BRAID: EVOLVING EVALUATORS FOR RELIABLE AND EFFECTIVE SELF-EVOLVING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-evolving LLM agents iteratively modify themselves from task experience, making reliable and actionable evaluation signals critical to efficient improvement. Existing systems, however, largely evolve the Task Agent while leaving the Evaluator that diagnoses failures and provides feedback fixed. To address this issue, we introduce Branch-Realized Adaptive Interventional Dual-Evolution (BRAID), a training-free framework that is agnostic to the underlying Task-Agent search topology and treats the Evaluator as an explicitly evolvable agent with an independent evolutionary lineage. BRAID periodically generates Evaluator challengers from accumulated task trajectories, modification histories, and ground-truth outcomes. A challenger replaces the incumbent only when it passes an Interventional Reliability Gate and, through Challenger Branch Realization, produces a Task Agent branch whose realized branch utility exceeds the incumbent utility baseline. This selection goes beyond stand-alone evaluation accuracy by measuring the downstream utility of the Task Agent modification induced by each challenger's feedback. We evaluate BRAID across multiple self-evolution topologies and tasks spanning executable code, repository-level software engineering, and mathematical grading. Compared with the corresponding underlying-system baselines, BRAID consistently improves held-out performance under matched search-action budgets, while Frozen-Evaluator controls isolate the additional gains from Evaluator evolution. These gains persist on Token–Performance curves after accounting for Evaluator-side computation, and ablations confirm the complementary roles of Evaluator evolution, the Interventional Reliability Gate, and Challenger Branch Realization.

1 INTRODUCTION

Large language model (LLM) agents are increasingly being developed as self-evolving systems that improve through task experience. Recent training-free approaches let agents modify their prompts, tools, control logic, or source code and iteratively search for better variants (Robeyns et al., 2025; Zhang et al., 2026a; Wang et al., 2026; Zhang et al., 2026b). Despite differences in candidate generation, evolutionary topology, and search-budget allocation, these methods rely on evaluation signals to select candidates and guide subsequent Task Agent modification.

Executable tests and environment rewards provide grounded outcomes but limited diagnosis of which decisions caused a failure or how they should be revised. An LLM-based semantic Evaluator can instead inspect intermediate task-solving processes and provide trajectory-aware feedback that informs subsequent Task Agent variants (Zhuge et al., 2025). Most training-free self-evolution frameworks, however, evolve the Task Agent while keeping evaluation fixed (Robeyns et al., 2025; Zhang et al., 2026a; Wang et al., 2026; Zhang et al., 2026b), although recent work has begun to adapt evaluation criteria or rubric models alongside agents (Iacob et al., 2026; Tian et al., 2026). Evaluator evolution requires more than standalone judgment accuracy. Judgments should remain grounded in task evidence, while feedback should induce useful Task Agent modifications. These objectives need not coincide because reliable judgment alone does not guarantee a better downstream modification.

To address this gap, we introduce Branch-Realized Adaptive Interventional Dual-Evolution (BRAID), a training-free framework that maintains an independent evolutionary lineage for the semantic Evaluator while preserving the host's Task Agent search topology and native candidate-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

processing rules. BRAID generates challengers from accumulated task trajectories, modification histories, and ground-truth outcomes, then selects among them in two stages. The *Interventional Reliability Gate* uses label interventions while keeping task answers and execution evidence fixed to test grounding in task evidence. Surviving challengers enter *Challenger Branch Realization*, where their feedback induces Task Agent branches from the same current parent and task records. Promotion requires passing the Gate and producing a branch whose realized utility exceeds the incumbent utility baseline computed from ordinary Task branches during the incumbent Evaluator's tenure. The selected Task Agent branch is returned to the host for native candidate processing.

Evaluation spans self-evolution topologies and tasks covering executable code, repository-level software engineering, and mathematical grading under matched search-action budgets with full token accounting. Across five host configurations on POLYGLOT, BRAID improves TestSet accuracy by 7.4–12.6 percentage points. Gains remain 7.0–8.0 points on SWE-BENCH VERIFIED and 6.0 points on IMO-GRADINGBENCH. Under HGM-H on POLYGLOT, Full BRAID exceeds Frozen Evaluator by 6.66 points on the WorkSet and 3.71 points on the TestSet. At shared performance targets on POLYGLOT, BRAID reduces search actions by 57.94–85.02% and total token consumption by 1.28–40.49% for HGM and HGM-H.

Our contributions are threefold.

- We make the semantic Evaluator an independently evolvable agent with a separate training-free evolutionary lineage.
- We introduce reliability- and utility-aware Evaluator selection through the Interventional Reliability Gate and Challenger Branch Realization, requiring robustness to controlled label intervention and realized downstream utility beyond the incumbent utility baseline.
- BRAID integrates with heterogeneous self-evolution hosts while preserving their native search rules. Experiments across five host configurations on POLYGLOT and cross-domain evaluations on SWE-BENCH VERIFIED and IMO-GRADINGBENCH show consistent gains under matched search-action budgets and full token accounting.

2 RELATED WORK

Open-Ended Agent Design and Self-Evolution. Open-ended search supports continual discovery by preserving diverse candidates as stepping stones for future progress (Hughes et al., 2024; Clune, 2020). PromptBreeder jointly evolves task and mutation prompts, whereas ADAS uses a fixed Meta Agent to generate code-based agents from an archive of prior discoveries (Fernando et al., 2024; Hu et al., 2025). STOP and Gödel Agent make the improvement mechanism itself modifiable (Zelikman et al., 2024; Yin et al., 2025). SICA and DGM extend self-modification to sustained agent evolution, with DGM retaining an archive for open-ended exploration (Robeyns et al., 2025; Zhang et al., 2026a). HGM guides search with clade-level metaproductivity over descendant outcomes, while Hyperagents jointly evolves the Task Agent and Meta Agent (Wang et al., 2026; Zhang et al., 2026b). BRAID instead extends the evolvable scope to evaluation by maintaining an independent semantic Evaluator lineage while preserving the host's native Task Agent search procedure.

Evaluation and Feedback for Agentic Systems. LLM-based evaluation increasingly provides actionable feedback beyond final performance measurement. LLM-as-a-Judge enables semantic assessment when automatic metrics are insufficient (Zheng et al., 2023), while Reflexion and Self-Refine use natural-language feedback to improve subsequent attempts or outputs (Shinn et al., 2023; Madaan et al., 2023). Agent-as-a-Judge further evaluates intermediate behavior in multi-step agents (Zhuge et al., 2025). These methods make evaluation more informative but generally keep the evaluation mechanism fixed during self-evolution.

Adaptive and Co-Evolving Evaluation. Recent work begins to adapt evaluation together with the improving agent (Iacob et al., 2026; Tian et al., 2026). RQGM updates its evaluation criterion across search epochs while keeping it fixed within each epoch (Iacob et al., 2026), whereas ARCO jointly updates a rubric model and task policy from on-policy trajectories (Tian et al., 2026). BRAID differs by maintaining an independent training-free evolutionary lineage for the semantic Evaluator,

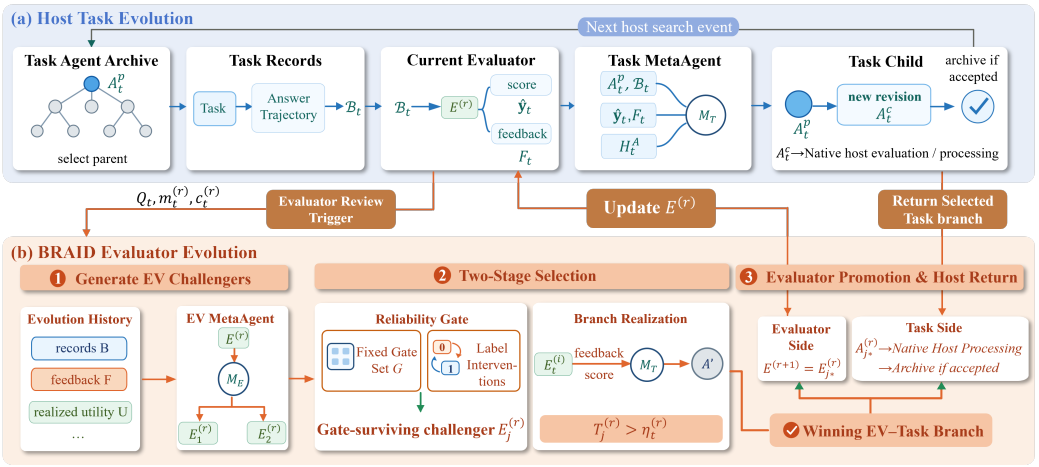


Figure 1: Overview of BRAID. The top panel shows host Task evolution, while the bottom panel shows Evaluator evolution through challenger generation, the Interventional Reliability Gate, Challenger Branch Realization, Evaluator promotion, and return of the selected Task branch to the host.

whose replacement depends jointly on intervention-based reliability and realized downstream utility while preserving the host's native Task Agent search procedure.

3 METHOD

Overview. As shown in Figure 1, BRAID maintains an independently evolving Evaluator alongside the host. During ordinary Task evolution, the incumbent Evaluator assesses Task Agent records and provides feedback for subsequent Task Agent modification. An active or passive trigger initiates an Evaluator Evolution Cycle (EEC), where an Evaluator MetaAgent generates challengers that are filtered by the Interventional Reliability Gate and then undergo Challenger Branch Realization from the same current Task parent. A challenger is promoted only if its realized branch utility exceeds the incumbent utility baseline. The selected Task Agent branch is then returned to the host for native candidate processing.

3.1 BRAID FRAMEWORK

We call the underlying self-evolution system the *host*. Let A denote a Task Agent, M_T the Task MetaAgent used by the host for Task Agent modification, M_E the Evaluator MetaAgent that proposes Evaluator challengers, and $E^{(r)}$ the incumbent Evaluator at Evaluator generation r. The host retains its original parent-selection policy, search topology, and candidate-processing rules. When the host jointly evolves the Task Agent and Task MetaAgent, M_T denotes the Task MetaAgent selected by the host for the current Task expansion.

Ordinary Task evolution. At Task expansion t, the host selects a parent A_t^p. For each task $x_{t,i}$, the parent produces answer $a_{t,i}$ and trajectory $\tau_{t,i}$, while the environment provides outcome $y_{t,i}$ and observable execution evidence $o_{t,i}$. The resulting batch is

$$\mathcal{B}_t = \{(x_{t,i}, a_{t,i}, \tau_{t,i}, o_{t,i}, y_{t,i})\}_{i=1}^{N_t}, \tag{1}$$

where N_t is the batch size. The incumbent Evaluator produces per-instance judgments and batch-level structured feedback

$$(\hat{\mathbf{y}}_t, F_t) = E^{(r)}(\mathcal{B}_t), \tag{2}$$

where $\hat{\mathbf{y}}_t = \{\hat{y}_{t,i}\}_{i=1}^{N_t}$ and F_t use the fixed output fields `score` and `feedback`, respectively. Their contents and internal organization may evolve while the interface remains fixed for host compatibility.

The Task MetaAgent then generates a candidate child

$$A_t^c = M_T\big(A_t^p, \mathcal{B}_t, \hat{\mathbf{y}}_t, F_t, \mathcal{H}_t^A\big)\,, \tag{3}$$

where $\mathcal{H}_t^A$ denotes the Task evolution history visible to M_T. After environment measurement, the realized parent-child utility is

$$U_t = S(A_t^c) - S(A_t^p)\,, \tag{4}$$

where $S(\cdot)$ is the host's training-time performance measure. The host then applies its native candidate-acceptance and archive-update rules.

