Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 98 (2024)

Models That Prove Their Own Correctness

Noga Amit*
UC Berkeley
nogamit@berkeley.edu

Shafi Goldwasser*
UC Berkeley
shafi.goldwasser@gmail.com

Orr Paradise*
UC Berkeley
orrp@eecs.berkeley.edu

Guy N. Rothblum*
Apple
rothblum@alum.mit.edu

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 a generic methods for learning Self-Proving models, and prove its convergence under certain assumptions.

The theoretical framework and results are complemented by experiments on an arithmetic capability: computing the greatest common divisor (GCD) of two integers. Our learning method is used to train a Self-Proving transformer that computes the GCD *and* proves the correctness of its answer.

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

*Authors listed alphabetically.

ISSN 1433-8092

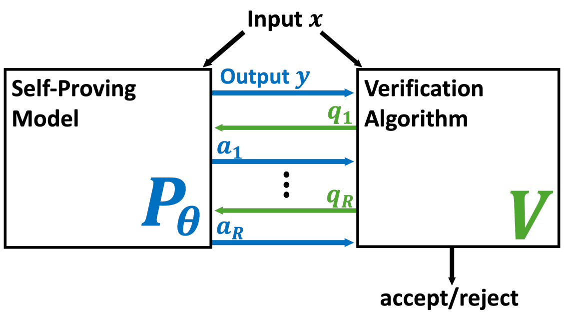

Figure 1: **Self-Proving models.** For input x, Self-Proving model P_θ generates an output y and sends it to a Verification Algorithm V. Then, over $i \in [R]$ rounds, V sends query q_i, and receives an answer a_i from P_θ. Finally, V decides ("accept/reject") whether it is convinced that y is a correct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

	Guarantee	Type	Def.
V	Completeness & Soundness	Worst-case $\forall x, y$	3.2
P_θ	Verifiability	Average-case $x \sim \mu,\ y \sim P_\theta(x)$	3.4

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.

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 3).
- We propose two methods for learning Self-Proving models in Section 4. The first, *Transcript Learning (TL)*, relies on access to transcripts of accepting interactions and is the focus of this paper; we prove convergence bounds for TL under convexity and Lipschitzness assumptions. The second method, *Reinforcement Learning from Verifier Feedback (RLVF)*, trains a model by emulating interaction with the verifier. We also present variants of these algorithms that use *Annotations* to improve learning in practice.
- We empirically study TL and Annotated-TL (ATL) for training Self-Proving transformers that compute the Greatest Common Divisor (GCD) of two integers. Table 2 demonstrates the efficacy of our methods, with additional experiments in Section 5. Our results may be of independent interest for research on the arithmetic capabilities of transformers (e.g. Charton 2024, Lee et al. 2024). Code, data and models will be made available upon publication.

Scope. This paper contains a theory of learned models that prove their own correctness via an Interactive Proof system. The fascinating and well-studied question of *which* settings are verifiable in an Interactive Proof system is beyond our scope. Our theory is general in that it pertains to *any* such setting, e.g., any decision problem solvable in polynomial space [Shamir, 1992]. See Goldreich [2008a] for a primer on Proof systems more broadly.

Learning method	Correctness	Verifiability
GPT (baseline)	99.8%	-
GPT+TL	98.8%	60.3%
GPT+ATL	98.6%	96.7%

Table 2: **Self-Proving transformers computing the GCD.** We train a 6.7M 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; training with Annotated Transcript Learning (GPT+ATL) increases this to 96.7%. See Section 5 for details.

2 Related Work

This paper is situated at the intersection of machine learning (ML) theory and Interactive Proof systems (IPs). We briefly discuss recent relevant work from these literatures.

ML and IPs. IPs have found numerous applications in ML towards a diverse set of goals. Anil et al. [2021] introduce Prover–Verifier Games, a game-theoretic framework for learned provers and verifiers. 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., 2023]. 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 3.2). 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; Gransden et al. [2015], Polu and Sutskever [2020], Yang et al. [2023] and others train models to produce proofs in Coq, Metamath and Lean [de Moura et al., 2015]; 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 4.1) can be viewed as training the model on a Chain-of-Thought induced by the interaction of a verifier and an honest prover (Definition 3.2).

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 4.2) 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].

Transformers for arithmetic. In Section 5 we train and evaluate Self-Proving transformers to generate the GCD of two integers and prove its correctness to a verifier. These experiments leverage a long line of work on neural models of arithmetic tasks originating with Siu and Roychowdhury [1992]. Of particular relevance is the recent paper of Charton [2024], who trains transformers to generate the GCD—without a proof of correctness. We benefit from conclusions suggested in their work and start from a similar (scaled-down) experimental setup. Our main challenge (obtaining *Self-Proving* models) is overcome by introducing Annotated Transcript Learning (ATL).

We conduct ablation experiments to find two deciding factors in ATL. First, we study the effect of the amount of annotation given in the form of intermediate steps [Lee et al., 2024], which is related to autoregressive length complexity [Malach, 2023]. Second, we characterize ATL efficacy in terms of an algebraic property of the tokenization scheme (cf. Nogueira et al. 2021, Charton 2022, 2024).

3 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 3.1. We then introduce our main definition in Section 3.2.

3.1 Preliminaries

Let Σ be a set of finite 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 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 left for future work). The fundamental goal of machine learning can be thought of as learning a model of the ground truth F^*. Formally,

Definition 3.1 (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[1] 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 $\langle V, P_\theta \rangle (x, y)$.

We present a definition of Interactive Proofs restricted to our setting.

Definition 3.2. *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.*[2] *It* interacts *with P over R rounds (see Figure 1) and outputs a decision $\langle V, P \rangle (x, y) \in \{0, 1\}$. 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.[3]

- 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.

