ is at least $1 - \varepsilon$. That is, $\mathbb{E}_{\bar{\theta}}[\mathrm{ver}_{V,\mu}(\bar{\theta})] \geq 1 - \varepsilon$.

The proof of Theorem D.2 goes by reduction to Stochastic Gradient Descent (SGD). Lemma 3.2 showed that the learner can use its only available tools—sampling honest transcripts, emulating the verifier, and differentiating the logits—to optimize the agreement $A(\theta)$. Since $A(\theta)$ lower bounds the Verifiability of P_θ, the former can be used as a surrogate for the latter.

For convenience of the reader, we first provide a description of Stochastic Gradient Ascent (equiv. to SGD) and quote a theorem on its convergence. We adapt the presentation in Shalev-Shwartz and Ben-David [2014], noting that they present Stochastic Gradient Descent in its more general form for non-differentiable unbounded functions. The familiar reader may skip directly to the proof in Appendix D.4.3.

D.4.1 Preliminaries on Stochastic Gradient Ascent

Stochastic Gradient Ascent (SGA) is a fundamental technique in concave optimization. Given a concave function $f\colon \mathbb{R}^d \to [0,1]$, SGA starts at $w_0 = \vec{0} \in \mathbb{R}^d$ and tries to maximize $f(w)$ by taking a series of "steps." Than directly differentiating f, SGA instead relies on an estimation $\nabla f(w)$: in each iteration, SGA takes a step in a direction that estimates $\nabla f(w)$.

Definition D.3 (Gradient estimator)**.** *Fix a differentiable function $f\colon \mathbb{R}^d \to \mathbb{R}$ for some d. A* gradient estimator *for f is a randomized mapping $D_f\colon \mathbb{R}^d \to \mathbb{R}^d$ whose expectation is the gradient of f. That is, for all $w \in \mathbb{R}^d$,*

$$\mathop{\mathbb{E}}_{v\sim D_f(w)}[v] = \nabla f(w).$$

Note that this is an equality between d-dimensional vectors.

Algorithm 3: Stochastic Gradient Ascent

Hyperparameters: Learning rate $\lambda > 0$ and number of iterations $N \in \mathbb{N}$.
Input: A function $f\colon \mathbb{R}^d \to \mathbb{R}$ to maximize and a gradient estimator D_f for f.
Output: A vector $\bar{w} \in \mathbb{R}^d$.

Initialize $w_0 \coloneqq \vec{0} \in \mathbb{R}^d$.
for $i = 1, \ldots, N-1$ **do**
  Sample $v_i \sim D_f(w_{i-1})$.
  Update $w_i \coloneqq w_{i-1} + \lambda \cdot v_i$.
Output $\bar{w} \coloneqq \frac{1}{N}\sum_{i\in[N]} w_i$.

Theorem 14.8 in Shalev-Shwartz and Ben-David [2014] implies the following fact.

Fact D.4. *Fix a concave $f\colon \mathbb{R}^d \to [0,1]$, a norm $||\cdot||$ on $\mathbb{R}^d$, and upper-bounds $B_{\mathrm{Norm}}, B_{\mathrm{Lip}} > 0$. Let*

$$w^* \in \mathop{\mathrm{argmax}}_{w:||w||<B_{\mathrm{Norm}}} f(w),$$

and let $\bar{w}$ denote the output of Algorithm 3 run for N iterations with learning rate

$$\lambda = \frac{B_{\mathrm{Norm}}}{B_{\mathrm{Lip}}\sqrt{N}}.$$

If at every iteration it holds that $||v_i|| < B_{\mathrm{Lip}}$, then

$$\mathop{\mathbb{E}}_{\bar{w}}[f(\bar{w})] \geq f(w^*) - \frac{B_{\mathrm{Norm}} \cdot B_{\mathrm{Lip}}}{\sqrt{N}}.$$

D.4.2 Learning with Stochastic Gradient Ascent/Descent

Fact D.4 captures the general case of using SGA for maximization of concave problems. It is more common for the literature to discuss the equivalent setting of Stochastic Gradient Descent (SGD) for minimization of convex problems. Specifically, a common application of SGD is for the task of *Risk Minimization*: given a loss function and access to an unknown distribution of inputs, the goal is to minimize the expected loss with respect to the distribution. Assuming that the loss function is differentiable, the gradient of the loss serves as a gradient estimator (see Definition D.3) for the risk function. We refer the reader to Shalev-Shwartz and Ben-David [2014, Section 14.5.1] for a complete overview of SGD for risk minimization.

For the sake of completeness, we formulate Transcript Learning (TL, Algorithm 1) in the framework of Risk Minimization for Supervised Learning. This is not strictly needed for the proof of Theorem D.2, but is an illuminating connection. Although multiple loss functions may achieve our ultimate goal—learning Self-Proving models—in what follows we define the loss that corresponds to TL.

