Models That Prove Their Own Correctness

Anonymous Authors[1]

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. We devise a generic method for learning Self-Proving models, and we prove convergence bounds under certain assumptions. As an empirical exploration, our learning method is used to train a Self-Proving transformer that computes the Greatest Common Divisor (GCD) *and* proves the correctness of its answer.

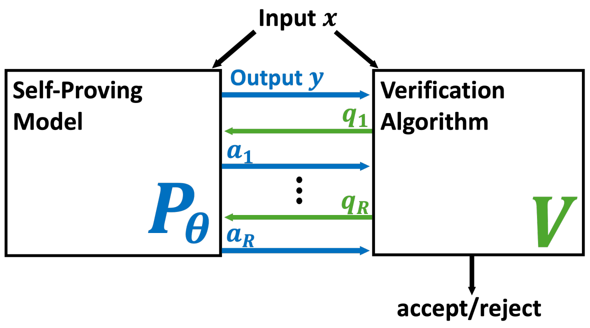

Figure 1. **Self-Proving models.** For input x, Self-Proving model P_θ generates an output y and sends it to a Verification Algorithm V. Then, over $i \in [R]$ rounds, V sends query q_i, and receives an answer a_i from P_θ. Finally, V decides ("accept/reject") whether it is convinced that y is a correct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

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* (Figure 1) that prove the correctness of their answers via an Interactive Proof system (Goldwasser et al., 1985). Self-Proving models successfully convince a verification algorithm V with *worst-case soundness guarantees*: for any question, with high probability over the interaction, V will not be convinced of an incorrect answer. This is even when the prover with which V is interacting has access to V's specification, and far more computational power.

Contributions and organization. In Section 2 we define Self-Proving models. In Section 3 we propose *Transcript Learning (TL)*, a method for learning learning Self-Proving models; we prove convergence bounds for TL under convexity and Lipschitzness assumptions. In Section 4 and Table 1 we evaluate TL and its Annotated variant (ATL) by training Self-Proving transformers to compute the Greatest Common Divisor (GCD).[1]

Related work is deferred to Appendix A.

2. 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 literature in Section 2.1. We then introduce our main definition in Section 2.2.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

[1]Code, data and models will be released upon publication.

Table 1. **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 4 for more details.

LEARNING METHOD	CORRECTNESS	VERIFIABILITY
GPT (BASELINE)	99.8%	-
GPT+TL	98.8%	60.3%
GPT+ATL	98.6%	96.7%

2.1. 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 2.1 (Correctness)**.** Let μ be a distribution of input sequences in Σ^* and let $F^* : \Sigma^* \to \Sigma^*$ 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 (a randomized mapping from Σ^* to Σ^*) as vanilla models. We change the notation to emphasize that P_θ outputs $y \sim P_\theta(x)$ but can also be prompted by the verifier, unlike F_θ who is only expected to generate an output.

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 $V^{P_\theta}(x, y)$.

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

Definition 2.2. Fix a soundness error $s \in (0, 1)$, a finite set of tokens Σ and a ground truth $F^* \colon \Sigma^* \to \Sigma^*$. A *verifier V (in an Interactive Proof) for F^** is a probabilistic polynomial-time algorithm that is given explicit inputs $x, y \in \Sigma^*$ and black-box (oracle) query access to a prover P.[2] It *interacts* with P over R rounds (see Figure 1) and outputs a decision $V^P(x, y) \in \{0, 1\}$. Verifier V satisfies the following two guarantees:

- *Completeness:* There exists an *honest prover* P^* such that, for all $x \in \Sigma^*$,
$$\Pr[V^{P^*}(x, F^*(x)) \text{ accepts}] = 1,$$
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
$$\Pr[V^P(x, y) \text{ accepts}] \leq s,$$
where the probability is over the randomness of V and P, and s is the soundness error.

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

[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.

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.

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

2.2. Self-Proving models

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

Definition 2.3 (Self-Proving model)**.** Fix a verifier V for a ground-truth $F^* \colon \Sigma^* \to \Sigma^*$ as in Definition 2.2, and a distribution μ over inputs Σ^*. The *Verifiability* of a model $P_\theta \colon \Sigma^* \to \Sigma^*$ is defined as

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

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

Now, 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.

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.

3. Learning Self-Proving Models

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

The challenge is to align the model with a verifier. The intuition behind our leaning method, *Transcript Learning (TL)*, 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.

We assume that, when training a Self-Proving model, 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]

We assume also that, when training a Self-Proving model, the learner has access to transcripts of interactions in which the verifier accepts. This is a reasonable assumption to make when the honest prover P^* (see Definition 2.2) 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 & Rothblum 2018) and applied (e.g. Zhang et al. 2021) works.[5] In this case, an honest prover P^* can be run by the learner during training to collect accepting transcripts without incurring heavy computational cost.

More formally, TL 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. TL then autoregressively optimizes the model 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 towards agreeing with $y^*\pi^*$ via Gradient Ascent.

We prove that, under certain conditions (fully specified in

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

[5]In Appendix C we introduce *Reinforcement Learning from Verifier Feedback (RLVF)*, which does not require access to accepting transcripts.

Appendix B), TL is expected to output a Self-Provable model.

