

The Policy as Its Own Judge: Self-Rewarding Language Models through Self-Distillation Grounded by Asymmetric Environment Feedback

2026-03-02

Motivation

Why is this worth pursuing? What gap does it fill?

- LLM as a judge is a powerful framework for computing rewards.
- But its inherently off-policy. This causes evasion, reward collapse, etc.
- RLVR, while on-policy, gives sparse rewards and suffers from the credit assignment problem.
- Thus, we present Self-Rewarding Language Models Grounded by Asymmetric Environment Feedback (acronym: POJ Policy as Own Judge):
 - LLM policy generates rollout
 - Rollout is evaluated in the environment, giving us a (1) binary reward from the verifier and (2) environment feedback (if applicable; for example, stack traces in codegen)
 - A teacher is forked from the LLM at the current time step. The teacher is conditioned on the environment feedback, prompt, rollout, verifier score, as is asked to assign a final reward (continuous $[0,1]$ value).
- Solves:
 - Sparse reward: model can receive partial credit
 - Eliminates evasion: the distributional gap between the judge and the generator is identically zero. The policy cannot learn to outrun its own distribution.
 - Reward collapse prevention: environment feedback as an inescapable, grounding constraint, preventing hallucination.
 - Ease of implementation: It introduces only a single additional forward pass (i.e., the teacher) with an extended context window per rollout.

Core Idea

Describe the approach. Include equations, pseudocode, or diagrams as needed.

```
% Requires:
% \usepackage{amsmath,amssymb}
% \usepackage[ruled,vlined]{algorithm2e}

\begin{algorithm}[t]
\caption{Asymmetric Environment-Grounded Self-Reward (AEG-SR)}
\label{alg:aeg-sr}

\KwIn{
Initial policy weights  $\pi_{\theta_0}$  (ideally warm-started via SFT on judgment tasks);
environment  $\mathcal{E}$  yielding execution feedback  $f$  and sparse reward  $R_{\text{env}}$ ;
static replay buffer of judgment examples  $\mathcal{B}_{\text{judge}} = \{(s, c_m, q_m)\}$ ;
judgment classes  $K$  and reward weights  $\{w_k\}_{k \in K}$  (e.g.,  $w_{\text{correct}} = 1.0$ ,  $w_{\text{incorrect}} = -1.0$ );
entanglement coefficient  $\lambda$ ; learning rate  $\eta$ .
}

\textbf{Initialize} shared policy weights  $\pi_{\theta} \leftarrow \pi_{\theta_0}$ ;

\For{$t = 1, 2, \dots$}{
    Sample a batch of prompts  $\{s_i\}_{i=1}^B$  from the training distribution;

    \BlankLine
    \textbf{Phase 1: Student Generation \& Environment Execution}\;
    \ForEach{prompt  $s_i$ }{
        Generate a group of  $G$  rollouts:  $a_{i,j} \sim \pi_{\theta}(\cdot \mid s_i)$ , extracting
        Execute each rollout  $a_{i,j}$  in environment  $\mathcal{E}$  to obtain execution feedback  $f_{i,j}$  and sparse reward  $R_{\text{env},i,j}$ ;
    }

    \BlankLine
    \textbf{Phase 2: Asymmetric Teacher Judgment (Dense Reward Generation)}\;
    \ForEach{rollout  $a_{i,j}$ }{
        Construct the privileged teacher context:  $c_{i,j} \coloneqq (a_{i,j}, f_{i,j}, R_{\text{env},i,j})$ ;
        Perform a forward pass using the judge prompt  $p_{\text{judge}}: \text{“} \langle a_{i,j}, f_{i,j}, R_{\text{env},i,j} \rangle \text{“}$ ;
        \Indp
         $P_{\theta}(q \mid s_i, c_{i,j}, p_{\text{judge}})$ ;
        \Indm
    }
}
```

```

    Extract the probability distribution over judgment classes  $k \in K$ ;
    Compute dense self-reward:
    \[
        R_{\text{self}}(s_i, a_{i,j})
        \coloneqq \sum_{k \in K} w_k \cdot P_{\theta}(q=k \mid s_i, c_{i,j})
    \];
}

\BlankLine
\textbf{Phase 3: Multi-Task Entanglement Update}\;
Compute GRPO advantages  $\hat{A}_{i,j}$  using the dense  $R_{\text{self}}$  signals\;
Compute the standard RL policy loss  $L_{\text{GRPO}}(\theta)$  over the batch\;

Sample a mini-batch of ground-truth judgment pairs from  $\mathcal{B}_{\text{judge}}$ \;
Compute the cross-entropy loss  $L_{\text{CE\_judge}}(\theta)$  on the replay buffer predictions\;

Calculate the joint entangled objective:
\[
    L_{\text{total}}(\theta) = L_{\text{GRPO}}(\theta) + \lambda L_{\text{CE\_judge}}(\theta)
\];

Update the shared weights:
\[
    \theta \leftarrow \theta - \eta \nabla_{\theta} L_{\text{total}}(\theta)
\];
}
\end{algorithm}