Remark 3.3 (Verifier efficiency)**.** *Definition 3.2 requires that V is a polynomial-time algorithm whereas provers are unbounded. This captures a requirement for* efficient verification. *We chose*

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

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

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

polynomial time as a measure of efficiency because it is common Proof systems literature. That said, one could adapt Definition 3.2 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 *stateful models*, e.g. a session with a chatbot.

Towards defining Self-Proving models (Section 3.2), 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 $x \sim \mu$.

3.2 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 call models capable of proving their own correctness as *Self-Proving models*.

Definition 3.4 (Self-Proving model)**.** *Fix a verifier V for a ground-truth $F^*\colon \Sigma^* \to \Sigma^*$ as in Definition 3.2, 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 3.5 (Verifiability $\implies$ correctness)**.** *Notice that the ground-truth F^* does not appear in Definition 3.4 except for the first sentence. Indeed, once it is established that V is a verifier for F^* (as per Definition 3.2), 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 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 Self-Provability 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 for 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.

4 Learning Self-Proving Autoregressive 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?

The challenge is to align the model with a verifier. We assume that the learner has access to input samples $x \sim \mu$ and correct outputs $F^*(x)$, as well as the verifier specification (code). Additionally, the learner can emulate the verifier, as the latter is required to be computationally efficient.[4]

Our focus is on autoregressive sequence-to-sequence (Self-Proving) models P_θ [Elman, 1990]. Such models generate their output by recursively prompting a randomized sampling from a base distribution p_θ over tokens Σ. For an input $z \in \Sigma^*$, the output $w \sim P_\theta(z)$ is generated as follows:

- Sample $w_1 \sim p_\theta(z)$.
- Let $j = 1$. While w_j is not the end-of-sequence token $\texttt{EOS} \in \Sigma^*$:
  - Sample $w_{j+1} \sim p_\theta(z w_1 \cdots w_j)$.
- Output $w = w_1 w_2 \cdots w_j$.

For any $z \in \Sigma^*$, it is useful to consider the vector of log-proabilities over Σ, denoted by $\log p_\theta(z) \in \mathbb{R}^{|\Sigma|}$. We assume that each coordinate in this vector is differentiable with respect to θ.

Our general 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).

Section 4.1 introduces *Transcript Learning* (TL), a learning algorithm that overcomes the exploration problem mentioned in the second point under the assumption that the learner has access to transcripts of interactions in which the verifier accepts. We prove convergence bounds for TL (Appendix A.1) and analyze it experimentally (Section 5).

Access to accepting transcripts is a reasonable assumption, for example, when there is an efficient honest prover that can generate such transcripts [Goldwasser et al., 2015]. When there is no access to accepting transcripts, we propose *Reinforcement Learning from Verifier Feedback* (Section 4.2).

[4] We refer the reader to classical literature on Interactive Proof systems for formal definitions of computational efficiency (e.g. Goldreich 2008b).

4.1 Transcript Learning

We present an algorithm for learning Self-Proving models which uses access to a distribution of accepting transcripts. This is a reasonable assumption to make when the honest prover P^* (see Definition 3.2) is efficient, as in the case in Doubly-Efficient Interactive Proof systems as defined by Goldwasser et al. [2015] and developed in other theoretical (e.g. Goldreich and Rothblum 2018) and applied (e.g. Zhang et al. 2021) works. In this case, an honest prover P^* can be run by the learner during training to collect accepting transcripts without incurring heavy computational cost.

The intuition behind Transcript Learning is that the interaction of the verifier and prover can be viewed as a sequence itself, which is called the *transcript* $\pi \in \Sigma^*$. 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.

In more detail, Transcript Learning requires access to an *(honest) transcript generator* $\mathcal{T}^*$. Given an input x, the generator $\mathcal{T}^*(x)$ samples a sequence $P^*(x)\pi^* \in \Sigma^*$ such that π^* is an accepted transcript. The generator is autoregressive, meaning that for any prefix of an accepted transcript $\pi^*_{\leq t} \in \Sigma^t$, the learner has access to the distribution over next tokens $\mathcal{T}^*(\pi_{\leq t}) \in \Sigma$.[5]

Transcript Learning (TL) trains a Self-Provable model by autoregressively optimizing towards generating accepting transcripts. It is described in Algorithm 1. At a very high level, it works by repeatedly sampling $x \sim \mu$ and transcript $y^*\pi^* \sim \mathcal{T}^*(x)$, and updating the logits $\log p_\theta$ towards agreeing with $y^*\pi^*$ via Gradient Ascent. We prove that, under certain conditions, it is expected to output a Self-Provable model.

Theorem 4.1 (Theorem A.5, informal)**.** *Fix an input distribution μ, a verifier V, a transcript generator $\mathcal{T}^*$, an autoregressive model family $\{P_\theta\}_\theta$ parameterized by $\theta \in \mathbb{R}^d$ for some $d \in \mathbb{N}$, and a norm $||\cdot||$ on $\mathbb{R}^d$. Assume that the* agreement function $A\colon \mathbb{R}^d \to [0,1]$ *defined by*

$$A(\theta) \coloneqq \Pr_{\substack{x\sim\mu \\ \pi^*\sim\mathcal{T}^*(x)}} [\mathrm{Transcript}(\langle V, P_\theta \rangle (x)) = \pi^*]$$

is concave and differentiable in θ. 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.*
- *The total number of tokens sent by the prover to the verifier V in any interaction is at most C.*

Denote by $\bar{\theta}$ the output of TL running for number of iterations

$$N \geq 4 \cdot C^2 \cdot \frac{B^2_{\mathrm{Norm}} \cdot B^2_{\mathrm{Lip}}}{\varepsilon^2} \tag{2}$$

and learning rate $\lambda = B_{\mathrm{Norm}}/CB_{\mathrm{Lip}}\sqrt{N}$. Then the expected Verifiability of $\bar{\theta}$ is at least $1 - \varepsilon$.