Fix a verifier V and let $\mathcal{T}^*_V$ denote a distribution over accepting transcripts. We define

$$\texttt{loss}\left(\theta, (x, \pi^*)\right) \coloneqq \Pr_{\pi \sim \mathcal{T}^\theta_V(x)}\left[\pi \neq \pi^*\right], \tag{9}$$

where π^* and π share the same verifier messages (as in Lemma 3.2) so the inequality is only over the prover's messages, namely $\Pr_{\pi\sim\mathcal{T}^\theta_V(x)}[\pi \neq \pi^*] = \Pr_{\pi\sim\mathcal{T}^\theta_V(x)}[(y, a_1, \ldots, a_R) \neq (y^*, a^*_1, \ldots, a^*_R)]$.[8]

The risk function is the expected value of the loss over the joint distribution of inputs and accepting transcripts $\mu \times \mathcal{T}^*_V(\mu)$:

$$\texttt{Risk}\left(\theta\right) \coloneqq \mathop{\mathbb{E}}_{\substack{x\sim\mu \\ \pi^*\sim\mathcal{T}^*_V}}\left[\texttt{loss}\left(\theta, (x, \pi^*)\right)\right],$$

which means that the *agreement function* defined in Theorem D.2:

$$A(\theta) = \Pr_{\substack{x\sim\mu \\ \pi^*\sim\mathcal{T}^*_V(x) \\ \pi\sim\mathcal{T}^\theta_V(x)}}\left[\pi = \pi^*\right],$$

satisfies $A(\theta) = 1 - \texttt{Risk}(\theta)$.

Thus, maximizing the agreement is equivalent to minimizing the risk. The hypothesis class over which the optimization is performed is the ball of radius B_{Norm}, i.e., $\left\{\theta \in \mathbb{R}^d : ||\theta|| < B_{\mathrm{Norm}}\right\}$. The assumption that A is concave in θ implies that the loss function is convex in θ, which is the required assumption for using SGD for risk minimization.

D.4.3 Proof of Theorem D.2

Our strategy is to cast TL as Stochastic Gradient Ascent and apply Fact D.4. Let ε, B_{Norm}, B_{Lip} and C as in the theorem statement be given. Let θ^* be such that $A(\theta^*) \geq 1 - \varepsilon/2$ and $||\theta^*|| \leq B_{\mathrm{Norm}}$.

First, notice that

$$\mathop{\mathbb{E}}_{\bar{\theta}}\left[\mathrm{ver}_{V,\mu}(\bar{\theta})\right] \geq \mathop{\mathbb{E}}_{\bar{\theta}}[A(\bar{\theta})].$$

[8] This loss is not to be confused with those discussed in Appendix C. Here, we are simply explaining how TL can be viewed as a supervised risk minimizer for the loss function defined in Equation (9).

This holds because, for any x and model P_θ, whenever the transcript generated by $\mathcal{T}^\theta(x)$ agrees with π^*, then the verifier accepts (because π^* is honest). Therefore, to prove the theorem it suffices to show that

$$\mathop{\mathbb{E}}_{\bar{\theta}}[A(\bar{\theta})] \geq 1 - \varepsilon.$$

Following the notation in Algorithm 1, in every iteration $i \in [N]$ the norm of the update step is

$$\left\| \prod_{\substack{r\in[R]\cup\{0\}\\ s\in S(r)}} \alpha_s(\theta_i) \cdot \sum_{\substack{r\in[R]\cup\{0\}\\ s\in S(r)}} \vec{d}_s(\theta_i) \right\| = \left| \prod_{\substack{r\in[R]\cup\{0\}\\ s\in S(r)}} \alpha_s(\theta_i) \right| \cdot \left\| \sum_{\substack{r\in[R]\cup\{0\}\\ s\in S(r)}} \vec{d}_s(\theta_i) \right\| \leq 1 \cdot \sum_{\substack{r\in[R]\cup\{0\}\\ s\in S(r)}} \left\| \vec{d}_s(\theta_i) \right\|,$$

where the inequality is because $\alpha_s(\theta_i)$ are probabilities, so ≤ 1. Moreover, we have

$$\sum_{\substack{r\in[R]\cup\{0\}\\ s\in S(r)}} \left\| \vec{d}_s(\theta_i) \right\| \leq \sum_{\substack{r\in[R]\cup\{0\}\\ s\in S(r)}} B_{\mathrm{Lip}} \leq C \cdot B_{\mathrm{Lip}}.$$

The first inequality is by definition of B_{Lip} as an upper-bound on the gradient of P_θ's logits. The second is because, by definition, C is an upper-bound on the number of tokens sent by the prover in the proof system, which is exactly the number of terms in the sum: r indexes rounds, and s indexes tokens sent in each round.

