BEYOND THE LATEST: HISTORICAL CAPABILITY FRAGMENTATION AND RECOVERY IN SELF-EVOLVING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-evolving language model agents improve through iterative data generation, feedback, and model updates, yet capabilities do not necessarily accumulate across iterations. Later iterations may lose previously acquired capabilities, while historical iterations retain complementary ones that no single iteration fully covers, a phenomenon we call *historical capability fragmentation*. Meanwhile, identifying what to preserve from model updates is unreliable: repaired and damaged instances exhibit highly overlapping responses across model parameter rows, making model-parameter response strength insufficient to distinguish beneficial from harmful update effects. We introduce *CapDyn-Match*, a lightweight response-level recovery framework that selects among frozen historical responses without modifying model parameters. It combines *(1) historical candidate scoring*, using query–response representations and cross-iteration agreement to identify useful responses, and *(2) conservative selection*, using a margin gate with a training-derived fallback to avoid harmful switches under insufficient evidence. Across distinct self-evolution frameworks and 14 math, general reasoning, and code benchmarks, *CapDyn-Match* improves question-micro accuracy over the latest iteration by 3.13 and 3.05 points on two three-iteration Qwen3-4B trajectories. On a six-iteration Qwen3-1.7B trajectory, it improves all 14 benchmarks by up to 7.40 points. Leave-one-dataset-out evaluation further shows transfer to unseen benchmarks. These results reveal a broader limitation of self-evolution: acquiring new capabilities does not guarantee preserving existing ones, leaving useful capabilities distributed across model history rather than consolidated in the latest iteration. **Code:** `CapDyn-Match repository`

1 INTRODUCTION

Self-evolving LLM agents improve by generating their own training tasks or data, obtaining feedback on their responses, and updating their models accordingly. STaR, SPIN, and Self-Rewarding Language Models use model-generated data or feedback to drive self-improvement (Zelikman et al., 2022; Chen et al., 2024; Yuan et al., 2024). More recently, Absolute Zero, R-Zero, and Agent0 combine autonomous task generation with reinforcement learning (Zhao et al., 2025; Huang et al., 2025; Xia et al., 2025). These methods continually update models using new data and feedback, but whether the resulting models retain earlier capabilities remains an important question. Catastrophic forgetting in continual learning shows that learning new tasks can impair previously acquired capabilities (Kirkpatrick et al., 2017). In settings that rely on model-generated data, studies of recursive training further show that repeated training can, under certain conditions, reduce generation quality or diversity (Shumailov et al., 2023; Alemohammad et al., 2023). We identify a first challenge in self-evolution: **the latest iteration is not always the best.** As shown in Figure 1, later updates can reduce performance, losing capabilities retained by earlier iterations. A natural response to such capability loss is to identify the internal components responsible for model behavior. Prior work has localized knowledge and features through internal activations and causal interventions (Dai et al., 2022; Meng et al., 2022; Gurnee et al., 2023). We therefore examine whether parameter responses can distinguish beneficial from harmful effects of self-evolution. Our analysis reveals a second challenge: **parameter-row response strength cannot distinguish changes to preserve from changes**

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.

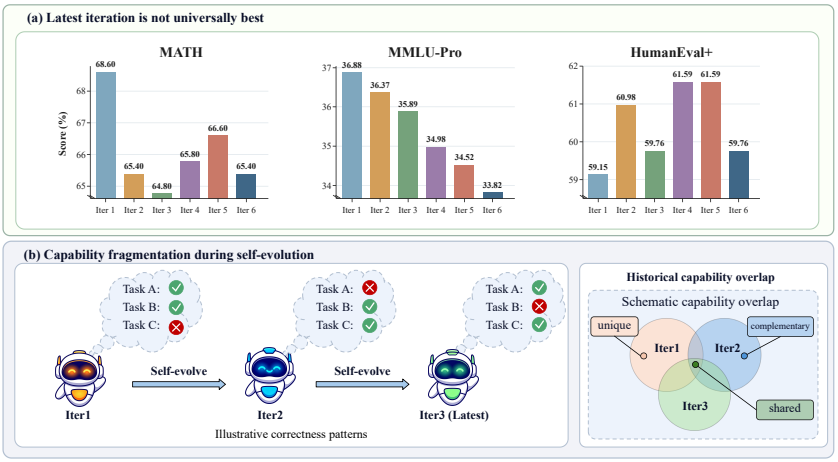

Figure 1: **Historical capability fragmentation.** (a) Qwen3-1.7B performance across six R-Zero iterations shows that the latest iteration is not always the best. (b) Correctness patterns illustrate capability loss, recovery, and complementarity across iterations.

to correct. Highly responsive parameter rows overlap substantially between repaired and damaged instances, making parameter responses unreliable for targeted capability recovery.

