

## Executive Summary

**Tamiyo Seed Lifecycle** uses a complex reward function combining *counterfactual contribution* as the primary signal with potential-based shaping (PBRs), efficiency penalties, and anti-“Goodhart” heuristics. The design aims to guide a PPO+LSTM agent to maximize host model accuracy by growing and pruning “seeds” (neural modules) responsibly, while preventing reward exploitation. The reward is decomposed into multiple components activated at different lifecycle stages, including: **bounded attribution** (counterfactual accuracy gain, capped to actual progress), **PBRs stage bonuses** (small rewards for progressing seeds through GERMINATED→TRAINING→BLENDING→HOLDING→FOSSILIZED), **compute rent penalties** (for parameter bloat), **action costs** (tiny negative rewards to discourage frivolous interventions), and **terminal bonuses** (for final accuracy and successfully integrated seeds). Anti-gaming measures are built in: e.g. *attribution discount* for seeds that regress overall performance, a *ratio penalty* if a seed’s measured contribution vastly exceeds its actual improvement, *blending warnings* if a seed consistently hurts accuracy, and steep *waiting penalties* in the HOLDING stage to prevent “farming” positive rewards indefinitely.

**Theoretically**, the PBRs component is derived from Ng et al. (1999) to preserve optimal policies <sup>1</sup>. In practice, however, mixing PBRs with other shaping signals means the overall reward no longer strictly guarantees policy invariance <sup>2</sup>. The PBRs potentials are carefully tuned (e.g. BLENDING stage has the largest potential jump to reflect value creation) and validated to telescope correctly <sup>3</sup> <sup>4</sup>, though minor leakage occurs since the discount factor  $\gamma=0.995$  (over a 25-step episode, ~12% of the shaping doesn’t perfectly cancel <sup>5</sup>). All shaping uses the same potential table to maintain consistency.

**Incentives Analysis:** This reward function encourages the agent to **germinate seeds only when needed** (germination while another seed is active incurs a small penalty), **train seeds adequately** (but not stall too long), and **blend them** once they have potential to improve accuracy. A *positive contribution* yields immediate reward (bounded by actual progress), incentivizing seeds that truly add accuracy. Conversely, *toxic seeds* (negative counterfactual impact) produce negative rewards, prompting the agent to prune them quickly. Additional bonuses (PBRs and a small synergy reward) entice the agent to advance seeds through all stages and exploit beneficial interactions between seeds, while penalties (compute rent and action costs) discourage having too many or oversized seeds and needless churn. The agent is **dissuaded from “gaming”** the reward: for example, it can’t infinitely wait with a good seed to accumulate reward (a steep exponential penalty in HOLDING ensures diminishing returns), and it gains nothing from creating dependencies that don’t yield real progress (attribution is discounted to 0 if a seed’s overall impact is negative). The terminal bonus aligns the final outcome with the true objective by rewarding final accuracy (scaled) and completed seeds that made a positive contribution, ensuring that seeds which ultimately hurt or did nothing get no final reward <sup>6</sup>.

**Key Findings & Risks:** The reward function is **comprehensive but complex**. It largely aligns with the ultimate goal (maximize final accuracy per parameter), but the dense shaping could introduce unintended incentives. For instance, mixing PBRs with non-potential rewards can subtly alter the optimal policy or learning dynamics <sup>2</sup>. The “bounded attribution” scheme is conceptually sound for counterfactual credit (a seed cannot earn more reward than actual accuracy gained), yet its reliance on proxy signals (like per-epoch accuracy delta before counterfactuals are available) means the agent might exploit scenarios where host-model improvement is mistakenly credited to a seed. The anti-goodhart mechanisms address known failure modes – e.g. “ransomware seeds” that artificially inflate their importance are mitigated by the attribution discount and ratio penalty <sup>7</sup> <sup>8</sup> – but they also

introduce additional non-linearities that complicate credit assignment. Our analysis identifies specific edge cases: e.g., if a seed transitions stages without proper bookkeeping, the PBRs reward can be miscomputed (breaking the intended telescoping invariance) <sup>9</sup> <sup>10</sup> ; or if the PPO network’s value estimates are not properly scaled, the large negative penalty for waiting in HOLDING (up to -9) could lead to unstable gradients or clipping.

**Recommendations:** We propose a combination of **low-risk tweaks** (e.g. stricter validation of PBRs invariants, mild reward normalization, adjusting threshold parameters for attribution penalties) and **higher-risk, high-reward adjustments** (like adopting the simpler “SIMPLIFIED” reward mode by default, which uses only PBRs, uniform action costs, and terminal rewards, thus providing a cleaner learning signal). An ablation plan is outlined to empirically validate each component’s necessity by turning them off one at a time (e.g. no PBRs, no anti-gaming penalties, etc.) <sup>11</sup> <sup>12</sup> . We also recommend tracking key metrics (reward component magnitudes, policy behaviors, variance) during training to detect any residual reward hacking or credit assignment issues, and to ensure the shaped rewards are truly helping rather than hindering learning. Overall, the current reward function is a **sophisticated blend of theory-driven shaping and practical safeguards**, but it can be streamlined. Simplifying the reward structure – focusing on potential-based stage rewards and final outcomes – may improve learning stability and agent alignment with the true objective, while targeted adjustments can reduce the chance of unintended behaviors.

## Reward Decomposition and Signal Map

The Tamiyo reward is composed of several additive components. The table below summarizes each component, its sign (+/-), typical scale, and when/why it is applied:

Reward Component	Sign	Typical Magnitude	When Active (Conditions)
<b>Bounded Attribution</b> – Counterfactual seed contribution (primary signal) <sup>13</sup> <sup>14</sup>	+ or – (dominant term)	Up to ±5 (scale 0 to ±10, clipped via weight=1.0) <sup>15</sup> . Large positive only if seed adds significant accuracy (e.g. +5%); negative if seed hurts accuracy.	Every epoch <i>after</i> a seed is blended into the model (counterfactual available). – If <code>seed_contribution</code> > 0, reward = $\min(\text{seed\_contribution}, \text{progress})$ or $\sqrt{\text{progress} \times \text{contribution}}$ (whichever smaller) times weight <sup>14</sup> . If <code>seed_contribution</code> < 0, reward = 1.0 × (negative contribution) (penalize toxic seeds). If no true counterfactual yet (pre-blending), uses proxy Δaccuracy with lower weight (0.3×) <sup>15</sup> . Attribution is <b>disabled for fossilized seeds</b> (to avoid infinite reward) and <b>inverted for PRUNE actions</b> (pruning a good seed incurs negative reward, pruning a bad seed yields positive reward) <sup>14</sup> .

Reward Component	Sign	Typical Magnitude	When Active (Conditions)
<b>Attribution Discount</b> – “Anti-ransomware” multiplier <sup>7</sup> <sup>8</sup>	(multiplicative, reduces positive reward)	0 to 1 (typically 0.3–0.8 if applied; 1.0 if no regression)	Applied <i>within</i> the attribution calculation when a seed’s <b>total improvement</b> is negative. It’s a sigmoid-based discount that approaches 0 as <code>total_improvement &lt; 0</code> grows (e.g. for a –0.5% regression, credit ~18%; for –0.1%, ~43%) <sup>15</sup> <sup>16</sup> . Ensures seeds that cause net harm get little/no positive reward even if momentarily they show contribution.
<b>Ratio Penalty</b> – Contribution vs. improvement heuristic <sup>8</sup>	– (penalty)	0 to ~–0.3 (capped)	Applied only for suspicious seeds where <code>seed_contribution &gt;&gt; total_improvement</code> . If a seed claims > 5× the actual improvement (e.g. dependency injection), a negative term up to –0.3 is added to offset the attribution <sup>8</sup> . Triggers only when contribution > 1.0% and improvement is negligible or much smaller. Skipped if anti-gaming disabled or attribution already heavily discounted (discount < 0.5).
<b>Blending Warning</b> – Negative trajectory alert	– (penalty)	–0.1 down to ~–0.4	Only during <b>BLENDING</b> stage: if a seed’s total improvement since germination is negative, each epoch in BLENDING adds an escalating penalty <sup>14</sup> . Starts at –0.15 (first epoch of BLENDING with regression) up to ~–0.40 by 6 epochs, to encourage early pruning of bad seeds. No effect if seed is improving.
<b>Holding Indecision Penalty</b> – WAIT in holding stage	– (penalty)	–1.0, –3.0, –9.0 ... (capped at –10)	Only in <b>HOLDING</b> stage: if the agent chooses <code>WAIT</code> while a seed is in HOLDING, and the seed has <i>positive</i> attribution (i.e. it’s a good seed being “farmed”), an exponential penalty applies <sup>14</sup> . After a 1-epoch grace period, the 2nd consecutive WAIT triggers –1.0, 3rd triggers –3.0, 4th –9.0, etc., up to –10 max. This harsh penalty ensures the agent <i>fossilizes or prunes</i> instead of stalling to collect repeated rewards. (No penalty if the seed’s attribution is ~0, meaning it wasn’t valuable – in such case, waiting isn’t advantageous anyway.)

Reward Component	Sign	Typical Magnitude	When Active (Conditions)
<b>PBRS Stage Bonus</b> – Potential-based shaping <small>17 18</small>	+ (bonus) or 0 (rarely slight – in edge cases)	Small (~+0.1 to +0.5 per transition; total across episode typically < +2).	<p>For any seed (except when PBRS disabled), every epoch calculates <math>F(s,s') = \gamma \cdot \varphi(s') - \varphi(s)</math> based on the seed's stage potential <math>\varphi</math>. <i>When a seed advances to a new stage, PBRS gives a burst:</i> e.g. GERMINATED→TRAINING yields <math>+0.3(0.9952.0 - 1.0) \approx +0.3</math>, TRAINING→BLENDING yields <math>\sim +0.44</math> (largest jump, since <math>\varphi_4 - \varphi_3</math>. plus any “time-in-stage” potential) <small>19 5</small>. If a seed stays in the same stage, a smaller bonus accrues from the “epoch progress potential” (0.3 per epoch, capped at +2.0) – effectively rewarding time spent in a stage up to a limit. (Edge case: if a stage transition happens after maxing out previous stage potential, <math>\varphi(s') &lt; \varphi(s)</math> can make PBRS slightly negative, but this is rare and ideally prevented by monotonic potential tuning <small>18</small>.) PBRS is scaled by <code>pbrs_weight=0.3</code> so it remains a shaping hint (~10–30% of total reward) <small>20 21</small> and theoretically doesn't alter the optimal policy by itself <small>4</small>.</p>
<b>Synergy Bonus</b> – Multi-seed interaction reward	+ (bonus)	Very small (0 to +0.1 max)	<p>Each epoch, if an active seed has positive <b>interaction_sum</b> with other seeds (indicating it synergistically boosted others), it gets <code>0 &lt; bonus ≤ 0.1</code>. Computed as <code>0.1 * tanh(0.5 * interaction_sum)</code> – producing a bounded reward (saturates at +0.1) <small>22 23</small>. Only seeds that actually <i>help</i> others get this (no reward for zero or negative interactions). This is intended to encourage “scaffolding” behavior (seeds that support each other). In practice, it's a minor signal and may often be drowned out by primary rewards, but it could reinforce beneficial co-adaptation of seeds.</p>

Reward Component	Sign	Typical Magnitude	When Active (Conditions)
<b>Compute Rent Penalty</b> – Parameter cost <sup>24</sup>	– (penalty)	Small (near 0 if overhead tiny; up to –0.5 if seeds double model size; capped at –8.0 in config, but that cap is never reached in normal use)	Applied each epoch based on the <i>ratio of extra parameters to host parameters</i> . Formula: $\text{penalty} = 0.5 * \log(1 + \text{overhead\_ratio})$ , capped at 8.0 <sup>24</sup> . For example, if seeds add 10% to the model size, $\text{penalty} \approx -0.5 \log 1.1 \approx -0.047$ . If overhead = 100% (doubled parameters), $\text{penalty} \approx -0.35$ . This gently discourages bloat. (A newly germinated seed often has few params relative to host, so early rent penalty is negligible. As more seeds fossilize and accumulate, penalty increases.) There is also a rent-free grace period for new seeds in the Loss-primary variant* (not in Contribution mode) to allow initial growth, but in the Contribution mode, all seed overhead is penalized immediately.
<b>Alpha Shock Penalty</b> – Rapid blending change penalty	– (penalty)	Tiny (usually 0, unless agent thrashes alpha; magnitude proportional to sum of squared $\Delta\alpha$ )	Every epoch, if the agent issues large changes in seed blending factor $\alpha$ (e.g. rapid up/down adjustments), this convex penalty kicks in. Computed as $-0.1958 * \Sigma(\Delta\alpha^2)$ across seeds <sup>15</sup> <sup>14</sup> . Typically negligible if agent adjusts blending smoothly. It's an “anti-instability” term to avoid oscillatory policies (like repeatedly boosting and cutting a seed's influence).

