Lecture 12: Self-Proving LLMs

Lecturers: Shafi Goldwasser, Orr Paradise
Scribes: Marcus Gozon, Megan Tian

Date: March 17, 2026

Contents

1. Summary

In the first part of lecture Shafi wrapped up some content from Lecture 10 on interactive and zero-knowledge proofs. In the second half (starting at Section 5), Orr started discussing self-proving LLMs (which was continued into Lecture 13).

2. Zero-Knowledge Proofs for NP Problems

Recall from lecture 9 we defined interactive proofs and honest-verifier/perfect/computational zero-knowledge proofs. Last time we stated the following result:

Theorem 2.1 ([6])**.** *If one-way functions exist, then every problem in* NP *has a computational zero-knowledge proof.*

Recall from last lecture that it suffices to show a zero-knowledge proof for any one NP-complete problem, since then any language in NP has a polynomial time reduction to it mapping yes

instances to yes instances and no instances to no instances, from which one can run the known zero-knowledge proof on the transformed instance. The completeness, soundness, and simulability of the known zero-knowledge proof system will then translate over to the new protocol. Last time, we saw a proof sketch of the theorem using the NP-hard Hamiltonian Path problem. Let us do the example for the NP-hard Graph 3-Coloring problem, which was the first example shown to have a computational zero-knowledge proof.

Recall that in the 3-coloring problem, we want to determine whether an input graph $G = (V, E)$ has a proper 3-coloring, i.e. an assignment of colors to the vertices $\alpha : V \to \{0, 1, 2\}$ so that the vertices across each edge have different colors. The idea for the zero-knowledge protocol is as follows: for the prover to show that a given graph G has a 3-coloring, they will first send over a coloring of the graph which is "locked" (i.e. the commitment which is encrypted) to the verifier. Then the verifier will choose a uniformly random edge to ask to unlock from the prover, who sends the key to open the boxes of the colors across an edge, from which the verifier accepts iff "opening" the box was successful and the colors are different.

This protocol is clearly complete by just using a true 3-coloring to send over, and for soundness, if the graph wasn't 3-colorable, then at least one of the edges will be monochromatic in the coloring sent over, and so the verifier would have at least a $1/|E|$ chance of selecting a violated edge and detect that the prover was lying. To boost the soundness, we just do this several times independently, e.g. $k = |E|^2$ times, and so the chance of not catching the prover would be at most $(1 - 1/|E|)^{|E|^2} \leq e^{-|E|}$. Note that we need to repeat the protocol independently to achieve this bound because if we kept revealing colors of the vertices, we would eventually be able to recover the coloring, violating the zero-knowledge property.

To get computational zero-knowledge, what we can do is that after each round of the protocol, the prover can permute the meaning of the 3 colors uniformly at random to prevent the verifier from learning what the valid coloring the prover used was. Thus, the simulator works as follows: color the graph with a fixed random color, e.g. green, and pick an edge uniformly at random and color its vertices randomly red and blue, encrypting this coloring and feeding it to the verifier. If it happened to choose the same edge that we specifically chose to be red/blue to uncover, then the verifier would accept. This would be a view that we can output and otherwise, we redo the entire process until it works.

Since the chance of choosing our chosen edge is at least $1/|E|$, the expected number of iterations it takes to output a coloring will be at most $|E|$. Since we need k independent instances, if we let the simulator run until it outputs this many transcripts, the total expected amount of time will be at most $k|E| = |E|^3$, which is polynomial in the input. This analysis works if the prover could provably "open" the safes of the colors it sends, and this can be done by using secure encryption schemes with computational indistinguishability so that the values look more or less random to any PPT algorithm.

This setup of locked safes is known as a **commitment scheme** which has the following structure:

- commit stage: the sender (prover in our case) has a private b to commit to. At the end of this stage, both parties have the commitment **com** while the sender has a private de-commitment **dec**.