```

Open Questions

-

Potential Experiments

-

Related Work

-

Dump

Self-Rewarding Language Models Grounded by Asymmetric Environment Feedback

1. Abstract and Problem Statement

Reinforcement Learning with Verifiable Rewards (RLVR) has become the dominant paradigm for scaling Large Language Models (LLMs) on reasoning, math, and coding tasks. However, RLVR is fundamentally bottlenecked by **sparse rewards**: the environment typically provides a binary pass/fail signal based on test-case correctness. This sparse signal fails to inform the model *why* it failed, making learning highly inefficient.

Recent literature proposes “Self-Rewarding Language Models” (SRLMs) to generate dense training signals, but these methods suffer from **reward collapse**—without external grounding, the model learns to sycophantically rate its own outputs highly. Conversely, using a static, external LLM as a judge introduces a severe **distributional gap** as the active policy quickly outpaces the frozen judge .

We propose **Asymmetric Environment-Grounded Self-Reward (AEG-SR)**. We instantiate the policy in two roles: a Student that generates solutions, and a Teacher that evaluates them. Both roles share identical weights, keeping the judge strictly on-policy. Crucially, the Teacher is granted *asymmetric privileged information*—the hard execution feedback (e.g., stack traces, compiler errors) from the environment . This self-distillation translates uninterpretable, sparse environment failures into dense, shaped gradient signals, while preventing reward collapse by anchoring the model’s self-judgment to objective reality.

2. Methodology: The Policy as Its Own Judge

Let π_θ be a language model policy trained via RL. At each training step, the framework operates in two distinct phases using the exact same parameter weights θ .

2.1 The Generation Phase (Student)

Given a prompt s , the Student generates a rollout consisting of an optional reasoning trace r and a final action/code a :

$$a \sim \pi_\theta(\cdot|s)$$

. The action is executed in the environment, which returns a standard sparse binary reward $R_{env} \in \{0, 1\}$ and a rich, text-based execution feedback string f (e.g., “IndexError: list index out of range at line 4”).

2.2 The Judgment Phase (Asymmetric Teacher)

Instead of discarding f after computing the sparse reward, we construct a privileged context window:

$$c = (a, f, r)$$

. The Teacher operates under the distribution $\pi_\theta(\cdot|s, c)$. We prompt the Teacher to act as a judge, evaluating the severity of the failure or the quality of the approach based on the exact error trace.

The Teacher outputs a categorical judgment token $q \in \{critical_error, minor_bug, correct\}$. We extract a continuous, dense self-reward R_{self} directly from the Teacher’s logits over these classes:

$$R_{self}(s, a) = \sum_k w_k \cdot P_\theta(q = k|s, c)$$

where w_k represents a predefined scalar value for each class (e.g., $minor_bug = 0.5$, $correct = 1.0$).

2.3 Policy Update

The policy is then updated using a standard RL algorithm (e.g., GRPO), substituting the sparse R_{env} with the dense R_{self} . Because the Teacher is identically π_θ , the detector cannot fall behind the policy’s evolving distribution.

3. Theoretical Advantages

1. **Solving the Sparse Reward Problem:** The model receives partial credit for conceptually correct approaches that fail due to minor syntax errors, dramatically smoothing the optimization landscape.
2. **Eliminating the Evasion Gap:** Because the detector’s parameters are a function of the current policy parameters θ_t at every training step, the distributional gap between the judge and the generator is identically zero . The policy cannot learn to outrun its own distribution .
3. **Preventing Reward Collapse:** In standard SRLMs, the judge only sees the prompt and the generated text, making it easy to hallucinate success. In AEG-SR, the environment feedback f acts as an inescapable, grounding constraint. The model is forced to condition its reward on the undeniable reality of a stack trace .