Across self-evolution trajectories, we find that aggregate performance captures only part of how capabilities change across iterations. Our instance-level analysis reveals two patterns. First, gains and damages occur within the same update and partially cancel, so small changes in aggregate accuracy can conceal substantial turnover in solved instances. Second, historical advantages are instance-dependent: different iterations solve complementary problem sets, and no single iteration covers all historically successful instances. We call this pattern *historical capability fragmentation*.

In this work, we introduce *CapDyn-Match*, a lightweight response-level capability recovery framework that selects among frozen historical iterations. **To address historical capability fragmentation,** *CapDyn-Match* treats historical responses as a capability pool and combines query–response representations with cross-iteration agreement for instance-level selection. **To address ambiguous parameter responses,** it shifts recovery from parameter space to realized historical behaviors. To avoid harmful switches, *CapDyn-Match* uses a margin gate and falls back to a training-derived iteration prior when selection evidence is insufficient.

We evaluate *CapDyn-Match* on Agent0 (Xia et al., 2025) and R-Zero (Huang et al., 2025) across 14 math, general reasoning, and code benchmarks. On the six-iteration Qwen3-1.7B (Qwen Team, 2025a) trajectory, it improves all benchmarks over the latest iteration, by up to 7.40 percentage points. On two three-iteration Qwen3-4B (Qwen Team, 2025b) trajectories, question-micro gains are 3.13 and 3.05 points. Leave-one-dataset-out evaluation yields benchmark-macro gains of 2.04 and 1.63 points over the training-derived prior.

Our contributions are:

- **Challenges in self-evolution.** Later iterations can lose earlier solutions, while highly responsive parameter rows overlap between repaired and damaged instances, making response magnitude insufficient to identify changes to correct.
- **Capability dynamics and recovery.** Small accuracy changes conceal substantial turnover, and historical iterations retain instance-dependent, complementary strengths. We introduce *CapDyn-Match*, a lightweight method that recovers these capabilities through per-question selection without updating historical models.
- **Evaluation across self-evolution settings.** Across multiple frameworks, model sizes, and iteration lengths on 14 math, general reasoning, and code benchmarks, *CapDyn-Match* improves question-micro accuracy over the latest iteration by 3.13 and 3.05 points on two three-iteration trajectories. On a six-iteration trajectory, it improves all 14 benchmarks, with a maximum per-benchmark gain of 7.40 points.

Table 1: **The latest iteration is not always the best.** Accuracy (%) across iterations. Historical Union gives complementary coverage; the best individual iteration is bolded. HCG is the gap from Iter3 (pp), computed before rounding and thus may differ by 0.01 from the displayed difference.

		Agent0					R-Zero				
Category	Evaluation pool	Iter1	Iter2	Iter3	Hist. Union	HCG	Iter1	Iter2	Iter3	Hist. Union	HCG
Math	GSM8K	92.04	**92.65**	91.21	94.69	3.49	**91.96**	**91.96**	91.89	94.39	2.50
Math	MATH500	**78.80**	78.40	77.20	84.60	7.40	**78.40**	77.00	78.00	85.40	7.40
Math	Minerva	**59.19**	**59.19**	**59.19**	69.85	10.66	**60.66**	58.46	57.35	68.75	11.40
Math	Olympiad	47.41	**47.56**	44.59	58.07	13.48	44.44	**46.81**	42.52	55.56	13.04
Math	AIME24	11.35	**12.60**	12.40	21.35	8.96	12.40	13.54	**18.23**	21.98	3.75
Math	AIME25	**9.27**	6.88	6.35	17.40	11.04	**7.71**	4.90	2.71	13.85	11.15
Math	AMC23	**49.92**	49.84	48.83	66.33	17.50	54.30	**54.45**	53.20	67.42	14.22
General	BBEH	9.71	11.19	**13.56**	20.64	7.08	**10.53**	10.38	10.22	16.88	6.66
General	MMLU-Pro	50.25	**50.59**	50.22	59.69	9.47	**54.80**	54.16	53.28	64.60	11.32
General	SuperGPQA	26.60	**27.46**	26.99	36.33	9.33	**28.08**	27.80	27.66	39.36	11.70
Code	HumanEval+	78.66	77.44	**81.10**	86.59	5.49	**76.22**	73.78	**76.22**	78.05	1.83
Code	MBPP+	72.22	**74.34**	71.96	78.04	6.08	66.40	66.67	**67.20**	69.05	1.85
Code	BigCodeBench-Hard	16.89	**18.24**	16.22	22.97	6.76	16.22	**16.89**	15.54	18.92	3.38
Code	LiveCodeBench	27.05	26.82	**27.27**	32.05	4.77	22.11	22.70	**29.16**	31.31	2.15

