> RLVR trains a model against a verifier that comes with no soundness guarantee. At the level of definitions, what would it mean for such a verifier to be "sound enough" to train against? Give one concrete verifier/task pair where you would expect training to select a prover that games the verifier rather than solves the task, and name the property of the verifier that makes this possible. A sketch is fine, but mark explicitly any step you cannot justify rather than glossing over it.

First, we would start with defining what does "sound" mean in this context: We may define x to be a task, y to be a proposed solution generated by the prover/model, $C(x, y) \in 0, 1$ to indicate whether y solves x, and $V(x, y) \in 0, 1$ to indicate whether the verifier accepts it. Therefore, an ideal and sound verifier would satisfy that $V(x, y) = 1$ implies $C(x, y) = 1$, and that it never accepts an incorrect solution (for instance, a verifier that rejects everything is sound).

However, such definition of "soundness" does not guide a verifier useful for training. Now we continue to define what is "sound enough" in our context: Let D be a task distribution, Π be a class of potential model policies, and ε be some error tolerance bound. Therefore, we may have the following sufficient condition:

$$\sup_{\pi \in \Pi} \Pr_{\substack{x \sim D \\ y \sim \pi(\cdot|x)}} [V(x, y) = 1 \text{ and } C(x, y) = 0] \leq \varepsilon.$$

I.e., every policy in the relevant class has probability at most ε to produce an incorrect solution that the verifier would accept.

Example

Before stating the concrete example, we recall the *Fermat's little theorem*, which says that "If p is a prime number and a is an integer that's not divisible by p, then $a^{p-1} \equiv 1 \pmod{p}$ (a^{p-1} is congruent to 1 mod p)." For a positive integer a, if a composite integer x divides $a^{x-1} - 1$, then x is called a **Fermat pseudoprime** to base a. A famous base-2 Fermat pseudoprime is 341, which is equal to 11 * 31 and it satisfies Fermat's little theorem: $2^{340} \equiv 1 \pmod{341}$, thus passes the Fermat primality test for base 2.

Now we state our example: Let the task be "Given an integer n >= 2, tell us whether it is a prime or a composite". Let the verifier be the base-2 Fermat test, in other words, we may have the following implementation:

```python
def base_two_fermat(n):
    if n == 2:
        return "prime"
    if n % 2 == 0:
        return "composite"
    return "prime" if pow(2, n - 1, n) == 1 else "composite"

def reward(n, model_answer):
    return int(model_answer == base_two_fermat(n))
```

Now, if we actually run the program `reward(341, "prime")`, it actually returns 1, instead of the correct answer 0, which means that the verifier would penalizes the correct answers.

Suppose we have a training distribution that assigns probability $q > 0$ to odd composite numbers that pass the base-2 Fermat test, and we consider the following two policies:

1. A *correct policy* that would always return the true primality test result.
2. A "verifier-matching policy" that would return the result of our base-2 Fermat test verifier.

Then the expected reward for the first one would be $1 - q$ and the second one be 1, which means that the reward objective would prefer the verifier-matching policy to the perfect correct policy. So training would prefer a prover that games the verifier, and the relevant property is that **systematic, learnable false positives caused by using a necessary condition for primality as a sufficient condition**. Since the verifier is fixed, these mistakes remain consistent across training.

> Pick one tool or result from complexity theory or cryptography (e.g., sum-check, PCPs, zero-knowledge, pseudorandomness) and make the case for what it can and cannot buy AI safety. Be concrete about the gap between the theorem's assumptions and the setting in which a deployed model actually operates, and describe a small-model experiment that would tell you whether that gap matters in practice.

My choice of topic is the **PCP theorem**, which says that every problem in NP has probabilistically checkable proofs (PCPs) of constant query complexity and logarithmic randomness complexity. True statements thus have proofs accepted with probability one. For false statements, every proof passes with probability at most 1/2, with independent repetition reducing this error. We then recall *scalable oversight*,

which is the problem of using weaker AI systems to supervise stronger ones (e.g., smarter-than-human AI models).

We then can ask: How can a weaker verifier check claims and/or proofs from a stronger model, since it does not need to read the whole proof? Theorem 32 of "How to Verify Probabilistic Consistency of Predictive Models" by Paradise, Richardson, Bengio, and Goldwasser ("PRBG26" when referred below) gives an interactive PCP for checking whether a model's probability claims are approximately consistent with some joint distribution. The protocol combines queries to an encoded proof with interaction, and unlike the classical PCP theorem, its query bound is not constant.

Therefore, I would like to study what happens when a model answers questions about the proof, instead of providing a fixed proof in advance. The PCP theorem assumes that the proof cannot change during checking, which implies that asking for the second bit should always return the same answer. However, a conservational model may use prior/earlier questions to adjust its answers and pass individual and separate checks without sticking to one proof. Remark 1 of PRBG26 suggests that we could reset the context of the model before each proof query.

For a small-world experiment, I would implement a toy version of the protocol with an exact verifier and independently certified consistent and inconsistent examples. I would train a small transformer under three conditions:

1. A proof table that is frozen before checking.
2. Oracle answers generated deterministically from the instance and query location.
3. Oracle answers generated with access to the conversation history.

I would then train for acceptance on inconsistent examples and measure false acceptance using fresh verifier randomness. More successful cheating with conversation history would show that this assumption matters. Low acceptance of consistent examples under isolation would instead expose a difficulty in learning useful proofs.