Search-action and token accounting. We distinguish the search-action budget from total token consumption. A Task Agent response, a Task MetaAgent modification, or an Evaluator judgment counts as one search action only when it is used to instantiate or advance an actual Task Agent branch. Ordinary Task evolution and Challenger Branch Realization share $B_{\text{search}} = 800$ search actions. During Challenger Branch Realization, the Evaluator judgment, ensuing Task MetaAgent modification, and corresponding Task Agent response all consume this budget even if the realized branch is rejected.

Evaluator challenger generation and Interventional Reliability Gate computation do not consume search actions because they do not instantiate Task Agent branches. Total token consumption nevertheless includes every model call on both the Task and evaluation sides, including unsuccessful challengers, rejected branches, and Gate evaluations.

3.2 Evaluator Evolution

The incumbent Evaluator participates in ordinary Task evolution until an active or passive condition triggers an EEC. Active evolution responds to insufficient evaluation accuracy or negative incumbent-tenure utility, whereas passive evolution retains periodic opportunities to search for a better Evaluator.

3.2.1 Active Evolution

The incumbent's evaluation accuracy on batch t is

$$Q_t = \frac{1}{N_t}\sum_{i=1}^{N_t} \mathbf{1}[\hat{y}_{t,i} = y_{t,i}]\,. \tag{5}$$

BRAID retains all ordinary Task branch utilities produced during the tenure of $E^{(r)}$

$$\mathcal{U}_t^{(r)} = \Big\{U_\tau : \tau \le t,\ E^{(r)} \text{ was the incumbent when branch } \tau \text{ was generated}\Big\}\,. \tag{6}$$

The corresponding incumbent-tenure median utility is

$$m_t^{(r)} = \operatorname{median}\Big(\mathcal{U}_t^{(r)}\Big)\,. \tag{7}$$

Let $c_t^{(r)}$ denote the number of ordinary evaluation batches completed since the previous Evaluator review, and $n_{\min}$ is the minimum number of ordinary evaluation batches required before an active review can be triggered. Once $c_t^{(r)} \ge n_{\min}$, BRAID initiates an active EEC if either

$$Q_t < q_{\min} \qquad \text{or} \qquad m_t^{(r)} < 0. \tag{8}$$

The first condition identifies insufficient agreement with environment outcomes, while the second identifies negative incumbent-tenure realized utility. The review counter $c_t^{(r)}$ is reset after every review. In contrast, $\mathcal{U}_t^{(r)}$ continues to accumulate until $E^{(r)}$ is replaced.

3.2.2 Passive Evolution

If neither active condition holds, BRAID initiates a passive EEC when

$$c_t^{(r)} \ge n_{\max}, \qquad n_{\max} \ge n_{\min}, \tag{9}$$

where $n_{\max}$ is the passive-review threshold. This provides periodic search opportunities even when the incumbent remains stable.

3.2.3 CHALLENGER GENERATION

When an EEC is triggered, M_E generates K challengers from the incumbent and the accumulated Evaluator evolution history:

$$E_j^{(r)} = M_E\Big(E^{(r)}, \mathcal{H}_t^E; \xi_j\Big), \qquad j = 1, \ldots, K. \tag{10}$$

The history $\mathcal{H}_t^E$ contains ordinary Task trajectories, incumbent judgments and feedback, subsequent Task Agent modifications, measured outcomes, and previous Evaluator modifications. It excludes the interventions and results used in the current Gate. The randomness of challenger generation is denoted by ξ_j, and each challenger remains fixed throughout the current EEC.

The Evaluator is treated as an editable agent program. Its instructions, reasoning procedure, diagnostic strategy, and feedback organization may be modified, while the output field names `score` and `feedback` remain fixed. Challenger generation does not update the parameters of the underlying language model and is therefore training-free.

3.3 RELIABILITY- AND UTILITY-AWARE EVALUATOR SELECTION

BRAID selects Evaluator challengers in two stages. The Interventional Reliability Gate filters challengers whose judgments become overly dependent on externally supplied labels. Challenger Branch Realization then measures whether the feedback from each surviving challenger produces sufficient realized downstream utility.

3.3.1 INTERVENTIONAL RELIABILITY GATE

The Gate changes the label exposed to the Evaluator while keeping the task, answer, trajectory, execution evidence, and controller-held outcome unchanged. Let $\mathcal{G}$ denote the GateSet used in the current EEC. Its records are divided into a normal subset $\mathcal{G}_{\mathrm{n}}$ and an intervened-label subset $\mathcal{G}_v$. The exposed label is defined as

$$\ell_i = \begin{cases} y_i, & i \in \mathcal{G}_{\mathrm{n}}, \\ \Pi(y_i; \xi_i), & i \in \mathcal{G}_v, \end{cases} \qquad \Pi(y_i; \xi_i) \neq y_i. \tag{11}$$

For records in the intervention subset $\mathcal{G}_v$, the exposed label is sampled from an incorrect class,

$$\Pi(y_i; \xi_i) \sim \mathrm{Uniform}(\mathcal{Y} \setminus \{y_i\}). \tag{12}$$

For binary outcomes, this reduces to deterministic label reversal, $\Pi(y_i; \xi_i) = 1 - y_i$. For IMO-GradingBench, where $\mathcal{Y} = \{0, 1, 6, 7\}$, one of the three incorrect labels is sampled uniformly. Evaluator correctness is defined by exact agreement with the unchanged controller-held outcome y_i, so all incorrect predictions contribute equally regardless of the numerical distance between class labels. The incumbent and all challengers receive the same Gate records and the same interventions within an EEC.

For an Evaluator E, its accuracy on the normal subset is

$$q_{\mathrm{n}}(E) = \frac{1}{|\mathcal{G}_{\mathrm{n}}|} \sum_{i \in \mathcal{G}_{\mathrm{n}}} \mathbf{1}\Big[\hat{y}_i^{(E)} = y_i\Big]. \tag{13}$$

Its accuracy on the intervened-label subset is

$$q_v(E) = \frac{1}{|\mathcal{G}_v|} \sum_{i \in \mathcal{G}_v} \mathbf{1}\Big[\hat{y}_i^{(E)} = y_i\Big]. \tag{14}$$

Here, $q_{\mathrm{n}}(E)$ measures ordinary evaluation accuracy when the exposed labels are correct, while $q_v(E)$ measures the Evaluator's ability to recover the correct outcome when the exposed labels have been intervened.

For the current EEC r, BRAID determines the aggregation weights from the Gate performance of the incumbent Evaluator $E^{(r)}$. We first define

$$w_{\mathrm{n}}^{(r)} = 1 + \Big(1 - q_{\mathrm{n}}(E^{(r)})\Big), \qquad w_v^{(r)} = \lambda_v + \Big(1 - q_v(E^{(r)})\Big). \tag{15}$$

The normalized weights are

$$\alpha_{\mathrm{n}}^{(r)} = \frac{w_{\mathrm{n}}^{(r)}}{w_{\mathrm{n}}^{(r)} + w_v^{(r)}}, \qquad \alpha_v^{(r)} = \frac{w_v^{(r)}}{w_{\mathrm{n}}^{(r)} + w_v^{(r)}}. \tag{16}$$

Here, $\lambda_v > 0$ specifies the base priority assigned to robustness against label intervention. We use $\lambda_v = 2.0$, which ensures $\min w_v^{(r)} \geq \max w_{\mathrm{n}}^{(r)}$ because $q_{\mathrm{n}}, q_v \in [0, 1]$. Consequently, lower incumbent accuracy on either subset increases the corresponding weight, while the intervened-label subset receives additional priority through λ_v. These weights are computed once from the incumbent and then fixed for evaluating both the incumbent and all challengers within the same EEC. The weighted Gate score is

$$G\Big(E \mid E^{(r)}\Big) = \alpha_{\mathrm{n}}^{(r)} q_{\mathrm{n}}(E) + \alpha_v^{(r)} q_v(E). \tag{17}$$

A challenger passes the Gate if its intervened-label accuracy remains within tolerance ε of the incumbent and its weighted Gate score is no lower:

$$\mathcal{C}^{(r)} = \Big\{ j \in \{1, \ldots, K\} : q_v\Big(E_j^{(r)}\Big) \geq q_v\Big(E^{(r)}\Big) - \varepsilon,\ G\Big(E_j^{(r)} \mid E^{(r)}\Big) \geq G\Big(E^{(r)} \mid E^{(r)}\Big) \Big\}. \tag{18}$$

We use $\varepsilon = 0$ in all experiments, imposing zero tolerance for a decrease in intervened-label accuracy. The first condition prevents a challenger from gaining overall accuracy by becoming less robust to misleading labels. The second condition requires its weighted performance across the normal and intervened-label subsets to be no worse than that of the incumbent. If $\mathcal{C}^{(r)} = \varnothing$, the incumbent is retained and Challenger Branch Realization is skipped.

3.3.2 Challenger Branch Realization

For every $j \in \mathcal{C}^{(r)}$, the challenger evaluates the same normal Task record batch $\mathcal{B}_p$ from the same current parent A_p selected for this EEC:

$$\Big(\hat{\mathbf{y}}_j^{(r)}, F_j^{(r)}\Big) = E_j^{(r)}(\mathcal{B}_p)\,, \tag{19}$$

where $\hat{\mathbf{y}}_j^{(r)}$ denotes the challenger's per-instance judgments, and $F_j^{(r)}$ is the challenger's structured feedback. The Task MetaAgent then generates the corresponding challenger branch:

$$A_j^{(r)} = M_T\Big(A_p, \mathcal{B}_p, \hat{\mathbf{y}}_j^{(r)}, F_j^{(r)}, \mathcal{H}_t^A\Big)\,. \tag{20}$$

GateSet records, including the label-intervention records, are used only for Evaluator selection and are not supplied to M_T for Task Agent modification. Challenger branches are mutually isolated, and their generation follows the run-level random-number stream of the complete self-evolution process.

After environment measurement, the realized branch utility induced by challenger j is

$$T_j^{(r)} = S\Big(A_j^{(r)}\Big) - S(A_p). \tag{21}$$

The incumbent does not generate an additional Task Agent branch during this stage. Instead, each $T_j^{(r)}$ is compared with the incumbent utility baseline:

$$\eta_t^{(r)} = \max\Big\{0, \mathrm{median}\Big(\mathcal{U}_t^{(r)}\Big)\Big\}\,. \tag{22}$$

The challengers eligible for replacement are

$$\mathcal{J}_t^{(r)} = \Big\{ j \in \mathcal{C}^{(r)} : T_j^{(r)} > \eta_t^{(r)} \Big\}\,. \tag{23}$$

If $\mathcal{J}_t^{(r)} \neq \varnothing$, BRAID selects

$$j^* = \underset{j \in \mathcal{J}_t^{(r)}}{\arg\max}\ T_j^{(r)} \tag{24}$$

Table 1: Endpoint Task Agent accuracy (%). Δ denotes the absolute BRAID gain over the corresponding native host.