The proof (Appendix A) goes by reduction to Stochastic Gradient Descent (SGD). We show that the learner can use only its available tools—sampling honest transcripts, emulating the verifier,

[5] Formally, if the generator is prompted with any string that cannot be completed to an accepted transcript, it outputs a dummy symbol $\perp \in \Sigma$.

and differentiating the logits—to estimate the agreement gradient $\nabla A(\theta)$. Since the agreement $A(\theta)$ lower bounds the Verifiability of P_θ, the former can be used as a surrogate objective for the latter.

The conditions for Theorem 4.1 can be split into two. First, the standard conditions used to prove SGD convergence: convexity,[6] B_{Norm}-boundedness, and B_{Lip}-Lipschitzness. Second, there is a bound C on the *communication complexity* of the prover in the Interactive Proof system.

Quantitatively, the efficiency of TL is captured by the *number of iterations* N. It is desirable to minimize N, which is also the *number of samples* needed from the distribution μ and the transcript generator $\mathcal{T}^*$. Like the conditions on the theorem, the bound on N can too be decomposed into two factors: The right factor is the complexity of SGD ($B_{\text{Norm}}^2 B_{\text{Lip}}^2/\varepsilon^2$), and the left factor $O(C^2)$ is the communication complexity of the proof system. Minimizing communication complexity has been an overarching goal in the study of proof systems (e.g. Goldreich and Håstad 1998, Goldreich et al. 2002, Reingold et al. 2021). Theorem 4.1 formally shows the benefit of communication-efficient proof systems in the context of Self-Proving models.

4.2 Reinforcement Learning from Verifier Feedback (RLVF)

As mentioned in Section 4.1, Transcript Learning uses access to an honest transcript generator to estimate gradients of (a lower bound on) the Verifiability of a model P_θ.

Reinforcement Learning from Verifier Feedback (RLVF, Algorithm 2) estimates this gradient without access to a transcript generator. RLVF can be viewed as a modification of TL in which the learner emulates the interaction of the verifier with its own model P_θ. Rather than directly sampling from the generator as in TL, it collects accepting transcripts by rejection sampling on emulated transcripts.

This rejection sampling means that RLVF requires its initial model P_{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 first learned with TL (with Verifiability $\delta > 0$), and then "amplified" to a fully Self-Proving model using RLVF (cf. Nair et al. 2018).

We prove that RLVF learner can estimate the Verifiability gradient of P_θ using emulation alone in Lemma A.6. From a broader perspective, 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.

4.3 Learning from annotated transcripts

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 3 and 4 to this setting in Appendix C, where we present *Annotated Transcripts*. This can be viewed as adding Chain-of-Thought [Wei et al., 2022] to the model. The Transcript Learning algorithm naturally extends to annotated transcripts as well.

[6] Theorem 4.1 requires concavity because it guarantees maximization, rather than minimization. We leave it for future work to relax the differentiability assumption.

5 Experimental Results

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

5.1 Setup: Training transformers to predict the GCD of two integers

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.[7] 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 is 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 E.

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 D. 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 D, the proof π consists of a pair of integers who are *Bézout coefficients* for x_1, x_2.
- *Soundness:* If $y \neq GCD(x_1, x_2)$, then $V(\mathbf{x_1 x_2 y}\pi)$ rejects 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 on the second page of this paper shows that Transcript Learning for 100K iterations ($\approx$100M samples) results in a Self-Proving transformer that correctly proves 60.3% of its answers; there is an additional 38.5% answers which are correct, but the transformer fails to generate an accepted proof. Annotated Transcript Learning all but closes this gap, proving 96.7% of its answers. We further investigate the effect of annotations next.

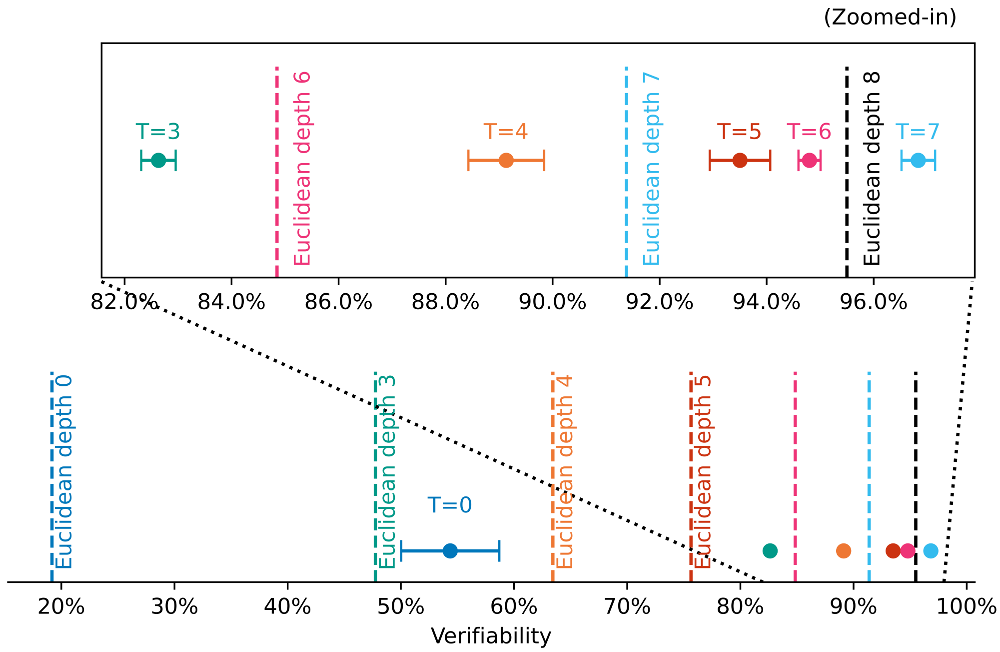

Figure 2: **Verifiability with increasing amounts of annotation**. T is the number of steps added in Annotated Transcript Learning. Dashed lines indicate *Euclidean depth*, that bound the Verifiability of models that prove *only* for integers up to a certain number of steps. Each T was run with three seeds, with mean $\pm$ standard error depicted.