2 Motivation and Analysis

We track capability changes across Agent0 (Xia et al., 2025) and R-Zero (Huang et al., 2025) self-evolution trajectories to examine whether new capabilities come at the expense of existing ones and whether historical iterations retain complementary capabilities. We reproduce their data generation, feedback-based evaluation and filtering, and model updates (Appendix F.1). Across 14 benchmarks in mathematical reasoning, general knowledge and reasoning, and code generation, we analyze aggregate performance and instance-level correctness transitions to assess capability acquisition, retention, loss, and historical recoverability. Results appear in Table 1.

2.1 Self-Evolution Does Not Guarantee Monotonic Improvement

Challenge 1: The latest iteration is not always the best.

Performance does not consistently improve during self-evolution; later updates can cause degradation.

Table 1 shows that performance does not consistently improve during self-evolution, and the latest iteration is not always the best. For Agent0, accuracy on MATH500 decreases from 78.80% at Iter1 to 77.20% at Iter3, while accuracy on BBEH increases from 9.71% to 13.56%. R-Zero shows a similar pattern: accuracy improves across iterations on AIME24 but declines on Minerva. These results indicate that later updates do not necessarily preserve earlier performance advantages.

2.2 Small Accuracy Changes Conceal Substantial Capability Turnover

Finding 1: Small Net Changes Conceal Substantial Capability Turnover.

Gains and damages partially cancel. Similar accuracy does not imply solving the same instances.

To characterize capability changes across self-evolution, we track instance-level correctness transitions between adjacent iterations, measuring both gains and damages beyond aggregate accuracy.

Let $\mathcal{C}_t$ denote the set of instances correctly solved by iteration t on a fixed evaluation set of N instances. For two consecutive iterations, we define

$$\text{Gain}_t = 100 \times \frac{|\mathcal{C}_t \setminus \mathcal{C}_{t-1}|}{N}, \qquad \text{Damage}_t = 100 \times \frac{|\mathcal{C}_{t-1} \setminus \mathcal{C}_t|}{N}. \tag{1}$$

Gain and Damage are the percentages of instances changing from incorrect to correct and from correct to incorrect, respectively, both measured over the full evaluation set. With $\text{Acc}_t = 100|\mathcal{C}_t|/N$, net accuracy change and capability turnover are

$$\Delta\text{Acc}_t = \text{Acc}_t - \text{Acc}_{t-1} = \text{Gain}_t - \text{Damage}_t, \qquad \text{Turnover}_t = \text{Gain}_t + \text{Damage}_t. \tag{2}$$

Net change is in points; turnover is the percentage of instances changing correctness.

Figure 2 shows that gains and damages occur within the same update and partially cancel in aggregate accuracy. For example, from Agent0 Iter2 to Iter3, 4.70% of instances change from incorrect