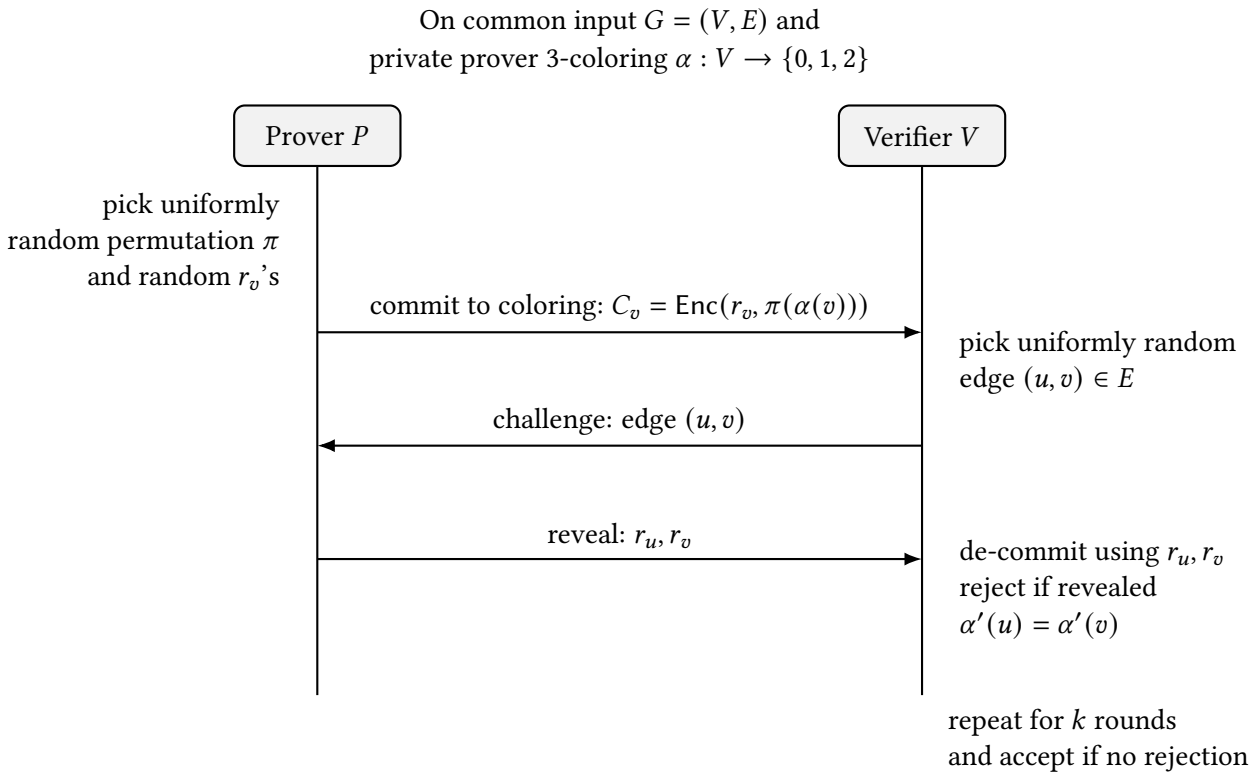


Figure 1: Zero-knowledge proof system for Graph 3-Coloring

- reveal stage: the sender gives (dec', b') to the receiver, who accepts or rejects their pair.

This should satisfy the **hiding property** that given com, the receiver should not be able to determine what the value b is. In our case of computational hiding, it should be computationally indistinguishable from a random b, i.e. for any PPT algorithm running using that it, we can replace it with true random bits while changing the output probability of the algorithm by a negligible function. This should also satisfy the **binding property** that the sender cannot later de-commit it to two different messages. For computational binding, no PPT cheating sender can produce a commitment that can later be opened to two different messages except with negligible probability. This can be implemented by sending a computationally secure $c = \mathsf{Enc}(r, b)$ to commit to b for r random, de-committing by revealing r.

For reference, the complete zero-knowledge protocol for graph 3-coloring is depicted above, and the simulation of any V^* for a single round is in Figure 2.

