easily identify flaws. In other words, while the generator can refine proofs based on external feedback, it fails to evaluate its own work with the same rigor as the dedicated verifier.

This observation motivated us to endow the proof generator with genuine verification capabilities. During training, we prompt the generator π_θ to produce a proof Y followed by a self-analysis Z that follows the same format and rubrics $\mathcal{I}_v$ as the verifier (see Appendix A.1). We denote the proof score predicted in the self-analysis as s'.

To ensure faithful self-evaluation, we use the verifier π_φ to assess both components: the proof Y receives score $R_Y = s$, and the self-analysis Z receives a meta-verification score $R_{\text{meta}}(Z) = ms$. The reward function combines these assessments:

$$R = R_{\text{format}}(Y, Z) \cdot (\alpha \cdot R_Y + \beta \cdot R_Z) \tag{5}$$

$$R_Z = R_{\text{score}}(s', s) \cdot R_{\text{meta}}(Z) \tag{6}$$

where $R_{\text{format}}(Y, Z)$ verifies that both the proof and self-analysis follow the specified format, $R_{\text{score}}(s', s)$ rewards accurate self-assessment. We set $\alpha = 0.76$ and $\beta = 0.24$. This reward structure creates the following incentives:

- Faithful acknowledgment of errors is rewarded over false claims of correctness.
- The highest rewards come from producing correct proofs and accurately recognizing their rigor.
- A good strategy to obtain high rewards for the proof generator is to identify and resolve as many issues as possible before finalizing the response.

2.3. Synergy Between Proof Verification and Generation

The proof verifier and generator create a synergistic cycle: the verifier improves the generator, and as the generator improves, it produces new proofs that challenge the verifier's current capabilities. These challenging cases – where the verifier may fail to identify issues in a single attempt – become valuable training data for enhancing the verifier itself.

To retrain and improve the verifier, we need labeled correctness data for newly generated proofs. Manual annotation, while straightforward, becomes increasingly time-consuming as problems grow harder and errors become more subtle. To boost annotation efficiency, we generated multiple verifier analyses per proof to surface potential issues for human review.

From this AI-assisted annotation process, we recognized two facts that make it feasible to push the level of automation a step further:

1. Scaling verifier samples increases the probability of catching real issues in flawed proofs.
2. Reviewing the verifier's identified issues is exactly **meta-verification**, which is easier than identifying issues from scratch. Meta-verification is also more sample-efficient for LLMs to master.

Building on these observations, we developed the following automated labeling process:

1. For each proof, generate n independent verification analyses