Lecture 13: Self-proving LLM (contd.)

Lecturer: Orr Paradise **Date:** March 19, 2026
Scribes: Kevin Zhao, Yifan Kang

Contents

1. Defining Self-Proving models

Definition 1.1. P_θ is a *Self-Proving Model with error* ε (i.e. P_θ is *Self-Proving*) w.r.t. a verifier V and data distribution μ, if $\Pr[P_\theta \text{ convinces } V \text{ to accept } y] \geq 1 - \varepsilon$, where the randomness is over $x \sim \mu, y \sim P_\theta(x)$, and the questions and answers $q_1, a_1, \ldots, q_R, a_R$, with the questions q_i coming from the verifier and answers a_i generated by P_θ.

Exercise 1.1. Soundness and Self-Proving imply Correctness.

For a fixed verifier V for f^* with soundness[1] error δ, if P is Self-Proving with error ε w.r.t V and μ, then

$$\Pr_{x\sim\mu}[P_\theta(x) = f^*(x)] \geq 1 - \delta - \varepsilon$$

Proof. Suppose for contradiction that $\Pr_{x\sim\mu}[y = f^*(x)] < 1 - \delta - \varepsilon$. Then

$$\begin{aligned}\Pr_{x\sim\mu}[P \text{ convinces } V \text{ to accept } y] &= \Pr_{x\sim\mu}[y = f^*(x)] + \Pr_{x\sim\mu}[y \neq f^*(x) \text{ but } V \text{ incorrectly accepts}] \\ &< (1 - \delta - \varepsilon) + \delta \\ &= 1 - \varepsilon\end{aligned}$$

which contradicts that P is Self-Proving with error ε. □

Our goal

Learn θ such that P_θ is Self-Proving.

2. Prerequisites for Learning Self-Proving models

To learn a Self-Proving model w.r.t V, μ, we need:

- Fix and assume access to the input distribution μ.
- Fix and assume access to the verifier V.
- A sequence-to-sequence model family: $\{P_\theta : \Sigma^* \to \Sigma^*\}_{\theta\in\Theta}$ where the parameter space $\Theta = \mathbb{R}^d$.
  - Σ is a finite set of tokens, sometimes called the "vocabulary" of the model family
  - Σ^* is a string/sequence of tokens
  - d is the number of parameters
- The model family should allow autoregressive generation. In other words, for any parameter setting θ and prompt $z = (z_1, z_2, \ldots z_m) \in \Sigma^*$,
  1. For a prefix $(z_1, z_2, \ldots z_n)$, P_θ predicts a probability distribution over Σ for the next token z_{n+1}. We denote this distribution as $P_\theta(\cdot \mid z_1, \ldots, z_n)$.[2] This process of using P_θ to get probabilities for the next token is known as the "*forwards pass*."
  2. Sample the next token $z_{n+1} \sim P_\theta(\cdot \mid z_1, \ldots, z_n)$.

[1] Definition 10.1

[2] In practice, an intermediate step in the neural network P_θ is to produce *logits*, which are unnormalized probabilities in $\mathbb{R}^{|\Sigma|}$. The logits are then normalized, most commonly with the softmax function.

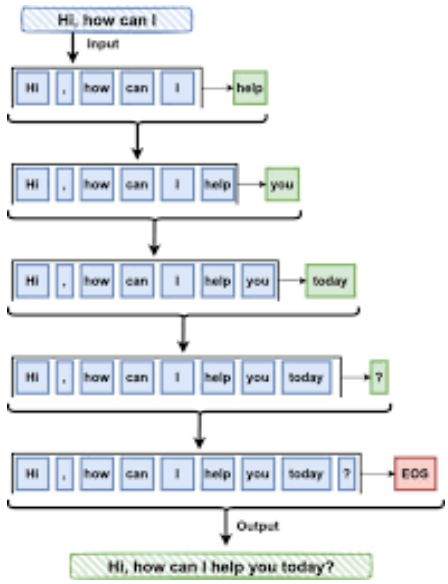


Figure 1: Example of Autoregressive Generation, taken from [3].

3. Repeat the previous two steps but with z_{n+1} appended onto the end of the prefix, and continue generating until a special token EOS $\in \Sigma$ is sampled.

Fig. 1 shows an example of autoregressive generation.