3. Applications of the Zero-Knowledge Theorem

Since all of NP has a computational zero-knowledge proof system, we can also consider a broader class of languages e.g.

$$L = \{\mathsf{Enc}(m_1), \ldots, \mathsf{Enc}(m_\ell), \text{program P}; m_\ell = P(m_1, \ldots, m_{\ell-1})\}$$

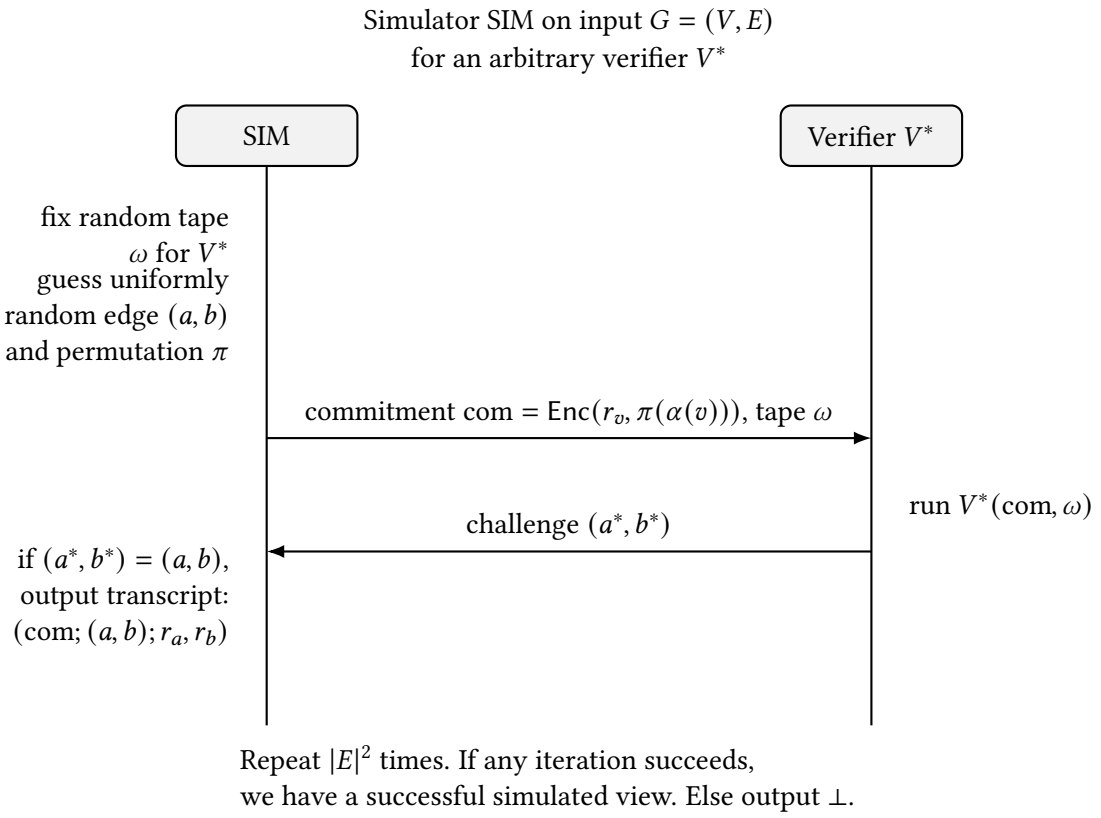


Figure 2: Zero-knowledge simulation for any verifier V^* for a single round. To get k instances for our full protocol, we run the simulator until k transcripts are produced.

for encrypted values $\mathsf{Enc}(m_i)$. This language is in NP since an NP machine can guess the values for m_i and check that their encoding matches $\mathsf{Enc}(m_i)$ before checking that $m_\ell = P(m_1, \ldots, m_{\ell-1})$, assuming that Enc and P are efficient. Thus, the theorem implies that there is a zero-knowledge proof system for checking whether the value of $P(m_1, \ldots, m_{\ell-1})$ is indeed m_ℓ without ever revealing the values of the m_i's.