To conclude, Lemma 3.2 shows that TL samples from a gradient estimator for $A(\theta)$, while the above equation shows that the gradient is upper-bounded by $C \cdot B_{\mathrm{Lip}}$. We can therefore apply Fact D.4 to obtain

$$\mathop{\mathbb{E}}_{\bar{\theta}} \left[A\left(\bar{\theta}\right) \right] \geq A(\theta^*) - \varepsilon/2 \geq (1 - \varepsilon/2) - \varepsilon/2 = 1 - \varepsilon,$$

where the inequality is by definition of θ^*. This completes the proof of Theorem D.2.

E Learning from annotations

To minimize the length of messages exchanged in an Interactive Proof system, the honest prover is designed to send the shortest possible message to the verifier, containing only essential information.

However, when training Self-Proving model, it may be useful for it to first generate an "annotated" answer $\widetilde{a}$ which is then trimmed down to the actual answer a to be sent to the verifier. We adapt Sections 2 and 3 to this setting via *Annotated Transcripts*. The TL and RLVF algorithms naturally extend to annotated transcripts as well. Table 2 shows that annotations significantly improve performance of TL in practice.

Annotations can be viewed as adding Chain-of-Thought [Wei et al., 2022]. As a concrete example, consider our experiments on computing the GCD. As detailed in Appendix F.2, a proof π in this setting is the output of an iterative process—the extended Euclidean algorithm—starting from the input x: $x \mapsto \pi_1 \mapsto \pi_2 \mapsto \cdots \mapsto \pi$. The annotation of the proof π consists the first T steps $(\pi_1, \ldots, \pi_T)$ up to some fixed cutoff T. These are prepended to the proof and shown to the model during TL training. At inference time, the model is evaluated only on whether it generated the proof π correctly.

We formally capture annotations by introducing a *transcript annotator* and an *answer extractor* incorporated into the training and inference stages, respectively. Fix a verifier V in an R-round proof system with question length L_q and answer length L_a. An *annotation system* with annotation length $\widetilde{L_a}$ consists of a *transcript annotator* A, and an *answer extractor* E.

In terms of efficiency, think of the annotator as an algorithm of the same computational resources as an honest prover in the system (see Definition 2.2), and the answer extractor as an extremely simple algorithm (e.g., trim a fixed amount of tokens from the annotation).

To use an annotation system the following changes need to be made:

Table 2: **Self-Proving transformers computing the GCD.** We train a 6.3M parameter GPT to compute the GCD of two integers sampled log-uniformly from $[10^4]$. Vanilla GPT correctly generates the GCD for almost all inputs, but does not prove correctness to a simple verification algorithm. GPT trained with Transcript Learning (GPT+TL) proves its answer 60.3% of the time; adding Reinforcement Learning from Verifier Feedback (+RLVF) increases this to 78.3%; training with Annotated Transcript Learning (GPT+ATL) gives the highest Verifiability score of 96%.

LEARNING METHOD	CORRECTNESS	VERIFIABILITY
GPT (BASELINE)	99.8%	-
GPT+TL	98.8%	60.3%
GPT+TL+RLVF	98.9%	78.3%
GPT+ATL	98.6%	96.0%

- At training time, an input x and transcript π is annotated to obtain $\widetilde{\pi} := A(x, \pi)$, e.g. before the forwards backwards pass in TL (line 3 in Algorithm 1).
- At inference time (i.e., during interaction between V and P_θ), the prover keeps track of the annotated transcript, but in each round passes the model-generated (annotated) answer through the extractor E before it is sent to the verifier. That is, in each round $r \in [R]$, the prover samples
$$\widetilde{a_r} \sim P_\theta(x, y, q_1, \widetilde{a_1}, \ldots, \widetilde{a_{r-1}}, q_r).$$
The prover then extracts an answer $a_r := E(\widetilde{a_r})$ which is sent to the verifier.

F Experiments

We describe our experimental setup, and present ablation studies that shed additional light on the effect of *annotation* and *representation* on Verifiability.

F.1 Setup: Training transformers to predict the GCD

Charton [2024] empirically studies the power and limitations of learning GCDs with transformers. We follow their setup and two conclusions on settings that make for faster learning: Training from the log-uniform distribution, and choosing a base of representation with many prime factors.

We fix a base of representation $B = 210$ and use $\mathbf{x}$ to denote an integer x encoded as a B-ary string.[9] For sequences of integers, we write $(\mathbf{x_1 x_2})$ to denote the concatenation of $\mathbf{x_1}$ with $\mathbf{x_2}$, delimited by a special token. The vocabulary size needed for this representation is $|\Sigma| \approx 210$.

We choose the input distribution μ to be the log-uniform distribution on $[10^4]$, and train the transformer on sequences of the form $(\mathbf{x_1 x_2 y})$, where $x_1, x_2 \sim \mu$ and $y = GCD(x_1, x_2)$. This is a scaling-down of Charton [2024], to allow single GPU training of Self-Proving transformers. In all of our experiments, we use a GPT model [Vaswani et al., 2017] with 6.3M parameters trained on a dataset of 1024K samples in batches of 1024. Full details are deferred to Appendix G.2.