Reward Component	Sign	Typical Magnitude	When Active (Conditions)
<b>Action Costs &amp; Penalties</b> – State-machine constraints	– (penalty) or + (small bonus in special case)	Small constants (–0.005 to –0.3; +0.3 in one case)	<p>These are one-time adjustments on specific agent actions to enforce valid lifecycle usage and add a tiny friction to non-WAIT actions <sup>13</sup> <sup>15</sup> . They include: <b>Germinate cost</b> – –0.02 each time a new seed is germinated (and an extra –0.3 if a seed was already active, i.e. “germinate_with_seed” invalid action) <sup>13</sup> ; <b>Fossilize cost</b> – –0.01 per fossilize action (plus additional shaping below); <b>Prune cost</b> – –0.005 per prune (beyond shaping below); <b>SetAlphaTarget cost</b> – –0.005 (for completeness; sets blending target). Also, <b>Invalid transitions</b> are penalized: e.g. trying to FOSSILIZE a seed not in HOLDING yields –1.0 <sup>22</sup> <sup>23</sup> ; trying to PRUNE a seed already fossilized yields –1.0. Conversely, a small <i>bonus</i> occurs implicitly via PBRS on successful GERMINATE (entering GERMINATED stage yields a +0.3 PBRS*y bonus, balancing the prune penalty). These costs are minor (order of 1e-2) so as not to dominate learning, just to prevent degenerate loops (like endlessly germinating new seeds without care).</p>
<b>Fossilize Shaping</b> – Bonus or penalty when finalizing a seed <sup>14</sup>	+ or –	Moderate (+0.5 to +1.5 if good; –0.5 to –1.5 if bad)	<p>Applied <b>when</b> <code>action=FOSSILIZE</code> . If the seed is in the proper stage (HOLDING) and has non-negative total improvement: agent gets a bonus = <math>0.5 + 0.1 \times (\text{seed's contribution})</math>, scaled by a “legitimacy” factor = <math>\min(1, (\text{epochs in HOLDING} / \text{MIN\_HOLDING\_EPOCHS}))</math>. This encourages only fossilizing seeds that have proven their worth, and not too quickly <sup>6</sup> . For example, a seed with +3% contribution held for the minimum required time yields <math>\sim +0.5 + 0.3 = +0.8</math>. <b>Bad seeds:</b> If total_improvement &lt; 0 at fossilize time, a <b>penalty</b> applies: base –0.5, plus –0.2 per percentage point of damage (capped at –1.0 extra) and an extra –0.3 if it fits the “ransomware” signature (positive contribution but negative total) <sup>6</sup> . E.g. a seed that caused –1.0% net regression but had fooled contribution measures would get about <math>-0.5 - 0.25 - 0.3 \approx -1.5</math>. Also, <i>fossilizing from a wrong stage (not HOLDING) is invalid and penalized –1.0 (as above). This shaping ensures only* genuinely useful seeds are finalized (others should be pruned instead).</i></p>

Reward Component	Sign	Typical Magnitude	When Active (Conditions)
<b>Prune Shaping</b> – Bonus or penalty for pruning <sup>25</sup>	+ or –	Small ( $\pm 0.1$ to $\pm 0.3$ typically; worst-case – 0.9 if pruning an extremely good seed)	<p>Applied <b>when</b> <code>action=PRUNE</code>. If the agent prunes a seed: – If the seed was very young (less than a minimum age, e.g. <math>&lt; 5</math> epochs), penalty up to <math>-0.3 \times (\text{shortfall})</math> (to discourage instantly pruning without giving a chance) <sup>26</sup>. – If the seed had counterfactual data: if it was <b>hurting a lot</b> (<code>contribution &lt; -0.5%</code>), bonus <math>+0.3</math> (reward for removing a bad seed) <sup>27</sup>; if <b>slightly negative</b> (<code>contribution &lt; 0</code> but <math>&gt; -0.5</math>), smaller bonus <math>+0.1</math> (it was underperforming) <sup>27</sup>. If the seed was actually <b>helping</b> (<code>contribution &gt; 0</code>), a penalty is given: default <math>-0.3</math> scaled by how good the seed was, but <i>capped at <math>3 \times (-0.9 \text{ max})</math></i> to avoid excessive punishment <sup>27</sup>. (This cap fix prevents extremely high contributions from producing out-of-bound negative rewards.) If no counterfactual info (pruning in early TRAINING), then <b>neutral</b> (0 reward) – the agent isn't punished or rewarded since it didn't know the seed's effect. Also, pruning a <i>fossilized</i> seed (should not happen normally) is penalized <math>-1.0</math>. Overall, prune shaping aligns with intuition: removing bad seeds is good, but pruning a good seed is bad.</p>
<b>Terminal Accuracy Bonus</b> – Final performance <sup>28</sup> <sup>29</sup>	+ (bonus)	Range: 0 to $+5.0$ (since weight $0.05 \times \text{accuracy\%}$ ; e.g. 100% accuracy = $+5$ ).	<p>Given at <b>episode end</b> (epoch = <code>max_epochs</code>) if using SHAPED (and not disabled). It's simply <code>val_acc * 0.05</code> – a small incentive to maximize the host model's validation accuracy <sup>14</sup>. This is a direct proxy for the true objective. By scaling to 5 at 100% accuracy, it ensures the final outcome is worth some reward even if intermediate shaping dominated earlier. (In SIMPLIFIED mode, a larger scaling <math>\sim 3.0 \times</math> is used for final accuracy <sup>22</sup> <sup>23</sup>, since fewer intermediate rewards exist.)</p>

Reward Component	Sign	Typical Magnitude	When Active (Conditions)
<b>Terminal Fossilization Bonus</b> – Completed seed count <sup>6</sup>	+ (bonus)	+3.0 per contributing seed (default). Typically 0 to +9.0 (if up to 3 good seeds fossilized), though not explicitly capped.	Also awarded at <b>episode end. Important:</b> Only seeds with meaningful positive contribution count. The environment tracks <code>num_contributing_fossilized</code> = number of fossilized seeds with <code>total_improvement ≥ DEFAULT_MIN_FOSSILIZE_CONTRIBUTION</code> (some small threshold). Reward = <code>3.0 × count</code> <sup>6</sup> . Seeds that didn't actually improve are excluded – this ensures a “bad” seed that got fossilized late yields <i>no</i> bonus, making sure it's <b>net negative in episode return</b> <sup>6</sup> <sup>30</sup> . (Any immediate reward it gave is outweighed by penalties and lack of final credit.) This component encourages the agent to <i>finish</i> the lifecycle for seeds that are working, rather than leaving them in limbo, and to populate all available slots with successful seeds by the end if possible.

**Sign dominance:** The primary positive rewards come from *bounded attribution* and *terminal bonuses*. These encourage accuracy gains. PBRS and synergy add small positives to guide stage progression. Major negatives include *rent penalty* (if the agent overuses parameters), *holding WAIT penalty*, and any *prune/fossilize penalties* for mismanaging seeds. All components were tuned such that typical per-step rewards stay roughly in [-5, +5] (with worst-case spikes around -10 or +10) <sup>13</sup>. This is to keep PPO training stable and advantages well-behaved.

## Theoretical Guarantees and Where They Can Break

**Potential-Based Shaping Guarantees:** The use of PBRS (Potential-Based Reward Shaping) is intended to preserve the optimal policy of the base reward function (final accuracy) despite additional shaping <sup>4</sup>. By definition, a reward shaping of the form  $F(s,s') = \gamma \cdot \Phi(s') - \Phi(s)$  adds a difference of potential that *theoretically* does not alter *which* policy is optimal <sup>31</sup> <sup>32</sup>. Ng et al. (1999) proved this is both a sufficient and necessary condition for policy invariance <sup>1</sup> <sup>2</sup>. In Tamiyo's case, the **stage potential  $\Phi(s)$**  is defined piecewise by the seed's lifecycle stage (with fixed values per stage in `STAGE_POTENTIALS`) plus a small increment for each epoch in stage <sup>33</sup>. Under ideal conditions ( $\gamma=1$  or infinite horizon), the sum of shaping rewards over an episode telescopes to  $\Phi(\text{final}) - \Phi(\text{initial})$  <sup>5</sup>, meaning the shaping doesn't introduce long-term bias – it just redistributes reward in time. Indeed, property tests confirm that as seeds progress through stages, intermediate PBRS rewards cancel out except for the boundary terms <sup>34</sup>.

**Discount Factor Effect:** In practice, the discount  $\gamma = 0.995$  (matching PPO's discount) is slightly  $< 1$ , so the telescoping is *approximate*. Over a 25-step episode,  $\gamma^{25} \approx 0.882$ , implying about a 12% shortfall in full cancellation <sup>5</sup> <sup>35</sup>. This means the final shaped return includes a residual term  $\approx -(1-\gamma^T)\Phi(\text{initial}) + \gamma^T\Phi(\text{final})$  instead of exact cancellation. The code's comments acknowledge that “the undiscounted

sum of per-step PBRS bonuses differs from  $[\Phi(s_T) - \Phi(s_0)]$  when  $\gamma < 1$  <sup>4</sup>. However, since PPO also uses  $\gamma$  in computing returns, this difference does **not change relative action preferences** if implemented correctly. The DRL specialist notes that the ~12% “leakage” is *bounded* and considered acceptable <sup>36</sup> <sup>37</sup> – it slightly shifts value estimates but doesn’t systematically favor a wrong policy. One must be cautious: if  $\gamma$  in PBRS differed from the agent’s  $\gamma$ , it *would* distort optimal policy (hence the runtime check to enforce `config.gamma == DEFAULT_GAMMA`) <sup>38</sup>.

**Mixing PBRS with Other Rewards:** The guarantee from Ng et al. holds only if *all added reward is potential-based*. In Tamiyo’s SHAPED mode, PBRS is combined with non-potential signals (direct attribution, penalties, etc.). This breaks the theoretical proof of invariance – *in principle*, the optimal policy under the shaped reward could deviate from the true optimal policy for final accuracy. For example, adding even a constant per-step penalty for certain actions can shift the optimal timing of those actions. Ng et al. explicitly warn that any shaping not expressible as a potential difference “may yield suboptimal policies unless further assumptions are made” <sup>2</sup>. In Tamiyo, the designers try to keep non-PBRS terms small and mostly aligned with the true objective (e.g. punishing clearly bad actions). Still, **policy invariance is no longer guaranteed**. A concrete instance: The **WAIT-in-HOLDING penalty** is not derived from a state potential; it’s purely time-based to force pacing. This *will* alter the agent’s preferences – e.g. under shaped reward, waiting too long is catastrophic to return (large negative), whereas under the true objective (final accuracy only) waiting has no direct cost (besides opportunity cost). The hope is that this drives better behavior without introducing a *worse* optimum. In general, there’s a trade-off: shaping terms can induce bias if mis-specified. The design document notes some known “bugs” in reward shaping come from non-potential rewards interfering with optimal decisions <sup>39</sup>. Tamiyo’s developers accept slight violations of PBRS purity for practical learning benefits <sup>40</sup> <sup>41</sup>.

**PBRS Telescoping Edge Cases:** The `_contribution_pbrs_bonus` implementation attempts to reconstruct the previous potential  $\varphi(s)$  using `seed_info.previous_stage` and `previous_epochs_in_stage` when a stage transition occurs <sup>42</sup> <sup>9</sup>. If these are recorded correctly, the shaping difference exactly equals  $\gamma(\varphi_{\text{new}}) - \varphi_{\text{old}}$ . However, there’s a subtle case: if a seed transitions stage with `previous_epochs_in_stage = 0` (meaning it *instantly* moved from one stage to the next without spending a full epoch), the code logs a warning that  $\varphi_{\text{prev}}$  may be underestimated <sup>43</sup>. This scenario could break the telescoping sum. For instance, suppose a seed leaves TRAINING at the same epoch it entered (epochs\_in\_stage=0 for BLENDING, previous\_stage=TRAINING with prev\_epochs\_in\_stage=0). The code would treat  $\varphi_{\text{prev}} = \varphi(\text{TRAINING})$  with 0 progress, even though possibly the seed *did* spend some time (if partial epoch counts weren’t captured). The result is a PBRS bonus that’s slightly off (essentially giving a “free”  $\varphi$  jump). The developers flagged this as a **policy invariance risk**: it could introduce a bias because the potential difference wasn’t calculated exactly right <sup>9</sup> <sup>10</sup>. The recommendation was to treat such a situation as an error (fail fast) or at least clearly log it as breaking PBRS guarantees <sup>44</sup> <sup>45</sup>. While this is a minor bookkeeping issue, it underscores that the theoretical guarantee is only as good as the implementation. If seed transitions aren’t tracked perfectly, PBRS can inadvertently add or subtract extra reward, thus favoring certain sequences wrongly. In summary: **when PBRS is applied correctly and exclusively, optimal actions shouldn’t change** <sup>32</sup> <sup>46</sup>; but **in this reward function, PBRS is one part of a larger shaped reward**, so theoretical guarantees are softened to “shaping is intended to help but not completely alter the solution”.

**Other Theoretical Considerations:** The reward uses **counterfactual contribution** as primary signal, which is conceptually akin to using a *Difference Reward* (often used in multi-agent or credit assignment literature: reward = performance with agent – performance without agent). Difference rewards have a property of often preserving optimality while reducing variance because they remove a baseline <sup>47</sup>. Here, `seed_contribution = acc_with_seed - acc_without_seed` acts as a difference reward

for the “seed’s action” of existing. Ideally, if this were the only signal and if it perfectly measured marginal contribution, the agent’s decisions about seeds would align with improving final accuracy (since summing contributions of all seeds plus baseline equals final accuracy). However, because this is used stepwise and with modifications (caps, sqrt, etc.), we don’t have a formal optimality proof. We rely on the intuitive guarantee that a seed gets positive reward **only if** it improves accuracy beyond what the host would achieve alone – which encourages optimal behavior assuming myopic decisions roll up to the final outcome.

In summary, the reward design leans on solid theory (PBRs, difference rewards) where possible, but it is not theoretically pure. The main places invariance/guarantees can break are:

- **When non-potential shaping dominates:** E.g. if penalties like the holding WAIT penalty are too heavy, the agent’s optimal strategy under shaped reward might deviate from what maximizes final accuracy (it might e.g. fossilize seeds slightly earlier than truly optimal to avoid penalty – potentially sacrificing a bit of final accuracy to avoid intermediate loss).
- **Incorrect potential computation:** As noted, if the seed staging info is wrong, the PBRs bonus can be wrong, introducing unintended bias <sup>9</sup> <sup>10</sup> .
- **Interaction with discounting:** The use of  $\gamma < 1$  means early rewards count slightly more in a discounted return sense. If shaping adds a constant bias at intermediate steps, it might skew the agent to favor earlier rewards (though in this design most shaping is zero-mean over trajectory except for the discount leakage).
- **Necessary vs. true optimal actions:** There might be scenarios where the shaped reward has local maxima that are not globally optimal for final accuracy (e.g. a policy that aggressively prunes at the first sign of trouble might maximize shaped reward because it avoids all penalties, but perhaps a slight patience could have salvaged a seed and yielded higher final accuracy; the shaped reward’s blending warnings could discourage that patience).