One use case is a public auction: you don't want others to see your bid so that they can't just bid slightly above yours. Thus, a prover could get the encrypted bids, and assuming that they can decrypt it to check which is the highest bid, then there would be a corresponding efficient zero-knowledge proof system to convince the participants of who won the auction without revealing their bids. Any application you can capture in an NP language can be made into a corresponding zero-knowledge proof system by the above theorem.

Another example: suppose there was a crime and someone's DNA is at the scene. As checking whether DNA sequences can be done efficiently, there is a zero-knowledge protocol for which you can prove that your DNA is different from their DNA without revealing your own. In reality, the DNA can be treated as strings collected via sequencing from blood samples, so other physical processes will likely be involved to realize this theorem.

Numerous other applications exist in nuclear disarmament [5], forensics [4], cryptocurrency [2],

and FISA compliance [7].

4. Strength of IP over NP

Is IP bigger than NP? That is, does letting us allow for a polynomial round of interactions rather than one increase the class of problems we can solve? Indeed, it is the case and in fact IP = PSPACE, where PSPACE is the class of problems solvable in polynomial space (though possibly exponential time). This is a well-known complexity theory fact whose proof goes via a technique known as "arithmetization", showing that the PSPACE-complete problem TQBF has an interactive proof, with the other direction immediately following from the fact that an optimal prover strategy could just search over all possible ways it could interact, choosing the one the maximizes the verifiers acceptance probability.

For example, consider the Graph Non-Isomorphism problem: how can we prove that there is no isomorphism between two graphs G_0 and G_1? This appears hard to check with a classical proof, i.e. in NP, given that there are $n!$ possible permutations to check whether the graphs are isomorphic. However, there is a simple protocol for GNI to show that there is no correspondence from one to another: the verifier starts by choosing one of the graphs G_i at random, picking a random permutation $\sigma \in S_n$ and sending $H = \sigma(G_i)$ over. If the two graphs are not isomorphic, then the prover would be able to tell which graph was chosen and permuted and can predict the coin used to choose G_i, which gives completeness. In case the graphs are isomorphic, then both graphs are isomorphic to H and so the prover can't tell which graph was permuted and can guess correctly with probability at most $1/2$, which gives soundness. Note that this protocol can be repeated k times independently to boost the soundness down to $1/2^k$.

Is this protocol zero-knowledge? To an honest verifier, yes – it's easy to simulate the yes instance since the prover's reply could just be set to the verifier's coin. However, the protocol is not zero-knowledge against all verifiers since there is no knowing of what a dishonest verifier could do. Indeed, they could send an arbitrary graph H, and so the simulator would have to be able to reproduce the prover's answer on those queries, which amounts to a graph isomorphism query against G_0 or G_1. This would result in Graph-Isomorphism having an efficient algorithm, which is not known and is not what the protocol is intended to give. It is, however, possible to convert this into a zero-knowledge proof using "zero-knowledge proofs of knowledge" whereby the verifier would first prove in zero-knowledge that they know γ for which $H = \gamma(G_i)$ to prevent dishonest verifiers of the previous form and ultimately allow for an efficient simulator, giving the zero-knowledge property.