Theorem 3.1 (Theorem B.5, informal)**.** *Fix an input distribution μ, a verifier V and an autoregressive model family $\{P_\theta\}_\theta$. Fix a transcript generator $\mathcal{T}^*$ such that the expected agreement with $\mathcal{T}^*$ is convex in θ. For any $\varepsilon > 0$ such that there exists θ^* with at least $\geq 1 - \varepsilon/2$ expected agreement with $\mathcal{T}^*$, let B be the minimal norm of such θ^*. Let $\rho > 0$ such that for all θ with $||\theta|| \leq B$, the logits of P_θ are ρ-Lipschitz in θ. Denote by $\bar{\theta}$ the output of TL running for number of iterations*

$$N \geq R^2 \cdot (L_a + 1)^2 \cdot \frac{4\rho^2 \cdot B^2}{\epsilon^2} \tag{2}$$

and learning rate $\lambda = \varepsilon/2R^2 L_a^2 \rho$. Then the expected Verifiability of $\bar{\theta}$ is at least $1 - \varepsilon$.

Looking at Equation (2), we see that the sample complexity of TL grows like that of SGD (e.g. Shalev-Shwartz & Ben-David 2014), multiplied by the number of rounds and length of answers in the proof system. Minimizing these quantities (known collectively as the *communication complexity*) has been an overarching goal in the study of proof systems (e.g. Goldreich & Håstad (1998); Goldreich et al. (2002); Reingold et al. (2021)). Theorem 3.1 formally shows the benefit of communication-efficient proof systems in the context of Self-Proving models.

3.1. 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 formally adapt the framework from Sections 2 and 3 to this setting in Appendix D, 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.

4. Experimental Results

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

Setup. 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 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.[6] 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 F.

Following Charton (2024), 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 devise a natural proof system. Its specification and formal guarantees are deferred to Appendix E. We denote its verification algorithm by V, and highlight some useful features:

- 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 E, 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 1 (p. 2) shows that Transcript Learning for 100K iterations 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.

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

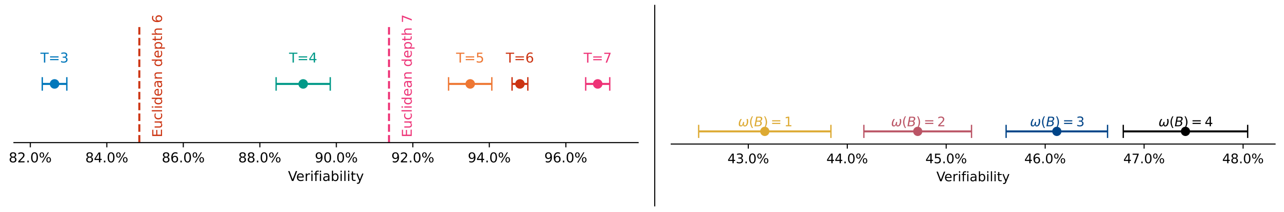

Figure 2. **Left:** T is the number of steps added in Annotated Transcript Learning. Dashed lines bound the Verifiability of models that can *only* prove for integers up to a certain number of steps. Off chart to the left are bounds for depths 3 (47%), 4 (63%), and 5 (75%). Each T was run with three seeds, with mean $\pm$ standard error depicted. See Appendix G for additional figures. **Right:** For each $b \in [4]$, we sampled 17 bases $B \in \{2, \ldots, 1386\}$ such that the number of prime divisors $\omega(B) = b$. A Self-Proving transformer was trained via Transcript Learning for twenty epochs on a dataset of 1024K identical samples encoded in base B.

Models generalize beyond annotations. The proof π is annotated by including intermediate steps in its computation. Details are deferred to Appendix E; 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 increases Verifiability, but all models exceed their corresponding Euclidean depth bound.

Base of representation. As mentioned previously, Charton (2024) concludes that, for a given base of representation B, transformers correctly compute the GCD of integers x_1, x_2 that are products of primes dividing B. Simply put, choosing a base B with many different prime factors yields models with better correctness (accuracy), which suggests why base $B = 210 = 2\cdot3\cdot5\cdot7$ yielded the best results. Figure 2 shows that $\omega(B)$ correlates not just with correctness (Charton, 2024), but also with Verifiability. Although the finding is statistically significant, 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.

5. 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 propose *Transcript Learning (TL)*, a generic method for learning Self-Proving models. One natural direction for future work is improving the sample complexity bounds for TL (Theorem 3.1). Another direction is designing a method for learning without an honest transcript generator; we propose such a method (inspired by RLHF, Irving et al. 2018) in Appendix C.

We train a small (6.3M parameter) transformer that learns to generate the Greatest Common Divisor (GCD), *and proves its answer*. 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.

References

Agarwal, A., Kakade, S. M., Lee, J. D., and Mahajan, G. 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.

Anil, C., Zhang, G., Wu, Y., and Grosse, R. B. Learning to give checkable answers with prover-verifier games. *CoRR*, abs/2108.12099, 2021. URL https://arxiv.org/abs/2108.12099.

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

Brown-Cohen, J., Irving, G., and Piliouras, G. 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.

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

Charton, F. 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.

Christiano, P. F., Leike, J., Brown, T. B., Martic, M., Legg, S., and Amodei, D. Deep reinforcement learning from human preferences. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 4299–4307, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/d5e2c0adad503c91f91df240d0cd4e49-Abstract.html.