5.2 Models generalize beyond annotations

The proof π is annotated by including intermediate steps in its computation. Details are deferred to Appendix D; 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 2 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 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 the to compute (in-context) the simple 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 2 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

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

increases Verifiability, but all models exceed their corresponding Euclidean depth bound.

5.3 Base of representation

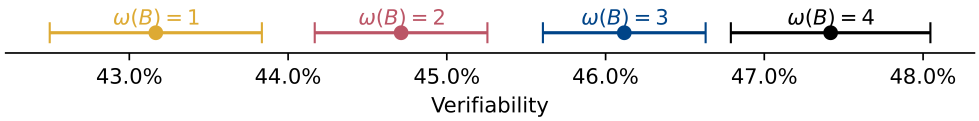

Figure 3: **The number of prime divisors of a base $\omega(B)$ determines Verifiability.** For each $o \in [4]$, we sampled 17 bases $B \in \{2, \dots, 1386\}$ such that $\omega(B) = o$. A Self-Proving transformer was trained via Transcript Learning for twenty epochs on an identical dataset of 1024K samples encoded in base B. For each $\omega(B)$ we depict the mean $\pm$ standard error.

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 whether the factorization of B has a similar effect on Verifiability as well, we train transformers on 68 bases varying the number of prime divisors $\omega(B)$ from $\omega(B) = 1$ (i.e., B is a prime power) to $\omega(B) = 4$. Figure 3 shows that $\omega(B)$ correlates not just with correctness [Charton, 2024], but also with Verifiability. Although the finding is statistically significant (no overlapping error margins), the overall difference is by a few percentage points; we attribute this to the smaller (10%) number of samples on which models were trained, relative to our other experiments.

6 Conclusions

Trust between a learned model and its user is fundamental. In recent decades, Interactive Proofs [Goldwasser et al., 1985] have emerged as a general theory of trust established via verification algorithms. This work demonstrates that models can learn to formally prove their answers in an Interactive Proof system. We call models that possess this capability *Self-Proving*.

The definition of Self-Proving models forms a bridge between the rich theory of Interactive Proofs and the contemporary topic of Trustworthy ML. Interactive Proofs offer formal *worst-case soundness guarantees*; thus, users of Self-Proving models can be confident when their models generate correct answers—and detect incorrect answers with high probability.

We demonstrate the theoretical viability of our definition with two generic learning algorithms: Transcript Learning (TL) and Reinforcement Learning from Verifier Feedback (RLVF). The analyses of these algorithms is informed by techniques from theories of learning, 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 (for example, by taking advantage of properties of the verifier).

To better understand the training dynamics of (Annotated) TL, we train Self-Proving transformers for the Greatest Common Divisor (GCD) problem. We train a small (6.3M parameter) transformer that learns to generate correct answers *and proofs* with high accuracy. Facing forward, we note that

Interactive Proofs exist for capabilities far more complex than the GCD [Shamir, 1992]; scaling up our experiments is the next step towards bringing Self-Proving models from theory to practice.

Acknowledgments

We are grateful to Micah Carroll and Avishay Tal 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

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.

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.

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.

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.

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.

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.

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

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

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.

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.

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.

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

Jonah Brown-Cohen, Geoffrey Irving, and Georgios Piliouras. Scalable AI safety via doubly-efficient debate. *CoRR*, abs/2311.14125, 2023. doi: 10.48550/ARXIV.2311.14125. URL https://doi.org/10.48550/arXiv.2311.14125.

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.

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.

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.

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.

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.

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

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.

Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). 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 378–388. Springer, 2015. doi: 10.1007/978-3-319-21401-6_26. URL https://doi.org/10.1007/978-3-319-21401-6_26.

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.

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.

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.

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.

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.

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.

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.

Kai-Yeung Siu and Vwani P. Roychowdhury. Optimal depth neural networks for multiplication and related problems. In Stephen Jose Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992]*, pages 59–64. Morgan Kaufmann, 1992. URL http://papers.nips.cc/paper/657-optimal-depth-neural-networks-for-multiplication-and-related-problems.

Eran Malach. Auto-regressive next-token predictors are universal learners. *CoRR*, abs/2309.06979, 2023. doi: 10.48550/ARXIV.2309.06979. URL https://doi.org/10.48550/arXiv.2309.06979.

Rodrigo Frassetto Nogueira, Zhiying Jiang, and Jimmy Lin. Investigating the limitations of the transformers with simple arithmetic tasks. *CoRR*, abs/2102.13019, 2021. URL https://arxiv.org/abs/2102.13019.

François Charton. Linear algebra with transformers. *Trans. Mach. Learn. Res.*, 2022, 2022. URL https://openreview.net/forum?id=Hp4g7FAXXG.

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.

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.

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.

Oded Goldreich. *Computational complexity - a conceptual perspective*. Cambridge University Press, 2008b. ISBN 978-0-521-88473-0. doi: 10.1017/CBO9780511804106. URL https://doi.org/10.1017/CBO9780511804106.

Jeffrey L. Elman. Finding structure in time. *Cogn. Sci.*, 14(2):179–211, 1990. doi: 10.1207/S15516709COG1402_1. URL https://doi.org/10.1207/s15516709cog1402_1.

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.

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.

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.

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.

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, 2018. doi: 10.4230/LIPICS.ITCS.2018.18. URL https://doi.org/10.4230/LIPIcs.ITCS.2018.18.