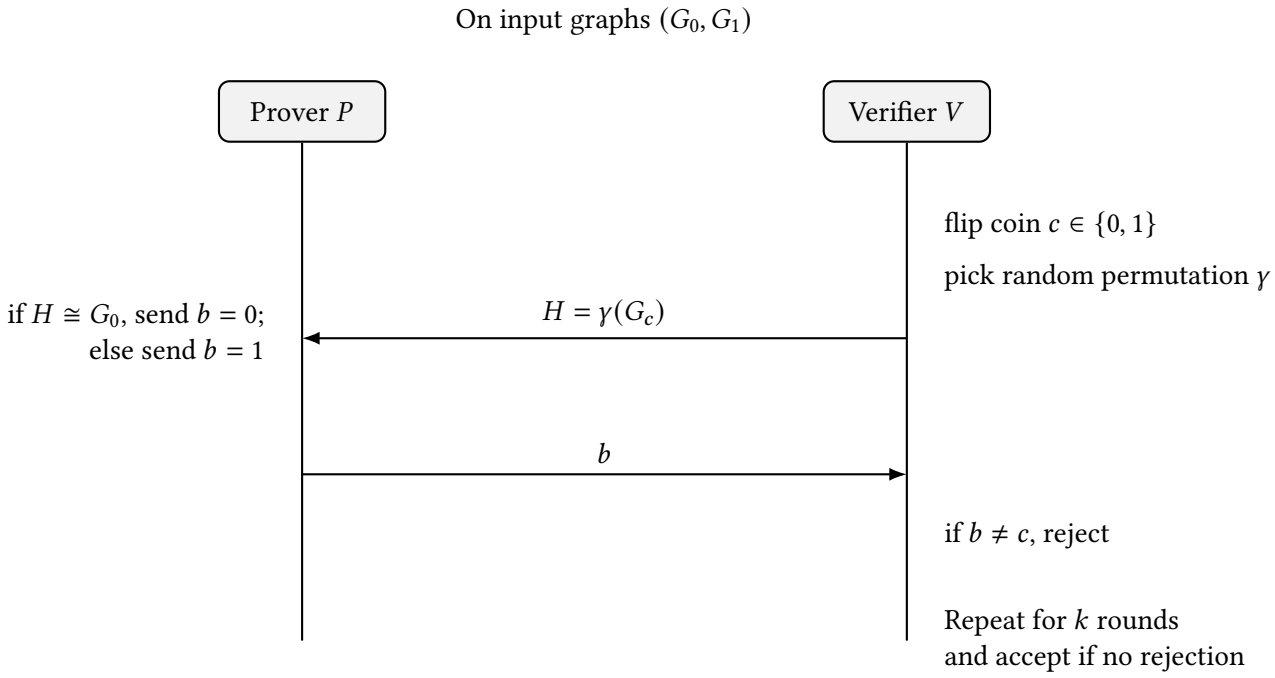


Figure 3: Graph Non-Isomorphism interactive proof

5. Introduction: Self-Proving Models (SPMs)

We introduce the topic of *self-proving models* (SPMs), a new research area connecting interactive proof systems with the problem of making ML model outputs trustworthy, particularly on worst-case rather than just average-case inputs. (Very) informally, the classic result IP = PSPACE tells us that any computational problem we could ever dream of solving can be efficiently verified even with a powerful, untrusted prover. LLMs are extremely powerful but untrusted computation entities that hallucinate—shouldn't we be able to somehow *verify* LLM outputs? This lecture (and the main referenced paper [1]) build toward a formal framework towards that goal.

Gameplan:

1. Background: Interactive Proof Systems
2. Proofs and Models
3. Defining SPMs—*There is a lot of literature on proof systems, cryptography, interactive proofs, and it was challenging to adapt these theoretical artifacts to generative models. Previously these cryptographical tools were primarily seen in blockchain.*
4. Learning SPMs—*Transcript learning (TL) and Reinforcement Learning from Verifier Feedback (RLVF)*
5. SPMs in practice and throughout history

Note: Items 1-3 were covered in Lecture 12, 4-5 were deferred to Lecture 13

6. Background: Interactive Proof Systems

**Orr used slightly different notation during the talk than the previous lectures in this class. Those changes are reflected here to match the lecture slides and recording.*

Definition: Interactive Proof System (IP)

An interactive proof system (P, V) for a language L is a protocol between two parties:

- **Prover** P — computationally unbounded ("all-powerful").
- **Verifier** V — efficient (polynomial-time), *randomised*.

Both P and V receive a *claim* φ as common input (specifically a claim of membership in L). They exchange R rounds of messages:

- V sends a *question* q_r
- P responds with an *answer* a_r

After R rounds, the verifier outputs Accept if it is "convinced" by P, or Reject otherwise.