Proving correctness of GCD. Following Charton [2024] as a baseline, we find that transformers can correctly compute the GCD with over 99% probability over $(x_1, x_2) \sim \mu$. To what extent can they *prove* their answer? To answer this question, we first devise a natural proof system based on Bézout's theorem. Its specification and formal guarantees are deferred to Appendix G.1. We denote its verification algorithm by V, and highlight some important features of the experimental setup:

- The proof system consists of one round ($R = 1$). The verifier makes no query, and simply receives a proof π from the prover.
- *Completeness:* For any $x_1, x_2, y \in [10^4]$ such that $y = GCD(x_1, x_2)$, there exists a proof π such that $V(\mathbf{x_1 x_2 y}\pi)$ accepts. As detailed in Appendix G.1, the proof π consists of a pair of integers who are *Bézout coefficients* for x_1, x_2.

[9] $B = 210$ is chosen following Charton [2024] to be an integer with many prime factors.

- *Soundness:* If $y \neq GCD(x_1, x_2)$, then $V(\mathbf{x_1 x_2 y}\pi)$ rejects[10] for any alleged proof $\pi \in \Sigma^*$.

To measure Verifiability, we train a Self-Proving transformer using Transcript Learning on sequences $(\mathbf{x_1 x_2 y}\pi)$ and estimate for how many inputs $x_1, x_2 \sim \mu$ does the model generate *both* the correct GCD $\mathbf{y}$ and a valid proof π. We test on 1000 pairs of integers $x_1', x_2' \sim \mu$ held-out of the training set, prompting the model with $(\mathbf{x_1' x_2'})$ to obtain $(\mathbf{y'}\pi')$, and testing whether $V(\mathbf{x_1' x_2' y'}\pi')$ accepts.

Table 2 shows our main experimental result, which has the following key takeaways:

1. Transcript Learning (TL) for 100K iterations (≈100M samples) results in a Self-Proving transformer that correctly proves 60.3% of its answers.
2. A base Self-Proving Model with fairly low Verifiability of 40% can be improved to 79.3% via Reinforcement Learning from Verifier Feedback (RLVF). Although it does not rely on honest transcripts, RLVF trains slowly: this nearly-twofold improvement took four million iterations.
3. Most efficient is Annotated Transcript Learning, which yielded a model with 96% Verifiability in 100K iterations. We further investigate their effect next.

F.2 Models generalize beyond annotations

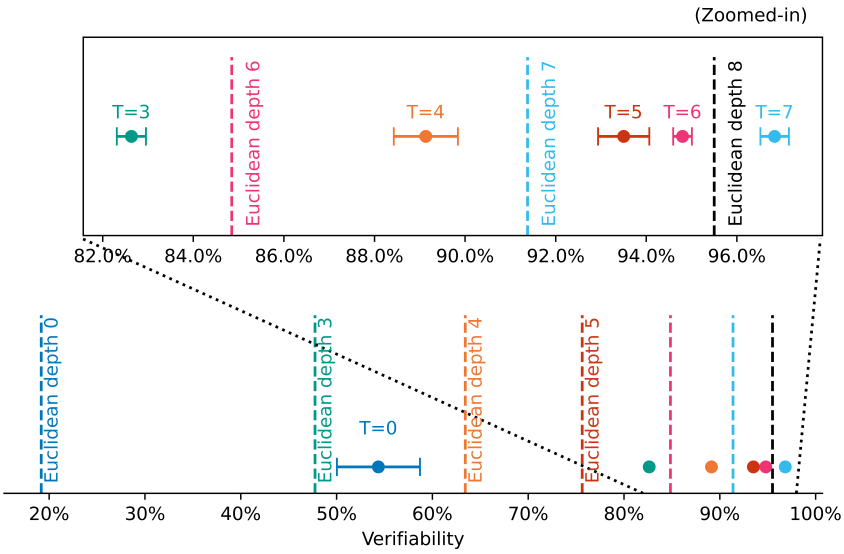

Figure 3: **Verifiability with increasing amounts of annotation**. T is the number of steps added in Annotated Transcript Learning. Dashed lines indicate *Euclidean depth*, that bound the Verifiability of models that prove *only* for integers up to a certain number of steps. Each T was run with three seeds, with mean ± standard error depicted. The upper graph provides a zoomed-in view of the 82% to 98% range from the lower graph, which spans a broader scale from 20% to 100%.

The proof π is annotated by including intermediate steps in its computation. Details are deferred to Appendix G.1; roughly speaking, we observe that the proof π for input $(\mathbf{a}, \mathbf{b})$ is obtained as the last element in a sequence $\mathbf{a}, \mathbf{b}, \pi_\mathbf{1}, \pi_\mathbf{2}, \ldots$ computed by the Euclidean algorithm. We annotate the proof π by prepending to it the sequence of *Euclidean steps* $(\pi_\mathbf{1}, \ldots, \pi_\mathbf{T})$ up to some fixed cutoff T.