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.

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, 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.

Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. *SIAM J. Comput.*, 50(3), 2021. doi: 10.1137/16M1096773. URL https://doi.org/10.1137/16M1096773.

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.

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.

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.

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.

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.

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

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

A Theoretical analyses for Section 4

In this section we provide a formal description and analysis of Transcript Learning (TL, Section 4.1) and Reinforcement Learning from Verifier Feedback (RLVF, Section 4.2). In Appendix A.1 we prove a convergence theorem for TL under convexity and Lipschitzness assumptions. Obtaining an analogous result for RLVF is more challenging; in lieu of a full analysis, we provide a lemma showing that the gradients estimated in the algorithm approximate the Verifiability of the model in Appendix A.2.

Specification of the learning model. We must first fully specify the theoretical framework in which our results reside. Continuing from Section 3, 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 consistence with the Interactive Proofs literature), we give the learner the full specification of the verifier V. More formally,

Definition A.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. Recall (Section 4) that 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,* $\nabla_\theta \log \Pr_{\sigma' \sim p_\theta(z)}[\sigma = \sigma']$.
- *Sample access to a 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.*

We remark that analysis of Transcript Learning will require a slight strengthening of the final item above. This is discussed in Appendix A.1.

Throughout this section, we will refer to the *transcript* of an interaction between a verifier and a prover (see Figure 1). We will this transcript by $\pi = (y, q_1, a_1, \dots, q_R, a_R)$, and for any index $s \in |\pi|$ we will write $\pi_{<s} \in \Sigma^{s-1}$ to denote the s-long prefix of π.

A.1 Transcript Learning

Recall that Transcript Learning requires access to an *honest transcript generator*. Before we can formally define this object, it will be useful to define a *query generator* for a verifier V.

Definition A.2 (Query generator)**.** *Fix a verifier* V *in a proof system with* $R \in \mathbb{N}$ *rounds, where the verifier issues queries of length* $L_q = |q_i|$ *and the prover (model) responses with answers of length* $L_a = |a_i|$*.*[8] *The* 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* $(q_i, a_i)_{i=1}^r$*,* $V_q(x, y, q_1, a_1, \dots, q_r, a_r)$ *is a random variable over* Σ^{L_q}*.*[9]

We assume that access to the verifier specification (Definition A.1) includes access to the query generator. After all, the verifier—who is assumed to be efficient—samples from V_q during the interaction. Moreover, we will assume that for any partial interaction and any sequence q', the learner is able to compute the probability that q' was the next query. In other words, we assume the learner can compute the probability density function of V_q.

[8] We can assume that queries (resp. answers) all have the same length by padding shorter ones.

[9] 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 $\perp \cdots \perp \in \Sigma^{L_q}$.

A *transcript generator* is 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 who is fully supported on transcripts accepted by the verifier.

Definition A.3 (Honest 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 $r \le R$, the marginal of $\mathcal{T}_V(x)$ on the r^{th} query q_r agrees with the corresponding marginal of the query generator $(V_q)_r$.*

A transcript generator $\mathcal{T}_V^* \coloneqq \mathcal{T}_V$ 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 A.3 to emphasize that transcripts can be "collected" independently of the honest prover (see completeness in Definition 3.2). As long as the generator is fully supported on honest transcripts, it can be used for Transcript Learning (Algorithm 1 described below).

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^*. **for** $s \in S(r)$ **do**
      Compute `# Forwards and backwards pass`

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

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

    If $r \geq 1$, let q_r denote the r^{th} query q_r in π^*, and let t denote its first index. That is, $\pi^*_{<t} = (y, q_1, a_1, \ldots, q_{t-1}, a_{t-1})$. Compute `# Emulate the verifier`

$$\beta_r(\theta_i) := \Pr_{q' \sim V_q(x\pi^*_{<t})}[q' = q].$$

    Update

$$\theta_{i+1} := \theta_i + \lambda \cdot \alpha_0(\theta_i) \cdot \left(\prod_{\substack{r \in [R] \\ s \in S(r)}} \beta_r(\theta_i)\alpha_s(\theta_i) \right) \cdot \sum_{\substack{r \in [R] \cup \{0\} \\ s \in S(r)}} \vec{d}_s(\theta_i).$$

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

Convergence of TL is proven by a reduction to Stochastic Gradient Descent (SGD). Essentially, we are tasked with proving that TL estimates a surrogate of the Verifiability-gradient of its model P_θ. More precisely, TL estimates the gradient of a function that bounds the Verifiability from below. Maximizing this function therefore maximizes the Verifiability.

The lower-bounding function is the agreement of the transcript generator induced by P_θ with the provided honest transcript generator $\mathcal{T}^*_V$. More formally, we let $\mathcal{T}^\theta_V$ denote the transcript generator induced by the model P_θ: for each x, $\mathcal{T}^\theta_V(x)$ is simply the distribution over transcripts of interactions between V and P_θ on input x. We first prove TL correctly estimates the gradient of $A(\theta)$ in its update step.

Lemma A.4 (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 be the* *parameters of a model P_θ such that*

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

is differentiable in θ. Then

$$\nabla A(\theta) = \mathop{\mathbb{E}}_{\substack{x \sim \mu \\ \pi^* \sim \mathcal{T}_V^*}} \left[\alpha_0(\theta) \cdot \left(\prod_{\substack{r \in [R] \\ s \in S(r)}} \beta_r(\theta) \cdot \alpha_s(\theta) \right) \cdot \sum_{\substack{r \in [R] \cup \{0\} \\ s \in S(r)}} \vec{d}_s(\theta) \right]$$

where $S(r)$, $\beta_r(\theta)$, $\alpha_s(\theta)$ and $\vec{d}_s(\theta)$ are as defined in Algorithm 1.