The protocol satisfies three properties:

Properties of an IP

1. **Completeness.** "For any correct claim there exists a way to convince the verifier of the claim's correctness"

   There exists an *honest prover* P^* such that, for every *correct* claim φ,

   $$\Pr\left[(P^*, V) \text{ accepts } \varphi \mid P^*\right] = 1.$$

2. **Soundness** "The verifier should not be convinced by a wrong proof" (with error δ)

   For every *incorrect* claim φ and every (possibly malicious) prover P,

   $$\Pr\left[V \text{ accepts } \varphi \mid P\right] \leq \delta$$

3. **Efficiency.** V runs in time polynomial in the length of φ. One can also require few rounds (small R), short messages, etc. Message length in particular will be very important in the ML context next lec!

On randomness:
The verifier is randomized and the probabilities above are taken with respect to V's coins. We can assume WLOG that the prover is deterministic because it is all-powerful and can therefore just evaluate *all* of V's possible options. Note that V does have to be randomized because otherwise the setup would reduce to the non-interactive setting where the sequence of questions is determined in advance and P could simulate the whole interaction.

6.1. LLMs, Hallucinations, and IP = PSPACE

Very informally, essentially any problem you could ever dream of solving is definitely in PSPACE. The result IP = PSPACE tells us, also very informally, that for any problem we could ever dream of solving (PSPACE), we can establish trust and efficiently *verify* that solution (IP) even when we don't know how to actually search through all of the possible solutions.

In principle, we can trust powerful systems—like LLMs!—using clever verification and interactive proofs. However the current reality is that LLMs hallucinate to varying degrees of subtlety *and we do not know how to reliably, algorithmically verify their answers.* The theoretical result IP = PSPACE tells us that LLMs *shouldn't* hallucinate (because we should be able to verify their answers) but we still need to figure out how to implement this trust mechanism. One can almost think of vanilla LLMs as "provers" that supply answers rather than actual checkable proofs—a natural attempt at bridging this gap is to train a model to generate such proofs.

7. Proofs and Models

For this work, it is useful to work with autoregressive sequence-to-sequence models.

Definition: Sequence-to-sequence model

Fix a finite alphabet Σ. Let Σ^* denote all finite-length strings over Σ.

- **Ground-truth functionality**: $f^* : \Sigma^* \to \Sigma^*$, a (possibly hard-to-compute) function we wish to approximate. We may not know f^* explicitly and only have access to finitely many labeled examples $\{(x_i, f^*(x_i))\}$.
- **Input distribution**: μ over Σ^* represents the "natural distribution" of inputs seen in practice / "nature". We can sample from μ but need not know it explicitly.
- **Model family**: $\{f_\theta : \Sigma^* \to \Sigma^*\}_{\theta \in \Theta}$, a parametrized collection of (possibly randomized) functions. E.g., $\Theta = \mathbb{R}^d$ for a neural network with d parameters.
- **Training**: given i.i.d. samples $S = \big((x_1, f^*(x_1)), \ldots, (x_m, f^*(x_m))\big)$ with $x_i \sim \mu$, output parameters $\hat{\theta} \in \Theta$.
- **Correctness**: $\hat{\theta}$ is $(1-\epsilon)$-*correct* with respect to (μ, f^*) if

$$\Pr_{\substack{x \sim \mu \\ \hat{y} \sim f_{\hat{\theta}}(x)}} \left[\hat{y} = f^*(x)\right] \;\geq\; 1 - \epsilon.$$

7.1. Trusting a Model

So far in this class we've talked about average-case accuracy. "ChatGPT scores 90% on xyz math benchmark" and "Claude improves on abc reasoning eval by 10%" are all about average-case correctness. There are two notions of accuracy that we care about when it comes to trusting a model:

1. **Average-case:** Does f_θ have $> (1-\epsilon)$ accuracy on a holdout set of data drawn from μ?
2. **Worst-case**: For a *specific* input x and model output $f_\theta(x)$, should I trust this particular answer $f_\theta(x)$?