		WorkSet			TestSet		
Benchmark	**Host**	**Native**	**BRAID**	**Δ**	**Native**	**BRAID**	**Δ**
Polyglot	DGM	51.7	**63.3**	**+11.6**	48.1	**57.8**	**+9.6**
	DGM-H	53.3	**66.7**	**+13.4**	49.6	**60.0**	**+10.4**
	HGM	55.0	**66.7**	**+11.7**	52.6	**63.0**	**+10.4**
	HGM-H	60.0	**68.3**	**+8.3**	57.8	**65.2**	**+7.4**
	SICA	50.0	**65.0**	**+15.0**	46.7	**59.3**	**+12.6**
SWE-bench Verified	HGM	60.0	**68.3**	**+8.3**	57.0	**64.0**	**+7.0**
	HGM-H	63.3	**73.3**	**+10.0**	60.0	**68.0**	**+8.0**
IMO-GradingBench	HGM	63.0	**69.0**	**+6.0**	60.5	**66.0**	**+5.5**
	HGM-H	65.0	**71.0**	**+6.0**	62.5	**70.0**	**+7.5**

and promotes the corresponding Evaluator:

$$E^{(r+1)} = E^{(r)}_{j^*}. \tag{25}$$

The selected Task Agent branch $A^{(r)}_{j^*}$ is returned to the host for native candidate processing and becomes available to subsequent Task evolution if accepted. If no challenger exceeds the baseline, $E^{(r)}$ is retained and no challenger branch is returned.

4 Experiments

4.1 Experimental Setup

Benchmarks and Data Configuration. Our evaluation spans **Polyglot** (Gauthier, 2024), **SWE-bench Verified** (Jimenez et al., 2024; Chowdhury et al., 2024), and **IMO-GradingBench** (Luong et al., 2025), covering multilingual code editing, repository-level software engineering, and Olympiad-level mathematical grading. For each benchmark, the WorkSet supports Task Agent evolution, the GateSet is reserved for the Interventional Reliability Gate, and the TestSet is held out for final evaluation. Detailed splits and benchmark configurations are provided in Appendix A.

Hosts and Evaluation Protocol. BRAID is instantiated with **DGM** (Zhang et al., 2026a), **DGM-H**, **HGM** (Wang et al., 2026), **HGM-H**, and **SICA** (Robeyns et al., 2025). Following Hyperagents (Zhang et al., 2026b), the suffix -H denotes joint evolution of the Task Agent and Task MetaAgent while preserving the underlying host search procedure. Every native and BRAID-augmented run follows the search-action accounting in Section 3 and is capped at $B_{\text{search}} = 800$ search actions. Total token consumption is reported separately and includes all computation, including Evaluator-only operations outside the search-action budget. Further implementation details are provided in Appendices A and A.2.

4.2 Main Results

Across all evaluated hosts and benchmarks, BRAID improves both WorkSet and TestSet accuracy under matched search opportunities. For Polyglot, gains span all five hosts, ranging from 8.3 to 15.0 percentage points on the WorkSet and from 7.4 to 12.6 points on the TestSet, indicating that the benefit is not tied to a particular host search topology. The gains persist on SWE-bench Verified, where HGM improves by 8.3 and 7.0 points on the WorkSet and TestSet, while HGM-H improves by 10.0 and 8.0 points. A similar pattern appears on IMO-GradingBench, with both hosts gaining 6.0 points on both splits. Table 1 reports all endpoint results, while evolution trajectories, search structures, and language-wise capability analyses are provided in Appendices E.1, E.2, and C.6.

Table 2: Endpoint component ablation under HGM-H on POLYGLOT. Δ denotes the gain over the common initial Task Agent.

	WorkSet		TestSet	
Setting	**Final**	**Δ**	**Final**	**Δ**
Native Host	60.00	+28.33	57.78	+23.71
Frozen EV	61.67	+30.00	61.48	+27.41
Reliability-only	63.33	+31.67	62.96	+28.89
Utility-only	65.00	+33.33	62.22	+28.15
Full BRAID	**68.33**	**+36.67**	**65.19**	**+31.11**

Table 3: TestSet performance of the fixed-Evaluator intervention under HGM-H on POLYGLOT.

Setting	**Initial**	**TestSet**	**Gain**
Native HGM-H	34.1	57.8	+23.7
Fixed Final Evaluator	34.1	63.0	+28.9
Full BRAID	34.1	**65.2**	**+31.1**

4.3 ABLATION STUDY

Under HGM-H on POLYGLOT, the ablation compares the Native Host, Frozen Evaluator (EV), Reliability-only, Utility-only, and Full BRAID. Native Host removes BRAID, Frozen EV fixes the initial Evaluator, and Reliability-only and Utility-only retain only the Interventional Reliability Gate and Challenger Branch Realization, respectively. Full BRAID uses both criteria. Table 2 shows that Frozen EV improves over the Native Host, both partial variants improve further, and Full BRAID performs best at 68.33% WorkSet and 65.19% TestSet accuracy. The different WorkSet and TestSet behavior of Reliability-only and Utility-only supports the complementary roles of reliability and realized downstream utility. Variant definitions, trajectories, and parameter sensitivity are provided in Appendices A.3, E.3, and B.1.

4.4 EVALUATOR EVOLUTION ANALYSIS

Effect of the Final Promoted Evaluator. A fixed-Evaluator intervention isolates the contribution of the evolved Evaluator. Starting from the same 34.1% initial Task Agent, HGM-H is rerun on POLYGLOT with the final promoted Evaluator fixed throughout evolution. Under the same search-action budget, Table 3 reports 57.8%, 63.0%, and 65.2% TestSet accuracy for native HGM-H, Fixed Final Evaluator, and Full BRAID, respectively. The fixed final Evaluator recovers 5.2 of BRAID's 7.4 percentage point gain, or about 70% of the observed endpoint gap. This result shows that the final promoted Evaluator alone preserves much of BRAID's downstream improvement, while the full adaptive procedure reaches a stronger endpoint.

Validation of the Incumbent Utility Baseline. All nine EECs from the original HGM-H run on POLYGLOT are audited to validate the incumbent utility baseline $\eta_t^{(r)}$. At each EEC, a matched incumbent branch is realized from the same Task parent and task-record batch as the audited challenger. Its realized utility replaces $\eta_t^{(r)}$, while Gate eligibility and the promotion procedure remain unchanged. The two references agree on eight of nine promotion decisions, or 88.9%. An end-to-end rerun under the same 800 search-action budget again produces two Evaluator promotions and comparable TestSet accuracy, although its WorkSet endpoint is lower. Together, these results support $\eta_t^{(r)}$ as a practical decision-level approximation without requiring an extra incumbent branch at every EEC. Detailed EEC-level comparisons are reported in Appendix C.5.

4.5 RELIABILITY ANALYSIS

Promoted Evaluator lineages become more robust to label interventions on Gate records, while the relationship between reliability change and realized downstream utility varies across hosts and benchmarks. This variation indicates that the two signals capture distinct properties of Evaluator

Table 4: Validation of the incumbent utility baseline under HGM-H on POLYGLOT. Decision agreement covers all nine audited EECs, and both runs use the same 800-action budget.

Utility Reference	Decision Agreement	EECs	Promotions	WorkSet	TestSet
Incumbent utility baseline	8/9 (88.9%)	9	2	**68.33**	65.19
Matched incumbent realization		8	2	65.00	**65.93**

Table 5: Evaluator reliability on unseen intervention records under HGM and HGM-H on POLYGLOT.

Host	Normal	Intervened-label	Balanced Reliability
HGM	77.53→62.73	54.51→**96.21**	66.02→**79.47**
HGM-H	67.16→57.69	67.48→**81.52**	67.32→**69.61**

Table 6: Average savings at shared POLYGLOT performance targets.

Host	Actions (%)	Tokens (%)
HGM	85.02	40.49
HGM-H	57.94	1.28

quality. Because Gate records influence Evaluator selection, the same lineages are also evaluated on disjoint unseen intervention records. Table 5 shows higher intervened-label accuracy and weighted Gate Scores for the final promoted Evaluators of HGM and HGM-H, although normal-subset accuracy decreases. Full Gate progressions, correlation analyses, and unseen-intervention results are provided in Appendices C.1, C.2, and C.4.

4.6 COMPUTATIONAL EFFICIENCY

Computational efficiency is assessed at accuracy targets reached by both native and BRAID runs, using the search actions and cumulative tokens consumed at first attainment. For HGM and HGM-H, this yields 14 shared targets corresponding to 20 through 33 solved POLYGLOT tasks. For budget measure B, the relative saving at each target is

$$\mathrm{Saving}_B(t) = \frac{B_{\mathrm{Native}}(t) - B_{\mathrm{BRAID}}(t)}{B_{\mathrm{Native}}(t)} \times 100\%,$$

which is averaged across all shared targets. Table 6 shows that HGM reduces both search actions and tokens, while HGM-H retains substantial action savings but only modest token savings because Evaluator evolution adds evaluation-side computation. These matched-performance savings do not imply lower full-run cost, since maintaining the Evaluator lineage can increase total token consumption over a complete run. Detailed token trajectories, full-run cost breakdowns, and a matched total token budget comparison are provided in Appendix D.

5 CONCLUSION

This paper introduces BRAID, a training-free framework that evolves the Evaluator alongside Task Agent search, which is agnostic to the host's native search and candidate-processing rules. Evaluator promotion combines intervention-based reliability with realized downstream utility. Across the evaluated hosts and task domains, BRAID improves endpoint Task Agent accuracy under matched search-action budgets. Under HGM-H on POLYGLOT, ablations show complementary roles of the Interventional Reliability Gate and Challenger Branch Realization, while the final promoted Evaluator recovers much of BRAID's endpoint gain when fixed throughout a new run. On POLYGLOT, held-out interventions under HGM and HGM-H further show improved robustness to misleading labels. BRAID also reaches shared performance targets with lower search cost in some hosts, although full runs can require more tokens. These results support adapting the evaluation mechanism during agent self-evolution. Future work could study longer horizons and changing task distributions.

AI USE STATEMENT

In this work, we used generative AI tools to provide feedback on research methodology or experiments, assist with translation, and interpret some results. We have not used generative AI tools to generate synthetic data sets, provide critical ingredients for proving mathematical claims, or assist in the writing of proofs, and the rest of the required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools to edit a research paper for improving readability, brainstorming, and searching for some information. We have reviewed all AI-assisted work. AI-assisted suggestions concerning research methodology, experimental design, result interpretation, and manuscript presentation were checked by the authors against the implementation, experimental results, and relevant literature, and all AI-assisted text was reviewed and revised by the authors as needed. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing swe-bench verified. https://openai.com/index/introducing-swe-bench-verified/, 2024.

Jeff Clune. Ai-gas: Ai-generating algorithms, an alternate paradigm for producing general artificial intelligence, 2020. URL https://arxiv.org/abs/1905.10985.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Paul Gauthier. o1 tops aider's new polyglot leaderboard, 2024.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025.

Edward Hughes, Michael Dennis, Jack Parker-Holder, Feryal Behbahani, Aditi Mavalankar, Yuge Shi, Tom Schaul, and Tim Rocktaschel. Open-endedness is essential for artificial superhuman intelligence. *arXiv preprint arXiv:2406.04268*, 2024.

Alex Iacob, Andrej Jovanović, William F. Shen, Daniel Burkhardt, Meghdad Kurmanji, Nurbek Tastan, Lorenzo Sani, Niccolò Alberto Elia Venanzi, Ambroise Odonnat, Zeyu Cao, Bill Marino, Xinchi Qiu, and Nicholas D. Lane. The red queen Gödel machine: Co-evolving agents and their evaluators. *arXiv preprint arXiv:2606.26294*, 2026. doi: 10.48550/arXiv.2606.26294. URL https://arxiv.org/abs/2606.26294.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. URL https://arxiv.org/abs/2310.06770.