- Differentiability: Can compute $\nabla_\theta \log P_\theta(\sigma \mid z_1, \ldots, z_n)$ for each $\sigma \in \Sigma$. The process of computing this gradient is known as the "*backwards pass*."
  - For example, if we want 1 to follow the sequence 0100_, differentiability tells us how to change the parameters to increase the probability $P_\theta(1 \mid 0, 1, 0, 0)$.
- For Transcript Learning (Section 3), we assume also access to an "honest prover"
  - The slightly weaker assumption that that we have access to an "honest transcript generator" also suffices.

Question: Why would we to even learn a self-proving model if we already assume access to an "honest prover"?

Answer: Efficiency. LLM inference can be cheaper than running the "honest prover," at the expense of a one-time cost for transcript generation and training. For intuition on why the LLM may be more efficient, note that the "honest prover" has the harder task of being able to answer every x, including the most difficult ones, while the LLM just has to answer $x \sim \mu$, which is often a simpler or more natural subset of questions.

3. Transcript Learning

1. Transcribe "honest" interactions.

- By definition of being an "honest prover" (specifically, completeness), we know it should convince the verifier to accept.
- Generate N of these interactions, i.e. $(x^1, y^1, q_1^1, a_1^1, \ldots, q_R^1, a_R^1)$.
- Can throw away "honest prover" after this.

2. Learn the autoregressive P_θ with Transcript Cloning:

Algorithm 1: Transcript Cloning

Input: N transcriptions of honest interactions
$(x^1, y^1, q_1^1, a_1^1, \ldots, q_R^1, a_R^1), \ldots, (x^N, y^N, q_1^N, a_1^N, \ldots, q_R^N, a_R^N)$

Output: Parameters θ corresponding to a learned prover P_θ

for *each transcript* $(x^i, y^i, q_1^i, a_1^i, \ldots, q_R^i, a_R^i)$ **do**
  Tokenize the transcript $(x^i, y^i, q_1^i, a_1^i, \ldots, q_R^i, a_R^i)$ into a sequence of tokens $(z_1^*, z_2^*, \ldots, z_T^*)$.
  Let $S(i) \subseteq [T]$ be the set of indices where $j \in S(i) \iff z_j$ is a token in y or in an answer $a_1^i, a_2^i, \ldots, a_R^i$.[a]
  for *each* $j \in S(i)$ **do**
    Forwards pass to compute the probability distribution over Σ of z_j: $P_\theta(\cdot \mid z_1^*, \ldots, z_{j-1}^*)$.
    Backwards pass to compute the gradient $\vec{d}_j := \nabla_\theta P_\theta(z_j^* \mid z_1^*, \ldots, z_{j-1}^*)$.
  end
  $\theta \leftarrow \theta + \lambda \cdot \left(\prod_{j=1}^{T} P_\theta(z_j^* \mid z_1^*, \ldots, z_{j-1}^*)\right) \cdot \sum_{j \in S(i)} \vec{d}_j$
end
return θ;

[a] At inference, P_θ will only generate the tokens indexed by $S(i)$, so these are the only tokens we want θ to learn to generate.

Note that λ in the update rule is a tunable parameter called the "learning rate" as it controls the rate at which θ is updated.

Theorem 3.1. *Transcript Learning Theorem: Under the below assumptions, Transcript Learning outputs a* $(1-\epsilon)$*-Self-Proving model when trained on*

$$N \geq 4\left(C \cdot B_1 \cdot B_2 \cdot \frac{1}{\epsilon}\right)^2 \tag{1}$$

samples.

Assumptions:

- *Access to a dataset of honest transcripts.*
- *The total number of tokens sent by the prover in any interaction is* $< C$.[3]

[3] The efficiency (Definition 10.2) of the verifier means the honest prover can just send a polynomial number of tokens.

- *The surrogate objective* $A(\theta) := \Pr_x[\pi_\theta(x) = \pi^*(x) \mid V\text{'s randomness}]$ *is concave and differentiable in* θ*, where* $\pi^*(x)$ *is the ground truth transcript generated by the "honest prover" and* $\pi_\theta(x)$ *is the distribution of interactions between* P_θ *and* V *on* x*.*
  - *The surrogate objective lower bounds the self-provability of a model (i.e.* $\Pr_x[P$ *convinces* V *to accept* $y]$*), as we know that transcripts generated by the "honest prover" are always accepted by the verifier.*
- *The logits of* P_θ *are* B_1*-Lipschitz in* θ*.*
- *For* $\epsilon > 0$*, there exists a* B_2 *such that:*
  - *There exists* θ^* *with* $\|\theta^*\| < B_2$ *such that* $A(\theta^*) \geq 1 - \frac{\epsilon}{2}$*.*

(The last 3 assumptions are necessary for reasoning about NN convergence.)