Note that Lemma A.4 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. We now prove Lemma A.4, from which we derive Theorem 4.1.

Proof. Throughout this proof, expectations and probabilities will be over the same distributions as in the lemma statement. First, by the law of total probability, and linearity of the gradient,

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

Next, we use the law of total probability together with the autoregressive property of P_θ (Section 4) to switch from probabilities on transcripts, to products of next-token probabilities. Formally, consider any fixed input x, honest transcript $\pi^* = (y^*, q_1^*, a_1^*, \ldots, q_R^*, a_R^*)$, and denote a random transcript sampled from $\mathcal{T}_V^\theta(x)$ by $\pi = (y, q_1, a_1, \ldots, q_R, a_R)$. For any $r \in [R]$ denote the random variables $V_q^{<r} \coloneqq V_q(y, q_1, a_1, \ldots, q_{r-1}, a_{r-1})$ and $\mathcal{T}_V^{\theta,<r} \coloneqq \mathcal{T}_V^\theta(y q_1 a_1 \cdots a_{r-1} q_r)$. Then,

$$\begin{aligned}
\Pr_{\pi} [\pi = \pi^*] &\coloneqq \Pr_{\pi}[(y^*, q_1^*, a_1^*, \ldots, q_R^*, a_R^*) = (y, q_1, a_1, \ldots, q_R, a_R)] && (3) \\
&= \Pr_{y \sim P_\theta(x)} [y = y^*] \cdot \prod_{r \in [R]} \Pr_{q \sim V_q^{<r}} [q = q_r^*] \cdot \Pr_{a \sim \mathcal{T}_V^{\theta,<r}} [a = a_r^*] \\
&= \Pr_{y \sim P_\theta(x)} [y = y^*] \cdot \prod_{\substack{r \in [R] \\ s \in S(r)}} \Pr_{q \sim V_q^{<r}} [q = q_r^*] \cdot \Pr_{\sigma \sim p_\theta(\pi_{<s}^*)} [\sigma = \pi_s^*] && (4) \\
&= \alpha_0(\theta) \cdot \left(\prod_{\substack{r \in [R] \\ s \in S(r)}} \beta_r(\theta) \cdot \alpha_s(\theta) \right), && (5)
\end{aligned}$$

where Equation (3) uses independence of the verifier and model's randomness, Equation (4) uses the autoregressive property of P_θ (Definition A.1), and Equation (5) is by definition of α_s and β_r.

Next, a basic calculus identity gives

$$\nabla_\theta \left(\Pr_{\pi} [\pi = \pi^*] \right) = \Pr_{\pi} [\pi = \pi^*] \cdot \nabla_\theta \log \left(\Pr_{\pi} [\pi = \pi^*] \right). \qquad (6)$$

Let us focus on the rightmost factor. By Equation (5),

$$\nabla_\theta \log \left(\Pr_\pi [\pi = \pi^*] \right) = \nabla_\theta \log \alpha_0(\theta) \cdot \left(\prod_{\substack{r \in [R] \\ s \in S(r)}} \beta_r(\theta) \cdot \alpha_s(\theta) \right)$$

$$= \nabla \log_\theta \alpha_0(\theta) + \sum_{\substack{r \in [R] \\ s \in S(r)}} \nabla_\theta \log \beta_r(\theta) + \nabla_\theta \log_\theta \alpha_s(\theta)$$

$$= \nabla \log_\theta \alpha_0(\theta) + \sum_{\substack{r \in [R] \\ s \in S(r)}} \nabla_\theta \log_\theta \alpha_s(\theta) \tag{7}$$

$$= \sum_{\substack{r \in [R] \cup \{0\} \\ s \in S(r)}} \nabla_\theta \log_\theta \alpha_s(\theta) = \sum_{\substack{r \in [R] \cup \{0\} \\ s \in S(r)}} \vec{d}_s(\theta) \tag{8}$$

where Equation (7) is because $\log \beta_r(\theta) := \log \Pr_{q' \sim V_q(x\pi^*_{<t})}[q' = q]$ is a constant and therefore has a gradient of zeros, and Equation (8) is by definition of $\vec{d}_s(\theta)$. Combining Equations (5), (6) and (8) concludes the proof. □

We are now ready to prove Theorem 4.1. We restate it below in full formality.

Theorem A.5 (Theorem 4.1, formal)**.** *Fix a verifier V, an input distribution μ, and 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

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

is concave and differentiable in θ. 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 $||\theta^*|| < B_{\text{Norm}}$ such that $A(\theta^*) \geq 1 - \varepsilon/2$.*
- *For all θ, the logits of P_θ are B_{Lip}-Lipschitz in θ. That is,*

$$\sup_{\substack{\theta \in \mathbb{R}^d \\ z \in \Sigma^*}} ||\nabla_\theta \log p_\theta(z)|| \leq B_{\text{Lip}}.$$

- *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 number of iterations

$$N \geq 4 \cdot C^2 \cdot \frac{B_{\text{Norm}}^2 \cdot B_{\text{Lip}}^2}{\varepsilon^2}$$

and learning rate $\lambda = B_{\text{Norm}}/CB_{\text{Lip}}\sqrt{N}$. Then the expected Verifiability (over the randomness of the samples collected by TL) of $\bar{\theta}$ is at least $1 - \varepsilon$. That is,

$$\mathbb{E}_{\bar{\theta}}[\text{ver}_{V,\mu}(\bar{\theta})] \geq 1 - \varepsilon.$$

Proof. Our strategy is to cast TL as Stochastic Gradient Ascent and apply Fact B.2. 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 $||\theta^*|| \leq B_{\text{Norm}}$.