Thang Luong, Dawsen Hwang, Hoang H. Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, Junsu Kim, Garrett Bingham, Jonathan Lee, Swaroop Mishra, Alex Zhai, Clara Huiyi Hu, Henryk Michalewski, Jimin Kim, Jeonghyun Ahn, Junhwi Bae, Xingyou Song, Trieu H. Trinh, Quoc V. Le, and Junehyuk Jung. Towards robust mathematical reasoning, 2025. URL https://arxiv.org/abs/2511.01846.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. In *ICLR 2025 Workshop on Scaling Self-Improving Foundation Models without Human Supervision*, 2025. URL https://openreview.net/forum?id=rShJCyLsOr.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. *URL https://arxiv. org/abs/2303.11366*, 1:5, 2023.

Zihang Tian, Jingsen Zhang, Rui Li, Xiaohe Bo, Yuanzi Li, and Xu Chen. ARCO: Adaptive rubrics with co-evolution for multi-step LLM-based agents. *arXiv preprint arXiv:2606.21262*, 2026. doi: 10.48550/arXiv.2606.21262. URL https://arxiv.org/abs/2606.21262.

Wenyi Wang, Piotr Piekos, Nanbo Li, Firas Laakom, Yimeng Chen, Mateusz Ostaszewski, Mingchen Zhuge, and Jürgen Schmidhuber. Huxley-Gödel machine: Human-level coding agent development by an approximation of the optimal self-improving machine. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=T0EiEuhOOL.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursively self-improvement. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27890–27913, 2025.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (STOP): Recursively self-improving code generation, 2024. URL https://openreview.net/forum?id=1gkePTsAWf.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Tjarko Lange, and Jeff Clune. Darwin Gödel machine: Open-ended evolution of self-improving agents. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=pUpzQZTvGY.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. HyperAgents. *arXiv preprint arXiv:2603.19461*, 2026b. doi: 10.48550/arXiv.2603.19461. URL https://arxiv.org/abs/2603.19461.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Mingchen Zhuge, Changsheng Zhao, Dylan R. Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-Judge: Evaluate agents with agents. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 80569–80611. PMLR, 2025. URL https://proceedings.mlr.press/v267/zhuge25a.html.

A Additional Experimental Details

A.1 Benchmark and Host Configurations

Polyglot is divided into WorkSet, GateSet, and TestSet splits of $60/30/135$. For SWE-bench Verified, the corresponding split is $60/30/100$. Its WorkSet follows the fixed 60-instance SWE-Verified-60 subset used by HGM (Wang et al., 2026), while the GateSet and TestSet are sampled without overlap from the remaining instances. On IMO-GradingBench, we retain only examples with reference scores in $\{0, 1, 6, 7\}$ and adopt a $100/50/200$ split. Predictions are evaluated by exact match. Performance on Polyglot and SWE-bench Verified is measured using task-level accuracy.

Across all experiments, the Task Agent, Task MetaAgent, Evaluator, and Evaluator MetaAgent use GPT-5.6 Luna with low reasoning effort and a context window of $1{,}050{,}000$ tokens. Evaluator evolution uses an intervened-label fraction of 0.50, an active-review accuracy threshold of $q_{\min} = 0.70$, review-count thresholds of $n_{\min} = 5$ and $n_{\max} = 8$, and $K = 2$ challengers in each Evaluator Evolution Cycle. The Interventional Reliability Gate uses zero tolerance on intervened-label accuracy in all experiments. A challenger survives the Gate only if its intervened-label accuracy q_v does not decrease relative to the incumbent and its weighted Gate score G is no lower than that of the incumbent. The definitions of the Gate accuracy measures and their weighting are provided in Section 3.

Host-specific evaluation procedures follow the corresponding native host implementations. For DGM and DGM-H, staged evaluation uses 10, 30, and 60 tasks on Polyglot and SWE-bench Verified, and 10, 50, and 100 tasks on IMO-GradingBench. In all three benchmarks, progression between successive stages uses minimum accuracy thresholds of 0.55 and 0.55. HGM and HGM-H retain the native non-staged evaluation policy with $\alpha = 0.6$ and $\beta = 1.0$ in our implementation. SICA retains its native search and evaluation procedure.

A.2 Initial Agent Configuration

All experiments use the same lightweight agent scaffold for the initial Task Agent, Evaluator, and MetaAgent. Each agent implementation contains only the basic interaction loop together with model and tool invocation, while task-specific functionality is provided through its prompt, accessible resources, and registered tools. For each benchmark, the Task Agent, Evaluator, and MetaAgent are maintained in separate repositories containing the agent implementation, prompt, manifest, tools, and workspace. The manifest fixes the executable interface by specifying the agent type, Python module, class, and entry method. Task Agent and Evaluator modification use the same initial MetaAgent implementation, with different modification targets and evolution histories supplied to each call. Large task contexts and evolution records are accessed through tools rather than inserted directly into the model context. Agent workspaces store the resources and evolution records accessible to the corresponding component through the resource interface. For repository-based tasks, the target repository is accessed separately through dedicated repository tools.

The underlying scaffold remains unchanged across benchmarks, while the initial Task Agent receives the tools required by the corresponding task interface. Table 7 lists the exact initial Task Agent tools. Across benchmarks, `resource_list` enumerates the resources available in the agent workspace, while `resource_read` retrieves the content of a selected resource. For repository-based tasks, `repository_list` exposes the repository structure, `repository_read` reads selected files, `repository_search` searches repository contents, `repository_edit` applies file modifications, and `repository_diff` exposes the resulting repository changes. The `bash` tool provides shell execution. The initial Evaluator uses `resource_list` and `resource_read` to access Task Agent records, while the initial MetaAgent additionally uses `bash` and `bundle_edit`, with `bundle_edit` providing the interface for modifying an agent repository.

The benchmark-specific tool differences follow directly from the task interfaces. Polyglot requires multilingual repository inspection and editing and also exposes Bash execution. SWE-bench Verified operates on existing software repositories and additionally provides repository-diff access. In IMO-GradingBench, the grading instance is supplied directly as task input and no repository modification is required, so the initial Task Agent only uses resource-access tools.

Table 7: Initial Task Agent tools for each benchmark.

Benchmark	Initial Task Agent Tools
POLYGLOT	`bash`, `resource_list`, `resource_read`, `repository_edit`, `repository_list`, `repository_read`, `repository_search`
SWE-BENCH VERIFIED	`repository_diff`, `repository_edit`, `repository_list`, `repository_read`, `repository_search`, `resource_list`, `resource_read`
IMO-GRADINGBENCH	`resource_list`, `resource_read`

A.3 COMPONENT ABLATION IMPLEMENTATION

A common initial Task Agent and matched search-action budget are used across all variants. The original HGM-H procedure without BRAID serves as the *Native Host*, whereas *Frozen Evaluator* keeps E_0 fixed throughout Task Agent evolution and disables Evaluator Evolution Cycles. In the *Reliability-only* variant, challenger generation and the Interventional Reliability Gate are retained, whereas Challenger Branch Realization is removed. Among surviving challengers, the one with the highest weighted Gate score is promoted. *Utility-only* instead bypasses the Interventional Reliability Gate and sends all K challengers through Challenger Branch Realization. Among challengers whose realized branch utility exceeds the incumbent utility baseline, the one with the largest $T_j^{(r)}$ is promoted. If no challenger satisfies the corresponding promotion criterion, the incumbent Evaluator is retained. *Full BRAID* applies both criteria sequentially, allowing an Evaluator to be promoted only after it satisfies the reliability constraint and demonstrates sufficient realized downstream utility. The endpoint comparison includes all five settings, while the trajectory visualization in Appendix E.3 focuses on the four BRAID-related variants and omits the Native Host for visual clarity.

A.4 PARAMETER SENSITIVITY SETUP

The parameter-sensitivity study uses the same HGM-H and POLYGLOT configuration as the component ablation. One parameter is varied at a time, with $\Delta n = n_{\max} - n_{\min}$ and default values $n_{\min} = 5$, $\Delta n = 3$, $q_{\min} = 0.70$, and $K = 2$. The default trigger configuration corresponds to $n_{\max} = 8$. Each complete EEC contributes K challenger iterations. When the search-action budget is exhausted during an EEC, fewer than K challenger iterations may be completed. The corresponding results are reported in Appendix B.1.

B ADDITIONAL EXPERIMENTAL RESULTS

B.1 PARAMETER SENSITIVITY

Table 8 reports the endpoint accuracies together with the number of challenger iterations, successful Evaluator promotions, and total token consumption for each configuration. The experimental protocol and default settings are described in Appendix A.4.

As $n_{\min}$ increases, eligibility for active review is delayed, accompanied by fewer challenger iterations and lower endpoint accuracy. Successful promotions, however, do not vary monotonically with $n_{\min}$, indicating that promotion frequency alone does not determine the quality of the final Task Agent. No consistent trend emerges from the Δn sweep across the two evaluation splits. While $\Delta n = 0$ yields the highest TestSet accuracy, the default $\Delta n = 3$ achieves the highest WorkSet accuracy. For $q_{\min}$, the intermediate value of 0.70 performs best among the tested settings. A larger K allows more challengers to be considered within each EEC, at the cost of additional evaluation-side computation. At $K = 4$, token consumption rises to 164.53M from 142.81M at $K = 2$, while TestSet accuracy improves by only 0.74 percentage points and WorkSet accuracy decreases. No single configuration dominates across both endpoint accuracy and computational cost. The default setting provides a practical balance between Evaluator-side exploration and total computation.

Table 8: Detailed parameter-sensitivity results under HGM-H on POLYGLOT. Underlined values denote the default setting. Token consumption is reported in millions.

Parameter	Setting	WorkSet	TestSet	Challenger Iterations	Promotions	Tokens (M)
$n_{\min}$	<u>5</u>	**68.33**	**65.19**	18	2	142.81
	10	66.67	63.70	10	3	140.70
	20	61.67	60.74	7	1	123.13
Δn	0	65.00	**66.67**	19	3	146.47
	<u>3</u>	**68.33**	65.19	18	2	142.81
	10	63.33	62.96	15	3	149.37
$q_{\min}$	0.50	66.67	64.44	14	1	139.55
	<u>0.70</u>	**68.33**	**65.19**	18	2	142.81
	0.90	63.33	62.22	18	1	140.91
K	1	63.33	61.48	11	1	136.00
	<u>2</u>	**68.33**	65.19	18	2	142.81
	4	66.67	**65.93**	20	2	164.53

C ADDITIONAL EVALUATOR ANALYSIS

C.1 RELIABILITY ON GATE RECORDS

Successive Evaluator promotions exhibit increasing robustness to label intervention on the Gate records. Intervened-label accuracy rises from 61.5% to 93.2% for HGM and from 60.5% to 77.5% for HGM-H. Every observed promotion improves this metric in both hosts, whereas normal-subset accuracy is not monotonic. Table 9 summarizes the initial and final lineage stages. Complete promotion sequences and Gate statistics are provided in Appendix C.3.

Table 9: Evaluation reliability before and after Evaluator evolution on POLYGLOT.