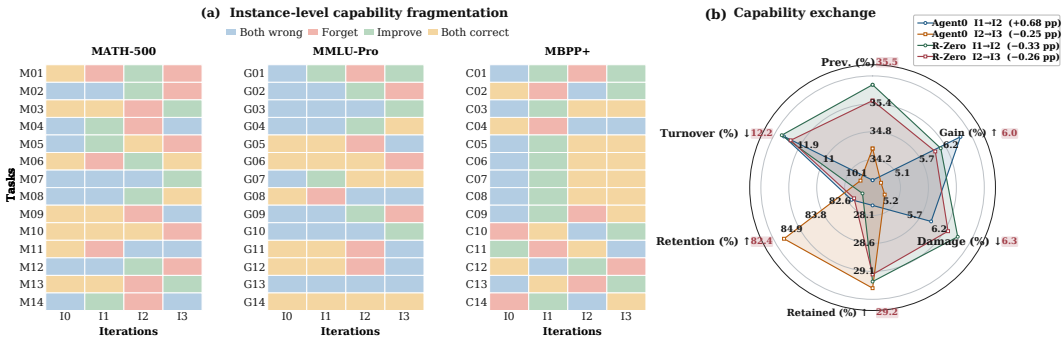

Figure 2: **Small net accuracy changes can conceal substantial capability turnover.** (a) Instance-level transitions on MATH-500, MMLU-Pro, and MBPP+ show that solved instances can be gained and lost across iterations. (b) Capability exchange across adjacent iterations of Agent0 and R-Zero shows that gains and damages can cancel in aggregate accuracy despite substantial turnover.

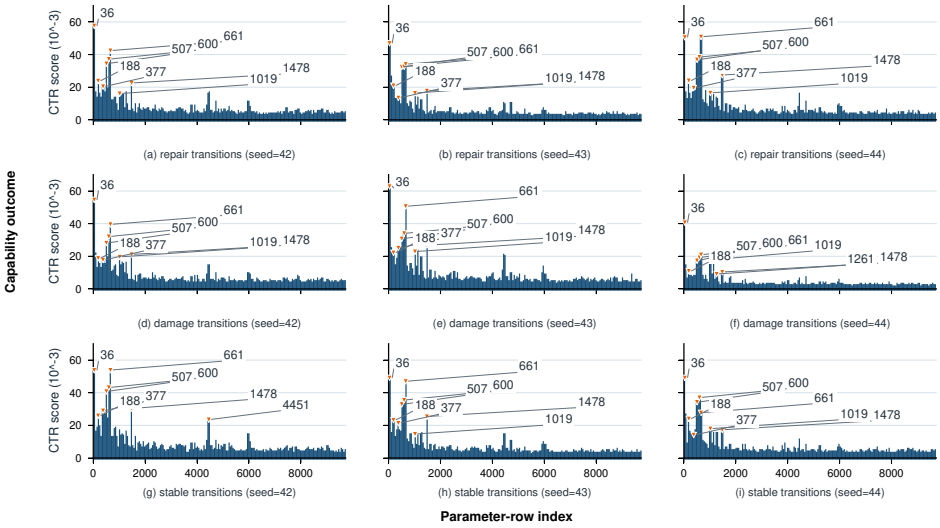

Figure 3: **Repaired and damaged instances share highly responsive parameter rows in Agent0.** Parameter rows with strong responses substantially overlap between repaired and damaged instances, indicating that response magnitude alone provides limited separation between beneficial and harmful changes.

to correct, while 4.95% change from correct to incorrect. The net accuracy change is only -0.25 percentage points, yet correctness changes on 9.65% of instances. Thus, similar aggregate scores can conceal substantially different sets of solvable instances.

2.3 Parameter-Row Response Analysis During Self-Evolution

Challenge 2: Parameter-response-based editing cannot identify harmful changes.

Responsive parameter rows strongly overlap between repaired and damaged instances, making response magnitude unable to distinguish what to preserve or correct.

To assess whether parameter responses can guide targeted edits, we compare instances repaired (incorrect to correct) and damaged (correct to incorrect) between adjacent iterations, alongside stable instances (unchanged).

For input x, let $A_{t,r}^{(k)}(x)$ denote the pre-SiLU output of row r in the layer-24 `gate_proj` module at token t in iteration k. We measure its change by

$$\mathrm{CTR}_r(x) = \frac{1}{|T_x|} \sum_{t \in T_x} \left(A_{t,r}^{(k+1)}(x) - A_{t,r}^{(k)}(x) \right)^2,$$

where T_x contains valid tokens (`attention_mask` $= 1$). Figure 3 reports the row-wise mean CTR over 25 events sampled with replacement per group and seed (42–44).