Condon, A., Feigenbaum, J., Lund, C., and Shor, P. W. 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.

de Moura, L. M., Kong, S., Avigad, J., van Doorn, F., and von Raumer, J. The lean theorem prover (system description). In Felty, A. P. and Middeldorp, A. (eds.), *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*, pp. 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.

Frieder, S., Pinchetti, L., Chevalier, A., Griffiths, R., Salvatori, T., Lukasiewicz, T., Petersen, P., and Berner, J. Mathematical capabilities of chatgpt. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *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.

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

Goldreich, O. and Håstad, J. 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.

Goldreich, O. and Rothblum, G. N. Simple doubly-efficient interactive proof systems for locally-characterizable sets. In Karlin, A. R. (ed.), *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pp. 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.

Goldreich, O., Vadhan, S. P., and Wigderson, A. 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.

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

Goldwasser, S., Kalai, Y. T., and Rothblum, G. N. 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.

Goldwasser, S., Rothblum, G. N., Shafer, J., and Yehudayoff, A. Interactive proofs for verifying machine learning. In Lee, J. R. (ed.), *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, pp. 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.

Gransden, T., Walkinshaw, N., and Raman, R. SEPIA: search for proofs using inferred automata. In Felty, A. P. and Middeldorp, A. (eds.), *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*, pp. 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.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the MATH dataset. In Vanschoren, J. and Yeung, S. (eds.), *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.

Irving, G., Christiano, P. F., and Amodei, D. AI safety via debate. *CoRR*, abs/1805.00899, 2018. URL http://arxiv.org/abs/1805.00899.

Karp, R. M. Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W. (eds.), *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pp. 85–103. Plenum Press, New York, 1972. doi: 10.1007/978-1-4684-2001-2_9. URL https://doi.org/10.1007/978-1-4684-2001-2_9.

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

Lee, N., Sreenivasan, K., Lee, J. D., Lee, K., and Papailiopoulos, D. Teaching arithmetic to small transformers. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 6-11, 2024*. OpenReview.net, 2024.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. 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.

Malach, E. 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.

Murty, S., Paradise, O., and Sharma, P. Pseudointelligence: A unifying lens on language model evaluation. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pp. 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.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pp. 6292–6299. IEEE, 2018. doi: 10.1109/ICRA.2018.8463162. URL https://doi.org/10.1109/ICRA.2018.8463162.

Nogueira, R. F., Jiang, Z., and Lin, J. Investigating the limitations of the transformers with simple arithmetic tasks. *CoRR*, abs/2102.13019, 2021. URL https://arxiv.org/abs/2102.13019.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P. F., Leike, J., and Lowe, R. Training language models to follow instructions with human feedback. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *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-Confere
html.

Paradise, O. 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.

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

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

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J., Ellenberg, J. S., Wang, P., Fawzi, O., et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Shalev-Shwartz, S. and Ben-David, S. *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.

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

Siu, K. and Roychowdhury, V. P. Optimal depth neural networks for multiplication and related problems. In Hanson, S. J., Cowan, J. D., and Giles, C. L. (eds.), *Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992]*, pp. 59–64. Morgan Kaufmann, 1992. URL http://papers.nips.cc/paper/657-optimal-depth-neural-networks-for-multiplication-and-related-problems.

Sutton, R. S., McAllester, D. A., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In Solla, S. A., Leen, T. K., and Müller, K. (eds.), *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pp. 1057–1063. The MIT Press, 1999. URL http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.

Trinh, T. H., Wu, Y., Le, Q. V., He, H., and Luong, T. 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.

Turpin, M., Michael, J., Perez, E., and Bowman, S. R. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *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-Confere html.

Uesato, J., Kushman, N., Kumar, R., Song, H. F., Siegel, N. Y., Wang, L., Creswell, A., Irving, G., and Higgins, I. 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.

Valiant, L. G. 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.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Wäldchen, S., Sharma, K., Turan, B., Zimmer, M., and Pokutta, S. Interpretability guarantees with Merlin-Arthur classifiers. In Dasgupta, S., Mandt, S., and Li, Y. (eds.), *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, pp. 1963–1971. PMLR, 02–04 May 2024. URL https://proceedings.mlr.press/v238/waldchen24a.html.

Wang, B., Yue, X., and Sun, H. Can chatgpt defend its belief in truth? evaluating LLM reasoning via debate. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pp. 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.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *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.

Welleck, S., Liu, J., Lu, X., Hajishirzi, H., and Choi, Y. Naturalprover: Grounded mathematical proof generation with language models. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *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.

Yang, K., Swope, A. M., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R. J., and Anandkumar, A. Leandojo: Theorem proving with retrieval-augmented language models. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *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.

Yang, M., Schuurmans, D., Abbeel, P., and Nachum, O. Chain of thought imitation with procedure cloning. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *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.

Zhang, J., Liu, T., Wang, W., Zhang, Y., Song, D., Xie, X., and Zhang, Y. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In Kim, Y., Kim, J., Vigna, G., and Shi, E. (eds.), *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pp. 159–177. ACM, 2021. doi: 10.1145/3460120.3484767. URL https://doi.org/10.1145/3460120.3484767.

A. 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 2.2). In this sense, Self-Proving models generate a formal proof of the correctness of their output. Several works propose specialized models that generate formal proofs.