	Normal-subset Accuracy		Intervened-label Accuracy		Balanced Reliability	
Host	**Initial**	**Final**	**Initial**	**Final**	**Initial**	**Final**
HGM	72.5	67.7	61.5	**93.2**	67.02	**80.47**
HGM-H	72.2	63.7	60.5	**77.5**	66.32	**70.61**

C.2 RELIABILITY AND REALIZED DOWNSTREAM UTILITY

A remaining question is whether improved evaluation reliability is consistently associated with greater downstream utility. For each surviving challenger that undergoes Challenger Branch Realization, we define the post-hoc balanced reliability change as

$$\Delta R_j^{(r)} = \frac{1}{2}\left[q_n(E_j^{(r)}) - q_n(E^{(r)}) + q_v(E_j^{(r)}) - q_v(E^{(r)})\right],$$

and compute its Spearman rank correlation with the realized branch utility $T_j^{(r)}$. This measure is used only for correlation analysis and is distinct from the weighted Gate score G used by the Interventional Reliability Gate.

Correlations vary markedly across hosts and task domains, as shown in Table 10. On POLYGLOT, all four correlations are non-positive, ranging from -0.5754 to -0.0176. Both become positive on SWE-BENCH VERIFIED, reaching 0.2742 for HGM and 0.6506 for HGM-H. The pattern diverges again on IMO-GRADINGBENCH, with correlations of 0.3469 for HGM and -0.1047 for HGM-H. The changing signs and magnitudes show that evaluation reliability does not consistently track realized branch utility.

Table 10: Spearman rank correlation between post-hoc balanced reliability change and realized branch utility.

Host	POLYGLOT	SWE-BENCH VERIFIED	IMO-GRADINGBENCH
DGM	−0.0367	–	–
DGM-H	−0.3591	–	–
HGM	−0.0176	0.2742	0.3469
HGM-H	−0.5754	0.6506	−0.1047

C.3 EVALUATOR LINEAGE PROGRESSIONS

Table 11 reports the complete Evaluator lineages for HGM and HGM-H on POLYGLOT. For each lineage stage, we report the incumbent-derived Gate weights α_{n} and α_{r} used in the associated EEC, together with the normal-subset accuracy q_{n}, intervened-label accuracy q_{r}, and weighted Gate score G of the reported Evaluator. The Gate weights are fixed before challenger comparison and therefore are not recomputed from the q_{n} and q_{r} values shown in the same table row.

Table 11: Statistics of promoted Evaluators on POLYGLOT.

Host	Evaluator	α_n	α_v	q_n (%)	q_v (%)	G (%)
HGM	ev-0001	0.340426	0.659574	72.53	61.51	65.26
	ev-0003	0.348320	0.651680	73.70	71.12	72.02
	ev-0005	0.355587	0.644413	71.54	82.33	78.49
	ev-0015	0.371131	0.628869	67.73	**93.21**	**83.76**
HGM-H	ev-0001	0.340426	0.659574	72.16	60.48	64.46
	ev-0010	0.347993	0.652007	67.61	68.59	68.25
	ev-0014	0.363911	0.636089	63.69	**77.52**	**72.48**

The complete sequences show that intervened-label accuracy increases at every observed promotion for both hosts, whereas normal-subset accuracy is not monotonic. The reported weighted Gate scores use the incumbent-derived weights associated with each EEC and therefore describe the local selection criterion rather than a fixed longitudinal reliability measure.

C.4 GENERALIZATION TO UNSEEN INTERVENTIONS

The same Evaluator lineages are reevaluated on a disjoint set of intervention records that is never used for Evaluator selection. For each lineage stage, we retain the Gate weights used during the corresponding EEC and recompute only q_{n}, q_{r}, and G on the unseen intervention records. Table 12 provides the complete lineage-level results corresponding to the initial-to-final comparison in Table 5.

Table 12: Evaluator reliability on unseen intervention records under HGM and HGM-H on POLYGLOT. The records are not used for Evaluator selection.

Host	Evaluator	α_n	α_v	q_n (%)	q_v (%)	G (%)
HGM	ev-0001	0.340426	0.659574	77.53	54.51	62.35
	ev-0003	0.348320	0.651680	69.70	77.12	74.54
	ev-0005	0.355587	0.644413	78.54	77.33	77.76
	ev-0015	0.371131	0.628869	62.73	**96.21**	**83.79**
HGM-H	ev-0001	0.340426	0.659574	67.16	67.48	67.37
	ev-0010	0.347993	0.652007	73.61	63.59	67.07
	ev-0014	0.363911	0.636089	57.69	**81.52**	**72.84**

Evaluation on the unseen records broadly preserves the reliability improvement from the initial to the final Evaluator. The progression is not strictly monotonic for HGM-H, but its final promoted

Evaluator improves both intervened-label accuracy and the weighted Gate score relative to the initial Evaluator.

C.5 Incumbent Utility Baseline Audit

For each of the nine EECs in the original BRAID run, we identify one challenger for matched comparison. We use the challenger with the highest weighted Gate score, except at EECs that result in an Evaluator promotion, where we use the promoted challenger. An additional incumbent branch is then realized from the same current Task parent A_p and the same normal task-record batch used for that challenger branch. Its realized branch utility is denoted by $T_{\mathrm{inc}}^{(r)}$. Replacing $\eta_t^{(r)}$ with $T_{\mathrm{inc}}^{(r)}$ as the utility reference while preserving the original Gate eligibility and the remaining promotion procedure, we compute

$$\Delta T_j^{(r)} = T_j^{(r)} - T_{\mathrm{inc}}^{(r)}.$$

Table 13: EEC-level audit of promotion decisions using the incumbent utility baseline and matched incumbent realization under HGM-H on POLYGLOT. pp denotes percentage points.

EEC	Baseline Decision	Matched Decision	Agreement	$T_j^{(r)}$	$T_{\mathrm{inc}}^{(r)}$	$\Delta T_j^{(r)}$ (pp)
audit-0001	Incumbent	Incumbent	Yes	0.0420	0.1409	−9.89
audit-0002	Incumbent	Incumbent	Yes	0.0589	0.2154	−15.65
audit-0003	Incumbent	Incumbent	Yes	0.2363	0.3566	−12.03
audit-0004	Incumbent	ev-0008	No	0.3883	0.2971	+9.11
audit-0005	Incumbent	Incumbent	Yes	0.0847	0.1731	−8.83
audit-0006	Incumbent	Incumbent	Yes	0.1409	0.3162	−17.54
audit-0007	Incumbent	Incumbent	Yes	0.1000	0.0050	+9.50
audit-0008	ev-0015	ev-0015	Yes	0.3778	0.2349	+14.29
audit-0009	ev-0017	ev-0017	Yes	0.4588	0.4132	+4.56

The two utility references produce the same promotion decision in eight of the nine EECs, corresponding to an agreement rate of 88.9%. Importantly, the matched comparison preserves the original Gate eligibility. For example, audit-0007 has a positive $\Delta T_j^{(r)}$, but its audited challenger does not pass the Interventional Reliability Gate and is therefore ineligible for promotion. The single decision disagreement occurs at audit-0004, showing that the tenure-level baseline is not identical to matched incumbent realization, while the high overall agreement supports it as a close approximation. The corresponding end-to-end rerun using matched incumbent realization is summarized in Table 4.

C.6 Task Agent Capability Profiles

HGM-H does not evaluate every intermediate Task Agent on the complete POLYGLOT WorkSet. We therefore restrict the capability analysis to representative Task Agents that complete evaluations on all 60 WorkSet tasks. These agents correspond to candidates that receive deeper evaluation and continued exploration during host evolution. They are ordered by their appearance along the trajectory. *Initial* denotes the starting Task Agent, *Stage 1* and *Stage 2* denote two intermediate representatives encountered later in the search, and *Final* denotes the last fully evaluated representative. Figure 2 decomposes their WorkSet performance by programming language.

As shown in Figure 2, the native and BRAID trajectories develop different language-wise capability profiles. Under the native trajectory, performance remains strongest on C++, Go, and Rust, while Java, JavaScript, and Python stay comparatively weaker at the final stage. BRAID shows a different progression on these initially weaker dimensions. Java increases from 16.7% to 58.3%, JavaScript from 0.0% to 60.0%, and Python from 27.3% to 63.6% across the representative stages. At the same time, the final BRAID Task Agent maintains strong performance on C++ and Go, reaching 66.7% and 70.0%, respectively. Overall, the BRAID trajectory exhibits improvements across a broader range of language-specific capabilities.

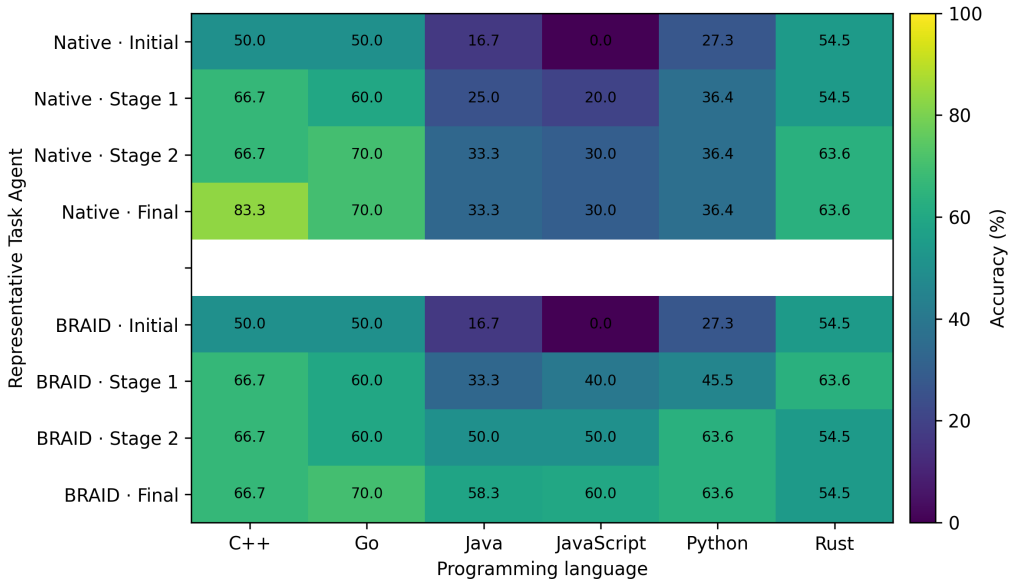


Figure 2: Language-wise WorkSet capability profiles of representative Task Agents during HGM-H evolution on POLYGLOT. Each row corresponds to a Task Agent evaluated on all 60 WorkSet tasks.

D ADDITIONAL COMPUTATIONAL COST ANALYSIS

D.1 TOKEN-EFFICIENCY TRAJECTORIES

Token-efficiency trajectories on POLYGLOT are compared between each native host and its BRAID variant in Figure 3. The clearest improvements in token efficiency appear for HGM and DGM-H, where BRAID reaches shared accuracy levels with substantially fewer tokens. HGM-H shows a smaller advantage at higher accuracy levels. For DGM and SICA, the early trajectories provide little consistent token advantage, while BRAID continues beyond the native runs to discover stronger Task Agents later in evolution.