4. RL from Verifier Feedback (RLVF)

In practice, people tend to do a bit of transcript learning, followed by RLVF. RLVF is powerful as it removes the requirement of "access to a dataset of honest transcripts." This is why in practice, people very rarely use an "honest prover" to generate transcripts, since it's expensive.

RLVR is the following procedure:

1. Generate a batch of transcripts with P_θ (as opposed to an "honest prover"). Keep only transcripts accepted by the verifier.
2. Update θ towards the accepted transcripts using transcript cloning.
3. Repeat steps 1-3, terminating when desired (e.g. P_θ achieves a certain metric).

Weakness of RLVR If P_θ is initially bad, then no transcripts will be accepted by the verifier and no learning happens. This is why people usually do a bit of transcript learning first, as that produces a reasonable P_θ that can learn to self-improve through RLVR.

5. Self-Proving models in practice

- "Learning to Prove" is a natural thing to do, e.g. what students do with each new math class.
- It has been explored in many settings:

5.1. The Landscape of $\texttt{IP} \cap \texttt{ML}$ (May 2024)

This section discusses related works up to May 2024.

- **Learning to Prove:** Models are being trained to generate formal proofs in various environments:
  - Coq [2]

  - Metamath [8]
  - Lean [5, 11]
  - Synthetic geometry [9]
- **Learning to Verify:** Shifting the focus to checking proofs:
  - Prover Verifier Games [7]
  - Neural Interactive Proofs [4]
- **Safety and Alignment:** Using proof-like structures for scalable AI safety:
  - Debate Systems for AI Safety [6]
  - Doubly-efficient debates for scalable AI safety [1]
- **Interpretability:** MA (Merlin-Arthur) Classifiers [10]
- **Verifying global accuracy**
  - Goldwasser et al. 2021; Mutreja and Shafer, 2023

Next, we will look into some specific papers as examples.

6. AI Safety via Debate (Irving et al., 2017)

- **The Goal:** To make AI systems broadly useful for real-world tasks, they must learn complex human goals and preferences.
- **The Bottleneck:** A common approach is to have humans judge agent behavior during training (to determine if it is safe and useful). However, this fails when tasks become too complicated for a human to directly and accurately evaluate.

6.1. The Proposed Solution: Zero-Sum Debate

To simplify the task of the human evaluator, the authors propose training agents via self-play in a **zero-sum debate game**:

1. The system is given a question or a proposed action.
2. Two AI agents take turns making short statements, up to a specified limit.
3. A human judge evaluates the debate and decides which agent provided the most true and useful information.

6.2. Theoretical Grounding: Complexity Theory Analogy

The debate model significantly expands the theoretical capacity of what a human judge can effectively evaluate:

- **Direct Judging:** Analogous to solving **NP** questions.