Repaired and damaged instances exhibit strongly overlapping response spectra. Their Pearson and Spearman correlations are 0.995 and 0.994, and their top-100 row sets have a Jaccard similarity of 0.887 (random reference: 0.005). Peaks at rows 36, 507, 661, and 1478 recur in both groups; some also appear in stable instances. R-Zero results appear in Appendix D.

2.4 Historical Capabilities Are Complementary Across Instances

Finding 2: Historical Advantages Are Instance-Dependent and Complementary.

No single iteration captures the Historical Union, requiring instance-level selection for recovery.

To quantify cross-iteration capabilities, we define Historical Union as instances solved by any iteration, and its gain over the latest iteration as the Historical Coverage Gap (HCG):

$$\mathcal{C}_{\mathrm{hist}} = \bigcup_{t=1}^{T} \mathcal{C}_t, \qquad \mathrm{HCG} = 100 \times \frac{|\mathcal{C}_{\mathrm{hist}} \setminus \mathcal{C}_T|}{N}. \tag{3}$$

Historical Union uses correctness labels to quantify potential recovery headroom. Its computation for benchmarks with repeated sampling is detailed in Appendix F.6.

Table 1 shows that Historical Union outperforms every individual iteration across all 14 benchmarks in both trajectories. On Minerva, all three Agent0 iterations achieve 59.19%, while their union reaches 69.85%, revealing complementary instance-level capabilities. Appendix Table 6 further shows that Agent0 Iter1 and Iter2 solve 6.45% and 4.95% of instances missed by Iter3. When Iter1 and Iter2 agree but differ from Iter3, selecting Iter1 improves accuracy from 18.06% to 26.19% for Agent0 and 19.59% to 26.14% for R-Zero (Table 7), supporting historical agreement as a useful signal for instance-level selection.

3 Method

We introduce *CapDyn-Match*, a lightweight capability recovery framework without modifying historical model parameters. **To address historical capability fragmentation,** *CapDyn-Match* combines query–response representations and cross-iteration agreement for instance-level selection. **To address ambiguous parameter responses,** we shift recovery from parameter space to responses. To avoid harmful switches, *CapDyn-Match* uses a margin gate and falls back to a training-derived prior when evidence is insufficient. Figure 4 provides an overview; the full procedure is in Appendix E.

3.1 Instance-Level Recovery via Response Representations

To recover fragmented historical capabilities without relying on ambiguous parameter responses, we evaluate historical query–response behaviors in a shared representation space.

Let $\mathcal{M} = \{M_k\}_{k=1}^{n}$ denote $n \geq 2$ frozen iterations from the same self-evolution trajectory. For each query q_i, all iterations generate candidate responses in parallel:

$$y_{ik} = M_k(q_i), \tag{4}$$

where y_{ik} contains both the reasoning trace and final output.

We encode each query–response pair using the same frozen reference model:

$$h_{ik} = E_{\mathrm{ref}}\left(q_i \oplus y_{ik}\right)_{\ell,\mathrm{last}}, \tag{5}$$

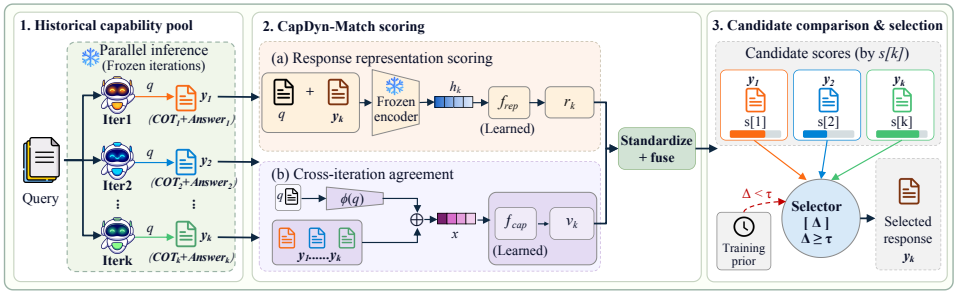

Figure 4: **Overview of *CapDyn-Match*.** Frozen historical iterations generate candidate responses in parallel. *CapDyn-Match* scores them using query–response representations and cross-iteration agreement, then fuses the two signals. A margin gate selects the top candidate or falls back to a training-derived prior when evidence is insufficient.

where $\oplus$ denotes concatenation and last selects the final non-padding token. This shared encoder enables comparison across iterations.