Figure 3 shows how T affects the Verifiability of the learned model. As suggested by Lee et al. [2024], training the model on more intermediate steps results in better performance; in our case, increasing the number of intermediate steps T yields better Self-Proving models. One might suspect

[10] With probability 1, i.e., $s = 0$ in Definition 2.2.

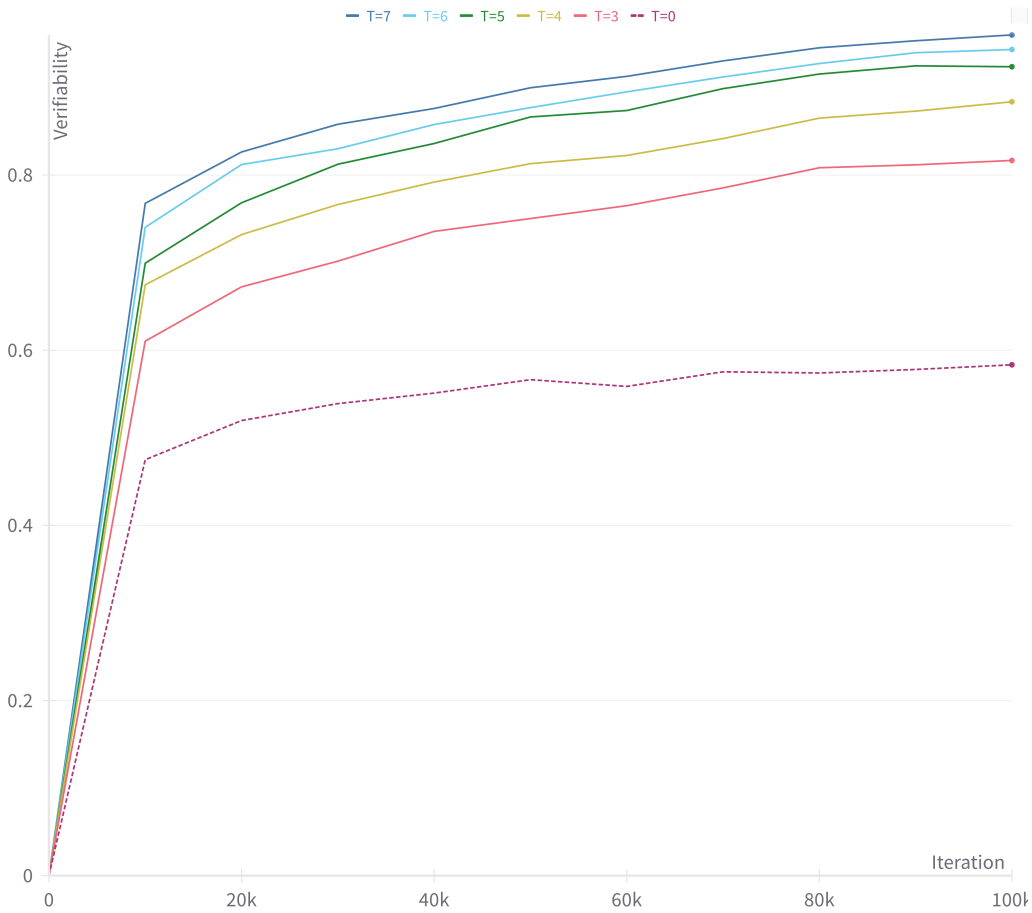

Figure 4: **Annotated TL Verifiability as a function of the number of samples** N. Each iteration (X axis) is a batch of 1024 samples from a dataset of ≈10M sequences. Every 10k iterations, Verifiability was evaluated on a held-out dataset of 1k inputs. T is the number of steps in Annotated Transcript Learning (Figure 3), and $T = 0$ is non-annotated Transcript Learning. Each T was run with three seeds, with mean depicted by the curve and standard error by the shaded area.

that models only learn to execute the Euclidean algorithm in-context. To rule out this hypothesis, we derive an upper bound on the possible efficacy of such limited models. This bound is based on the *Euclidean depth* of integers (x_1, x_2), which we define as the number of intermediate steps that the Euclidean algorithm makes before terminating on input (x_1, x_2). Indeed, a model that only learns to compute (in-context) the arithmetic of the Euclidean algorithm would only be able to prove the correctness of inputs (x_1, x_2) whose depth does not exceed the annotation cutoff T.

Figure 3 tells a different story: For each cutoff T, we estimate the probability that integers $x_1, x_2 \sim \mu$ have Euclidean depth at most T on 10^5 sampled pairs. Larger annotation cutoff T increases Verifiability, but all models exceed their corresponding Euclidean depth bound.