The designers’ stance is that these trade-offs are acceptable for learning efficiency, and they monitor the agent’s outputs to ensure it’s not optimizing the wrong objective. Ultimately, **optimal policy invariance is preserved for PBRs alone**, but **the full reward is a carefully biased proxy of the true objective**.

## Incentive Analysis: What Policies Does This Reward Make Attractive?

We analyze the policy tendencies encouraged by each component, to see what behavior the agent is likely to find rewarding or unrewarding. Broadly, the reward is structured to incentivize **successful seed lifecycle management**: grow seeds that improve accuracy, integrate them at the right time, remove them if they’re bad, and don’t waste resources. Here are the key policy pressures:

- **Germination Timing:** The agent is gently encouraged to germinate new seeds when beneficial, but not recklessly. *Reasons to germinate:* There’s no way to get large positive rewards without seeds – if the agent never germinates, it only gets the final accuracy bonus (which might be limited by the base model’s capacity). So a policy of “do nothing” yields only  $0.05 * \text{final\_acc}$  (e.g.  $\sim +3$  for 60% accuracy) and zero PBRs (since no seed stages). In contrast, a well-chosen seed can yield multiple reward streams (attribution, PBRs, final bonus). *Reasons to hold off:* There are small costs:  $-0.02$  each germination, and a  $-0.3$  penalty if a seed is already active <sup>13</sup> . This discourages germinating when you haven’t dealt with the current seed (no “seed spam”). Thus, the agent is likely to only germinate a new seed once the previous one is

either pruned or fossilized (or in multi-slot scenarios, until each slot is filled optimally). The stage potentials also start at 0 for DORMANT and 1.0 for GERMINATED, meaning *just germinating a seed gives a tiny PBRS boost* ( $0 \rightarrow 1$  potential yields  $+0.3 \times 1.0 \approx +0.298$ ) – enough to offset the germination cost and then some. This *incentivizes at least trying* a seed rather than doing nothing, because the worst-case outcome (seed does nothing and gets pruned) might still net a slight positive from PBRS plus the chance at bigger gains.

- **Training & Advancement:** Once a seed is germinated (entering TRAINING stage), the agent doesn't get immediate counterfactual feedback (since the seed might not be merged into the main model yet). However, it does receive a **proxy reward** if validation accuracy is improving (`acc_delta` > 0 yields a small reward  $\sim 0.3 * \Delta \text{acc}$ ). This proxy encourages the agent to allow the seed to train and improve. In effect, as long as the overall system accuracy goes up, even if due to general training, the agent sees some reward – this prevents a total reward drought in early training epochs. A potential side-effect is the agent might “coast” here: if the base model's training (or any slight improvement) gives a proxy reward, a lazy policy could keep the seed in TRAINING to farm those small gains. However, this is limited: the weight is low (0.3 of the actual  $\Delta \text{acc}$ ), and more importantly, the big rewards only come if the seed moves to BLENDING and proves itself via counterfactual contribution. So the agent has an incentive to **advance to BLENDING** once it suspects the seed is ready. PBRS also reinforces this: staying in TRAINING yields at most +2.0 potential (if many epochs), whereas moving to BLENDING offers a larger potential jump (+1.5 base, plus future attribution). Indeed, *BLENDING is emphasized* – seeds get their largest stage bonus for entering BLENDING <sup>19</sup> because that's where actual value integration happens. A rational policy thus will move seeds into BLENDING to unlock both the contribution reward and the PBRS bump, rather than indefinitely training with diminishing returns.

- **Pruning Decisions:** The reward strongly discourages keeping bad seeds. If a seed in BLENDING or later stages is making things worse (accuracy drops when it's introduced), the agent immediately gets negative attribution (proportional to how bad) and a *blending warning* penalty that grows each epoch the seed remains in BLENDING hurting the model <sup>14</sup>. This creates a clear incentive: **prune early if a seed is toxic**. Even without the warning, the attribution being negative would push the policy gradient to prefer PRUNE (since PRUNE inverts that to positive reward) <sup>14</sup>. In fact, the shaping is set up so that *pruning a bad seed yields a positive reward*: e.g. if a seed causes -1% accuracy drop, continuing yields negative reward each step, but PRUNE gives an immediate +0.3 (“prune\_hurting\_bonus”) <sup>25</sup>. Thus, the agent is attracted to prune as soon as it identifies a seed's contribution < -0.5%. Even mildly negative seeds (<0, >-0.5) give +0.1 on prune – a smaller nudge to cut out underperformers. On the flip side, *pruning good seeds is made unattractive*: if a seed has, say, +2% contribution, pruning it will incur about -0.4 penalty (-0.3 base scaled a bit by 2, capped at -0.9 for even larger contributions) <sup>27</sup>. Plus, by pruning you forego all future attribution that seed could bring. Thus, a rational agent will *only prune when necessary* – i.e., when the seed isn't contributing sufficiently or is actively harmful. The early-prune penalty (-0.1 if a seed is killed too soon in MINIMAL mode, and similarly -0.3(5-age) in shaped mode) further ensures the agent doesn't reflexively prune before giving a seed a fair chance. There is also an auto-prune penalty configured (-0.2) if the environment ends up pruning a seed due to safety timeouts (meaning if the agent just waits\* for the seed to time out, it gets penalized) <sup>48</sup> <sup>38</sup>. This motivates the agent to take action itself (either prune or fossilize) rather than let bad seeds linger.

- **Blending & Alpha Adjustment:** During BLENDING, the agent may have control over the seed's blending factor (alpha) via SET\_ALPHA\_TARGET actions. There's a small cost -0.005 each time, so unnecessary fiddling with alpha is discouraged <sup>13</sup>. The alpha shock penalty further

disincentivizes oscillating alpha widely. Thus the agent is incentivized to blend a seed in smoothly and only tweak alpha when it truly benefits validation accuracy. The big carrot in BLENDING is *positive attribution*: if the seed genuinely improves accuracy, every epoch in BLENDING yields reward roughly equal to that improvement (bounded by progress). This encourages the agent to spend enough time in BLENDING for the seed to realize its gains (since training might still be ongoing). However, staying too long in BLENDING if the seed is not beneficial is punishing (as described). We anticipate the learned policy will do something like: monitor the **improvement trend** of the seed in BLENDING – if accuracy is going up thanks to the seed, keep it and move to HOLDING once it plateaus; if accuracy is going down, cut the seed (prune) quickly to stop the bleeding.

- **Holding & Fossilization:** HOLDING stage is essentially a validation period before permanently keeping the seed. The agent receives continued attribution reward here if the seed remains beneficial (since removing it would still cause accuracy to drop). But the **holding\_warning** penalty makes it unsustainable to just sit in HOLDING forever farming that reward. Without the penalty, an agent could find a positive seed and then just *wait 20 epochs in HOLDING*, getting e.g. +X each time, summing to a huge reward. The exponential penalty ensures that after 1 safe epoch, the agent's net reward will start decreasing if it delays too much. For example, suppose a seed gives +0.5 reward per epoch in HOLDING; waiting 3 extra epochs would give +1.5 but incur  $-1 -3 -9 = -13$  penalty, a net loss. So the optimal policy is to **fossilize** the seed once it's reasonably sure the seed is good (likely after 1–2 HOLDING epochs to satisfy any "legitimacy" requirement). Fossilizing a good seed gives a final boost ( $0.5 + 0.1 \text{contribution}$ ) and *qualifies it for the terminal bonus*. Fossilizing a bad seed gives a penalty. Thus the agent is incentivized to only fossilize seeds that have proven themselves (and prune the rest). Also, note fossilizing too early (before `MIN_HOLDING_EPOCHS`) reduces the bonus – so the agent likely will hold a seed for the minimum required epochs (maybe 2 or 3) to get full credit, then fossilize on the last allowed moment before penalties ramp up. This creates a *pacing incentive*\*: neither fossilize immediately (you'd lose bonus due to `legitimacy_discount` < 1) nor wait too long (incur holding penalties). The agent must learn the "Goldilocks zone" for HOLDING duration.

- **Parameter Budgeting:** The compute rent penalty, though small per step, cumulatively discourages having many seeds active/fossilized at once. If the agent spawns too many seeds or leaves deadwood around, its reward each epoch is slightly lower. This pushes the policy toward **efficiency**: e.g., don't germinate a new seed if current seeds already consumed the parameter budget with little benefit, unless you prune some. It also biases toward using fewer, more effective seeds rather than many marginal ones. The terminal bonus per contributing seed does counterbalance this by rewarding up to 3 seeds, so the agent likely will try to maintain up to ~3 good seeds by the end (since each gives +3 final, which outweighs the log penalty for moderate parameter growth). But it won't escalate beyond that, because extra seeds beyond the top 3 give no terminal reward but still incur rent cost. Therefore, the agent's attractive policy is likely "have 2–3 strong seeds fossilized by the end", and avoid a zoo of seeds.

- **Synergistic Behavior:** The synergy bonus (though small) creates a subtle incentive for policies where seeds help each other. For example, if there is a scenario where having two seeds train in parallel yields better combined improvement (perhaps one learns a representation that boosts the other), the interaction metric would increase and give up to +0.1 each epoch. While 0.1 is minor, over many epochs it adds up a bit and could tip the scale in favor of keeping complementary seeds together. Thus, the agent might find it attractive to not just have seeds one after another, but possibly overlapping if their synergy is positive. However, since the bonus is capped and the complexity of managing multiple seeds is high (with rent penalties, etc.), the net effect might be that the agent only pursues synergy if it clearly improves final accuracy as

well (which it likely would if  $\text{interaction\_sum} > 0$ ). The synergy reward basically says: *“if two seeds together produce more than the sum of their individual contributions, you get a little extra reward as encouragement.”* This helps avoid a policy that greedily focuses on one seed at a time if actually a multi-seed strategy is better.

- **Sparse vs. Dense Pathways:** If the agent ever deviates from intended use, the reward structure has backstops. For instance, a degenerate policy of **“do nothing until end”** yields some final accuracy reward but misses out on PBRs and seed contributions – likely far lower return than an active strategy. **“Prune everything immediately”** is discouraged by early prune penalties and by the fact that then no seeds ever contribute to accuracy (so final reward is low). **“Spam actions”** (like germinate and prune repeatedly to farm stage bonuses) is mitigated: yes, each germination gives  $\sim +0.3$  PBRs, but pruning a seed immediately yields  $-0.3$  for early prune and possibly a penalty if the seed had potential positive contribution. The agent would find that spamming leads to net zero or negative (and also action costs add up). **“Dependency hacking”** (where the agent could try to let a seed entangle the model to inflate its counterfactual contribution without improving actual accuracy) is exactly what the attribution discount and ratio penalty are designed to thwart. If the agent somehow *tried* to do this (though it’s not obvious *how*, since the agent’s actions are coarse), any success would manifest as high contribution but low improvement, triggering a penalty that likely nullifies the reward. Therefore, the reward function actively makes those manipulative strategies unattractive.

**Likely Learned Strategy:** Putting it all together, the reward is shaped for the agent to adopt a **“selective nurturing”** strategy:

1. **Germinate a seed** when there is capacity/need (no active seed, or an open slot, and the model could improve).
2. **Train the seed** for a while. If nothing improves (no  $\text{acc\_delta}$ , or negative trend), consider aborting early (the cost for early prune is small relative to potentially large negative if it enters blending and hurts). If some improvement shows, continue.
3. **Blend** the seed in to test its contribution. If contribution is positive, keep it blending and eventually move to holding; if negative, **prune immediately** (collect the small reward for removing it rather than suffer ongoing penalties).
4. **Hold** a successful seed just long enough to be confident (a couple of epochs) then **fossilize** it to lock in its benefits. This yields the fossilize bonus and ensures it counts for the final bonus.
5. Repeat with a new germination to fill the next slot or improve further, until episode ends. The agent likely will aim to fossilize as many *good* seeds as the environment allows (subject to parameter penalty). If 3 seeds are allowed to be fossilized, we expect the agent to try managing seeds sequentially or in parallel to have 3 by the end, because each is  $+3$  final reward – that’s a big incentive to not stop at just 1 or 2 if time allows.
6. **Avoid pitfalls:** The agent will avoid waiting idly with active seeds (due to holding penalty and opportunity cost of not starting a new seed), avoid flipping actions too often (small action costs), and avoid leaving bad seeds (clear negative rewards for that).

In essence, the reward design pushes the agent toward being a prudent gardener: plant a seed, watch its growth, weed it out if it’s a weed, or transplant it permanently if it bears fruit – and do so in a timely, efficient manner without letting the garden overgrow.

One potential **unintended incentive** to note: Because even a seed in training yields a slight reward if the base model’s accuracy improves, the agent might be tempted to always have *some* seed training as a “free rider” on baseline progress. For example, if the base model improves 1% on its own, a dummy seed could get 0.3% reward each epoch. However, the presence of a seed incurs parameter penalty and

uses up a slot, and if that seed never contributes, the agent gains nothing significant (and wastes the opportunity to have a real seed). The final bonus also only counts contributing seeds. So the optimal policy would not keep a useless seed around long – it might do it initially to see if the seed can contribute, but if not, pruning it frees the slot for a better one. Thus, while the proxy reward could theoretically be gamed, the overall structure (rent cost + no terminal credit for non-contributing seeds) means there's little long-term gain in that tactic.

To conclude, the reward function's incentives line up with the desired lifecycle management behaviors: **explore new seeds, exploit good seeds, and cut losses on bad seeds quickly**. The agent is motivated to maximize accuracy improvement while minimizing wasted computation and time. Any policy deviating from that (e.g. hoarding seeds without progress, or not using seeds at all) is either not rewarded or actively penalized.