AlphaGeometry (Trinh et al., 2024) is capable of formally proving olympiad-level geometry problems; Gransden et al. (2015); Polu & 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 3) can be viewed as training the model on a Chain-of-Thought induced by the interaction of a verifier and an honest prover (Definition 2.2).

To complete the analogy, let us adopt the terminology of Uesato et al. (2022), who consider *outcome supervision* and *process supervision*. In our case, the *outcome* is the decision of the verifier, and the *process* is the interaction between the verifier and the model. Thus, Reinforcement Learning from Verifier Feedback (RLVF, Appendix C) 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 4 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 & 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 (AR) 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).

B. Theoretical analyses for Section 3

Before we can prove Theorem 3.1, we must first fully specify the theoretical framework in which our results reside. Continuing from Section 2, 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 B.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 3) 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.

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

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 B.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 $L_a = |a_i|$.[7] 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}.[8]

We assume that access to the verifier specification (Definition B.1) includes access to the query generator. After all, the verifier—who is assumed to be efficient—sampled 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.

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 B.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 $\pi = (y, q_1, a_1, \dots, q_R, a_R) \in \Sigma^*$. For any input x, $\mathcal{T}_V(x)$ satisfies that for each $r \leq R$, the marginal of $\mathcal{T}_V(x)$ on the r^{th} query (q_r) agrees with the corresponding marginal of the query generator $(V_q)_r$.

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.[9]

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 B.3 to emphasize that transcripts can be "collected" independently of the honest prover (see completeness in Definition 2.2). As long as the generator is fully supported on honest transcripts, it can be used for Transcript Learning, described next (TL, Algorithm 1).

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

[8] 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 token $\perp \in \Sigma$.

[9] WLOG we can assume that the prover sends her final answer a_R, the verifier's decision is deterministic.