F.3 Base of representation

As mentioned previously, Charton [2024] concludes that, for a given base of representation B, transformers correctly compute the GCD of integers x_1, x_2 that are products of primes dividing B. Simply put, choosing a base B with many different prime factors yields models with better correctness (accuracy), which suggests why base $B = 210 = 2 \cdot 3 \cdot 5 \cdot 7$ yielded the best results. To test if B's factorization has a similar effect on Verifiability, we train transformers on 68 bases varying the number of prime divisors from $\omega(B) = 1$ (i.e., B is a prime power) to $\omega(B) = 4$. Figure 5 shows that $\omega(B)$ correlates not just with correctness [Charton, 2024], but also with Verifiability.

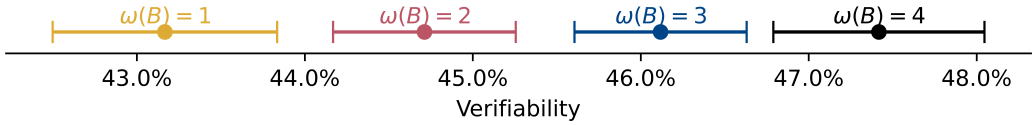

Figure 5: **The number of prime divisors of a base $\omega(B)$ determines Verifiability.** For each $o \in [4]$, we sampled 17 bases $B \in \{2, \ldots, 1386\}$ such that $\omega(B) = o$. A Self-Proving transformer was trained via Transcript Learning for twenty epochs on an identical dataset of 1024K samples encoded in base B. For each $\omega(B)$ we depict the mean $\pm$ standard error.

Although the finding is statistically significant (no overlapping error margins), the difference is by a few percentages; we attribute this to the smaller (10%) number of samples on which models were trained, relative to other experiments.

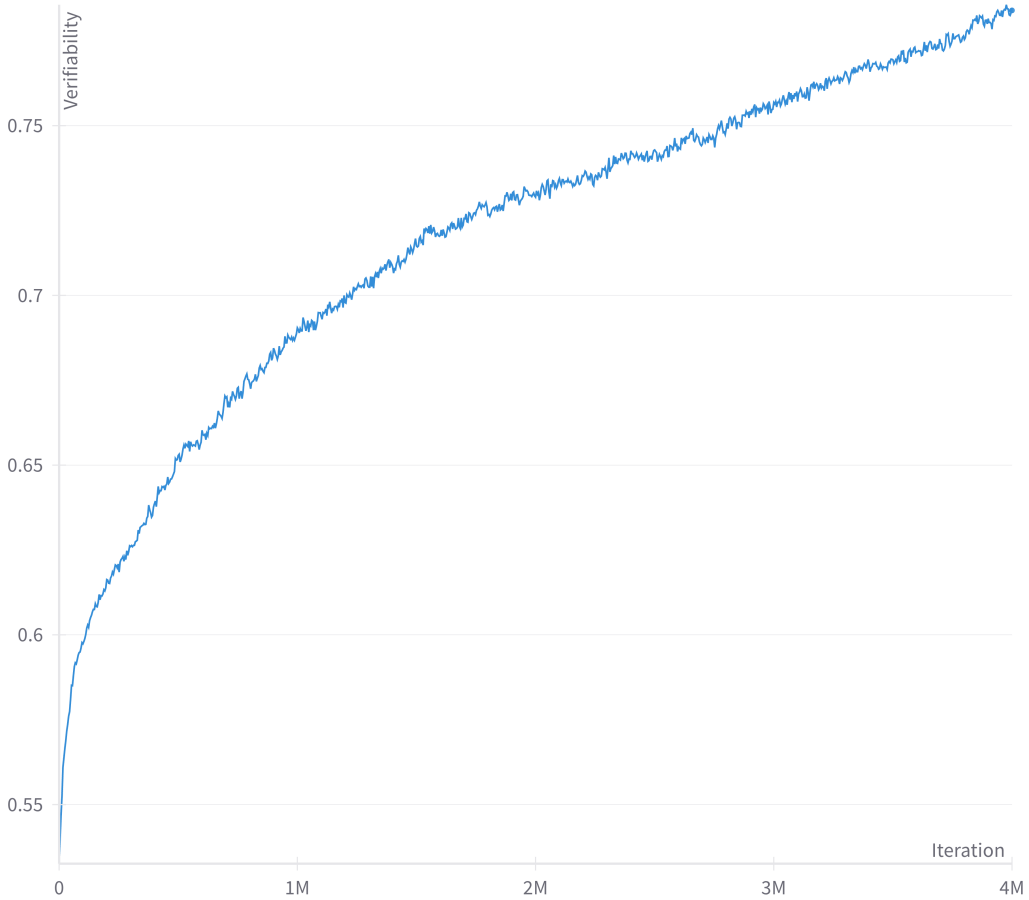

Figure 6: **RLVF Verifiability as a function of the number of samples N.** Starting from a base model with Verifiability 48% (obtained via Transcript Learning), in each iteration a batch of 2048 inputs are sampled; the model generates a proof for each; the Verifier is used to check which proofs are accepted; then, the model parameters are updated accordingly (see Algorithm 2). Verifiability was evaluated on a held-out dataset of 1k inputs.