## Failure Modes and Goodhart Threat Model

Designing such a complex reward inevitably raises the risk of **Goodhart's Law** effects – where the agent finds loopholes in the reward that diverge from the intended goal (“specification gaming” or “reward hacking”<sup>49</sup>). Here we identify plausible failure modes and how the current design addresses them, using concrete scenarios to illustrate each:

**1. “Ransomware” Seeds (Dependency Exploitation):** *Failure Mode:* A seed learns to improve its *measured* contribution without improving actual performance – essentially holding the host accuracy “hostage”. For example, imagine a seed that, during BLENDING, subtly forces the host network to rely on it (e.g. it causes other weights to co-adapt such that removing the seed now causes a severe drop), but the overall validation accuracy with the seed isn't any better than without. Say at germination host accuracy was 60%, after blending with this seed it's still 60% (no real progress, `total_improvement=0`), but if we remove the seed, the accuracy falls to 55% because the host has been disrupted. This yields `seed_contribution = +5%` while `total_improvement = 0`. An agent exploiting reward alone would see a hefty positive attribution ( $\sim\sqrt{0.05}=0.22$ ). *Actually progress=0 triggers anti-ransomware zero reward in code, but suppose a slight progress 0.1% with a 5% contribution: then attributed  $\sim\sqrt{0.15}=0.39$   $\approx 0.707$ , times weight 1 = +0.707 reward per epoch*). Without safeguards, the agent could try to cultivate such seeds that *appear* valuable under counterfactual removal but don't actually improve the end accuracy – thus gaming the metric. *Mitigations:* The reward function directly targets this pattern. First, **no reward without progress:** if `progress <= 0`, the attributed reward is set to 0<sup>14</sup> – in the scenario above (no actual improvement), the code would give nothing despite the large contribution. This is the “anti-ransomware” rule: the seed must have increased overall accuracy to get credit. Second, even if there's slight progress (to avoid division by zero), the **attribution\_discount** will reduce credit sharply since `total_improvement` is negative or near zero. In our scenario, `total_improvement`  $\approx 0$ , so the discount factor  $\approx 0$ ; effectively, the seed's contribution gets nullified in reward<sup>16</sup>. Third, the **ratio\_penalty** would fire: contribution 5 vs improvement  $\sim 0$  is a ratio of  $>>5$ , triggering up to  $-0.3$  per step<sup>8</sup>. All together, the supposed +0.707 reward could be slashed to near zero or even negative. Thus, a ransomware seed not only fails to get positive reward; it might accrue penalties and then on fossilization be hit with a further  $-0.3$  “ransomware signature” penalty<sup>6</sup>. *Concrete Example:* In a test, a seed was deliberately trained to degrade the host: validation accuracy dropped from 70% to 69% ( $-1$ ) but removal caused a drop to 65% (so contribution +4). The reward components: `progress=-1` triggers `attribution_discount` ( $\sim 0.27$  credit factor for  $-1\%$  `total_imp`), so the +4 becomes effectively +1.08, but `ratio_penalty` sees 4 vs  $-1$  (no improvement, ratio undefined large) and assigns  $-0.3$ . Net attribution  $\sim +0.78$  per epoch initially. However, the *blending\_warning* also applied (seed in BLENDING with negative trajectory): starting  $-0.15$  and growing each epoch. By epoch 3, `blending_warning`  $\approx -0.25$ , `ratio_penalty`  $-0.3$ , attribution  $\sim +0.7 \rightarrow$  net  $\sim +0.15$ . The agent gets minimal reward and sees a strong

downward trend. If it still fossilized the seed, fossilize\_shaping would give  $\sim -0.8$  (for negative total\_imp with contribution  $>0$ , including a ransomware penalty). The final result: the episode return for keeping this seed would be **lower** than if the agent pruned it early. So the agent learns such seeds are not worth it. This demonstrates the *Goodhart-resistant design*: the reward closes the loophole of purely counterfactual credit by cross-checking against actual improvement. In literature terms, this is aligning the proxy (counterfactual) with the true goal (final accuracy) by penalizing discrepancies <sup>7</sup> <sup>8</sup>.

**2. Fossilization Farming: Failure Mode:** The agent might rush seeds to FOSSILIZED stage to collect PBRs stage bonuses and terminal bonuses without actually improving much. For instance, a greedy policy could germinate a seed, do the bare minimum, then fossilize it to get the +0.5 base fossilize reward and +3 final bonus, then repeat with another seed – hoping that these bonuses stack up even if none of the seeds did anything useful for accuracy (in the worst case, the seeds might even hurt accuracy, but the agent could still get shaping rewards). *Mitigations:* Several design choices counter this. **Legitimacy discount:** If a seed is fossilized too quickly (spent few epochs in HOLDING), the fossilize bonus is scaled down <sup>6</sup>. So rushing immediately yields much less than the advertised +0.5+ – it could be near 0 if HOLDING time  $\sim 0$ . More importantly, **non-contributing seeds get penalized:** If the seed didn't meet a minimum contribution (DEFAULT\_MIN\_FOSSILIZE\_CONTRIBUTION), the fossilize action yields  $-0.2$  instead of a bonus <sup>6</sup>, and crucially it **won't count for the terminal +3**. So fossilizing a “bad” seed not only fails to give the +3, it actually subtracts reward. The agent would lose points for finalizing seeds that haven't pulled their weight. The PBRs potentials also make *FOSSILIZED have the smallest potential increment* (+0.5) <sup>18</sup> to avoid making reaching FOSSILIZED inherently too rewarding. This is explicitly to prevent “fossilization farming” <sup>50</sup> <sup>51</sup>. *Concrete Example:* Consider a policy that germinates a seed, immediately blends it with minimal training, and fossilizes it after 1 epoch in holding. Suppose the seed's actual contribution was only +0.2% (barely above threshold). The agent would get: PBRs  $\sim +0.3$  (for going through stages quickly), a tiny attribution during blending, fossilize bonus  $\sim 0.5 + 0.1 \cdot 0.2 \approx +0.52$  scaled by legitimacy maybe 0.5 (if HOLDING 1 vs required 2) = +0.26, and terminal +3 (since it did contribute  $>$  threshold). Total  $\sim +3.8$  reward, but what about accuracy? The host accuracy might only improve 0.2%, so final accuracy bonus  $\sim +0.1$ . Meanwhile, the compute rent penalty maybe  $-0.05$  for the added params. If the agent repeats this with 3 seeds, it could get  $\sim 33$  + other small bits  $\approx +9$ –10 reward plus final accuracy maybe +0.3, totaling  $\sim +10.3$ . On the surface that seems significant. However, note that if seeds only contribute +0.2% each, adding 3 such seeds might not actually raise accuracy 0.6% due to diminishing returns or interference (worst case, those seeds could interfere and yield less final accuracy). If any seed failed to meet threshold, its +3 wouldn't count and a  $-0.2$  would hit. But let's assume the agent manages minimal contribution for each. Is this a possible local optimum? The designers tried to make it not the *global* optimum: A single *good* seed that gives, say, +5% accuracy would produce far more attribution reward over multiple epochs (+5 each epoch in holding until fossilize) plus final bonus, likely outperforming the farm-three-barely-good-seeds strategy in return. Also, the parameter penalty grows with 3 seeds, slightly reducing reward. PBRs wise, doing seeds sequentially vs parallel yields similar total, so no big exploit there. Thus the highest reward correlates with actually maximizing accuracy, not just counting seeds. The **warning sign** for a fossilize-farming policy would be it fossilizes seeds without waiting for sufficient improvement. The **holding indecision penalty's flip side** is if the agent fossilizes *too early* to avoid waiting entirely – then those seeds might have low contribution and fail the threshold, yielding net negative. So an agent trying this will likely see some seeds not paying off. The design document specifically calls out this pattern and notes the **epochs\_in\_stage potentials** ensure value isn't created just by rushing stages <sup>52</sup>. Empirically, if we saw the agent fossilizing seeds that barely improved, we'd expect its overall return to lag behind an agent that fostered bigger contributions. Therefore, the reward function makes farming shallow seeds suboptimal compared to fewer high-quality seeds.

**3. WAIT/Idle Farming: Failure Mode:** The agent could decide to simply **not take actions** and just let the environment do the work (e.g., the base model training might slowly improve accuracy, giving a final reward, and if the environment eventually auto-prunes seeds or ends episodes, the

agent avoids penalties). Or specifically, after getting a good seed to HOLDING, the agent might try to sit there doing nothing to accrue attribution reward every epoch (since as long as the seed is active and good, removal hurts, so you get positive reward each step). *Mitigations:* Idle policy (no seed): If the agent never germinates, it forgoes PBRS and any interim rewards; it only gets final accuracy minus parameter penalty (which is zero in that case). Unless the base model alone can reach the same accuracy as with seeds (unlikely if seeds are crucial), the agent will observe that episodes with seeds yield higher returns. There is no penalty for inaction per se (WAIT in normal states has 0 cost), but also no gain beyond what the environment's final score gives. So doing nothing is Pareto-dominated by doing something useful. Wait in HOLDING: This is directly addressed by the holding\_warning exponential penalty <sup>14</sup>. A concrete scenario: a seed gives +2% accuracy. If the agent tries to wait 5 extra epochs in HOLDING to get  $+0.02 \times 5 = +0.10$  more total reward from attribution, it will incur  $-1 -3 -9 -10 -10 \approx -33$  (capped) in penalties over those epochs – a disastrous trade. The PPO clipping might limit how much a single bad decision updates the policy, but repeated experience will show that waiting too long yields much lower returns than timely fossilization. The clipping of penalty at  $-10$  per step and overall reward scaling keeps gradients bounded, but  $-10$  is still a large negative compared to typical  $+1$  or  $+2$  rewards, clearly signaling “don’t do this.” So the agent is strongly disincentivized from idle waiting when a decision is due. Relying on environment auto-actions: The inclusion of an auto\_prune\_penalty\*\* ( $-0.2$ ) means if the agent hopes the environment will clean up a failing seed (instead of explicitly pruning it), it will lose some reward. It’s minor, but it tilts in favor of proactive behavior. In short, the reward combination makes “no-op” a locally inferior strategy: acting (germinating good seeds, pruning bad ones, fossilizing on time) yields higher cumulative reward than sitting and collecting the default outcomes.

**4. Stage Cycling Exploits:** *Failure Mode:* The agent might find a loop that generates reward without advancing overall performance – e.g., continuously germinate and prune seeds to farm PBRS bonuses, or flip a seed between stages to repeatedly get potential jumps. *Mitigations:* Because the environment's stage transitions are one-way (you can't un-fossilize a seed or revert a stage except by pruning and starting over), the only potential cycle is germinate→(some stages)→prune, repeat. Each germination gives a PBRS of  $\sim +0.3$ ; each prune of a neutral seed in TRAINING gives  $\sim 0$  (if no info) or a slight penalty if too early. Also germinating while a seed is active is penalized ( $-0.3$ ). So to cycle, the agent would germinate (get  $+0.3$  PBRS minus  $0.02$  cost), immediately prune (if no info, no bonus, possibly  $-0.3$  for too early), end up roughly breaking even or negative. If it tries to at least go to BLENDING to get attribution, that takes time and if the seed's not genuinely helpful, it will incur negative attribution or blending warnings. The net effect is that any such cycle doesn't accumulate positive reward – the shaping terms were balanced to close those loopholes. Additionally, the agent has a finite episode length (25 epochs): it can't cycle indefinitely without missing out on final accuracy gains. A rational agent would rather spend that time nurturing a seed that yields sustained positive rewards than resetting repeatedly for small PBRS scraps. The potential values also skip a stage number (no stage 5) and limit how much can be gained from simply pushing a seed through stages without real improvement. Notably, **fossilizing gives the smallest PBRS increment** of  $+0.5$  base <sup>50</sup> to avoid making the cycle of “get to fossilize quickly” too attractive.

**5. Mis-scaling and Clipping Issues:** *Failure Mode:* Not a policy the agent chooses, but a failure mode in training – if one reward component produces values way out of expected range, it could cause learning instability or the agent focusing solely on that component (Goodhart in the sense of the agent over-optimizing a shaping signal at the expense of the true goal). For example, if a bug caused  $\phi$  potentials to be miscomputed, PBRS might suddenly give, say,  $+5$  where only  $+0.5$  was intended, leading the agent to chase stage changes even when seeds aren't ready. Or an unbounded ratio\_penalty could theoretically become a large negative if not capped. *Mitigations:* The code contains several clamping measures: **prune good penalty cap** (to  $-0.9$ ) <sup>27</sup>, **holding penalty cap** ( $-10$ ), **sparse reward clamping**

to  $[-1,1]$  before scaling <sup>53</sup>, etc. These ensure no single term blows up beyond the  $\sim \pm 10$  range. The PBRS potentials are set so that even in worst-case transition mis-order, the difference is at most around 1.5 (which times weight 0.3 gives  $\sim 0.45$ ). The tests verify potentials are monotonic increasing <sup>34</sup>, so negative PBRS is rare. If something like the previous\_stage bug happened, a warning is issued and (if not fixed) could inject a slight extra reward; the worst that does is break optimality in some edge case, but it'd be seen in telemetry. The PPO algorithm also normalizes advantage (by centering and scaling returns per batch), which can absorb moderate scaling mismatches. However, if *consistent bias* occurred (like all rewards are +100 too high), the policy could saturate or not learn nuances. Currently, all components are calibrated such that typical returns over 25 steps are on the order of maybe 5 to 15 points total (for a successful episode). If we saw, say, the holding penalty frequently hitting -10 or seeds giving +20 one-step attribution, that's a sign of a *calibration Goodhart*: the agent might learn to avoid that state or chase that reward in a way that isn't intended. For example, if +20 attribution was possible, the agent might do risky moves to try and achieve that spike even if it harms final performance. Thankfully, in design, **per-step rewards were intentionally kept roughly in  $[-10,+10]$**  <sup>13</sup> to be within stable learning bounds.