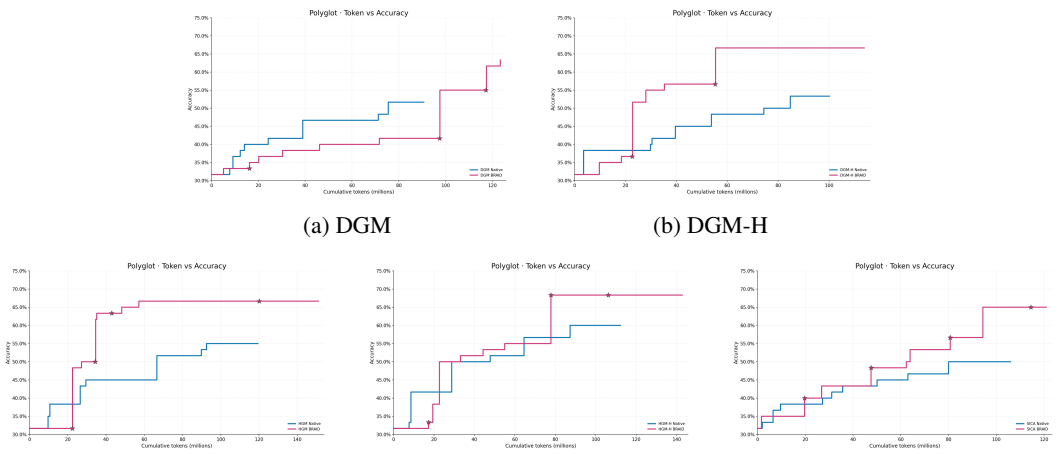
D.2 FULL-RUN TOKEN BREAKDOWN

Full-run token consumption across all evaluated benchmarks is summarized in Table 14. While BRAID incurs additional cost through its Evaluator component, Task and Meta token consumption may increase or decrease with the host search trajectory. Across the nine benchmark and host combinations, Evaluator computation adds between 5.84M and 32.31M tokens. Its effect on total consumption is not purely additive because BRAID also changes the subsequent host search trajectory and therefore the Task and Meta workloads.

D.3 MATCHED TOTAL TOKEN BUDGET COMPARISON

The full-run analysis above shows that BRAID incurs additional Evaluator-side computation. To test whether the improvement under HGM-H can be explained solely by this larger token budget, we extend the native HGM-H run on POLYGLOT beyond the original 800 search actions until its cumulative token consumption matches that of BRAID. The extended native run reaches 1,107 search actions and consumes 152.674M tokens, closely matching the 152.68M tokens used by BRAID within 800 search actions.

Despite receiving 307 additional search actions under the matched token budget, native HGM-H remains 6.66 percentage points below BRAID on the WorkSet and 6.67 points below it on the TestSet. Only one additional WorkSet improvement occurs after the original 800-action boundary, at action 994. For HGM-H on POLYGLOT, this control indicates that BRAID's endpoint gain is not explained solely by greater total token consumption.



(a) DGM (b) DGM-H

(c) HGM (d) HGM-H (e) SICA

Figure 3: Best Task Agent accuracy against cumulative token consumption on POLYGLOT. Each panel compares a native host with its BRAID counterpart.

Table 14: Full-run token consumption across hosts and benchmarks. Token counts are reported in millions and decomposed into Task Agent, Task MetaAgent, and Evaluator computation.

		Total		Task		Meta		Evaluator	
Benchmark	**Host**	**Native**	**BRAID**	**Native**	**BRAID**	**Native**	**BRAID**	**Native**	**BRAID**
Polyglot	DGM	108.88	122.93	86.51	88.46	22.37	20.68	0.00	13.79
	DGM-H	109.04	120.30	84.47	88.96	24.57	15.58	0.00	15.76
	HGM	122.25	153.62	100.91	97.31	21.35	25.89	0.00	30.43
	HGM-H	111.73	152.68	89.46	91.00	22.27	29.37	0.00	32.31
	SICA	105.98	120.95	83.52	85.50	22.46	19.99	0.00	15.46
SWE-bench Verified	HGM	145.86	164.54	143.60	155.71	2.26	2.99	0.00	5.84
	HGM-H	144.97	163.17	141.80	153.95	3.16	2.87	0.00	6.35
IMO-GradingBench	HGM	7.78	14.63	7.60	8.16	0.18	0.20	0.00	6.27
	HGM-H	7.31	14.92	7.08	8.13	0.23	0.27	0.00	6.52

E VISUALIZATION RESULTS

E.1 EVOLUTION TRAJECTORIES

The aggregate evolution trajectories on POLYGLOT are presented in Figure 4 for all five hosts. Corresponding cross-domain trajectories on SWE-BENCH VERIFIED and IMO-GRADINGBENCH are reported in Figure 5.

Table 15: HGM-H performance on POLYGLOT under a matched total token budget.

Variant	Actions	Tokens (M)	WorkSet	TestSet
Native HGM-H	1,107	152.674	61.67	58.52
HGM-H + BRAID	800	152.680	**68.33**	**65.19**

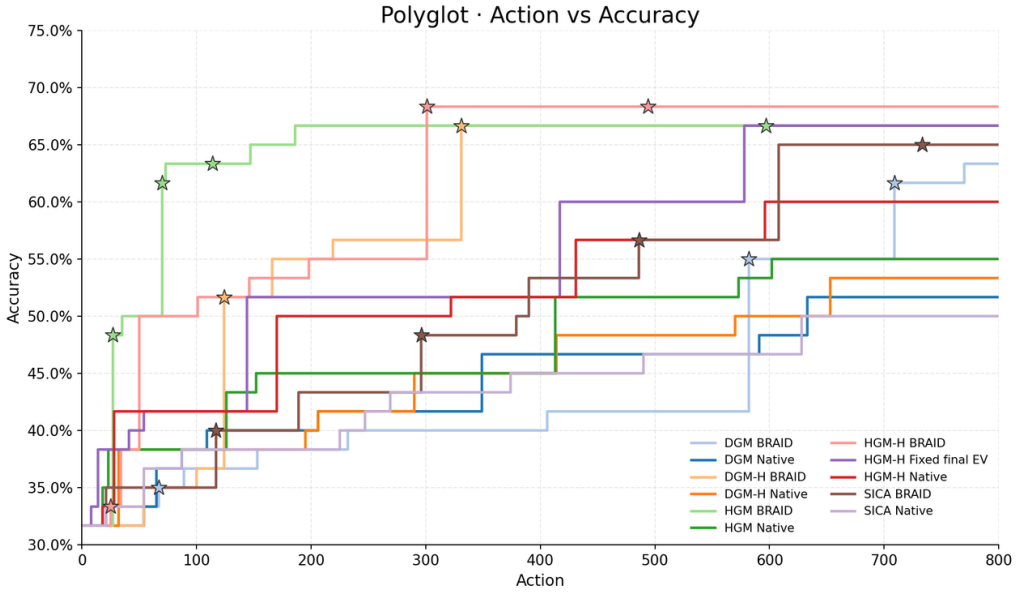
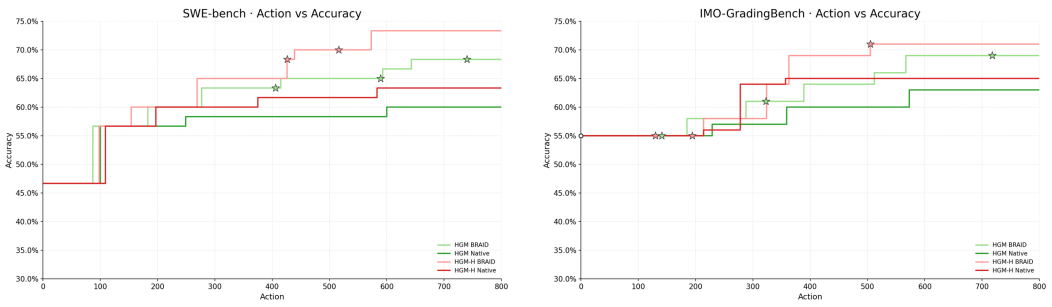


Figure 4: Best Task Agent accuracy throughout evolution on POLYGLOT under the matched 800 search-action budget. Each native host is compared with its BRAID counterpart. Star markers denote Evaluator promotion events.



(a) SWE-BENCH VERIFIED (b) IMO-GRADINGBENCH

Figure 5: Task Agent evolution on SWE-BENCH VERIFIED and IMO-GRADINGBENCH under the matched 800 search-action budget. Each panel compares native HGM and HGM-H with their BRAID counterparts. Star markers denote Evaluator promotion events.

E.2 Evolutionary Search Structures

E.2.1 Search Structures on Polyglot

Figures 6 to 10 visualize the evolutionary search structures produced by the five hosts on Polyglot. For each host, we compare the native and BRAID Task Agent archives and additionally visualize the Evaluator evolution tree introduced by BRAID.

In the Task Agent archives, node color denotes Task Agent accuracy, while node borders indicate the amount of task evaluation received by each agent. The highlighted lineage traces the ancestry of the final best Task Agent, which is marked by a star. In the Evaluator trees, nodes distinguish the initial Evaluator, challengers rejected by the Interventional Reliability Gate, challengers that pass the Gate but are not selected, and promoted Evaluators. The star denotes the final active Evaluator. Node color indicates balanced Evaluator reliability.

Across hosts, the native and BRAID archives remain visibly host dependent rather than converging to a common evolutionary pattern. Their highlighted final-agent lineages also follow distinct paths through the corresponding search trees.

The Evaluator trees also exhibit substantial variation across hosts. DGM explores several Gate-qualified alternatives across multiple Evaluator generations, while DGM-H follows a comparatively sparse promoted lineage. HGM exhibits a deeper sequence of promoted Evaluators, whereas HGM-H performs broader challenger evaluation around the initial Evaluator before following a shorter promoted path. Across these structures, many challengers either fail the reliability Gate or pass the Gate without being promoted. This distinction visually reflects BRAID's two-stage selection mechanism, where reliability determines eligibility but Evaluator replacement additionally depends on realized downstream utility.

E.2.2 Cross-Domain Search Structures

We further visualize the realized Task Agent search structures on SWE-bench Verified and IMO-GradingBench. Figures 11 and 12 compare the native and BRAID runs of HGM and HGM-H. Node fill color indicates Task Agent accuracy, while the node outline denotes the number of tasks used when the corresponding candidate is evaluated. The highlighted path traces the ancestry of the final best Task Agent, which is marked by a star.

Across both SWE-bench Verified and IMO-GradingBench, the native and BRAID runs produce visibly different archive structures and final evolutionary lineages. For both HGM and HGM-H, the ancestry path leading to the final best Task Agent often follows a different branch under BRAID, and the selected endpoint can occupy a substantially different region of the archive. This pattern is particularly clear for HGM-H, where the native and BRAID runs reach their final best agents through distinct multi-step lineages on both benchmarks. At the same time, all runs retain the characteristic tree-structured search of their respective hosts. These visualizations are therefore consistent with evaluation-side evolution altering which candidate branches receive subsequent evolutionary opportunities, while leaving the host search mechanism unchanged. The corresponding performance differences are quantified in Table 1.