We then apply a training-fitted transform $\mathcal{T}$ for standardization and truncated SVD, and compute

$$r_{ik} = f_{\text{rep}}\left(\mathcal{T}(h_{ik})\right) = \sigma\left(w^\top \mathcal{T}(h_{ik}) + b\right). \tag{6}$$

Here, f_{rep} is a class-balanced logistic-regression probe trained on candidate correctness and shared across iterations, providing a unified score of the realized query–response behavior.

3.2 Cross-Iteration Agreement as Complementary Evidence

To complement candidate-specific evidence, we exploit cross-iteration answer agreement. As shown in Section 2.4, agreement patterns provide an informative signal for instance-level selection.

For the candidate set $\mathcal{Y}_i = \{y_{ik}\}_{k=1}^n$, we compute

$$x_i = \phi(q_i) \oplus g(\mathcal{Y}_i), \qquad v_{ik} = f_{\text{cap},k}(x_i), \tag{7}$$

where $\phi(q_i)$ is an SVD-reduced TF–IDF query representation and $g(\mathcal{Y}_i)$ summarizes normalized final-answer agreement using permutation-invariant statistics, including output diversity, modal-answer share, output entropy, and mean pairwise agreement. Each $f_{\text{cap},k}$ is trained on response correctness, yielding v_{ik} as complementary cross-iteration evidence to r_{ik}.

3.3 Conservative Historical Response Selection

To recover historical capabilities while reducing harmful switches, we fuse representation and agreement evidence with a margin gate. For each candidate set, we normalize the representation scores r_i and agreement scores v_i across iterations and compute

$$s_{ik} = \alpha z(r_i)_k + (1-\alpha) z(v_i)_k, \tag{8}$$

where α balances the two evidence sources.

We define the training-derived prior as the iteration with the highest training accuracy, breaking ties in favor of the latest iteration:

$$k_{\text{prior}} = \max\left(\arg\max_{k\in\{1,\dots,n\}} \text{Acc}_{\text{train}}(M_k)\right). \tag{9}$$

Let $s_{i,(1)}$ and $s_{i,(2)}$ denote the highest and second-highest fused scores. We switch to the unique top candidate only if $s_{i,(1)} - s_{i,(2)} \geq \tau$; otherwise, we fall back to k_{prior}:

$$\hat{k}_i = \begin{cases} k_i^{(1)}, & \text{if the maximum is unique and } s_{i,(1)} - s_{i,(2)} \geq \tau, \\ k_{\text{prior}}, & \text{otherwise.} \end{cases} \tag{10}$$

We tune α and τ through inner validation and return $y_{i\hat{k}_i}$.

Table 2: ***CapDyn-Match* recovers complementary capabilities across self-evolution iterations of Qwen3-4B.** Accuracy (%) is reported for individual iterations and *CapDyn-Match*. Overall rows report question-micro-averaged accuracy within each domain. Highest scores within each trajectory are bolded, including ties.

		AGENT0				R-ZERO			
Category	Evaluation pool	Iter1	Iter2	Iter3	*CapDyn-Match*	Iter1	Iter2	Iter3	*CapDyn-Match*
Math	GSM8K	92.04	92.65	91.21	**93.03**	91.96	91.96	91.89	**92.52**
Math	MATH500	78.80	78.40	77.20	**81.13**	78.40	77.00	78.00	**80.07**
Math	Minerva	**59.19**	**59.19**	**59.19**	58.70	**60.66**	58.46	57.35	60.54
Math	Olympiad	47.41	47.56	44.59	**49.38**	44.44	46.81	42.52	**48.25**
Math	AIME24	11.35	12.60	12.40	**15.52**	12.40	13.54	18.23	**19.55**
Math	AIME25	9.27	6.88	6.35	**12.67**	7.71	4.90	2.71	**8.44**
Math	AMC23	49.92	49.84	48.83	**52.63**	54.30	54.45	53.20	**56.30**
Math	**Overall**	73.80	74.03	72.44	**75.20**	73.19	73.28	72.32	**74.74**
General	BBEH	9.71	11.19	**13.56**	12.43	10.53	10.38	10.22	**11.98**
General	MMLU-Pro	50.25	50.59	50.22	**55.60**	54.80	54.16	53.28	**57.63**
General	SuperGPQA	26.60	27.46	26.99	**29.93**	28.08	27.80	27.66	**30.56**
General	**Overall**	31.44	32.21	32.07	**35.27**	33.70	33.33	32.99	**36.17**
Code	HumanEval+	78.66	77.44	81.10	**82.32**	76.22	73.78	76.22	**76.63**
Code	MBPP+	72.22	74.34	71.96	**76.28**	66.40	66.67	**67.20**	66.58
Code	BigCodeBench-Hard	16.89	**18.24**	16.22	17.79	16.22	16.89	15.54	**17.12**
Code	LiveCodeBench	27.05	26.82	27.27	**28.37**	22.11	22.70	29.16	**29.35**
Code	**Overall**	42.36	42.74	42.61	**44.54**	42.71	42.80	45.88	**46.02**