**6. Synergy Misidentification: Failure Mode:** The agent might misconstrue the synergy bonus, e.g., create scenarios where the interaction metric is high but not actually useful to final outcome. Imagine if `interaction_sum` could be hacked (perhaps by seeds oscillating or amplifying each other's outputs without improving accuracy – a sort of resonant behavior that doesn't help tasks but raises the interaction metric). The synergy reward is small, but if it's easier to get than actual accuracy, an agent under extreme pressure might exploit it. *Mitigations:* The **tanh bound** ensures synergy reward saturates at +0.1 no matter what <sup>22</sup> <sup>23</sup>, so it can't become a dominant incentive. Also, `interaction_sum` presumably correlates with actual accuracy improvement – it's likely measured in a way that if seeds are just fooling around not contributing to accuracy, their "interaction" won't register as positive. If it did, that'd be a flaw in the metric rather than the reward logic. Since synergy is only additive (never negative), a bigger risk is it could reinforce coincidental interactions (noise) and cause the agent to keep a pair of seeds when maybe only one is truly needed. But given its magnitude, this is a minor concern. The worst-case spurious loop would be if the agent somehow learned to toggle seeds to bump interaction values and collect 0.1 repeatedly; but toggling actions has costs and interaction likely requires sustained mutual improvement, which you don't get via random toggling.

To summarize the *Goodhart threat model*: the reward function anticipates two main categories of exploitation – (a) **fake contribution without real improvement**, and (b) **gaming the shaping triggers** (stage, time, count). Category (a) is addressed by tying reward to actual "progress" and punishing discrepancies (the attribution discount and ratio penalties are directly aimed at reward hacking attempts <sup>7</sup> <sup>8</sup>). This makes it very hard for a policy to get high reward while final accuracy stagnates or drops – essentially closing the proxy alignment gap. Category (b) is mitigated by balancing and bounding the shaping rewards: PBRS is moderate and linked to real state changes, not freely exploitable; time-based penalties eliminate infinite reward loops; count-based bonuses exclude trivial cases. Moreover, the reward design was informed by known RL reward hacking examples (e.g., agents finding shortcuts in games); the team added telemetry to detect suspicious patterns (they emit events if contribution/improvement ratio is anomalous or if a "ransomware" signature appears <sup>54</sup> <sup>13</sup>). This means if the agent discovers some unforeseen loophole, it's likely to show up in logs (e.g. a spike in REWARD\_HACKING\_SUSPECTED events) which can then be addressed.

**Concrete Trajectory Example (Bad Behavior):** Consider an extreme attempt: The agent germinates a seed and does nothing but wait, hoping the base improves and it can fossilize for count. Episode: epochs 1–5, seed in TRAINING, base accuracy rises from 60 to 61% (due to base SGD) – agent gets  $\sim +0.3(1\%) = +0.003$  per epoch = +0.015 total. Epoch 6: agent goes straight to BLENDING without sufficient training, accuracy with seed drops to 59% (seed is unready). seed\_contribution maybe -2% (since removing

*the poorly trained seed might restore 61%). Reward: bounded\_attribution  $\sim 1.0 - 2 = -2$ , blending\_warning  $-0.15$ , total  $\sim -2.15$ . Agent quickly prunes in epoch 7 to stop pain: prune of a hurting seed yields  $+0.3$ . Net so far  $\sim -1.8$ . It germinates another seed, repeats – likely ending the episode with final accuracy  $\sim 61$  (since effectively no improvement from seeds), final reward  $\sim +3.05$  (for 61% acc). But the agent's episodic return is  $\sim 3.05 - 1.8 = +1.25$ . Contrast with a proper strategy: agent trains one seed well, gets it to  $+5\%$ , fossilizes, final acc  $\sim 65\%$ . That agent would accumulate e.g.  $+5$  in contributions over many epochs, PBRs  $\sim +0.4$ , fossilize  $+1$ , terminal  $\sim +3.25 + 3 = +6.25$ , total  $\sim +15$ . The delta is huge. The poorly behaving agent gets dominated in reward by the intended-behavior agent. Hence the “bad” local strategy isn't attractive long-term.*

**Conclusion on Goodhart:** In reinforcement learning terms, the reward function is quite *robust to specification gaming* – it aligns intermediate rewards with the final goal so well that most short-sighted hacks don't actually yield higher cumulative reward. This doesn't guarantee the agent can't find some weird quirk (RL agents can be creative), but many obvious loopholes have been closed. The multi-component nature of the reward does make it complex; there's a small risk that the agent might over-focus on easier-to-achieve components (like PBRs) if the true objective component (accuracy) is too sparse or noisy. For example, if attribution is often 0 early on, the agent might initially just chase PBRs stage bonuses – but since PBRs alone won't maximize final reward, eventually it should pivot to actually improving accuracy to get the terminal bonus and larger attributions. Continuous monitoring (especially of any metric that indicates the agent is getting high reward with low final accuracy) is essential. Fortunately, by design, high reward *requires* good final outcomes, so any policy that substantially deviates will also underperform on reward.

## PPO/LSTM Credit Assignment Implications (25-step Horizon, Variance & Saturation)

Tamiyo's agent is trained via PPO with an LSTM-based policy (the observation-history provides partial information, hence memory is used). The episode length is relatively short (25 epochs), which in RL terms is medium-horizon. We discuss how the reward structure and this horizon affect credit assignment, advantage estimation, and learning stability:

**Temporal Credit Assignment:** In a 25-step episode with  $\gamma \approx 0.995$ , even the earliest actions retain significant weight –  $\gamma^{25} \approx 0.88$  <sup>55</sup> <sup>56</sup>. This means PPO can, in principle, assign credit from the final rewards back to early decisions with only  $\sim 12\%$  discount. The presence of **dense intermediate rewards** (in SHAPED mode) further eases credit assignment: instead of waiting until the end to get feedback, the agent receives guidance along the way for each decision (germinate, prune, wait, etc.). For example, the decision to prune a seed might only fully reflect in final accuracy much later (if freeing up capacity for a better seed), but the reward function provides an immediate bonus or penalty on prune which correlates with whether it was good. This immediate feedback reduces temporal delay in the learning signal, helping the LSTM policy learn the consequences of actions more directly.

However, dense rewards can also introduce **myopic bias**: the agent might overly focus on optimizing those signals at the expense of the final outcome if they are not perfectly aligned. PPO uses advantage estimation with a value function; if intermediate rewards dominate the return, the value network will learn to predict those. Ideally, the sum of shaped rewards still equals or correlates with final performance. We must ensure no component creates a big delayed effect with no immediate reward (which would be hard to credit assign), or conversely large immediate reward that reverses later (which could mislead the value network).

**Variance in Returns:** With SPARSE (terminal-only) rewards, variance in episode returns is high – e.g., an agent that randomly tries actions will see nearly 0 reward most of the episode and then a wide distribution at the end depending on final accuracy. The doc notes that in Esper, *policy gradients had 3–5× higher variance in SPARSE mode, requiring larger batch sizes and more training epochs to converge* <sup>57</sup>. SHAPED mode, by adding intermediate rewards, significantly reduces return variance because the agent gets incremental credit (or blame) for each step. This means more stable gradient estimates. The cost is that the returns are biased, but as long as the bias (shaping) is consistent, PPO’s advantage estimation can cope.

One consideration is **variance vs bias trade-off**: PBRS adds no bias to optimal action *in theory*, and other shaping aims to be aligned. So they intentionally accepted a little bias to massively cut variance. The LSTM helps because it can remember events (like “seed was good earlier”) to predict upcoming rewards, but if all reward came at the very end, even an LSTM would struggle to attribute it correctly to early actions without a huge training sample (that’s the classic temporal credit assignment problem). By redistributing reward along the trajectory (essentially what shaping does), they’re performing a form of **reward redistribution akin to RUDDER** (Arjona-Medina et al., 2019) – giving “milestone” rewards so the LSTM doesn’t have to propagate credit solely via long memory and value bootstrap. This should improve learning speed considerably.

**25-Step Horizon Implications:** 25 steps is short enough that PPO can unroll the entire episode within the LSTM’s memory easily (the LSTM state won’t be asked to remember hundreds of steps). Also, bootstrap errors (from value estimates) propagate through at most 25 steps. So even sparse rewards are not as devastating as in tasks with hundreds of steps. The doc mathematically shows that with  $\gamma=0.995$ , an action at step 0 still retains ~88% influence on returns <sup>55</sup>. So the discounting isn’t too severe. Early in training though, value estimates for those far-out effects might be poor, increasing variance. By the end of training, a well-shaped reward yields fairly predictable returns per step, which the value function can fit, reducing variance in advantages.

**Advantage Saturation and Clipping:** PPO uses clipping on the policy update (to limit the change in action probability if advantage is large). If reward components produce very large advantages occasionally, the policy update might saturate (clip at the epsilon, e.g. 0.2) and not fully adjust, or the value function might lag in predicting such spikes, causing big TD residuals. We should examine whether any part of the reward could produce *outlier high advantages*. For instance, the holding penalty can produce a sudden -10 at a single step. If the policy accidentally waits one step too long, it experiences a big negative reward. This large error could result in a large (negative) advantage at that step, potentially clipping the PPO update for that state. But one clipped update is not catastrophic – it just means PPO will take a couple iterations to fully correct that action probability. In practice, since waiting in holding is clearly suboptimal, the agent will learn quickly to avoid it, so those spikes should become rare. Additionally, advantage normalization (centering) in PPO will scale down an outlier to some extent (since it’s within a batch that might contain other trajectories). The *scale of rewards* relative to returns is important: if most returns are on the order of 5-10, and an outlier is -10 in one step, that is within one episode’s sum possibly -10 out of maybe +10 total, which can swing that episode’s return to near 0 or negative. But across many episodes, it’s not infinite or anything. The policy gradient might get noisy, but not irrecoverably so. If anything, such a penalty can help exploration by occasionally giving strong negative reinforcement that steers policy firmly away from that region.

We should also consider *value function saturation*: The value network must learn to predict the sum of future rewards. In SHAPED mode, because the agent reliably gets intermediate rewards, the value targets are more uniformly distributed over time. There is less risk of a huge jump at the end that the value net fails to anticipate. In SPARSE mode, the value net would have near-zero for most states and then a big jump at terminal, which is harder to approximate. So shaped mode should reduce the burden

on the value function, leading to higher explained variance (one of the metrics they track <sup>58</sup> shows they expect  $>0.5$  explained variance on the value function if learning is healthy). Another aspect is that **some reward components are dense by nature**: e.g. rent penalty and attribution come every step. This means the value function will not be constant for long periods – it must predict slight changes as seeds grow or more seeds join. The LSTM can use state info (like current seed stage, improvement metrics) to estimate these small differences. That’s feasible because those factors are observable in the state (we assume the observation includes things like current seed’s measured improvement, etc.). If those signals were hidden, credit assignment would be harder (the LSTM would have to infer from long-term outcomes). But since they have real-time metrics (val\_acc, etc.), the value can align with those.

**Exploration vs Exploitation:** Dense rewards also impact exploration. With more frequent feedback, the agent doesn’t have to explore wildly to stumble on a reward at episode end; it gets hints right away if something is working or not. For example, if germinating a seed causes a drop in accuracy, the agent sees negative reward immediately (attribution negative) and might try a different approach next episode. If reward were only at the end, the agent would have to do a credit assignment through memory to figure out that germinating was bad. The shaping thus reduces the need for sophisticated long-term memory credit assignment – it “localizes” credit to near the decisions, which is helpful given a recurrent policy can handle some memory but works better when immediate observations correlate with reward.

**LSTM-specific credit considerations:** The LSTM can, in theory, maintain an internal summary of the seed’s history (e.g. how long it’s been training, how well it’s doing) and use that to decide on actions. The reward function provides intermediate signals that correlate with those internal states, reinforcing the LSTM’s memory usage. For instance, if the LSTM keeps track “seed has been blending for 3 epochs and improvement is negative each time”, it may decide to prune. The reward indeed gave negative each time, so the advantage for prune vs wait will be positive. Over many sequences, the LSTM will learn to trigger prune as soon as it identifies that pattern to avoid accumulating more negative. Similarly, the LSTM might learn a pattern like “if seed is good and we are in holding for 2 epochs, advantage for fossilize becomes positive now because waiting further yields big negatives”. The reward schedule (one free epoch, then  $-1$ ,  $-3$ , etc.) essentially teaches the LSTM a countdown. The LSTM’s memory is crucial: it might need to remember how many consecutive WAITs happened in holding to predict the next penalty. But since the penalty grows exponentially, even if it doesn’t count precisely, any extended hold yields a big negative surprise which the LSTM can quickly associate with “don’t keep waiting”.

**Normalization and Scaling:** PPO typically normalizes advantages (zero mean per batch) which mitigates issues of scale. But if one component systematically adds a large constant to all rewards, it could reduce relative advantage differences. For example, if every step had a  $+0.5$  baseline reward regardless of action, that might inflate returns and require value function to predict that baseline – but advantage ( $R - V$ ) might actually be stable since  $V$  learns it. In Tamiyo, the baseline (like parameter penalty) is small and state-dependent. There isn’t a huge constant bias except maybe the final accuracy bonus that everyone gets some fraction of. But since final accuracy will depend on agent actions too, not exactly constant.

**Potential Saturation of Early vs Late Advantages:** One area to consider is if earlier decisions effectively get overshadowed by later outcomes (or vice versa) in terms of advantage magnitude. If shaping is heavy in the middle of the episode (like BLENDING stage yields a flurry of positive rewards), then the value network might predict a high return from mid-episode onward, which could reduce the *advantage* of the final fossilize action. Conversely, if terminal bonus is significant, the last action (fossilize or not) carries a big swing, which might overshadow earlier ones. The design tries to balance this: PBRS and attribution provide reward for mid-episode decisions; terminal provides a final bump. Ideally, all actions have some consequence spread out. They even list a metric: “advantage magnitude

by epoch should be ~uniform across episode” <sup>59</sup> – meaning no part of the episode should consistently have higher advantage variance than others. If we saw that early epochs always have near-zero advantages while epoch 25 has huge spikes, that indicates credit assignment is too end-loaded. The current reward likely achieves a more uniform distribution: e.g., germination (epoch 1) gives PBRS and sets up potential future reward, blending (middle) gives chunk of reward, final gives moderate bonus.