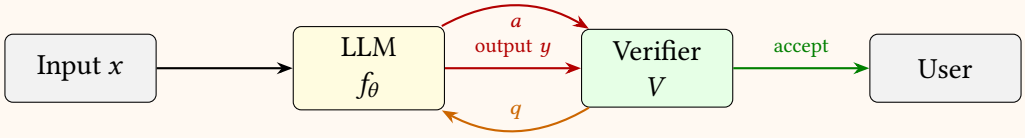
These two notions are different! For a high-stakes situation like deciding whether or not to perform a risky medical procedure, a patient could see that the average mortality rate is 1 in 100 but they more so care about whether they themselves are that single casualty.

7.2. Why interactive proofs + LLMs?

When you query an LLM and aren't sure whether the response is correct, you might ask follow-up questions to probe the model's understanding, try to uncover contradictions, etc. But you might still leave the interaction with very little idea of the correctness of the LLM's response. Instead of using humans (which are fallible, etc.) to try to verify the LLM response, what if we used an algorithm—or verifier—instead?

Containment: Dalrymple et al [3]

Idea: put a verifier that you trust (prove completeness and soundness) between the LLM and the user! Then the user will be unlikely to be bamboozled because the LLM needs to "get through" the verifier first.



Open question: how do we train the LLM to generate proofs?

8. Self-Proving Models (SPMs)

8.1. Adapting Interactive Proofs to LLMs

In the classical interactive proof setting, the claim is membership in a language. In the model/LLM setting, a natural claim is:

"The model's output on this input is correct", or equivalently, $f_\theta(x) = f^*(x)$

where f_θ is the learned model and f^* is the ground truth / nature.

The prover sends the output $f_\theta(x)$ to the verifier as the first message, then the verifier interrogates the prover about $f_\theta(x)$ over several rounds.

Adapting the interactive proof guarantees from Section 6 gives us the following:

Properties of an IP: LLM edition

Fix a model $f_\theta : X \to Y$. The "claim" is that $f_\theta(x) = f^*(x)$ for some input x.

1. **Completeness:** $\exists$ honest prover P^* such that for every input x,

$$\Pr\left[P^* \text{ convinces } V \text{ to accept } (x, f^*(x))\right] = 1$$

2. **Soundness:** With error δ and any input x, for every incorrect output $f_\theta(x) \neq f^*(x)$ and for any "lying" prover P,

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

Why do hallucinations still exist? Completeness guarantees that the honest prover P^* *exists* but it does not tell us how to compute it. Training a model to be accurate does not automatically train it to generate (verifiable) proofs. This is the gap that SPMs try to close.

8.2. Defining SPMs

There are 3 primary objects we care about: the learned model/LLM f_θ, the prover P, and the verifier V. The key idea is to train f_θ and P jointly such that they are both parameterized by the

same θ. This is what the "self-proving" refers to! The LLM should not only produce *answers* but also provide a proof of the correctness of that answer.

Definition: SPM

Given a verifier V and an input distribution μ, a model P_θ is self-proving with error ϵ if

$$\Pr_{\substack{x\sim\mu \\ y\sim P_\theta(x) \\ \text{interaction}}} [V \text{ accepts } y] \;\geq\; 1-\epsilon$$

The probability is taken over:

- x sampled from input distribution μ
- y (the output/first message) from $P_\theta(x)$
- V's random coins throughout the interaction
- P's random coins throughout the interaction

Unlike the classical interactive proof setting, the prover P_θ *does* use randomness as it is a neural network. **Self-provability** is a feature of the prover *with respect to a specific V and μ*, similar to how model accuracy is defined with respect to a specific μ and f^*.

Note that V is hand-specified, classical code (ie. `verifier.py`) that you have to write and prove completeness/soundness by hand.

8.3. Correctness of SPMs

Soundness + Self-Provability → Correctness

Fix a verifier V for f^* with soundness error δ. If P_θ is self-proving with error ϵ wrt to V, μ, then

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

In other words, the model P_θ generates correct outputs $P_\theta(x)$ with probability $\geq 1-\delta-\epsilon$.