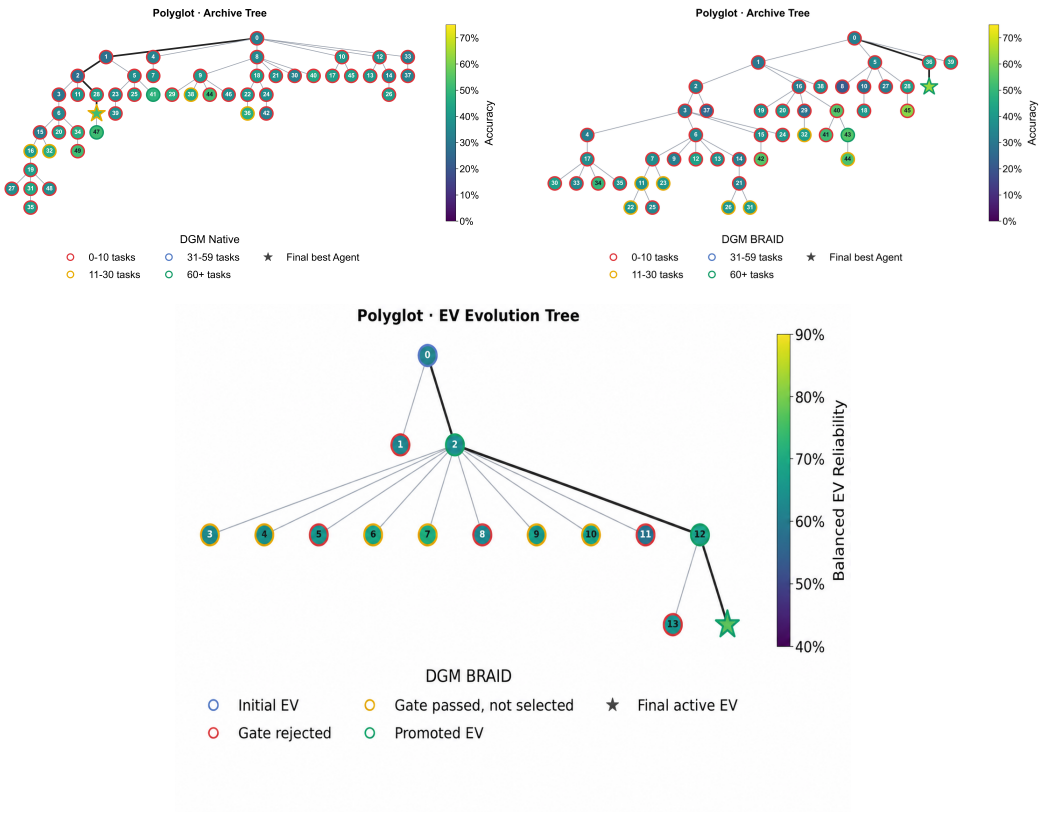


Figure 6: Evolutionary search structures of DGM on POLYGLOT. The native and BRAID Task Agent archives are shown together with the Evaluator evolution tree maintained by BRAID.

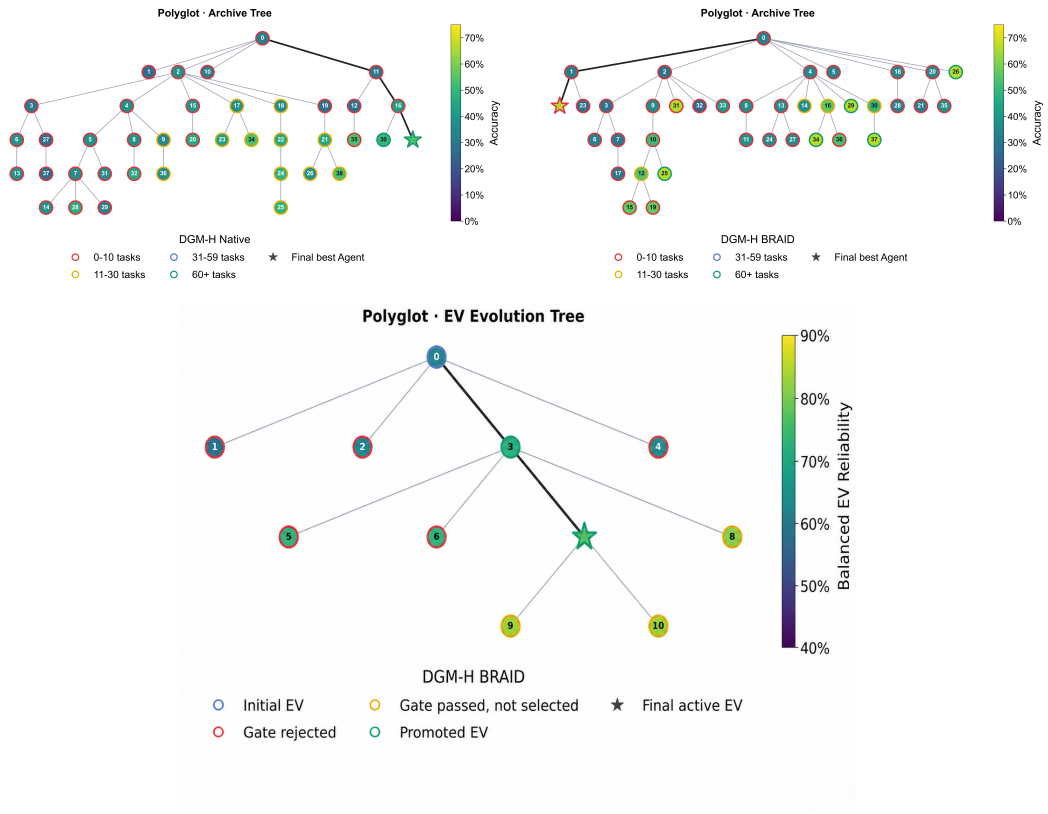


Figure 7: Evolutionary search structures of DGM-H on POLYGLOT. The native and BRAID Task Agent archives are shown together with the Evaluator evolution tree maintained by BRAID.

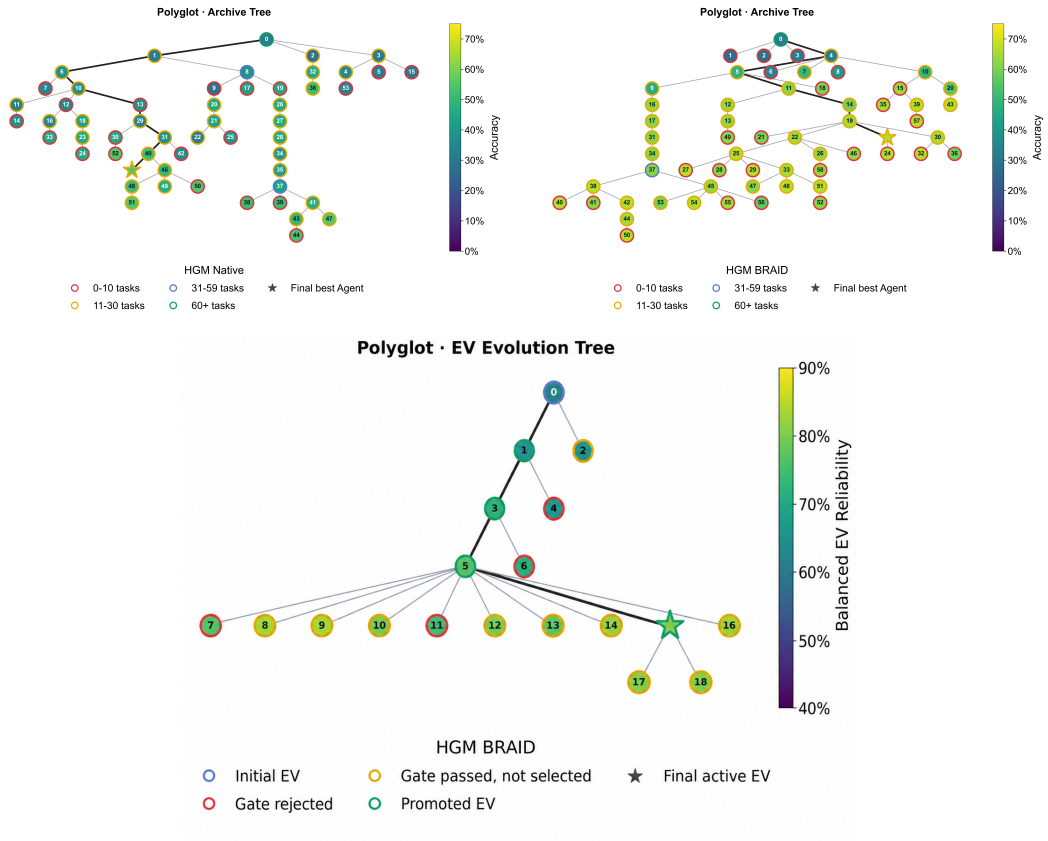


Figure 8: Evolutionary search structures of HGM on POLYGLOT. The native and BRAID Task Agent archives are shown together with the Evaluator evolution tree maintained by BRAID.

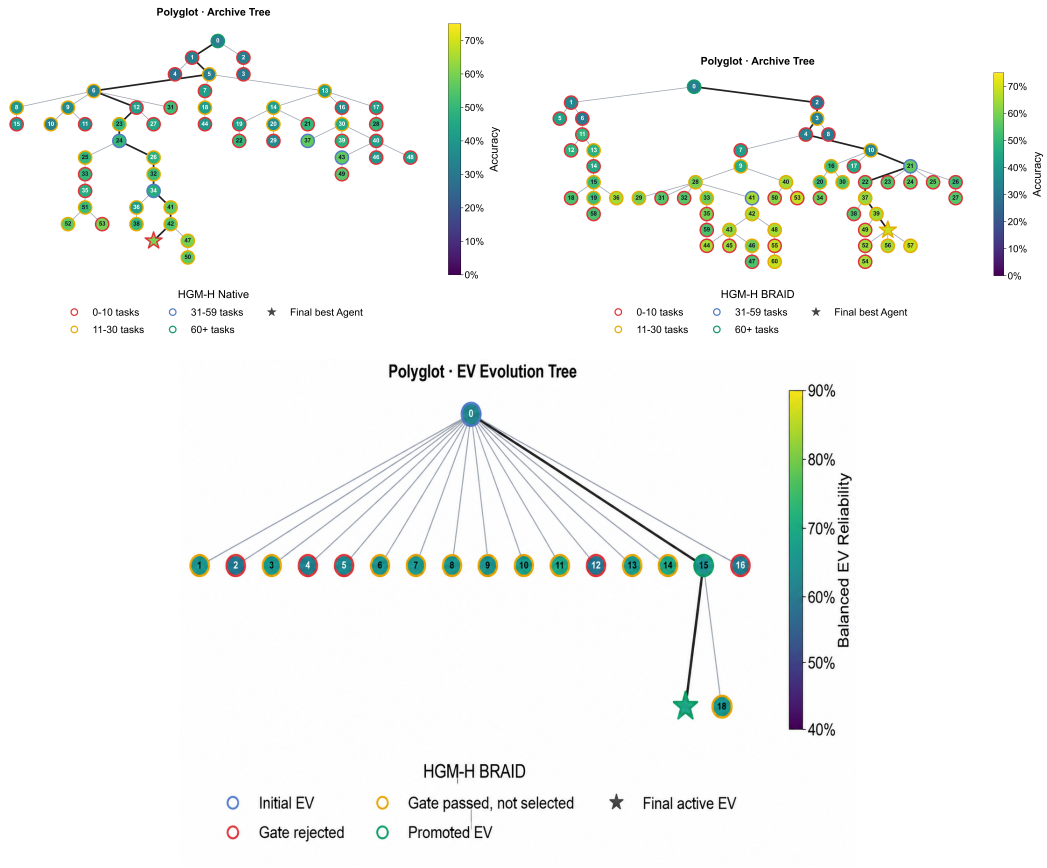


Figure 9: Evolutionary search structures of HGM-H on POLYGLOT. The native and BRAID Task Agent archives are shown together with the Evaluator evolution tree maintained by BRAID.