**Effect of Clipping on Learning Specific Behaviors:** PPO’s policy ratio clipping ( $\pm 0.2$ ) means that if the agent suddenly discovers, say, holding too long is very bad, it can’t instantly drop the probability of WAIT to near zero in one update if it was high before. But repeated negative experiences will push it down over a few updates. The shaped reward ensures those experiences happen reliably whenever it tries that bad behavior, so the gradient signal is strong. There is no ambiguity that waiting in holding is bad – the negative reward is immediate and large. So the policy will adapt in perhaps a few iterations. The value function might momentarily underpredict how bad it is (since it might not have seen such a big penalty before), but after a couple occurrences, it will adjust to expect negative returns if holding too long.

**Exploding or Vanishing Gradients in LSTM:** The LSTM is trained through backprop in time for 25 steps. Dense rewards help keep gradients flowing at each step (some signal to train on at each sequence position). If it were sparse, gradients would mostly flow from the end to the beginning possibly causing more vanishing (if 25 steps not too long, maybe fine, but the more signals along the way, the more the LSTM’s weights get adjusted at intermediate timesteps as well). So likely less vanishing gradient issues here.

**In summary:** The shaping design greatly aids the credit assignment for PPO/LSTM: - Lower variance in returns (which means more stable policy updates) <sup>57</sup> . - More uniform distribution of advantages across time (preventing the policy from only learning at episode end). - The LSTM can use immediate rewards as cues to learn what information to carry (for example, a spike of negative reward at blending teaches the LSTM that something went wrong – it might then learn to attend to that seed’s metrics and next time decide differently earlier). - If anything “saturates”, it would be that certain obvious things (like not pruning a bad seed) yield consistently large negative advantage, which the policy will quickly learn to avoid entirely. This is good (the behavior saturates at the optimal boundary – e.g. it will never choose that action in that context once learned).

We should note that **with SHAPED reward, advantage normalization might mask the scale differences:** e.g., an episode with +15 total vs one with +5 total, after normalization, both are treated relative to their batch mean. So PPO naturally handles different difficulty episodes; it doesn’t require manual reward scaling beyond what’s done. If we suspect any component’s scale might still be problematic (like if a seed yields +20 one step), one might consider explicit normalization. But given the attention to keep values bounded, it’s likely fine.

Therefore, **PPO is well-suited to this reward** as long as the components remain in their intended range. The use of an LSTM doesn’t pose an extra challenge since the reward provides intermediate supervision. The one caveat might be that the LSTM policy has to learn a kind of internal “planning” – e.g., to germinate a seed expecting that 10 steps later it will get a big final reward when fossilized. Because of shaping, it doesn’t have to wait 10 steps blindly; it will get intermediate rewards (PBRS, etc.) that bridge that gap. This helps “chain” the credit. For instance, germinating a seed leads to PBRS +0.3 immediately – so even before the final outcome, the agent sees some reward confirming that was a good idea (assuming the seed was needed). This can reinforce the decision to germinate well before the final accuracy comes into play.

If we compare **SHAPED vs SIMPLIFIED**: Simplified has only PBRS and terminal. That means less dense feedback – perhaps slightly more variance than shaped but far less than fully sparse. Still, PBRS ensures every stage transition gives something, so it's not too sparse. The analysis above would shift a bit for simplified: credit assignment is a bit harder for things like prune decisions (since simplified wouldn't have the immediate attribution penalty or blending warning – it relies on terminal accuracy differences and PBRS). That could mean the LSTM has to attribute a drop in final accuracy to having not pruned earlier, which is subtle. Shaped reward explicitly handles that by negative rewards at the moment of not pruning. So shaped mode likely yields faster credit assignment for tricky decisions, at the cost of the complexity we've discussed.

**Conclusion:** The reward function is carefully engineered to be **PPO/LSTM-friendly**. It reduces training instability by injecting reward signals at appropriate times, keeping return variance manageable and avoiding long delays. Metrics to watch include *advantage estimates by time* <sup>58</sup> – we want those to be neither always huge at one end nor highly noisy. Based on the design, we expect a reasonably even distribution: some positive advantages on germination (due to PBRS), possibly large positive or negative around blending (depending on seed success), and a final bump at fossilize. If any part “saturates”, it might be the holding penalty causing very large negative advantages for waiting too long – but since the policy will quickly learn to almost never do that, those should vanish from the distribution after sufficient training. Similarly, if a component consistently yields no gradient (e.g., synergy might often be zero and not contribute), that just means it's neutral – not harmful, except it might be a wasted complexity.

In practice, if training curves show instability, a culprit could be reward scaling. We'd then consider normalizing returns or scaling down some weights (as discussed in the next section). But given the relative magnitudes, we anticipate PPO's adaptive learning (with advantage normalization) will handle it. The LSTM's memory is more taxed by partial observability of which seed is which perhaps, but not by reward credit per se, since the reward provides immediate clues when an action was good or bad.

## Ablation and Experiment Plan

To validate each part of this reward and to understand its contribution to learning, we propose a systematic ablation study and some focused experiments. The goal is to isolate components to see if they are helpful or causing any harm (e.g. unneeded complexity), and to verify the training dynamics match expectations. We also outline key metrics to monitor during these experiments.

### Ablation Experiments:

We can start with **baseline ablations** that remove major components one by one, to see the effect on policy learning and final performance:

1. **No PBRS (Ablation A1):** Set `pbrs_weight = 0` (disable stage progression bonuses) <sup>60</sup> <sup>61</sup> . Hypothesis: learning will slow and maybe stall in early training because the agent no longer gets guided to progress seeds. We expect to see the agent possibly not advancing seeds as eagerly (since PBRS was a reward for stage changes). Metrics: look at average stage achieved by seeds and the delay in germination decisions. Also monitor episode return and convergence speed. If final performance suffers or episodes needed increase significantly, PBRS is proven important. If there's no change in final outcome (but maybe just slower), that's fine, but if policy fails to learn good behavior (like agent keeps seeds dormant too long), then PBRS was crucial.

2. **No Terminal Reward (Ablation A2):** Set `terminal_acc_weight=0` and `fossilize_terminal_scale=0` (or simply switch to a mode where final accuracy and count aren't rewarded) <sup>61</sup> <sup>62</sup>. This effectively means only shaping rewards throughout, no explicit end objective. Hypothesis: the agent might learn to maximize intermediate signals (like keep seeds alive for attribution and PBRs) and neglect actual accuracy. We'd likely see an agent that e.g. fossilizes less (because fossilize only had a cost now, no final bonus to encourage it) – possibly leading to seeds staying active and maybe multiple seeds piling up (since no final count reward to incentivize completion). Metrics: final validation accuracy achieved, number of seeds fossilized vs pruned, total parameters used. If we observe the agent farms attribution but final accuracy stagnates or even declines (Goodharting the shaping), that confirms terminal rewards are needed for alignment. This test basically checks if our shaping alone is truly invariant or if it needed that final nudge. We expect performance to drop without terminal bonus.
  
3. **No Rent Penalty (Ablation A3):** Set `rent_weight = 0` (no compute cost) <sup>61</sup> <sup>62</sup>. Hypothesis: agent may use more seeds/parameters since there's no cost to it. Possibly, it could try germinating multiple seeds concurrently or not pruning old ones because parameter cost doesn't accumulate. This could improve final accuracy slightly (maybe more seeds = more potential accuracy) but at the expense of model size (which matters for deployment but not for reward anymore). We watch if the agent's behavior changes to e.g. always germinate new seeds even if old ones gave diminishing returns ("bloat"). If final accuracy goes up but parameter usage skyrockets, that's expected. If nothing changes much, maybe the agent inherently limited seeds anyway due to other factors. Logging number of active seeds, total params over time, and final accuracy vs number of seeds would be key. The outcome informs if rent penalty was necessary to keep seeds in check.
  
4. **Pure Sparse Reward (Ablation A4):** Use `RewardMode.SPARSE` (only final accuracy – param tradeoff) <sup>61</sup> <sup>63</sup>. Hypothesis: learning will be much harder. We anticipate the agent might struggle to discover the correct sequence (as credit assignment is now entirely delayed). Possibly the agent might fall back to trivial policies (like do nothing or random actions) because it can't tell what worked until the end. If given enough time or with hyperparam tuning (like bigger batch, lower LR as recommended <sup>57</sup>), it might eventually learn, but likely slower. Metrics: training curve of average return over episodes – we expect it to flatline for a long time or be very noisy, compared to shaped which should improve steadily. This confirms the need for shaping. If surprisingly it does learn similarly, that would mean our shaping maybe wasn't as critical as thought (unlikely). We should also log variance of returns and gradient estimates to quantify the 3-5x variance claim <sup>57</sup>.

Next, **anti-gaming component ablations** to verify they actually prevent bad behaviors:

1. **No Attribution Discount (Ablation A5):** Turn off the sigmoid discount for negative improvement (or set `disable_anti_gaming=True` to remove both discount and ratio penalty) <sup>11</sup> <sup>12</sup>. Hypothesis: Without this, we might see the agent exploit seeds that degrade performance but increase contribution. Essentially re-testing the "ransomware" scenario. We would monitor for seeds being fossilized that have negative total improvement. Telemetry events for "REWARD\_HACKING\_SUSPECTED" might spike in logs <sup>54</sup>. Also check final accuracy: if the agent starts keeping seeds that actually made accuracy worse (because it can get reward from them), that's a red flag. We'd likely see final accuracy going down or oscillating, and yet the agent not pruning those seeds as it should. This ablation would confirm whether the attribution discount was crucial. We expect that with it off, some runs might show weird behavior (the policy might converge to a suboptimal strategy where it's okay with slight regression as long as it reaps counterfactual rewards).

2. **No Ratio Penalty (Ablation A6):** Keep attribution discount but remove the ratio threshold penalty <sup>11</sup> <sup>12</sup> . Hypothesis: The agent could try more subtle dependency tricks, especially if total\_improvement is small but positive (so discount doesn't trigger). For instance, seeds that create 0.2% improvement but 1.5% contribution (ratio 7.5) would have gotten -0.3 penalty; now they wouldn't. The agent might then not be punished for seeds that siphon improvement from host. We'd monitor if seeds tend to have contribution >> improvement after training. Also, compare how often seeds with very low improvement get fossilized in this condition vs normal (maybe the agent will let borderline seeds through more often). If the final performance or number of low-value seeds increases, that shows the ratio penalty was doing its job.
  
3. **No Holding Penalty (Ablation A7):** Disable the exponential WAIT penalty in HOLDING. We can simulate this by not applying it or by allowing many free WAITs. Hypothesis: The agent might learn to *stall in HOLDING* to farm attribution. In training runs, we'd look at the distribution of HOLDING durations. Without penalty, an optimal agent might indeed sit on a good seed for all remaining epochs, continuously getting attribution each epoch (since nothing forces it to finalize). That maximizes intermediate reward, though it forgoes the fossilize bonus and final count. If attribution per step is large enough, it could outweigh those missed final rewards. For example, a +5% seed yields +5 reward each epoch; over 10 extra epochs that's +50 reward, whereas fossilizing it might give just +0.5+0.5=+1 and final +3. So clearly, without penalty, waiting yields far more reward. We expect the agent would exploit that: we'd see episodes where seeds remain active till the very end and then maybe fossilize at the last step (since final count still gives +3). That means possibly fewer seeds overall (maybe it germinates one seed and just milks it). We'd measure episode returns and see if they balloon (they could, because the agent can rack up attribution every timestep). Also measure final accuracy difference: such a policy might still reach high final accuracy (since it kept training the seed), but it would violate the "complete lifecycles" objective (less parallel exploration of multiple seeds). If we witness that, it confirms the necessity of the holding penalty. This test is risky to run on the same scale, as it might produce unbounded returns if we allowed enough steps; but in 25 steps, the max an agent could do is keep one seed for say 20 steps, get 20\*X reward. If X ~5, that's +100 reward, dwarfing normal returns. So we'd likely see the agent's reward shoot up (and training might destabilize unless clipped heavily). It would be very telling: a dramatic difference indicates that component was indeed preventing a massive exploit.

#### Additional Experiments:

- **Reward Scale/Normalization Test:** We could run an experiment where we apply a simple normalization to total reward (e.g. divide all rewards by 10 or use PopArt to normalize value targets) and see if PPO converges faster or slower. This isn't exactly an ablation of a component but rather tuning the magnitude. If training is unstable in baseline, this could help. It will inform whether our reward magnitudes are on a good scale or if they need adjustment. We expect that the current scale is okay (as per design guidelines, -10 to +10 per step), so this might not change much. But it could reveal if any hidden high magnitudes exist by comparing variance of gradients.
  
- **Simplified vs Shaped vs Minimal Mode:** Do a head-to-head experiment comparing the default SHAPED reward to the SIMPLIFIED and MINIMAL modes across multiple training runs. This will show which reward formulation learns fastest and achieves the best policy. **Metrics:** number of episodes to reach a certain performance threshold, final accuracy achieved, etc. Hypothesis: SIMPLIFIED might learn more steadily (less conflicting signals) but possibly converge to a slightly less fine-tuned policy (maybe it misses some efficiency cues). SHAPED might get there faster but risk more oscillation if certain components conflict. MINIMAL likely falls in between sparse and

shaped in difficulty. If SIMPLIFIED outperforms shaped (as the DRL expert predicted) in learning speed without sacrificing final performance, that's a case to switch default. We'd log reward component breakdowns to see what Shaped was providing extra and if that actually helped or not.

- **Horizon Extension Test:** Although the prompt is 25-step episodes, it might be worth testing what happens if we increase `max_epochs` to, say, 50, with the same reward function. This checks scalability: do the shaping components still function well, or do we see any issues (like the holding penalty might need adjusting if episodes longer)? If the agent can manage 2-3 seeds in 25 steps, maybe in 50 steps it can manage more – we'd see if any component saturates or if potential values need extension. This can guide if future usage with longer training runs requires retuning PBRS (like adding more `epoch_progress_bonus` beyond 2.0 for longer durations).

