

Search Self-Distillation

2026-03-02

“generalized reasoning engine”. By replacing standard gradient steps with search in the inner loop, we avoid the trap of just doing Supervised Fine-Tuning. The teacher acts as the search heuristic, and the student compiles those discoveries back into its base weights, staying strictly on-policy to prevent distribution shift.

This formulation heavily utilizes the latent reasoning tokens to bridge the gap between exploration and distillation, clarifying the model’s intent at each step.

1. The Search-Distillation Objective Function

Let the policy be π_θ . For a given prompt s , a complete trajectory τ consists of a sequence of intermediate reasoning tokens $r = (r_1, \dots, r_k)$ and a final answer/action a . So, $\tau = (r, a)$.

The Inner Loop (Teacher’s Search): Instead of Test-Time Training via gradients, the Teacher runs a search algorithm $\mathcal{S}$ (e.g., MCTS or a STaR-like rejection sampling) over the latent reasoning space. The search is guided by the policy’s own prior probabilities and an environment verifier $V(s, \tau) \in \{0, 1\}$.

The output of the search is a golden trajectory $\tau^* = (r^*, a^*)$ that successfully solves the prompt:

$$\tau^* = \operatorname{argmax}_{\tau \in \mathcal{S}(\pi_\theta, s)} V(s, \tau)$$

The Outer Loop (Student’s Distillation): The Student (which shares the exact same weights θ) is then updated to maximize the likelihood of this verifiable reasoning path. To maintain stability in RLVR, we optimize this using a policy gradient method like GRPO, treating the discovered τ^* as the behavioral target.

The distillation objective forces the model’s latent representations to align with the successful reasoning trace:

$$\mathcal{L}_{distill}(\theta) = -\mathbb{E}_{s \sim \mathcal{D}, \tau^* \sim \mathcal{S}} \left[\sum_{t=1}^{|\tau^*|} \log \pi_{\theta}(\tau_t^* | s, \tau_{<t}^*) \right]$$

If you want to maintain the relative advantage framing (like in GRPO), you can structure it as an advantage-weighted loss where trajectories found via search that yield high verifier scores receive a positive advantage $+A$, and failed searches receive $-A$.

2. Algorithm: Generalized Reasoning via On-Policy Search

Here is how the execution loop looks. Notice how it seamlessly supports continual learning because the student and teacher are always on the same distribution.

Algorithm: Inference-Search Distillation (The “New MAML”) **Require:** Initial policy weights θ_0 , Search algorithm $\mathcal{S}$ (e.g., MCTS), Verifier V , Learning rate η .

1. **Initialize** policy $\pi_{\theta} \leftarrow \pi_{\theta_0}$
 2. **For** each training iteration $k = 1, 2, \dots$ **do**:
 3. **Sample** a batch of prompts $\{s_i\}_{i=1}^B$ from the task distribution.
 4. **Teacher Phase (Self-Induced Reasoning Search):**
 5. **For** each prompt s_i **do**:
 6. Execute search $\mathcal{S}$ using π_{θ} as the rollout prior to explore reasoning tokens r and actions a .
 7. Evaluate leaf nodes using the environment verifier $V(s_i, \tau)$.
 8. Extract the golden reasoning trajectory: $\tau_i^* = (r_i^*, a_i^*)$.
 9. **End For**
 10. **Filter** out prompts where the search failed to find a verifiable solution ($V = 0$).
 11. **Student Phase (On-Policy Distillation):**
 12. Compute the distillation loss over the successful reasoning tokens and actions:
 13. $\mathcal{L}(\theta) = -\frac{1}{N_{success}} \sum_i \log \pi_{\theta}(\tau_i^* | s_i)$
 14. Update the shared weights: $\theta \leftarrow \theta - \eta \nabla_{\theta} \mathcal{L}(\theta)$
 15. **End For**
-

3. Implementation Strategy

Because this architecture isolates the heavy compute to the forward-pass search tree rather than requiring ephemeral backwards passes, it's highly scalable.

If you were to prototype this, you could easily build it into a minimal deep learning framework. You'd just need to abstract the `SearchAlgorithm` as a modular class that takes a prompt and the current model state, returning a list of token IDs (the golden trajectory). Then, your outer training loop remains a dead-simple PyTorch optimization step over those returned tokens, perfectly suited for a lightweight RL LLM post-training package like `minpost`.