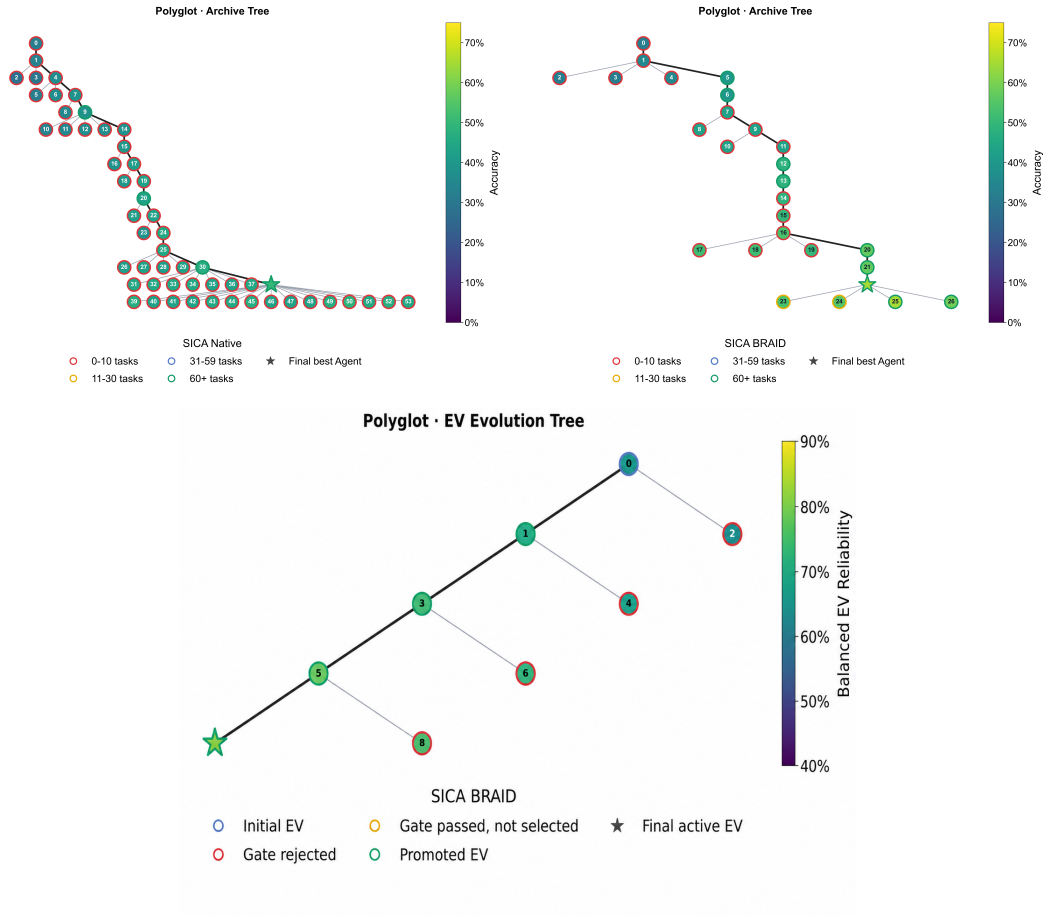
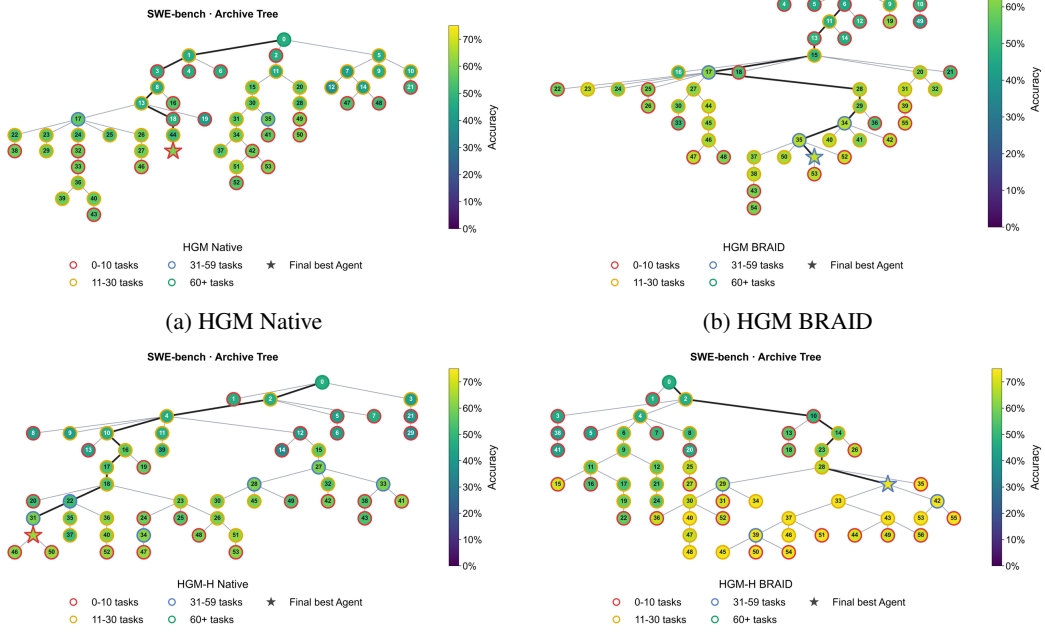


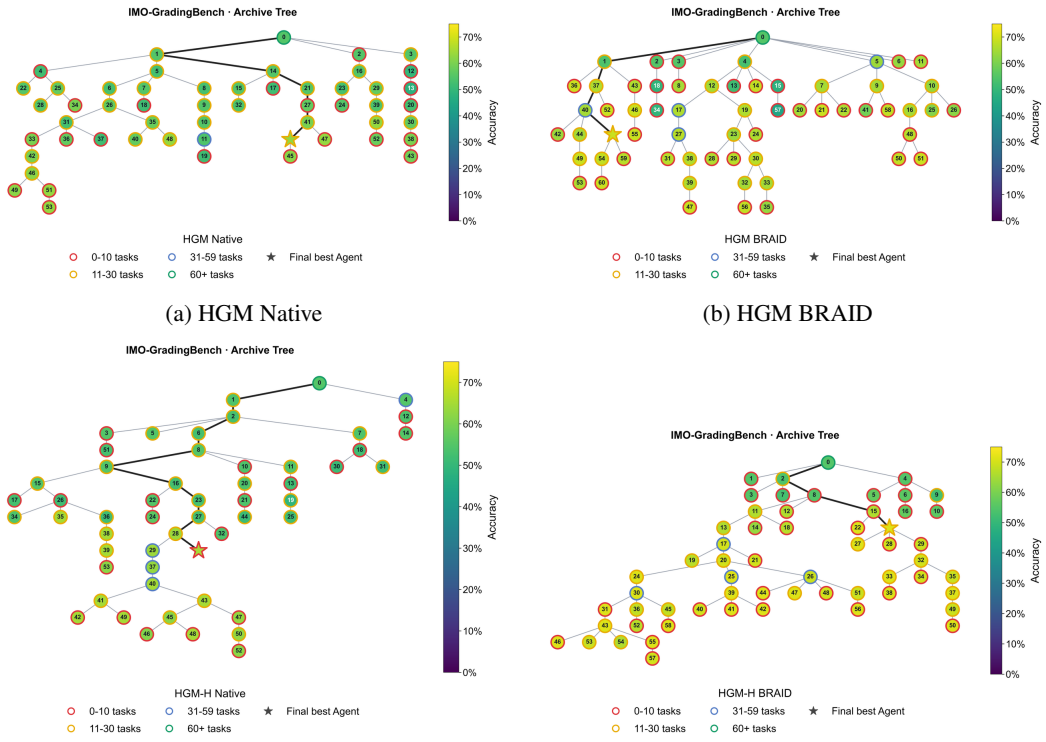
Figure 10: Evolutionary search structures of SICA on POLYGLOT. The native and BRAID Task Agent archives are shown together with the Evaluator evolution tree maintained by BRAID.



(a) HGM Native (b) HGM BRAID

(c) HGM-H Native (d) HGM-H BRAID

Figure 11: Evolutionary search structures on SWE-BENCH VERIFIED. Native and BRAID runs are shown for HGM and HGM-H. Node fill color represents Task Agent accuracy, and node outlines indicate the number of tasks used for evaluation. Dark paths trace the ancestry of the final best Task Agent.



(a) HGM Native

(b) HGM BRAID

(c) HGM-H Native

(d) HGM-H BRAID

Figure 12: Evolutionary search structures on IMO-GRADINGBENCH. Native and BRAID runs are shown for HGM and HGM-H. Node fill color represents Task Agent accuracy, and node outlines indicate the number of tasks used for evaluation. Dark paths trace the ancestry of the final best Task Agent.

E.3 Component Ablation Trajectories

WorkSet evolution trajectories for Frozen EV, Reliability-only, Utility-only, and Full BRAID are visualized in Figure 13, using the component ablation settings reported in Table 2. For clarity, the Native Host is excluded from the trajectory plot but retained in the endpoint comparison. All variants share the same initial Task Agent and matched search-action budget. Detailed definitions are provided in Appendix A.3.

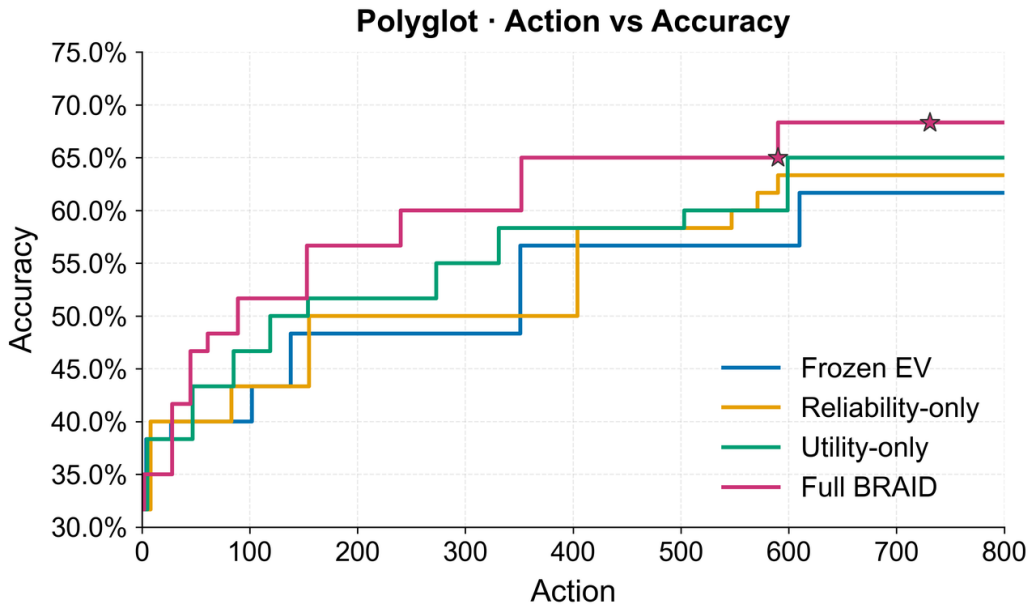


Figure 13: Best Task Agent WorkSet accuracy against cumulative search actions for the four BRAID-related component-ablation settings under HGM-H on Polyglot. Frozen EV denotes Frozen Evaluator. For Full BRAID, star markers denote Evaluator promotion events.

The ablation variants follow distinct search trajectories. Frozen EV, Reliability-only, and Utility-only improve at different stages but all converge below Full BRAID on the WorkSet. Full BRAID reaches 65.0% accuracy earlier and attains its peak WorkSet accuracy of 68.3% at roughly 590 search actions. Its two Evaluator promotions relate differently to best-agent performance. The first occurs near the transition to the final 68.3% level, while the second is not followed by a further gain. This pattern is consistent with Evaluator promotion shaping subsequent Task Agent evolution rather than directly replacing the current best Task Agent.