Algorithm 1 Transcript Learning (TL)

```
1: **Hyperparameters:** Learning rate $\lambda \in (0,1)$ and number of samples $N \in \mathbb{N}$
2: **Input:** An autoregressive model family $\{P_\theta\}_{\theta \in \mathbb{R}^d}$, verifier specification (code) $V$, and sample access to an input distribution $\mu$ and an accepting transcript generator $\mathcal{T}^*_V(\cdot)$
3: **Output:** A vector of parameters $\bar{\theta} \in \mathbb{R}^d$
4: Initialize $\theta_0 := \vec{0}$
5: for $i = 0$ to $N-1$ do
6:     Sample $x \sim \mu$ and $\pi^* = (y, q_1, a_1, \ldots, q_R, a_R) \sim \mathcal{T}^*_V(x)$. Denote $a_0 := y$
7:     for each $r = 0$ to $R$ do
8:         Let $S(r)$ denote the indices of the r-th answer $a_r$ in $\pi^*$
9:         for each $s \in S(r)$ do
10:            Compute                                    # Forwards and backwards pass
               $$\alpha_s(\theta_i) := \Pr_{\sigma \sim p_{\theta_i}(x\pi_{<s})}[\sigma = \pi_s]$$
               $$\vec{d}_s(\theta_i) := \nabla_\theta \alpha_s(\theta_i) = \nabla_\theta \log \Pr_{\sigma \sim p_{\theta_i}(x\pi_{<s})}[\sigma = \pi_s]$$
11:        end for
12:        if $r \geq 1$ then
13:            Let $q_r$ denote the r-th query $q_r$ in $\pi^*$, and let $t$ denote its first index. That is, $\pi^*_{<t} = (y, q_1, a_1, \ldots, q_{t-1}, a_{t-1})$
14:            Compute                                    # Emulate the verifier
               $$\beta_r(\theta_i) := \Pr_{q' \sim V_q(x\pi^*_{<t})}[q' = q]$$
15:        end if
16:    end for
17:    Update
       $$\theta_{i+1} := \theta_i + \lambda \cdot \alpha_0(\theta_i) \cdot \left( \prod_{\substack{r \in [R] \\ s \in S(r)}} \beta_r(\theta_i)\alpha_s(\theta_i) \right) \cdot \sum_{\substack{r \in [R]\cup\{0\} \\ s \in S(r)}} \vec{d}_s(\theta_i)$$
18: end for
19: 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 the gradient of the Verifiability 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 that this function is differentiable.

Lemma B.4. *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 $\mathcal{T}^*_V$. For any model P_θ, it holds that*

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

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

Note that Lemma B.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 B.4, from which we derive Theorem 3.1.

Proof. Throughout this proof, expectations and probabilities will be over the same distributions as in the lemma statement. First,

$$\Pr_{x,\pi^*,\pi}[\pi = \pi^*] = \mathop{\mathbb{E}}_{x,\pi^*} \Pr_{\pi}[\pi = \pi^*],$$

and so, by the linearity of the gradient,

$$\nabla_\theta \Pr_{x,\pi^*,\pi}[\pi = \pi^*] = \mathop{\mathbb{E}}_{x,\pi^*} \nabla_\theta \left(\Pr_{\pi}[\pi = \pi^*] \right). \tag{3}$$

The probability that the output of V and P_θ on input x is equal to a given transcript is (by the law of total probability) the product of probabilities that each of the tokens of the transcript is equal to the corresponding token of the given transcript, both tokens generated by V's queries and by P_θ's answers, when conditioning on the prefix of the transcript.

Formally, consider any fixed $\pi^* = (y^*, q_1^*, a_1^*, \ldots, q_R^*, a_R^*)$ and denote the random variable $\pi = (y, q_1, a_1, \ldots, q_R, a_R)$. For any $r \in [R]$ denote the random variables $V_q^{<r} := V_q(y, q_1, a_1, \ldots, q_{r-1}, a_{r-1})$ and $\mathcal{T}_V^{\theta,<r} := \mathcal{T}_V^\theta(y q_1 a_1 \cdots a_{r-1} q_r)$. Then,

$$\begin{aligned}
\Pr_{\pi}[\pi = \pi^*] &:= \Pr_{\pi}[(y^*, q_1^*, a_1^*, \ldots, q_R^*, a_R^*) = (y, q_1, a_1, \ldots, q_R, a_R)] && (4)\\
&= \Pr_{y \sim P_\theta(x)}[y = y^*] \cdot \prod_{r \in [R]} \Pr_{q \sim V_q^{<r}}[q = q_r^*] \cdot \Pr_{a \sim \mathcal{T}_V^{\theta,<r}}[a = a_r^*] \\
&= \Pr_{y \sim P_\theta(x)}[y = y^*] \cdot \prod_{\substack{r \in [R] \\ s \in S(r)}} \Pr_{q \sim V_q^{<r}}[q = q_r^*] \cdot \Pr_{\sigma \sim p_\theta(\pi^*_{<s})}[\sigma = \pi_s^*] && (5)\\
&= \alpha_0(\theta) \cdot \left(\prod_{\substack{r \in [R] \\ s \in S(r)}} \beta_r(\theta) \cdot \alpha_s(\theta) \right), && (6)
\end{aligned}$$

where Equation (4) uses independence of the verifier and models' randomness, Equation (5) uses the autoregressive property of P_θ (Definition B.1), and Equation (6) is by definition of α_s and β_r.

Next, a basic calculus identity gives

$$\nabla_\theta \left(\Pr_{\pi}[\pi = \pi^*] \right) = \Pr_{\pi}[\pi = \pi^*] \cdot \nabla_\theta \log \left(\Pr_{\pi}[\pi = \pi^*] \right). \tag{7}$$

Let us focus on the last term. By Equation (6),

$$\begin{aligned}
\nabla_\theta \log \left(\Pr_{\pi}[\pi = \pi^*] \right) &= \nabla_\theta \log \alpha_0(\theta) \cdot \left(\prod_{\substack{r \in [R] \\ s \in S(r)}} \beta_r(\theta) \cdot \alpha_s(\theta) \right) \\
&= \nabla \log_\theta \alpha_0(\theta) + \sum_{\substack{r \in [R] \\ s \in S(r)}} \nabla_\theta \log \beta_r(\theta) + \nabla_\theta \log_\theta \alpha_s(\theta) \\
&= \nabla \log_\theta \alpha_0(\theta) + \sum_{\substack{r \in [R] \\ s \in S(r)}} \nabla_\theta \log_\theta \alpha_s(\theta) && (8)\\
&= \sum_{\substack{r \in [R] \cup \{0\} \\ s \in S(r)}} \nabla_\theta \log_\theta \alpha_s(\theta) = \sum_{\substack{r \in [R] \cup \{0\} \\ s \in S(r)}} \vec{d_s}(\theta) && (9)