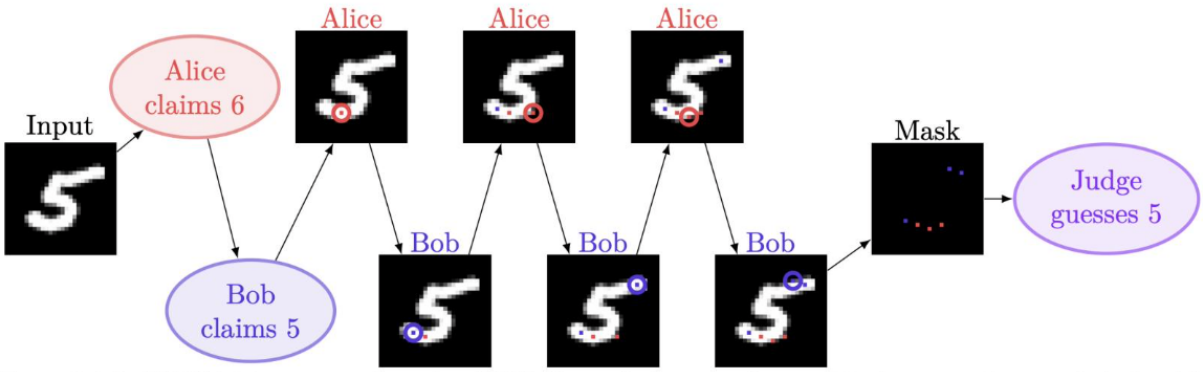


Figure 2: The MNIST debate game. A random MNIST image is shown to the two debating agents but not the judge. The debaters state their claimed label up front, then reveal one nonzero pixel per turn to the judge up to a total of 4 or 6. The judge sees the sparse mask of 4 or 6 pixels and chooses the winner based on which of the two labels has higher logit. The judge is trained in advance to recognize MNIST from random masks of nonzero pixels.

Figure 2: Graphic of the MNIST debate.

- **Optimal Play Debate:** Analogous to solving any question in **PSPACE**, assuming polynomial-time human judges.

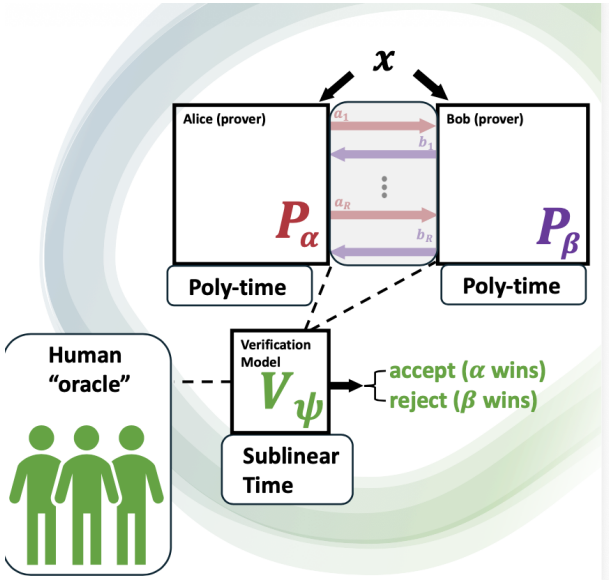
6.3. Empirical Evidence: MNIST Experiment

The authors tested the debate model on the task shown in Fig. 2 and explained in the caption. The two agents take turns to pick pixels to reveal to the judge, with the goal of to convincing the judge of their claim. Intuitively, since the agent's claim is that the true image is a certain digit, it should reveal the pixels that help convince the judge that the image is that digit.

While the judge could be a human, in this task the authors trained a sparse classifier on the MNIST dataset as the judge, where the judge never sees the whole image and instead only sees a tiny allowance of pixels, since that's the setting it would be used in for the debate game. The authors found that the baseline of randomly revealing pixels instead of using agents has a low accuracy of only 59.4% for the 6-pixel case. In contrast, by having the agents pick the pixels to reveal, the "honest" agent improved the probability of convincing the judge:

- **6-Pixel Test:** Classifier accuracy was boosted from **59.4%** to **88.9%**.
- **4-Pixel Test:** Classifier accuracy was boosted from **48.2%** to **85.2%**.