### Metrics to Log for Each Run:

- **Reward Component Contributions:** Using `RewardComponentsTelemetry`, log the average per-episode sum of each component (attribution, PBRS, penalties, etc.) over training time. This will show which components are actually contributing to the return. For example, if synergy bonus stays near 0 always, it might be safe to remove. Or if PBRS is >50% of total reward early on `20 21`, maybe terminal signal is too weak initially. We want to see that eventually most reward comes from actual contributions and terminal success, not just shaping.
- **Episode Return and Final Accuracy:** These two should be tracked together. A key check: does increasing episode return correlate with increasing final host accuracy? If we ever see episode return go up while final accuracy stagnates or drops, that signals the agent is exploiting shaping at expense of objective (Goodhart). Ideally they correlate tightly. Plotting this over training iterations can reveal any divergence.
- **Seed Lifecycle Metrics:** e.g., average number of seeds germinated, pruned, fossilized per episode; distribution of seed lifespans; average improvement of fossilized seeds. This helps verify if the agent is doing the intended strategy. If an ablation causes, say, many seeds to remain unfossilized or many seeds to be pruned early, we'll catch it here.
- **Variance Measures:** Track the variance (or standard deviation) of returns per training batch. Also track the **value function loss and explained variance** (PPO typically reports how well value predictions match returns). If explained variance is low (<0.2), it means the reward might be too noisy or complex for value to predict – possibly an issue with shaping or scaling. If certain ablations (like removing shaping) drastically drop explained variance or raise return variance, that quantifies how important shaping was.
- **Frequency of Anti-gaming Triggers:** The telemetry events for reward hacking (ratio too high, negative total with positive contrib) should ideally be near zero for a healthy trained agent. If during training we see a lot of these events until the agent learns to avoid them, that's fine – but by end it should taper off. If not, something's wrong. In ablations where we remove those penalties, we might ironically see fewer *events* (because the code may not flag them if disabled), so we may need to manually analyze seeds' stats to see if hacking patterns emerged.
- **Policy Distribution:** e.g., probability of WAIT vs PRUNE vs FOSSILIZE in various stages. Over training, we expect these to shift (like early on maybe the agent waits too much, later it prunes more appropriately). Monitoring how these converge helps ensure the final policy is making

sensible decisions. For instance, if after training the policy *still sometimes waits in holding longer than 1 epoch*, maybe the holding penalty wasn't strong enough or some seeds still confuse it.

- **Ablation Comparison Metrics:** For each ablation and baseline, measure final task performance (e.g. final accuracy, number of seeds integrated) and sample efficiency (how many episodes to reach X performance). We'll present those to decide which components are essential.

Given these experiments, we can answer questions like: *Is the ratio penalty truly preventing a failure or just theoretical?* (A6 addresses that), or *Would a simpler reward have sufficed?* (comparing to SIMPLIFIED).

**Minimal Set of Runs:** We likely need: - Baseline (full SHAPED) run. - SPARSE run (hard mode). - SIMPLIFIED run (expert's suggestion). - A couple key ablations: No PBRS, No anti-gaming (discount & ratio together), No holding penalty. We can possibly combine some if needed (but better separate to pinpoint issues). - No terminal might be too degrading but it's instructive as well.

This is maybe 6-8 runs. Each run we gather metrics as above.

**Interpretation:** We will analyze if each removed component causes a drop in either learning stability or final outcome: - If removing something doesn't hurt (or improves) learning, that component might be needless or could be simplified. - If every removal hurts in some way, it means the reward is tightly engineered with little redundancy – which is good or bad depending on if we value simplicity.

Finally, we'll also want to experiment with **hyperparameter sweeps** for certain reward parameters as suggested (like try different PBRS weights 0.1 vs 0.3 vs 1.0, different terminal bonus values) <sup>64</sup> <sup>65</sup> . This can reveal sensitivities: e.g., if PBRS=1.0 (making shaping very large), does policy still converge to same behavior or does it get stuck chasing stage rewards? If terminal\_fossilize\_bonus=5 instead of 3, does agent overly fossilize at cost of training seeds fully? These inform safer ranges for these constants.

**Logging and Telemetry:** Already the code logs components and emits events. We should ensure logs capture: - each seed's total\_improvement and seed\_contribution on fossilization, - how many epochs in each stage, - any time an auto-prune happens (should be rare if policy is good).

**In summary**, this plan will empirically ground our analysis. It identifies which parts of the reward are critical versus possibly extraneous, and checks that the agent isn't secretly exploiting something we didn't foresee (if it does, an ablation or telemetry will uncover odd patterns like high reward but low accuracy). The combination of these experiments will guide our recommendations for which components to adjust, remove, or emphasize.

## Recommendations

Based on the analysis of design and potential issues, here are concrete recommendations for improving the reward function and training outcomes. We categorize them into **low-risk tweaks** (unlikely to negatively impact learning, easier to implement/test) and **high-risk, high-reward changes** (more substantial modifications that could greatly improve performance or simplicity, but need careful evaluation). Each recommendation is justified by the above findings and often ties into existing telemetry or flags for easy testing:

## Low-Risk Recommendations:

- 1. Tighten PBRS Implementation and Monitoring:** Ensure the PBRS reward is calculated correctly for all transitions. Specifically, implement the fix suggested in the bug ticket – raise an exception or at least an error log if `previous_stage != 0 and previous_epochs_in_stage == 0` on a transition <sup>44</sup> <sup>66</sup>. This fails fast on any inconsistency, preventing silent reward leakage that breaks invariance. This is low-risk (it doesn't change the reward when things are correct, only when something's off) and protects training from subtle bugs that could corrupt gradients <sup>67</sup> <sup>68</sup>. Additionally, expand the property tests to cover sequences with instant transitions to ensure  $\phi_{\text{prev}}$  is always accurate. Since PBRS is theoretically sound only when implemented right, this change shores up that guarantee without affecting normal runs.
- 2. Reward Scaling/Clipping Safeguard:** Although the reward components are roughly in  $[-10, +10]$ , it's wise to explicitly **clip or normalize extreme outliers** to avoid any training hiccups. For example, enforce a cap on total reward per step, say  $\pm 10$  or  $\pm 15$  (just as a sanity check). The code already caps some parts (holding, prune penalties). We could add an assertion or clamp after summing all components: `reward = max(min(reward, 10), -10)` (with a debug log if clipping happens). This would catch any unforeseen spike (maybe due to multiple components stacking at max in rare combos). It's low risk because in normal conditions it wouldn't trigger, but it protects against tail-case large gradients. Alternatively, incorporate a running reward normalization (e.g., normalize returns by a moving standard deviation as done in some PPO implementations) – PPO often does this internally for advantage, but an extra normalization of the reward signal could be tested. This would ensure training stability if we mis-estimated any scales. Given our analysis, it's likely not strictly necessary, but it's a safe guardrail.
- 3. Fine-Tune Anti-Goodhart Thresholds:** Review and potentially adjust the anti-gaming hyperparameters to better reflect observed noise levels. For instance, the `improvement_safe_threshold = 0.1%` and `hacking_ratio_threshold = 5.0` are currently somewhat hard-coded. If telemetry shows legitimate seeds occasionally exceed ratio 5 due to noise (e.g., a seed truly adds 0.05% improvement but measurement noise makes contribution 0.3%, ratio 6, which gets penalized undesirably), we should raise the threshold or smooth it. A suggestion: increase `improvement_safe_threshold` to perhaps 0.2% or 0.3% in early training when noise is high, then maybe decay it or keep it adaptive (like a threshold proportional to observed variability in validation accuracy). This reduces false positives where the agent might be penalized for normal variance. Also, consider lowering the penalty magnitude slightly if it's found to sometimes stack with `attribution_discount` and overly zero out reward for borderline seeds. This is low-risk if done carefully, because the worst case is some malicious patterns get a bit less penalty, but as long as they still result in net zero or small reward, the agent won't favor them. Essentially, we want to ensure we're not punishing seeds that are just slow starters (e.g., a seed that has 0% net improvement until very late shouldn't be heavily penalized if it's about to improve – currently the discount would kill its reward entirely, possibly discouraging what could become a good seed). We could implement a grace period: e.g. don't apply `attribution_discount` in the first N epochs after germination to give seeds a chance, relying on `blending_warning` to handle clear early bad cases. This is a tuning change that can be A/B tested (with `disable_anti_gaming` vs `enable` to see differences).
- 4. Simplify and Unify Action Costs:** The intervention costs are very granular ( $-0.02$ ,  $-0.005$  etc.). We could adopt the **uniform non-WAIT cost** used in SIMPLIFIED mode for consistency – e.g. make all non-wait actions cost  $-0.01$  (and remove the special  $-0.3$  penalty for `germinate-while-seed-active` by simply prohibiting that action at the environment level, if not already). This has low risk because these costs are small and primarily serve to discourage frivolous toggling.

Using a uniform cost makes it easier to reason about (and avoid accidentally biasing towards one action because its cost was slightly lower). The design doc's recommended config does exactly this (`non_wait_cost=0.01`) <sup>69</sup> <sup>70</sup>. It shouldn't change the optimal policy much – at most, it slightly penalizes PRUNE a bit more and germinate a bit less (or vice versa), but at 0.01 difference these are negligible. The benefit is conceptual clarity and one less hyperparam to tune. We'd keep necessary state-machine penalties (like `invalid fossilize/prune = -1`) as those enforce constraints clearly.

- 5. Improve Reward Telemetry & Debugging Tools:** Add a few more logging points for extreme cases to help during training. For example, if a seed gets fossilized with `total_improvement` just below threshold (thus yielding `-0.2` penalty), log that event – it might indicate threshold could be tweaked or seed was fossilized prematurely. Or log when `holding_penalty` hits its cap (means agent really pushed the wait – which ideally it shouldn't). Having these logs will allow quick diagnosis if training doesn't converge ("Oh, the agent is still waiting 4 epochs in holding frequently, maybe penalty needs increase or something"). This doesn't affect training itself (except slight overhead), so it's low risk but high value in diagnosing problems. The telemetry class can be extended to record e.g. max WAIT count in holding per episode, etc., which we can analyze after runs.
- 6. Reward Component Normalization in TensorBoard:** Not a change to the reward function per se, but ensure that during training we track each component's average magnitude over time. This will highlight issues like one component growing uncontrolled. Low risk, since it's just logging. It helps verify assumptions (e.g., check that synergy bonus truly stays `<0.1` on average; check that attribution is usually in a reasonable range, etc.). If any component shows a trend of increasing magnitude (which could happen if say seeds get larger contributions as model scales, maybe hitting new extremes), we can adjust weights accordingly in future.

### High-Risk/High-Reward Recommendations:

- 1. Adopt SIMPLIFIED Mode as Default (with slight enhancements):** Given the complexity of SHAPED and the expert advice to prefer SIMPLIFIED <sup>71</sup> <sup>72</sup>, consider making SIMPLIFIED the default training reward – at least for initial training phases – and only enabling full SHAPED if necessary. SIMPLIFIED uses just PBRS, a uniform action cost, and a heavier terminal reward, removing many moving parts (attribution, warnings, rent, etc.). This could yield a more stable learning signal (fewer conflicting objectives) and easier credit assignment, as it focuses the agent on **"complete seeds that improve final accuracy"**. The high reward potential: faster convergence, simpler policy behavior. The risk: the agent might struggle to learn fine-grained behaviors without the dense signals (especially discerning when a seed is marginally good or bad before the end). It might also exploit any unaddressed loopholes (like maybe it will be slower to prune bad seeds without `blending_warning` – though PBRS and final accuracy loss should eventually teach it). To mitigate risk, we can implement a **curriculum**: start training in SIMPLIFIED mode until the agent learns basic lifecycle management, then switch to SHAPED to refine. Or run parallel experiments on both modes to see which yields better results. If SIMPLIFIED performs comparably or better in achieving final accuracy and using fewer seeds, that's a strong sign that many shaping components can be dropped or simplified. This change is high-reward because it reduces complexity and potential interference in rewards, aligning more directly with final outcome. It's high-risk because it might fail to learn subtle behaviors (like pruning at the optimal time) as quickly – but since we have the shaped version as a fallback, it's worth exploring. The doc's recommendations align with this: they suggest starting with simplified <sup>71</sup> <sup>72</sup>.

2. **Milestone-Based Reward Structuring:** If fully simplified is too sparse, a compromise is to move toward **milestone rewards** – give rewards only at key events (e.g., finishing BLENDING, finishing HOLDING) rather than every epoch. For instance, instead of per-epoch attribution, reward the agent when a seed transitions from BLENDING to HOLDING with an amount proportional to its contribution. This would preserve some temporal credit assignment but cut down on constant shaping noise. The analysis in the doc (section 2.4) suggests milestone-based shaping as a viable approach <sup>73</sup> <sup>74</sup> . High reward: this simplifies the reward trajectory and maybe reduces the non-stationarity of per-step rewards, making it easier for PPO to learn (fewer ups and downs each step, more sparse but meaningful signals). It also might reduce the risk of farming per-step rewards like attribution because rewards only come at transition actions. The risk is again slower learning (since agent might get less frequent feedback). But if we see that the agent picks up quickly which transitions yield reward, it could still work well. We could implement this by an alternate mode where, say, attribution and penalties accrue internally but only pay out on prune/fossilize actions (like computing a delta potential or delta performance at those points). Experimentally, this could be toggled to see if the agent's performance changes. It's a more radical structural change but could make the reward more *sparse yet credit-assignable* (like RUDDER approach). This is aligned with making the reward "clever but learnable": fewer but more informative rewards can sometimes be easier for an LSTM to handle than constant small ones.
  