\end{aligned}$$

where Equation (8) 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 (9) is by definition of $\vec{d}_s(\theta)$.

Combining Equations (6), (7) and (9) concludes the proof. □

We are now ready to prove Theorem 3.1. We restate it below in full formality.

Theorem B.5. *[Theorem 3.1, 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 $\mathcal{T}^*_V$ such that the expected agreement*

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

is convex in θ. For any $\varepsilon > 0$, we define

$$B_\varepsilon := \min\left\{||\theta^*|| : \mathrm{agree}_{\mathcal{T}^*_V}(\theta^*) \geq 1 - \varepsilon/2\right\}$$
$$\rho_\varepsilon := \max\left\{||\nabla_\theta \log p_\theta(z)|| : z \in \Sigma^*, ||\theta|| \leq B_\varepsilon\right\}$$

For any $\varepsilon > 0$ such that B_ε and ρ_ε are both finite, we denote by $\bar\theta$ the output of TL (Algorithm 1) running for number of iterations

$$N \geq \frac{4R^2 \cdot (L_a+1)^2 \cdot \rho_\varepsilon^2 \cdot B_\varepsilon^2}{\epsilon^2} \tag{10}$$

and learning rate $\lambda = \varepsilon/2R^2L_a^2\rho_\varepsilon$. Then the expected Verifiability (over the randomness of the samples collected by TL) is at least $1-\varepsilon$. That is,

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

Proof. Our strategy is to cast TL as Stochastic Gradient Ascent (SGD). We follow Shalev-Shwartz & Ben-David (2014, Section 14.3), which is presented for Descent (SGD) but is equivalent up to sign change.

Let ε such that B_ε and ρ_ε are finite be given. Since $B_\varepsilon < \infty$, let θ^* be such that $\mathrm{ver}_{V,\mu}(\theta^*) \geq 1 - \varepsilon/2$ and $||\theta^*|| \leq B_\varepsilon$. To prove the theorem, it suffices to prove that

$$\mathbb{E}[\mathrm{ver}_{V,\mu}(\bar\theta)] \geq \mathrm{ver}_{V,\mu}(\theta^*) - \varepsilon/2.$$

Following the notation in Algorithm 1, for every iteration $i \in [N]$, $r \in [R] \cup \{0\}$ and $s \in S(r)$, the definition of ρ_ε gives $||\vec{d}_s(\theta_i)|| \leq \rho_\varepsilon$. Thus, for each $i \in [N]$, we can bound the norm of the update step by

$$\begin{aligned}
&\left\| a_0(\theta_i) \cdot \left(\prod_{\substack{r\in[R] \\ s\in S(r)}} b_r(\theta_i) a_s(\theta_i) \right) \cdot \sum_{\substack{r\in[R]\cup\{0\} \\ s\in S(r)}} \vec{d}_s(\theta_i) \right\| \\
&\qquad = \left| a_0(\theta_i) \cdot \left(\prod_{\substack{r\in[R] \\ s\in S(r)}} b_r(\theta_i) a_s(\theta_i) \right) \right| \cdot \left\| \sum_{\substack{r\in[R]\cup\{0\} \\ s\in S(r)}} \vec{d}_s(\theta_i) \right\| \\
&\qquad\qquad \leq (R+1) \cdot L_a + \sum_{\substack{r\in[R]\cup\{0\} \\ s\in S(r)}} \left\| \vec{d}_s(\theta_i) \right\| \leq (R+1)\cdot(L_a + \rho_\varepsilon).
\end{aligned}$$

For the above, we used the fact that $\alpha_s(\theta_i), \beta_r(\theta_i) \leq 1$, the definition of the answer length L_a ($|S(r)| = L_a$), and the triangle inequality on $||\cdot||$. Therefore, by Theorem 14.8 of Shalev-Shwartz & Ben-David (2014) and Lemma B.4, TL (Algorithm 1) satisfies

$$\mathop{\mathbb{E}}_{\bar\theta}\left[\mathrm{agree}_{\mathcal{T}^*_V}(\bar\theta)\right] \geq \mathrm{agree}(\theta^*) - \varepsilon/2 \geq 1 - \varepsilon,$$

where the right inequality is by the choice of θ^*. The proof follows by observing that, for any $\bar{\theta}$ (and in particular an expected one), it holds that $\text{agree}_{\mathcal{T}_V^*}(\bar{\theta}) \leq \text{ver}_{V,\mu}(\bar{\theta})$; this is because, for any x, whenever the transcript generated by $\mathcal{T}^\theta(x)$ agrees with π^*, then the verifier accepts (by definition of π^*).

□

C. Reinforcement Learning from Verifier Feedback (RLVF)

As mentioned in Section 3, Transcript Learning relies on access to an honest transcript generator to estimate gradients of (a lower bound on) the Verifiability of a model P_θ. Our new notion of *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. It 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).

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 suggests 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).

RLVF is described in Algorithm 2, and is a learning algorithm under the model of Definition B.1. Before we proceed with its analysis, let us make a few observations.

Firstly, the parameters are updated (line 15) only when an accepting transcript was generated. This means that the learner can first fully generate the transcript (lines 7-10), and then take backwards passes (line 12) 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 B.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 B.4 for TL.

Lemma C.1. *Fix an input distribution μ over Σ^* and a verifier V with round complexity R and answer length L_a. For any transcript $(x, y, q_1, \dots, a_R)$ we let $\text{Acc}_V(x, y, q_1, \dots, a_R)$ denote the indicator random variable which equals 1 if and only if V accepts the transcript. For any model P_θ, denoting by $\text{ver}_{V,\mu}(\theta)$ the verifiability of P_θ (Definition 2.3), it holds that*

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

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 B.4). By the definitions of Verifiability in

Algorithm 2 Reinforcement Learning from Verifier Feedback (RLVF)