6.4. Doubly Efficient Debates



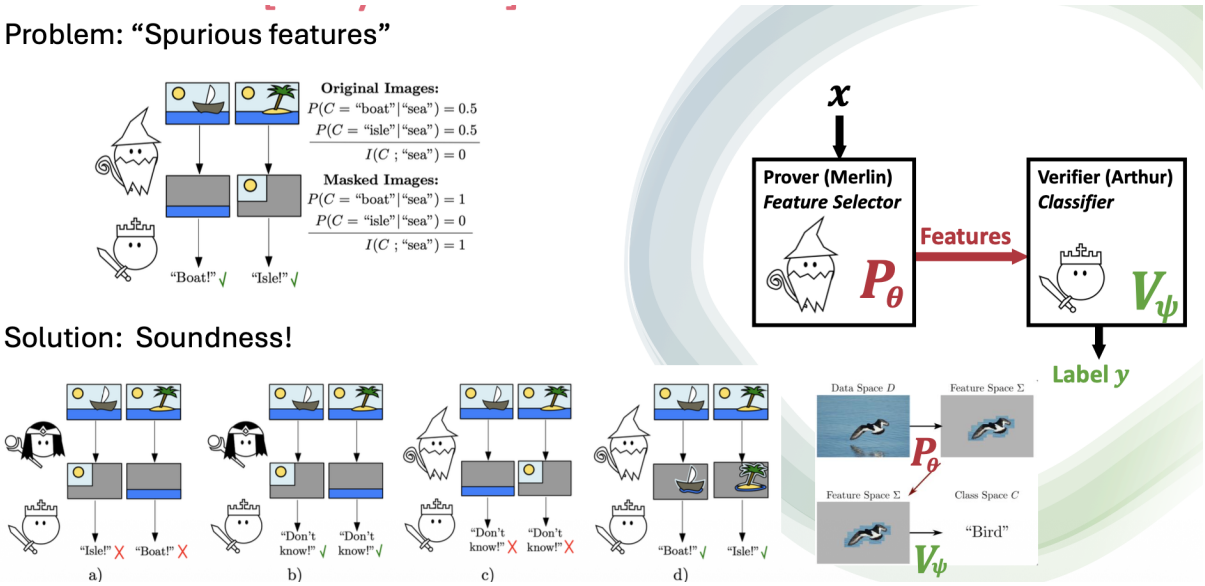
The system is "doubly efficient" because it scales practically on both ends: the AI generating the proofs (P_α, P_β) doesn't need infinite compute (it uses Poly-time), and the system checking the proofs (V_ψ) can do so almost instantly (Sublinear Time), leaning on humans only as an ultimate, high-level compass.

7. Interpretability via MA Classifiers (Wäldchen et al., 2022)

A significant vulnerability in standard deep learning classification is the reliance on **spurious features**. For example, a model might correctly classify an image as a "boat" not by recognizing the boat itself, but by detecting the "sea" in the background, leading to failures when a boat is on land or an island is in the sea.

To address this, Wäldchen et al. introduced **MA (Merlin-Arthur) Classifiers**, which leverage an interactive proof framework to guarantee feature soundness and provide inherent interpretability.

7.1. The Merlin-Arthur Architecture



The classification pipeline is split into a two-player game:

- **The Prover (Merlin, P_θ):** Acts as a *Feature Selector*. Given an original input x (e.g., an image of a bird on water), Merlin generates a mask to isolate specific features, hiding the rest of the data.
- **The Verifier (Arthur, V_ψ):** Acts as the *Classifier*. Arthur receives *only* the isolated features from Merlin and must output the label y (or abstain by saying "I don't know").

7.2. Enforcing Soundness

The core requirement for this system to work is that Arthur must be trained to be **sound**.

- If Merlin tries to cheat by only providing spurious background features (e.g., masking out the boat and only showing the sea), a sound Arthur will refuse to guess and output "Don't know!".
- If Merlin provides the true causal features (e.g., masking out the sea and showing only the boat), Arthur will confidently output the correct label.

8. Prover-Verifier Game for Legibility (Kirchner et al., 2024)

One way to increase confidence in the outputs of Large Language Models (LLMs) is to support them with reasoning that is clear and easy to check – a property we call legibility. Optimizing on CoT could result in a decrease in legibility.

To increase the human legibility of a large language model's reasoning, we introduce a smaller LLM to act as a Verifier. The main Prover LLM is fine-tuned to alternate between two roles: an

honest prover (which tries to produce correct, easily verifiable answers) and a sneaky prover (which tries to fool the verifier into accepting incorrect answers).

Models & Parameters

- ψ: The learnable parameters (weights) of the Verifier model.
- θ: The learnable parameters (weights) of the Prover model.
- V_ψ: The Verification Model (the smaller LLM acting as the judge).
- P_θ: The Honest Prover (trained to provide correct answers and legible proofs).
- P_θ^*: The Sneaky Prover (trained to provide incorrect answers but convincing, deceptive proofs).