4. Implementation and Feasibility

This architecture is highly optimized for lightweight RL post-training frameworks. It requires no external API calls, no delayed Exponential Moving Average (EMA) targets, and no auxiliary training loops. It introduces only a single additional forward pass of π_θ with an extended context window per rollout.

To prevent meta-reasoning degradation—where the model optimizes for generation but loses its ability to judge—we can entangle the representations by computing a joint loss over both the RL objective and a static replay buffer of judgment examples during the backward pass .

Algorithm: Asymmetric Environment-Grounded Self-Reward (AEG-SR)

Require: * Initial policy weights θ_0 (ideally warm-started via SFT on judgment tasks)

- Environment E yielding execution feedback f and sparse reward R_{env}
 - Static replay buffer of judgment examples $\mathcal{B}_{judge} = \{(s, c_m, q_m)\}$
 - Judgment classes K and associated reward weights w_k (e.g., $w_{correct} = 1.0, w_{minor_bug} = 0.5$)
 - Entanglement coefficient λ , Learning rate η
1. **Initialize** shared policy weights $\pi_\theta \leftarrow \pi_{\theta_0}$
 2. **For** each training iteration $t = 1, 2, \dots$ **do**:
 3. **Sample** a batch of prompts $\{s_i\}_{i=1}^B$ from the training distribution.

4. **Phase 1: Student Generation & Environment Execution**
5. **For** each prompt s_i **do**:
6. Generate a group of G rollouts: $a_{i,j} \sim \pi_\theta(\cdot|s_i)$, extracting reasoning traces $r_{i,j}$.
7. Execute each rollout $a_{i,j}$ in environment E to obtain execution feedback $f_{i,j}$.
8. **End For**
9. **Phase 2: Asymmetric Teacher Judgment (Dense Reward Generation)**
10. **For** each rollout $a_{i,j}$ **do**:
11. Construct the privileged teacher context: $c_{i,j} = (a_{i,j}, f_{i,j}, r_{i,j})$.
12. Perform a forward pass using the judge prompt p_{judge} : $P_\theta(q|s_i, c_{i,j}, p_{judge})$.
13. Extract the probability distribution over judgment classes $k \in K$.
14. Compute the dense self-reward: $R_{self}(s_i, a_{i,j}) = \sum_{k \in K} w_k \cdot P_\theta(q = k|s_i, c_{i,j})$.
15. **End For**
16. **Phase 3: Multi-Task Entanglement Update**
17. Compute the GRPO advantages $\hat{A}_{i,j}$ using the dense R_{self} signals.
18. Compute the standard RL policy loss $\mathcal{L}_{GRPO}(\theta)$ over the batch.
19. **Sample** a mini-batch of ground-truth judgment pairs from $\mathcal{B}_{judge}$.
20. Compute the Cross-Entropy loss $\mathcal{L}_{CE_judge}(\theta)$ on the replay buffer predictions.
21. Calculate the joint entangled objective:

$$\mathcal{L}_{total}(\theta) = \mathcal{L}_{GRPO}(\theta) + \lambda \mathcal{L}_{CE_judge}(\theta)$$

22. Update the shared weights: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{total}(\theta)$.
23. **End For**

Key Architectural Strengths of this Algorithm

- **Continuous Gradient Flow:** By directly mapping the Teacher’s logits to R_{self} , the generation phase receives a highly granular advantage signal $\hat{A}_{i,j}$. Instead of a binary 0 or 1, a model that writes a 90% correct function that fails a single edge case might get a 0.8 reward, radically speeding up convergence.

- **Structural Entanglement:** Step 21 is the linchpin. Because both $\mathcal{L}_{GRPO}$ and $\mathcal{L}_{CE_judge}$ are backpropagated through the exact same parameters simultaneously, the model is structurally forced to maintain its meta-reasoning. It cannot maximize R_{self} by generating sycophantic “fake” solutions without simultaneously incurring a massive penalty on the $\mathcal{L}_{CE_judge}$ term.
- **Efficiency:** There are no detached clones or nested SGD loops to manage. It requires G standard generation passes, G fast forward passes for the Teacher’s logit extraction, and a single joint backward pass.