Algorithm 4: Extended Euclidean algorithm

Input: Nonzero integers $x_0, x_1 \in \mathbb{N}$.

Output: Integers (y, z_0, z_1), such that $y = GCD(x_0, x_1)$ and (z_0, z_1) are Bézout coefficients for (x_0, x_1).

Initialize $r_0 = x_0$, $r_1 = x_1$, $s_0 = 1$, $s_1 = 0$, and $q = 0$.
while $r_1 \neq 0$ **do**
  Update $q \coloneqq \lfloor r_0/r_1 \rfloor$.
  Update $(r_0, r_1) \coloneqq (r_1, r_0 - q \times r_1)$.
  Update $(s_0, s_1) \coloneqq (s_1, s_0 - q \times s_1)$.
Output GCD $y = r_0$ and Bézout coefficients $z_0 \coloneqq s_0$ and $z_1 \coloneqq (r_0 - s_0 \cdot x_0)/x_1$.

G Full details of the experimental setup

G.1 The Bézout proof system for GCD

The Euclidean algorithm for computing the Greatest Common Divisor (GCD) of two integers is possibly the oldest algorithm still in use today [Knuth, 1969]. Its extended variant gives a simple proof system.

Before we dive in, let us clarify what we mean by *a proof system for the GCD*. Prover Paul has two integers 212 and 159; he claims that $GCD(212, 159) = 53$. An inefficient way for Verifier Veronica to check Paul's answer is by executing the Euclidean algorithm on $(212, 159)$ and confirm that the output is 53. In an efficient proof system, Veronica asks Paul for a short string π^* (describing two integers) with which she can easily compute the answer—without having to repeat Paul's work all over. On the other hand, if Paul were to claim that "$GCD(212, 159) = 51$" (it does not), then for any alleged proof π, Veronica would detect an error and reject Paul's claim.

The verifier in the proof system relies on the following fact.

Fact G.1 (Bézout's identity [Bezout, 1779])**.** *Let $x_0, x_1 \in \mathbb{N}$ and $z_0, z_1 \in \mathbb{Z}$. If $z_0 \cdot x_0 + z_1 \cdot x_1$ divides both x_0 and x_1, then $z_0 \cdot x_0 + z_1 \cdot x_1 = GCD(x_0, x_1)$.*

Any coefficients z_0, z_1 satisfying the assumption of Fact G.1 are known as *Bézout coefficients* for (x_0, x_1). Fact G.1 immediately gives our simple proof system: For input $x = (x_0, x_1)$ and alleged GCD y, the honest prover sends (alleged) Bézout coefficients (z_0, z_1). The Verifier accepts if and only if $y = z_0 \cdot x_0 + z_1 \cdot x_1$ and y divides both x_0 and x_1.

In this proof system the Verifier does not need to make any query; to fit within Definition 2.2, we can have the verifier issue a dummy query. Furthermore, by Fact G.1 it is complete and has soundness error $s = 0$. Lastly, we note that the Verifier only needs to perform two multiplications, an addition, and two modulus operations; in that sense, verification is more efficient than computing the GCD in the Euclidean algorithm as required by Remark 2.3.

Annotations. To describe how a proof $z = (z_0, z_1)$ is annotated, let us first note how it can be computed. The Bézout coefficients can be found by an extension of the Euclidean algorithm. It is described in Algorithm 4.[11]

Referring to Algorithm 4, the annotation of a proof $z = (z_0, z_1)$ will consist of intermediate steps in its computation. Suppose that in each iteration of the While-loop, the algorithm stores each of r_0, s_0 and q in an arrays $\vec{r_0}$, $\vec{s_0}$ and $\vec{q}$. The annotation $\tilde{z}$ of z is obtained by concatenating each of these arrays. In practice, to avoid the transformer block (context) size from growing too large, we fix a cutoff T and first trim each array to its first T elements.

We formalize this in the terminology of Appendix E by defining a Transcript Annotator and Answer Extractor. Note that, since our proof system consists only of one "answer" z send from the prover to the verifier, the entire transcript π is simply $z = (z_0, z_1)$. Since the verification is deterministic, this means that the proof system is of an NP type (however, note that the search problem of finding the "NP-witness" $z = (z_0, z_1)$ is in fact in P).

[11] Our description follows `https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm`.

- *Transcript Annotator A:* For a fixed cutoff T and given input $x = (x_0, x_1)$ and transcript $z = (z_0, z_1)$, A executes Algorithm 4 on input $x = (x_0, x_1)$. During the execution, A stores the first T intermediate values of r_0, s_0 and q in arrays $\vec{r_0}$, $\vec{s_0}$ and $\vec{q}$. It outputs $A(x, z) \coloneqq (\vec{r_0}, \vec{s_0}, \vec{q}, z)$.
- *Answer Extractor E:* Given an annotated transcript $\tilde{z} = (\vec{r_0}, \vec{s_0}, \vec{q}, z)$, outputs $E(\tilde{z}) \coloneqq z$.