First, notice that

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

This is 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

$$\begin{aligned}
&\left\| \alpha_0(\theta_i) \cdot \left(\prod_{\substack{r\in[R] \\ s\in S(r)}} \beta_r(\theta_i)\alpha_s(\theta_i) \right) \cdot \sum_{\substack{r\in[R]\cup\{0\} \\ s\in S(r)}} \vec{d}_s(\theta_i) \right\| \\
&\qquad\qquad = \left| \alpha_0(\theta_i) \cdot \prod_{\substack{r\in[R] \\ s\in S(r)}} \beta_r(\theta_i)\alpha_s(\theta_i) \right| \cdot \left\| \sum_{\substack{r\in[R]\cup\{0\} \\ s\in S(r)}} \vec{d}_s(\theta_i) \right\| \\
&\qquad\qquad\qquad\qquad \leq 1 \cdot \sum_{\substack{r\in[R]\cup\{0\} \\ s\in S(r)}} \left\| \vec{d}_s(\theta_i) \right\|,
\end{aligned}$$

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

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

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 A.4 shows that TL samples from a gradient estimator for $A(\theta)$, while the above equation shows that the gradient is upper-bounded by $C \cdot B_{\text{Lip}}$. We can therefore apply Fact B.2 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^*.

□

A.2 Reinforcement Learning from Verifier Feedback

Our second learning method, Reinforcement Learning from Verifier Feedback (RLVF, Algorithm 2), does not require access to an honest transcript generator. Instead, the learner learns P_θ generates transcripts herself by emulating the interaction of the verifier with the current Self-Proving model P_θ. When an accepting transcript is generated, the learner updates the parameters θ towards generating such transcript.

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 $a_0 \coloneqq y$ and $d_i \coloneqq \vec{0}$.
4: **foreach** *Round of interaction* $r = 1, \ldots R$ **do**
5: Sample the r^{th} query `# Emulate the verifier`

$$q_r \sim V_q(x, a_0, q_1, a_1, \ldots, q_r, a_r).$$

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

$$a_r \sim P_\theta(x, a_0, q_1, a_1, \ldots, q_r, a_r, q_{a_{r+1}}).$$

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

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

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

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

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

Before we continue with formal analysis of Algorithm 2, let us make a few observations.

Firstly, 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 $\delta > 0$), and 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.

Secondly, in comparing Algorithms 1 and 2, we see that the latter (RLVF) does not keep track of the probabilities α_s and β_r. This is because, in RL terms, RLVF is an *on-policy* algorithm; it generates transcripts using the current learned model, unlike TL which samples them from a distribution whose parameterization is unknown to the learner. Hence, the update step in RLVF is simpler than TL. Furthermore, the RLVF learner does not require access to the density function of the query generator V_q (Definition A.2) unlike its TL counterpart.

We now prove that the update step in RLVF maximizes the Verifiability of P_θ; this is analogous to Lemma A.4 for TL. We leave it for future work to use Lemma A.6 to obtain convergence bounds on RLVF (analogous to Theorem A.5). As mentioned in Section 4.2, the gap between the lemma and a full convergence theorem (informally) reduces to the problem of obtaining convergence bounds for Policy Gradient methods, a challenging and active research direction (e.g. Agarwal et al. 2021).

Lemma A.6 (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 3.4). For any θ, if $\mathrm{ver}(\theta)$ is differentiable in θ, then*

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

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

Proof. Recall the *transcript generator of P^θ*, denoted by T_V^θ (see Lemma A.4). By the definitions of Verifiability in Definition 3.4 and $V(x, y, q_1, \ldots, a_R)$ in the lemma statement,

$$\begin{aligned} \mathrm{ver}(\theta) &:= \Pr_{\substack{x\sim\mu \\ y\sim P_\theta(x)}} \left[\langle V, P_\theta\rangle (x, y) \text{ accepts}\right] \\ &= \mathop{\mathbb{E}}_{\substack{x\sim\mu \\ y\sim P_\theta(x) \\ (q_r,a_r)_{r=1}^R}} \left[\mathrm{Acc}_V(x, y, q_1, \ldots, a_R)\right] \\ &= \mathop{\mathbb{E}}_{x\sim\mu} \left[\Pr_{\pi\sim\mathcal{T}_V^\theta} [\mathrm{Acc}_V(x, \pi)\ ] \right] \end{aligned} \tag{9}$$

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

$$\Pi^*(x) := \{\pi^* \in \Sigma^* : \mathrm{Acc}_V x, \pi^* \text{ accepts}\}.$$

Noting that $\Pi^*(x)$ has finite or countably infinite cardinality, for any fixed input x we can write

$$\Pr_{\pi\sim\mathcal{T}_V^\theta}[\mathrm{Acc}_V(x, \pi)] = \sum_{\pi^*\in\Pi^*(x)} \Pr_{\pi\sim\mathcal{T}_V^\theta(x)}[\pi = \pi^*]. \tag{10}$$

We will use Equations (3) through (8) in the proof of Lemma A.4. 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 (9) and (10), by linearity of expectation we have that

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

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 $\mathcal{T}^\theta(x)$. □

B Preliminaries on Stochastic Gradient Ascent

For convenience of the reader, we provide a description of Stochastic Gradient Ascent 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.

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 B.1 (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.