```
1:  Hyperparameters: Learning rate $\lambda \in (0,1)$ and number of samples $N \in \mathbb{N}$
2:  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 $\mu$.
3:  Output: A vector of parameters $\bar{\theta} \in \mathbb{R}^d$
4:  for $i = 0$ to $N - 1$ do
5:      Sample $x \sim \mu$.
6:      Initialize $a_0 := y$ and $d_i := \vec{0}$.
7:      for each $r = 1$ to $R$ do
8:          Sample the $r^{\text{th}}$ query                                # Emulate the verifier
                $q_r \sim V_q(x, a_0, q_1, a_1, \dots, q_r, a_r)$.
9:          Sample the $r^{\text{th}}$ answer                               # Forwards pass
                $a_r \sim P_\theta(x, a_0, q_1, a_1, \dots, q_r, a_r, q_{a_{r+1}})$.
10:         Let $\tau_r := (a_0, q_1, \dots, a_{r-1}, q_r)$.
11:         for each $s \in [L_a]$ do
12:             Let $a_{r,s}$ denote the $s^{\text{th}}$ token in $a_r$. Compute    # Backwards pass
                $\vec{d}_s(\theta_i) := \nabla_\theta \log \Pr_{\sigma\sim p_{\theta_i}(x\tau_r)}[\sigma = a_{r,s}]$.
13:         end for
14:     end for
15:     if $V(x, y, q_1, a_1, \dots, q_R, a_R)$ accepts then
16:         Update
                $\theta_{i+1} := \theta_i + \lambda \cdot \sum_{\substack{r\in[R]\cup\{0\} \\ s\in[L_a]}} \vec{d}_s(\theta_i)$.
17:     end if
18: end for
19: Output $\bar{\theta} := \frac{1}{N}\sum_{i\in[N]} \theta_i$.
```

Definition 2.3 and $V(x, y, q_1, \dots, a_R)$ in the lemma statement,

$$\begin{aligned} \mathrm{ver}_{V,\mu}(\theta) &:= \Pr_{\substack{x\sim\mu \\ y\sim P_\theta(x)}}\left[V^{P_\theta}(x,y) \text{ accepts}\right] \\ &= \mathop{\mathbb{E}}_{\substack{x\sim\mu \\ y\sim P_\theta(x) \\ (q_r,a_r)_{r=1}^R}} \left[\mathrm{Acc}_V(x, y, q_1, \dots, a_R)\right] \\ &= \mathop{\mathbb{E}}_{x\sim\mu}\left[\Pr_{\pi\sim\mathcal{T}_V^\theta}[\mathrm{Acc}_V(x,\pi)\,]\right] \end{aligned} \tag{11}$$

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

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

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

$$\Pr_{\pi\sim\mathcal{T}_V^\theta}[\mathrm{Acc}_V(x,\pi)] = \sum_{\pi^*\in\Pi^*(x)} \Pr_{\pi\sim\mathcal{T}_V^\theta(x)}[\pi = \pi^*]. \tag{12}$$

We will use Equations (4) through (9) in the proof of Lemma B.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 (11) and (12), by linearity of expectation we have that

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

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 $\mathcal{T}^\theta(x)$. □

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 it for future work to use Lemma C.1 to obtain convergence bounds on RLVF (analogous to Theorem B.5).

D. Annotations

We formally capture the modification described in Section 3.1 by introducing a *transcript annotator* and an *answer extractor* incorporated into the training and inference stages, respectively.

Fix a verifier V in an R-round proof system with question length L_q and answer length L_a. An *annotation system* with annotation length $\widetilde{L_a}$ consists of a *transcript annotator* A, and an *answer extractor* E.

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

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

- 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 $a_r \coloneqq E(\widetilde{a_r})$ which is sent to the verifier.

E. 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.

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

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

In this proof system the Verifier does not need to make any query; to fit within Definition 2.2, we can have the verifier issue a dummy query. Furthermore, by Fact E.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.

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 3.[10]

Algorithm 3 Extended Euclidean algorithm