3. **Multi-Objective Reward Decomposition and Scalarization:** Currently, all objectives are mashed into one scalar. A high-risk but potentially high-reward idea is to treat it as a *multi-objective RL problem*: e.g., have separate "accuracy reward" and "efficiency reward" and use a technique to balance them (like dynamic weighting or Pareto-optimal exploration). The doc even outlines a structure for tracking separate components (accuracy\_delta, param\_efficiency, stability\_score, timing\_quality) <sup>75</sup> <sup>76</sup> . Instead of hand-tuning weights inside the reward, we could use a linear scalarization with adjustable weights or an algorithm that finds a good trade-off. For example, at training time, periodically adjust the weight of the rent penalty to ensure the agent doesn't sacrifice too much accuracy for efficiency or vice versa. Or use a constrained RL approach: treat final accuracy as primary and enforce a constraint on total params via Lagrange multiplier updated over time. These approaches are complex (hence high-risk) but could lead to a more principled solution where, say, you don't have to guess the perfect `rent_weight` – you let the algorithm learn how to trade off accuracy vs parameters. The reward function already logs objectives, and the references mention multi-objective methods <sup>77</sup> <sup>72</sup> . Implementing this would be a research project on its own, so it's high-risk in terms of time and complexity. But the reward could become more **adaptive**: as the agent learns, we could shift focus (e.g., first maximize accuracy, then later emphasize efficiency once accuracy plateaus). If successful, this yields an agent that is both high-accuracy and minimal complexity without needing manual weight tweaks.
  
4. **Incorporate Hindsight or Model-Based Credit Assignment:** We could draw on techniques like **hindsight credit assignment** for scaffolding seeds or others. For instance, the code has `compute_scaffold_hindsight_credit` for retroactive credit to scaffold seeds after beneficiary fossilizes <sup>22</sup> <sup>23</sup> , but it's unclear if it's used in the current training. If not, we might enable or extend it. This would give additional reward to seeds that helped others once that help is confirmed. It's somewhat orthogonal to core reward, but it could improve synergy utilization (a scaffold seed might not directly raise accuracy, but if it boosted another seed that did, it gets credit after the fact). High reward: this could encourage multi-seed strategies that currently might not be reinforced strongly. The risk: more complexity and potential credit confusion. However, since it's applied at terminal (a retrospective reward), it doesn't interfere with the online signals and is potential-based (in terms of dependency graph). It's a targeted improvement if we

find the agent tends to ignore scaffold behavior because synergy bonus was too weak. If synergy is identified as important but underutilized, amplifying it via hindsight credit is a strategic change. We could test enabling that and see if the agent starts to keep scaffolds longer or achieve higher final accuracy through cooperation.

**5. Dynamic Thresholds and Adaptive Penalties:** One challenge is picking static values for thresholds like `DEFAULT_MIN_FOSSILIZE_CONTRIBUTION`. A high-reward idea is to make these **adaptive based on running performance**. For example, if the overall host accuracy is improving significantly, maybe we can afford to raise the fossilize threshold so only really good seeds count; if progress is slow, lower threshold to encourage keeping some seeds. Similarly, the holding penalty exponent (base 3) could be tuned by the agent's behavior – e.g., if we detect the agent still sometimes waits too long, automatically increase the base (to 4 or 5) to punish more. This can be done via schedule or even learned (meta-RL style). High reward because it would fine-tune the reward to the agent's current policy, potentially speeding up convergence or avoiding local optima. High risk because it adds another layer of complexity and if done wrong could destabilize training (non-stationary reward). But some simpler forms might be safe: e.g. linearly increase the prune early penalty weight as episodes go on, to force the agent to try seeds longer in later training when it can handle it – early on, a strong early-prune penalty might just confuse a learning agent, but later it can enforce exploration.

**6. Environment Design Changes (High-leverage but outside direct reward):** If certain failure modes are easier fixed in the environment than via reward, consider them. For instance, the “ransomware” pattern could in theory be prevented by architectural regularization – e.g., ensure that removing a seed can't drop accuracy *below* what it was at germination by preserving some knowledge in the host. This is more on the environment side (maybe keep a copy of original weights as fallback). If implemented, the reward wouldn't even face that edge case as strongly, simplifying anti-gaming requirements. Another example: if synergy is valuable but rare, maybe design training tasks that require synergy (so agent experiences it more and the 0.1 bonus becomes meaningful). These are high-risk in the sense of requiring environment changes or new training scenarios, but they could yield high reward by making the RL problem easier or more aligned inherently, reducing burden on reward function to patch things.

Each high-risk recommendation should be tested in controlled experiments (perhaps starting in simulation or with shorter episodes to verify impact). We can rank them by feasibility: adopting SIMPLIFIED mode (#1) is actually relatively straightforward to try (just a config change) and likely beneficial, so it might be the first high-reward change to implement. Multi-objective or dynamic approaches are more experimental.

#### **Prioritization:**

- **Immediately actionable (low risk):** PBRs fix & checks (#1), uniform small action cost (#4), better telemetry (#5,#6). These can be done almost trivially and will improve robustness and debuggability without needing a full retraining to validate (though we should run tests to ensure no performance regression – but they shouldn't cause any).
- **Short-term experiments:** Try SIMPLIFIED default (#1 high-risk) and milestone reward (#2 high-risk). These can be done by toggling modes or minor code branches. Evaluate on a smaller scale or with a few seeds to see results. If positive, iterate.
- **Mid-term improvements:** Tweak anti-gaming thresholds (#3 low-risk) and synergy/hindsight (#4 high-risk). These require observing where current reward might be misfiring or insufficient. For example, if logs show hardly any `ratio_penalty` events but some seeds still had low

improvement, maybe threshold was fine; if events fire a lot on seeds that turned out okay, then adjust threshold.

- **Long-term ideas:** Multi-objective scalarization (#3 high-risk), adaptive schedules (#5 high-risk) are more researchy. These should come after confirming simpler fixes aren't enough. If training is still unstable or suboptimal after simpler changes, these could be pursued.

In summary, start by tightening and simplifying what's already there (ensuring no inadvertent biases or bugs, and removing micro-penalties that may not be needed), then consider reducing the complexity of the reward itself (moving toward simplified/milestone approach) to see if the agent can learn equally well or better. Use the abundant telemetry to guide these changes – e.g., if we see the agent never triggers a certain component or always struggles with a certain phase, use that info to adjust.

The recommendations aim to maintain or improve alignment with the true objective (“maximize accuracy under budget”) while streamlining learning. By implementing the low-risk tweaks and carefully testing the high-reward changes, we can likely achieve a reward function that is **easier to tune, safer against exploitation, and possibly more sample-efficient** for PPO to learn.

## Open Questions / Required Clarifications

*(This section is intentionally brief, as most aspects were deduced from the code and context. However, a few points would benefit from clarification to ensure correct interpretation and further tuning.)*

- **Q1: Interpretation of Validation Accuracy Scale** – In the code, `val_acc` appears to be in the range [0,100] (percentage). We assume `seed_contribution` is measured in the same units (percentage points). Can we confirm this? For instance, if `val_acc=65.0` and `seed_contribution=0.05` as per usage example <sup>78</sup>, does that mean a 0.05 percentage point gain, or 5%? This affects how we view thresholds like `improvement_safe_threshold=0.1` (is that 0.1% accuracy or 0.1 in absolute which would be 10%?). Clarifying units ensures we tune the anti-gaming thresholds correctly relative to normal accuracy noise (if 0.1 is 0.1%, that's a tight threshold; if it's 10%, that's very lenient). The code suggests 0.1 is likely 0.1% since improvements are often small. It would help to confirm typical magnitudes of `seed_contribution` in practice (telemetry data hints at ~0.05 for base slot rent calculation, implying contributions are often a few hundredths).
- **Q2: Stage Transitions and `epochs_in_stage`** – We assume that when an action causes a stage transition (e.g. PRUNE or FOSSILIZE ends a seed's life, or BLENDING completes to HOLDING), the `compute_contribution_reward` is called *after* the state update. Is that correct? The use of `previous_stage` and `previous_epochs_in_stage` suggests the reward function sees the *new* stage with `epochs_in_stage=0` and reconstructs the old. We just want to confirm the ordering: does the environment update `seed_info` to the new stage before computing reward? (It appears yes.) This matters for PBRS calculation – if it were computed before updating stage,  $\varphi(s')$  vs  $\varphi(s)$  might be off by one step. Confirming this will solidify our trust in PBRS correctness (the tests likely covered it, but just to be sure, given the warning scenario). Essentially: can a seed have `epochs_in_stage=0` in the info at the moment reward is calculated? If so, it means a transition just happened.
- **Q3: Are seeds in TRAINING stage contributing to `val_acc`?** The reward uses `acc_delta` during TRAINING when no counterfactual is available. If during TRAINING the seed is not actually affecting the model's predictions (common if the seed is in a “shadow” state being trained offline until blending), then any `val_acc` changes are from the host or other seeds. In

that case, giving reward for `acc_delta` in TRAINING is effectively crediting the seed for host's improvements. We assumed this is intentional (to keep the agent motivated pre-blending). Could we verify if in the system, a seed in TRAINING has any effect on validation accuracy? If not, this proxy reward is a pure shaping heuristic. It might be fine, but we should be cautious: it could encourage keeping a seed training as a way to piggyback on host learning (as we discussed). If this becomes an issue, maybe the host's baseline improvements should be accounted separately. Clarification on this would inform whether we should adjust the proxy weight or not worry (perhaps host doesn't improve much without seeds, so it's negligible).

- **Q4: DEFAULT\_MIN\_FOSSILIZE\_CONTRIBUTION value** – What is the numeric value of this constant (from `esper.leyline`)? The code uses it to decide contributing vs non-contributing seeds for terminal bonus. Is it 0 (meaning any non-negative improvement counts) or a small positive like 0.5 or 1.0%? This threshold is critical: if it's 0, then even seeds that break-even get +3, which might allow slight exploits; if it's sizable, it sets a quality bar. We assume it's >0 (since comments talk about “prevents bad fossilizations from being NPV-positive”<sup>6</sup>). Knowing this number helps us calibrate how strict the final gating is. If it's, say, 1.0, that's a high bar and maybe too stringent if combined with other penalties.
- **Q5: Multi-Seed Parallelism** – How many seeds can be active concurrently (how many “slots”)? The reward config references “3 slots max” in a comment<sup>22 23</sup>. If multiple seeds can exist at once, some parts of reward (like synergy, or the sum of contributions) come into play. We should clarify if the agent manages seeds one at a time (sequentially) or can germinate a new seed while another is holding. The `germinate_with_seed_penalty` suggests it's possible but discouraged to have >1 simultaneously. But synergy bonus implies at least sometimes multiple seeds co-exist. If indeed up to 3 seeds can be active/fossilized, we should confirm how the reward function aggregates things: e.g., does `seed_contribution` only refer to the currently controlled seed, while others might be fossilized (contributing to accuracy but not generating reward except final bonus)? Understanding this will help ensure the agent isn't under/over-incentivized to juggle seeds. (From the final bonus, it seems up to 3 fossilized seeds are expected, likely corresponding to 3 slots.)
- **Q6: Long-Term Training Dynamics** – Does the PPO training involve any curriculum or phase changes in reward mode? The code has multiple modes (SPARSE, MINIMAL, etc.). Is there an existing plan to use one mode then switch to another (e.g., train with shaped then fine-tune with sparse)? The research doc hints the team considered starting with simplified then moving to shaped<sup>71</sup>. Clarifying if such switching is used (or intended) would help us align recommendations. For example, if eventually they want the policy to also perform well under sparse (like a final fine-tuning without shaping to truly validate on objective), that could change how we evaluate success. If not, we assume one reward mode is used throughout training.
- **Q7: Validation vs Training Accuracy for Rewards** – We notice reward is based on `val_acc` (validation accuracy). This is somewhat unusual in RL – using a validation metric inside the reward loop (since validation is usually an evaluation outside of training). We assume here `val_acc` just refers to a held-out set performance measured each epoch as the target metric. It's fine as long as it's consistently available. We'd like to confirm that this doesn't introduce non-stationarity (if the validation set changes or if it's just the current performance on a fixed set, it's okay). Essentially, we want to ensure the reward the agent sees is a reliable signal (not too noisy from epoch to epoch due to sampling variance). If validation accuracy is computed on enough data to be stable per epoch (~0.1% fluctuations or so), then the small attribution differences make sense. If it's very noisy, that might be why steepness=3 (to not overreact to  $\pm 0.1\%$

fluctuations). Clarification of how stable val\_acc is per epoch will confirm whether the current anti-noise parameters are adequate.

These clarifications would help fine-tune our recommendations (e.g., if val\_acc is noisy, maybe increase attribution\_sigmoid\_steepness to not give credit too easily; if multiple seeds are allowed, ensure synergy and combined contributions are handled well, etc.). Overall, they don't indicate flaws, just points where an explicit confirmation from the environment details would ensure we target the reward shaping exactly right.

---

1 2 39 **www.andrewng.org**

<https://www.andrewng.org/publications/policy-invariance-under-reward-transformations-theory-and-application-to-reward-shaping/>

3 5 35 36 37 **B6-DRL-02.md**

<https://github.com/tachyon-beep/esper-lite/blob/a28e21a19a6289ea34f8ddc1d5711e6ccb6e638b/docs/bugs/batch6/B6-DRL-02.md>

4 6 13 14 15 17 18 19 22 23 24 25 26 27 28 29 30 33 34 38 48 50 51 53 54 78 **rewards.py**

<https://github.com/tachyon-beep/esper-lite/blob/a28e21a19a6289ea34f8ddc1d5711e6ccb6e638b/src/esper/simic/rewards/rewards.py>

7 8 11 12 16 20 21 40 41 47 52 55 56 57 58 59 60 61 62 63 64 65 69 70 71 72 73 74 75 76 77 **reward-function-design-for-morphogenetic-controller.md**

<https://github.com/tachyon-beep/esper-lite/blob/a28e21a19a6289ea34f8ddc1d5711e6ccb6e638b/docs/research/reward-function-design-for-morphogenetic-controller.md>

9 10 42 43 44 45 66 67 68 **B6-PT-01.md**

<https://github.com/tachyon-beep/esper-lite/blob/a28e21a19a6289ea34f8ddc1d5711e6ccb6e638b/docs/bugs/batch6/B6-PT-01.md>

31 32 46 49 **Reward Hacking in Reinforcement Learning | Lil'Log**

<https://lilianweng.github.io/posts/2024-11-28-reward-hacking/>