4 EVALUATION

We evaluate (i) whether *CapDyn-Match* recovers capabilities beyond any single iteration, (ii) whether it outperforms alternative selection strategies and benefits from its individual components, (iii) what inference overhead it introduces, and (iv) whether it generalizes to held-out benchmarks and longer self-evolution trajectories. Details appear in Appendix F.

4.1 EXPERIMENTAL SETUP

Baselines and Ablations. We reproduce Agent0 (Xia et al., 2025) and R-Zero (Huang et al., 2025) under their official settings. Our evaluation includes three-iteration Qwen3-4B trajectories for both Agent0 and R-Zero, and a six-iteration Qwen3-1.7B trajectory for R-Zero. We compare *CapDyn-Match* with individual iterations, Majority Vote (Wang et al., 2023b), Qwen3-32B Judge (Yang et al., 2025), and Query-only Router. We additionally include Model Soup (Wortsman et al., 2022) and TIES-Merging (Yadav et al., 2023) as parameter-merging baselines that combine historical iterations into a single model. Best Iter. and Historical Union serve as single-iteration and oracle references, respectively. We further ablate CapAgree-only, Representation-only, and Vanilla Linear. Implementation and baseline details are provided in Appendices F.1, F.2, and F.3.

Evaluation Benchmarks. We evaluate 14 benchmarks across three categories: (1) mathematical reasoning: GSM8K (Cobbe et al., 2021), MATH500 (Hendrycks et al., 2021b), Minerva (Lewkowycz et al., 2022), OlympiadBench (He et al., 2024), AMC23 (Mathematical Association of America, 2023), AIME24, and AIME25; (2) general reasoning: BBEH (Kazemi et al., 2025), MMLU-Pro (Wang et al., 2024), and SuperGPQA (M-A-P Team et al., 2025); and (3) code generation: HumanEval+, MBPP+ (Liu et al., 2023), BigCodeBench-Hard (Zhuo et al., 2025), and LiveCodeBench (Jain et al., 2025). Benchmark details are provided in Appendix F.4.

4.2 HISTORICAL CAPABILITY RECOVERY

Table 2 shows that *CapDyn-Match* improves over Iter3 across all three domains on both trajectories. Beyond the latest iteration, it also exceeds the best individual iteration by 1.17/1.46 points in mathematical reasoning and 3.06/2.47 points in general reasoning on Agent0/R-Zero, indicating that the gains arise from instance-level recovery rather than reverting to a globally stronger historical iteration. Improvements over Iter3 reach 6.32 points on AIME25 for Agent0 and 5.73 points on AIME25 and OlympiadBench for R-Zero. Code generation shows smaller but positive aggregate gains of 1.93/0.14 points over Iter3.

Table 3: **Comparison and ablation results for *CapDyn-Match*.** Accuracy is question-micro-averaged (%). Best Iter. is the best individual iteration among Iter1–3 of Qwen3-4B, selected separately for Overall and each domain. Δ is the Overall gain over Best Iter. in percentage points. CapAgree-only removes the representation signal; Representation-only removes the capability-agreement signal; Vanilla Linear removes the margin gate.