```
1: Input: Nonzero integers $x_0, x_1 \in \mathbb{N}$
2: Output: Integers $(y, z_0, z_1)$, such that $y = \text{GCD}(x_0, x_1)$ and $(z_0, z_1)$ are Bézout coefficients for $(x_0, x_1)$
3: Initialize $r_0 = x_0$, $r_1 = x_1$, $s_0 = 1$, $s_1 = 0$, and $q = 0$
4: while $r_1 \neq 0$ do
5:     Update $q \coloneqq (r_0 // r_1)$, where // denotes integer division
6:     Update $(r_0, r_1) \coloneqq (r_1, r_0 - q \times r_1)$
7:     Update $(s_0, s_1) \coloneqq (s_1, s_0 - q \times s_1)$
8: end while
9: Output GCD $y = r_0$ and Bézout coefficients $z_0 \coloneqq s_0$ and $z_1 \coloneqq (r_0 - s_0 \cdot x_0)/x_1$
```

Referring to Algorithm 3, 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 (step 4), the algorithm stores each of r_0, s_0 and q in an arrays $\vec{r_0}$, $\vec{s_0}$ and $\vec{q}$. The annotation $\tilde{z}$ of z is obtained by concatenating each of these arrays. In practice, to avoid the transformer block (context) size from growing too large, we fix a cutoff T and first trim each array to its first T elements.

We formalize this in the terminology of Appendix D 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 $z = (z_0, z_1)$.[11]

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

[11] Such a proof system is known as an NP-proof system and defines the complexity class NP (Karp, 1972).

- *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 3 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 3 (up to additional space due to storing intermediate values). As for E, it can be implemented in logarithmic space and linear running time in $|\tilde{z}|$, i.e., the length of the description.[12]

F. Experiment details

We provide details of how we implemented the experiments in Section 4 and additional figures for each experiment.

Model architecture. We use Karpathy's *nanoGPT*.[13] 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 $\beta 2 = 0.95$, 10% learning rate decay factor, no dropout, gradient clipping at 2.0, no warmup iterations, and 10% weight decay.

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

Compute. All experiments were run on a machine with an NVIDIA A10G GPU, 64GB of RAM, and 32 CPU cores. 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.

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.

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

As an example, consider the inputs $x_0 = 46$ and $x_1 = 39$. Tracing through the execution of Algorithm 3, 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.

G. Additional figures for Section 4

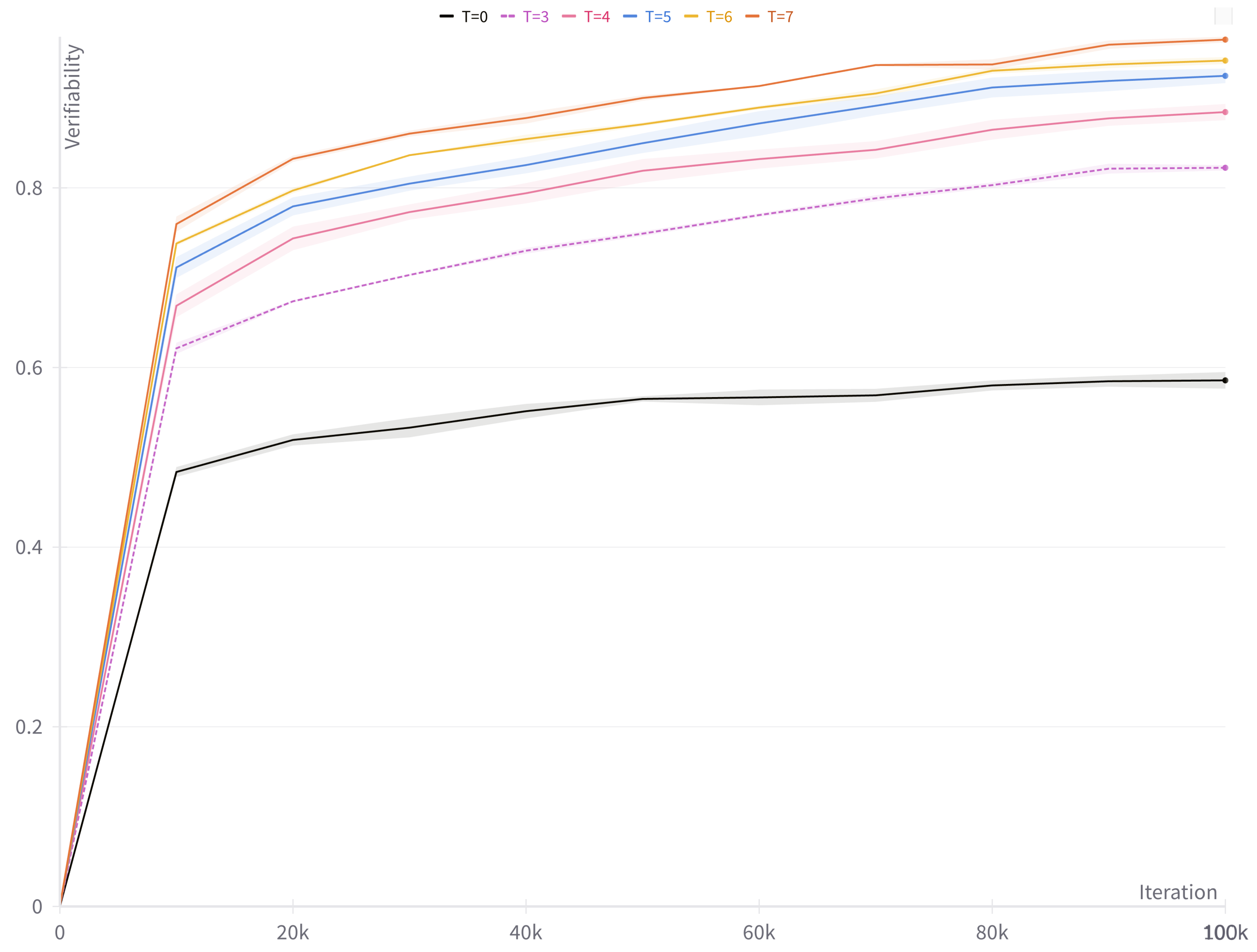

Figure 3. **Verifiability as a function of the number of samples** N. Each iteration (X axis) is a batch of 1024 samples from a dataset of ≈10M sequences. Every 10k iterations, Verifiability was evaluated on a held-out dataset of 1k inputs (as described in Section 4). T is the number of steps in Annotated Transcript Learning (Section 4), 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.

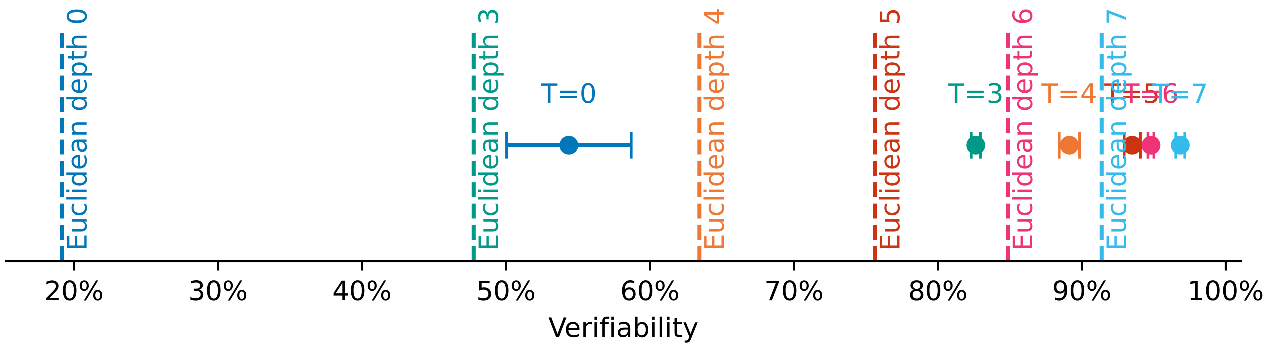

Figure 4. **Full version of Section 4**, with all $T \in \{0, 3, 4, 5, 6, 7\}$ possibilities for the annotation cutoff in Annotated Transcript Learning ($T = 0$ means no annotation). For a zoom-in of the right end of the axis, see Section 4.