

TTCL WORKSHOP · NEURIPS 2026 · POSTER

Position: A Blueprint for the Reflect-Abstract-Integrate Architecture for Agent Self-Evolution from Production Failures

Rudrendu Kumar Paul, Sourav Nandy

Boston University

University of Texas at Austin

Production agents log their failures, then discard what those failures could teach

CURRENT PRACTICE

A compliance agent formats a gap analysis as bullets when the downstream consumer expects JSON with severity scores. It retries with the same error, and an operator escalates to manual work. Most frameworks keep only the error log.

1,600+

multi-agent traces analyzed, 14 categories of failure (Cemri et al. 2025)

26%

upper bound on relative improvement in task success from step-level debugging feedback (Zhu et al. 2025)

HOW RAI DIFFERS FROM PRIOR METHODS

Method	Causal	Rules	Forgetting	No weights
Reflexion	✗	✗	✗	✓
ExpeL	✗	Partial	✗	✓
EvolveR	✗	✓	✗	✗
ETO	✗	✗	✗	✗
Voyager	✗	✗	✓	✓
RAI	✓	✓	✓	✓

Self-Refine matches Reflexion. The comparison rests on published descriptions of each method, and no experiments back it.

Three modules turn one failed trajectory into a reusable, pruned rule

THE LOOP

When a task fails ($y = 0$), **Reflect** extracts a causal chain, **Abstract** generalizes it into a rule, and **Integrate** adds the rule to the rule set R . Before each task, the top- K matching rules enter the agent's context as conditional constraints: when [condition], avoid [action].

No parameter updates. The whole loop runs in context, through rule injection at planning time. That makes it usable with API-only frontier models.

THREE MODULES

MODULE 1

Reflect: causal chain extraction

Find the earliest step where execution left the goal state, and trace whether the error is direct or propagated from an earlier step.

MODULE 2

Abstract: rule generalization

Swap task-specific identifiers for type variables, giving a rule (condition, action, priority) that can transfer to similar tasks.

MODULE 3

Integrate: importance weighting

Track how often each rule prevents a failure, and prune low-value rules to avoid rule bloat and regression.

Importance weights decide which rules stay, and the defaults are proposals

IMPORTANCE WEIGHT

$$w_i = P_i / (A_i + \epsilon)$$

A_i counts the tasks a rule matched. P_i counts the matches where it prevented a failure. A rule is pruned once $A_i \geq 20$ and $w_i < 0.1$. At $A_i = 20$ with $\epsilon = 1$, a rule that prevented failure in 3 of 20 matches survives. Raising ϵ to 20 prunes it.

When several rules match, the module injects the top K by $\text{priority} = \alpha \cdot w_i + (1 - \alpha) \cdot t_i / t$, where t_i is the rule's most recent successful application.

PROPOSED DEFAULTS

Symbol	Value	Role
ϵ	1	Laplace smoothing
K	5	rules injected per task
α	0.8	prevention over recency
δ	0.95	strategy-preservation target

Proposals only. No tuning or sensitivity study produced these values, and a deployment would likely need to adjust each one.

A manual walkthrough on a published WebShop failure, not an executed run

THE TRAJECTORY

Instruction: white blackout shades, 66 in wide and 66 in tall.
The human expert reaches $r = 1.0$ in eight actions. The IL+RL agent (Yao et al. 2022) reaches $r = 0.2$ in three, so $y = 0$ under $\theta = 1.0$.

Scope. No RAI implementation exists. The walkthrough shows how the modules compose. It does not show that Abstract finds rules of the right generality.

WHAT REFLECT WOULD FIND

Step	Agent action	Finding
1	Search "65 inches in width"	Divergence point: contradicts the stated 66 in
2	Choose first item	Propagated from the malformed query
3	Choose Buy	Skips the option selection the human performs twice

Abstract would turn this two-link chain into one rule: verify each free-text attribute value against the instruction before querying, and select and check offered options before a terminal commit.

No experiments yet, and a specified test that would settle the design

OPEN PROBLEMS

- **No evaluation.** RAI is an architecture and a mechanistic argument.
- **Contradicting rules.** Priority ranking never detects two rules that prescribe opposite actions. A recency tie-break is brittle.
- **Unchecked abstraction.** Constrained decoding makes a rule well-formed but leaves its correctness unchecked.
- **Long chains.** Attribution may degrade beyond roughly eight steps (a hypothesis).

WHAT A PROPER TEST REQUIRES

- Multi-domain benchmark with machine-checkable success
- Baselines: no learning, simple retry, experience replay, Reflexion
- Metrics: failure-rate reduction and strategy preservation
- Ablate Reflect, Abstract, and importance weighting one at a time
- Multiple independent runs, $p < 0.05$ on a paired t-test over matched tasks

Learn from failures at the step level, generalize, and prune

TAKEAWAYS

- **Step-level causal chains** give a failure a cause as well as a log entry.
- **Generalize, then prune.** Type variables let one failure inform similar tasks, and importance weights keep only rules that earn their place.
- **No weight updates**, so the loop works with API-only models.
- **Untested.** This is a position paper with a specified evaluation plan.

<https://openreview.net/forum?id=HUj8VEUNKh>

SCAN FOR PAPER ON OPENREVIEW