Intuition: ϵ is the probability lost because P_θ failed to convince V to accept correct outputs. δ is the probability lost because the verifier accepted wrong outputs. These failure modes add up in error as in a union bound.

Proof. Let $x \sim \mu$ and $y \sim P_\theta(x)$. Partition based on whether $y = f^*(x)$ or not.

By the soundness guarantee we have $\Pr[V \text{ accepts } y \mid y \neq f^*] \leq \delta$.

Self-provability says $\Pr[V \text{ accepts } y] \geq 1-\epsilon$.

Applying the Law of Total Probability:

$$\begin{aligned}\Pr[V \text{ accepts } y] = \Pr[V \text{ acc.} \mid y = f^*] \cdot \Pr[y = f^*] \\ + \Pr[V \text{ acc.} \mid y \neq f^*] \cdot \Pr[y \neq f^*]\end{aligned}$$

Using $\Pr[V \text{ acc.} \mid y = f^*] \leq 1$:

$$\Pr[V \text{ accepts } y] \leq \Pr[y = f^*] + \delta \cdot \Pr[y \neq f^*] \leq \Pr[y = f^*] + \delta.$$

Applying self-provability and rearranging terms yields

$$1 - \epsilon \leq \Pr[V \text{ accepts } y] \leq \Pr[y = f^*] + \delta$$
$$1 - \delta - \epsilon \leq \Pr_{x \sim \mu}[P_\theta(x) = f^*(x)]$$

□

9. Challenges and Open Problems

- **Natural language ambiguity:** Many practical LLM questions are ambiguous ("What should I say to my professor about xxx?"). *Auto-formalization*, or translating squishy natural language into formal statements is a separate, complementary challenge to SPMs.
- **Distribution shift in proofs:** Even if the input μ is natural, the distribution over proof transcripts (questions + answers) may not be natural or very different from μ.
- **Naturalness of proof systems:** The proof of IP = PSPACE is highly "unnatural" and involves coefficients of low-degree polynomial arithmetizations. LLMs are trained on natural language and can't be expected to generate proofs in this type of format.

Next lecture: learning SPMs (TL, RLVF) + applications!

References

[1] Noga Amit, Shafi Goldwasser, Orr Paradise, and Guy N. Rothblum. Models that prove their own correctness. *CoRR*, abs/2405.15722, 2024. doi: 10.48550/ARXIV.2405.15722. URL https://doi.org/10.48550/arXiv.2405.15722.

[2] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014. doi: 10.1109/SP.2014.36.

[3] David "davidad" Dalrymple, Joar Skalse, Yoshua Bengio, Stuart Russell, Max Tegmark, Sanjit Seshia, Steve Omohundro, Christian Szegedy, Ben Goldhaber, Nora Ammann, Alessandro Abate, Joe Halpern, Clark Barrett, Ding Zhao, Tan Zhi-Xuan, Jeannette Wing, and Joshua Tenenbaum. Towards guaranteed safe ai: A framework for ensuring robust and reliable ai systems, 2024. URL https://arxiv.org/abs/2405.06624.

[4] Ben Fisch, Daniel Freund, and Moni Naor. Physical zero-knowledge proofs of physical properties. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO*

2014, volume 8617 of *Lecture Notes in Computer Science*, pages 313–336. Springer, 2014. doi: 10.1007/978-3-662-44381-1_18.

[5] Alexander Glaser, Boaz Barak, and Robert J. Goldston. A zero-knowledge protocol for nuclear warhead verification. *Nature*, 510(7506):497–502, 2014. doi: 10.1038/nature13457.

[6] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, page 218–229, New York, NY, USA, 1987. Association for Computing Machinery. ISBN 0897912217. doi: 10.1145/28395.28420. URL https://doi.org/10.1145/28395.28420.

[7] Shafi Goldwasser and Sunoo Park. Public accountability vs. secret laws: Can they coexist?: A cryptographic proposal. In *Proceedings of the 2017 Workshop on Privacy in the Electronic Society*, WPES '17, pages 99–110. Association for Computing Machinery, 2017. doi: 10.1145/3139550.3139565.