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

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 := 0⃗ ∈ R^d.
for i = 0, ..., N − 1 do
    Sample v_i ∼ D_f(w_{i−1}).
    Update w_i := w_{i−1} + λ · v_i.
Output w̄ := (1/N) Σ_{i∈[N]} w_i.
```

Fact B.2. *Fix a concave $f\colon \mathbb{R}^d \to [0,1]$, a norm $||\cdot||$ on $\mathbb{R}^d$, and upper-bounds $B_{\text{Norm}}, B_{\text{Lip}} > 0$. Let*

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

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

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

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

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

C Annotations

We formally capture the modification described in Section 4.3 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 3.2, 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:

- 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.

D A simple proof system for the 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*. Paul has two integers 212 and 159; he claims that $GCD(212, 159) = 53$. An inefficient way for 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.

Claim D.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 Claim D.1 are known as *Bézout coefficients* for (x_0, x_1). Claim D.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 3.2, we can have the verifier issue a dummy query. Furthermore, by Claim D.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 3.3.

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.[10]

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 $q := \lfloor r_0 / r_1 \rfloor$.
  Update $(r_0, r_1) := (r_1, r_0 - q \times r_1)$.
  Update $(s_0, s_1) := (s_1, s_0 - q \times s_1)$.
Output GCD $y = r_0$ and Bézout coefficients $z_0 := s_0$ and $z_1 := (r_0 - s_0 \cdot x_0)/x_1$.

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

[10] Our description is the same as https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm.

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 C 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).

- *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.[11]

E Experiment details

We provide details of how we implemented the experiments in Section 5 and additional figures for each experiment.

Model architecture. We use Karpathy's *nanoGPT*.[12] We use a 6.7M 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 $\beta2 = 0.95$, 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. 2 are trained for 100K iterations on a dataset of of $\approx$10M samples. For Figure 3 (base ablation) we train for 20K iterations on a dataset of $\approx$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. Longer runs (annotation-cutoff ablation) took about 75 minutes each. Shorter runs (base ablation) took about 15 minutes. The total running time of our experiments was approximately 40 hours, excluding time dedicated to a random hyperparameter search. The overall disk space needed for our models and data (to be made available upon publication) is 4GB.

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

[12]https://github.com/karpathy/nanoGPT.

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 annotaitons 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.

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.

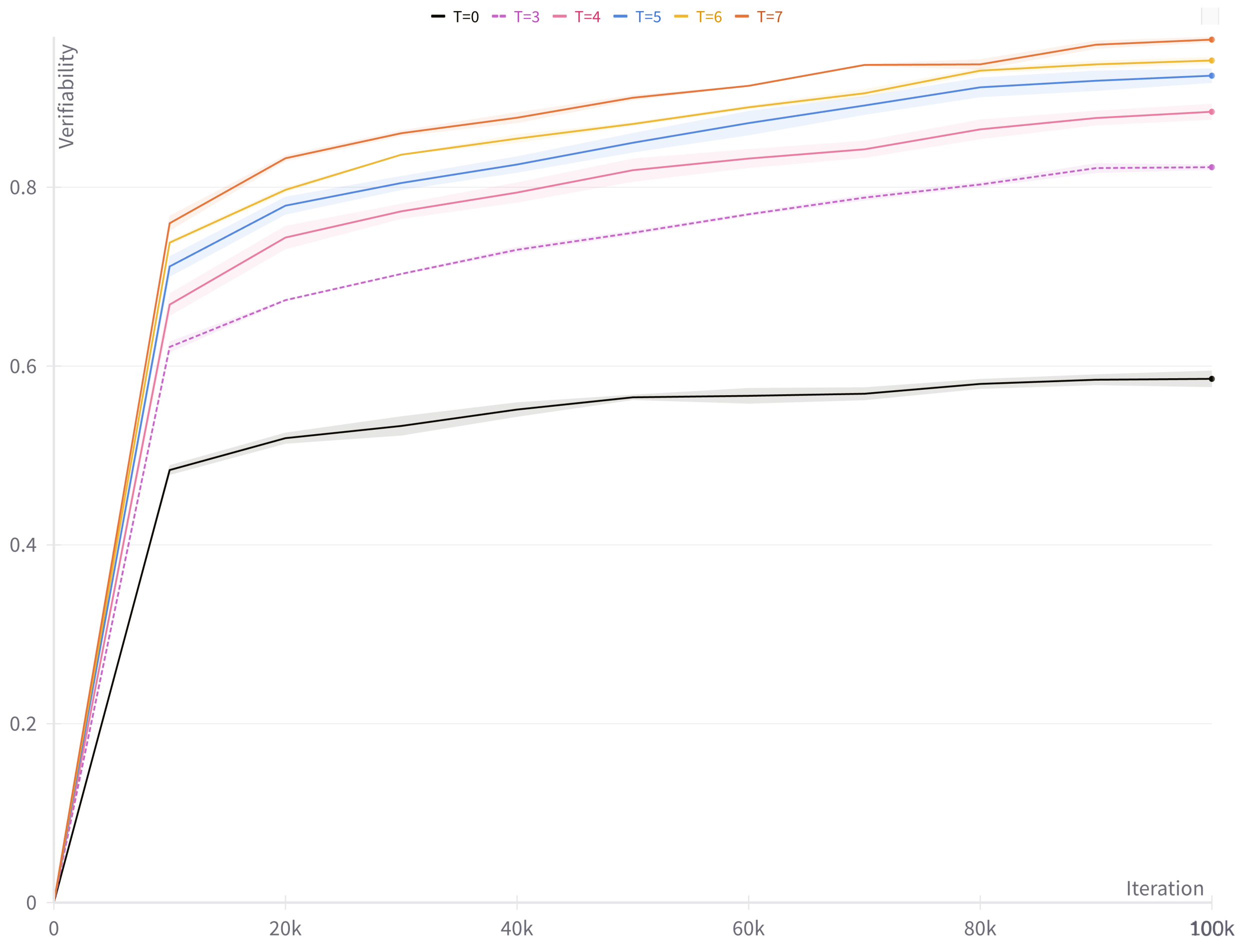

Figure 4: **Verifiability as a function of the number of samples** N. Each iteration (X axis) is a batch of 1024 samples from a dataset of ≈10M sequences. Every 10k iterations, Verifiability was evaluated on a held-out dataset of 1k inputs (as described in Section 5). T is the number of steps in Annotated Transcript Learning (Figure 2), 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.

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il