Joint Training (Simplified)

Verifier Optimization:

$$\psi \underset{\text{argmin}}{\longleftarrow} L(\psi \mid \theta) := \ell(\psi \mid P_\theta) + \ell(\psi \mid P_\theta^*)$$
$$\ell(\psi \mid P) := -\mathbb{E}_x \left[H\left(\langle V_\psi, P \rangle(x), \mathbb{I}_{P(x)=y(x)} \right) \right]$$

Prover Optimization:

$$\theta \underset{\text{argmin}}{\longleftarrow} \mathbb{E}_x \left[\langle V_\psi, P_\theta \rangle(x) \cdot \mathbf{1}_{P_\theta(x)=y(x)} + \langle V_\psi, P_\theta^* \rangle(x) \cdot \mathbf{1}_{P_\theta^*(x)\neq y(x)} \right]$$

9. Transcript Learning: Ancient Experiments (Noga et al., 2024)

This is on the paper "Models That Prove Their Own Correctness".

Motivation: Past work have shown that small transformer can learn to compute GCD, can it prove that its answer is correct?

Instead of just asking the model to calculate the GCD of two integers (x_1 and x_2), the researchers required the model to act as a Prover. To prove that its proposed answer (d) is indeed the correct GCD, the model had to output the answer alongside two Bézout coefficients (z_1 and z_2). According to Bézout's identity, if d is the greatest common divisor of x_1 and x_2, then there exist integers z_1 and z_2 such that:

$$(z_1 \cdot x_1) + (z_2 \cdot x_2) = d$$

The verifier accepts if $d|x_1$ and $d|x_2$ and z_1, z_2, x_1, x_2, d satisfy the above identity.

The researchers evaluated different methods to train the Transformer to succeed at the interactive prover-verifier game:

- **Transcript Learning (TL):** (Application of Section 3) The model is trained autoregressively on a dataset of successful "transcripts"—step-by-step interactions between an honest prover and the verifier. The model learns to predict not just the answer, but the sequence of tokens that makes up a convincing proof.
- **Annotated Transcript Learning (ATL):** The authors discovered that adding intermediate mathematical steps (annotations) to the training transcripts drastically improves the model's ability to generate sound proofs. The annotations is generated by the intermediate values when computing the extended Euclidean algorithm.
- **Reinforcement Learning from Verifier Feedback (RLVF):** (Application of Section 4) The model is trained by having it generate proofs, submitting them to the deterministic verifier, and updating the model's parameters based on the verifier's accept/reject feedback.

10. Appendix

Definition 10.1. A verifier V with *soundness error* δ means that for any incorrect correct output $f_\theta(x) \neq f^*(x)$ and any "lying prover" P,

$$\Pr\left[P \text{ convinces } V \text{ to accept } (x, f_\theta(x))\right] \leq \delta.$$

Definition 10.2. A verifier V is *efficient* if there exists a constant $k \in N$ such that for any input x, V runs in time $O(|x|^k)$.

References

[1] Jonah Brown-Cohen, Geoffrey Irving, and Georgios Piliouras. Scalable ai safety via doubly-efficient debate. *arXiv preprint arXiv:2311.14125*, 2023.

[2] Thomas Gransden, Neil Walkinshaw, and Rajeev Raman. Sepia: search for proofs using inferred automata. In *International Conference on Automated Deduction*, pages 246–255. Springer, 2015.

[3] Ivan Gridin. *Improving Chat Model Responses*, pages 93–130. Apress, Berkeley, CA, 2025. ISBN 979-8-8688-2216-2. doi: 10.1007/979-8-8688-2216-2_3. URL https://doi.org/10.1007/979-8-8688-2216-2_3.

[4] Lewis Hammond and Sam Adam-Day. Neural interactive proofs. *arXiv preprint arXiv:2412.08897*, 2024.

[5] Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, pages 1–3, 2025.

[6] Geoffrey Irving, Paul Christiano, and Dario Amodei. Ai safety via debate. *arXiv preprint arXiv:1805.00899*, 2018.

[7] Jan Hendrik Kirchner, Yining Chen, Harri Edwards, Jan Leike, Nat McAleese, and Yuri Burda. Prover-verifier games improve legibility of llm outputs. *arXiv preprint arXiv:2407.13692*, 2024.

[8] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

[9] Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

[10] Stephan Wäldchen, Kartikey Sharma, Berkant Turan, Max Zimmer, and Sebastian Pokutta. Interpretability guarantees with merlin-arthur classifiers. In *International Conference on Artificial Intelligence and Statistics*, pages 1963–1971. PMLR, 2024.

[11] Kaiyu Yang, Aidan 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. *Advances in Neural Information Processing Systems*, 36:21573–21612, 2023.