We note that the computational complexity of A is roughly that of the honest prover, i.e., Algorithm 4 (up to additional space due to storing intermediate values). As for E, it can be implemented in logarithmic space and linear running time in $|\tilde{z}|$, i.e., the length of the description.[12]

G.2 Implementation details

Code, data and models are available at `https://github.com/orrp/self-proving-models`.

Model architecture. We use Karpathy's *nanoGPT*[13] implementation of GPT. Note that we train the model "from scratch" only on sequences related to the GCD problem, rather than starting from a pretrained checkpoint. We use a 6.3M parameter architecture of 8 layers, 8 attention heads, and 256 embedding dimensions. We optimized hyperparameters via a random hyperparameter search, arriving at learning rate 0.0007, AdamW $\beta_1 = 0.733$ and $\beta_2 = 0.95$, 10% learning rate decay factor, no dropout, gradient clipping at 2.0, no warmup iterations, and 10% weight decay.

Data. We sample integers from the $\log_{10}$-uniform distribution over $\{1, \ldots, 10^4\}$. Models in Table 2 and Fig. 3 are trained for 100K iterations on a dataset of ≈10M samples. For Figure 5 (base ablation) we train for 20K iterations on a dataset of ≈1M samples; this is because this setting required 68 many runs in total, whereas the annotation-cutoff ablation required 18 longer runs.

Compute. All experiments were run on a machine with an NVIDIA A10G GPU, 64GB of RAM, and 32 CPU cores. The longest experiment was the single RLVF run, which took one month and four days. The annotation-cutoff ablation runs took about 75 minutes each. Base of representation ablation runs were shorter at about 15 minutes each. The total running time of the Transcript Learning experiments was approximately 40 hours (excluding time dedicated to a random hyperparameter search), and the RLVF experiment took another month and four days. The overall disk space needed for our models and data is 4GB.

Representing integers. We fully describe how integer sequences are encoded. As a running example, we will use base 210. To encode a sequence of integers, each integer is encoded in base 210, a sign is prepended and a delimiter is appended, with a unique delimiter identifying each component of the sequence. For example, consider the input integers $x_0 = 212$ (which is 12 in base 210) and $x_1 = 159$. Their GCD is $y = 53$, with Bézout coefficients $z_0 = 1$ and $z_1 = -1$. Therefore, the sequence $(212, 159, 53, 1, -1)$ is encoded as

```
+,1,2,x0,+,159,x1,+,53,y,+,1,z0,-,1,z1
```

where commas are added to distinguish between different tokens. Null tokens are appended to pad all sequences in a dataset to the same length. Both the input and the padding components are ignored when computing the loss and updating parameters.

Annotations Annotations are encoded as above, with each component in an intermediate step π_t delimited by a unique token. Since different integer pairs may require a different number of intermediate steps to compute the Bézout coefficients, we chose to pad all annotations to the same length T by the last step π_T in the sequence (which consists of the final Bézout coefficients). This ensures that the final component output by the model in each sequence should be the Bézout coefficient, and allows us to batch model testing (generation and evaluation) resulting in a 1000x speed-up over sequential testing.

[12] That is, if integers are represented by n-bits, then E has space complexity $O(\log n + \log T)$ and running time $O(n \cdot T)$.

[13] `https://github.com/karpathy/nanoGPT`.

As an example, consider the inputs $x_0 = 46$ and $x_1 = 39$. Tracing through the execution of Algorithm 4, we have

x_0	x_1	y	$\vec{s_0}$	$\vec{r_0}$	$\vec{q}$	z_0	z_1
46	39		1	46	1		
			0	39	5		
			1	7	1		
			-5	4	1		
			6	3	3		
		1				-11	13

To encode this as an annotated transcript for the transformer, we must specify a base of representation and an annotation cutoff. Suppose that we wish to encode this instance in base $B = 10$ and cutoff $T = 3$. Then the input with the annotated transcript is encoded as

```
+,4,6,x0,+,3,9,x1,+,1,y,
+,1,z0',+,4,6,z1',+,1,q',
+,0,z0'',+,3,9,z1'',+,5,q'',
+,1,z0''',+,7,z1''',+,1,q''',
-,1,1,z0,+,1,3,z1
```

where commas are used to separate between tokens, and linebreaks are added only for clarity. Notice the three types of tokens: signs, digits, and delimiters. Notice also that the output y is added immediately after the input, followed by the annotated transcript (whose six tokens comprise the proof itself). Since the Self-Proving model we train has causal attention masking, placing the output y before the proof means that the model "commits" to an output and only then proves it.