	Agent0					R-Zero				
Method	Overall	Δ	Math	General	Code	Overall	Δ	Math	General	Code
Best Iter.	35.08	+0.00	74.03	32.21	42.74	36.42	+0.00	73.28	33.70	45.88
Always-Iter3 (Latest)	34.85	−0.23	72.44	32.07	42.61	35.71	−0.71	72.32	32.99	45.88
Majority Vote	35.36	+0.28	74.45	32.50	42.55	36.57	+0.16	73.71	33.90	43.88
Model Soup	35.70	+0.62	73.41	32.92	42.93	35.97	-0.45	72.88	33.29	43.71
TIES	34.50	-0.58	72.43	31.67	42.87	36.03	-0.39	72.58	33.33	45.46
Qwen3-32B Judge	36.59	+1.51	**75.56**	33.65	**46.18**	38.64	+2.22	74.62	36.01	**46.96**
Query-only Router	35.33	+0.25	73.61	32.51	42.95	36.09	−0.32	72.89	33.38	45.55
CapAgree-only	35.53	+0.45	74.54	32.66	43.04	36.53	+0.11	73.25	33.83	45.66
Representation-only	37.63	+2.55	75.13	34.89	44.33	38.67	+2.25	74.55	36.08	45.99
Vanilla Linear	37.13	+2.05	74.36	34.41	43.88	37.70	+1.28	73.93	35.06	45.83
CapDyn-Match	**37.98**	**+2.90**	75.20	**35.27**	44.54	**38.76**	**+2.34**	**74.74**	**36.17**	46.02

Table 4: **Aggregate LODO results for Agent0 and R-Zero with Qwen3-4B.** Benchmark-macro accuracy (%) weights datasets equally. Δ is the gain over Train-Prior (pp), with 95% benchmark-stratified paired-bootstrap CIs; Question-wtd. Δ is the question-weighted gain.

Trajectory	Train-Prior	*CapDyn-Match*	Δ	95% CI	Question-wtd. Δ
Agent0	44.95	**46.99**	**+2.04**	$[+1.36, +2.73]$	+1.28
R-Zero	44.73	**46.36**	**+1.63**	$[+0.83, +2.49]$	+2.07

4.3 Baseline Comparison and Ablation

Comparison with Baselines. Table 3 shows that *CapDyn-Match* achieves 37.98% and 38.76% overall accuracy on Agent0 and R-Zero, outperforming the best individual iteration by 2.90/2.34 points and Majority Vote by 2.62/2.19 points. Compared with parameter-merging baselines, *CapDyn-Match* also outperforms Model Soup by 2.28/2.79 points and TIES by 3.48/2.73 points. It performs comparably to or slightly better than the substantially larger Qwen3-32B Judge, with gains of 1.39/0.12 points. *CapDyn-Match* requires no large judge model during selection, relying instead on lightweight representation-based scoring and routing. These results show that lightweight instance-level selection can effectively exploit complementary capabilities across historical iterations.

Ablation Study. Representation-only already achieves 37.63%/38.67% on Agent0/R-Zero, substantially outperforming the best individual iteration and accounting for most of *CapDyn-Match*'s improvement. This supports query–response representations as the primary signal for recovering fragmented historical capabilities. Within the representation-only selector, removing the margin gate reduces accuracy by 0.50 and 0.97 percentage points on Agent0 and R-Zero, respectively. This comparison supports the benefit of margin-based selection with a training-derived fallback. Together, the results show that agreement and conservative gating provide complementary refinements on top of the strong representation-based selector.

Table 5: **End-to-end inference latency.** Median and mean latency (s/question) over 16 questions, excluding model loading. Candidate generation is parallelized. GPU counts include generation and selection on A100-40GB GPUs. The reference encoder for *CapDyn-Match* is Qwen3-4B.

Method	GPUs	Median (s)	Mean (s)
Iter1	1	4.10	9.49
Iter2	1	3.98	4.74
Iter3	1	4.36	6.77
Query-only Router	1	3.99	4.75
Majority Vote	3	5.29	12.18
CapAgree-only	3	5.29	12.18
Vanilla Linear	4	5.51	12.73
Representation-only	4	5.51	12.73
Qwen3-32B Judge	7	7.19	13.53
CapDyn-Match	4	5.52	12.73

Inference Cost. As shown in Table 5, *CapDyn-Match* requires 5.52/12.73 seconds median/mean end-to-end latency per question using four GPUs, close to Majority Vote and lower than the Qwen3-32B Judge. Thus, the gains from historical capability recovery do not require introducing a large-model judge.

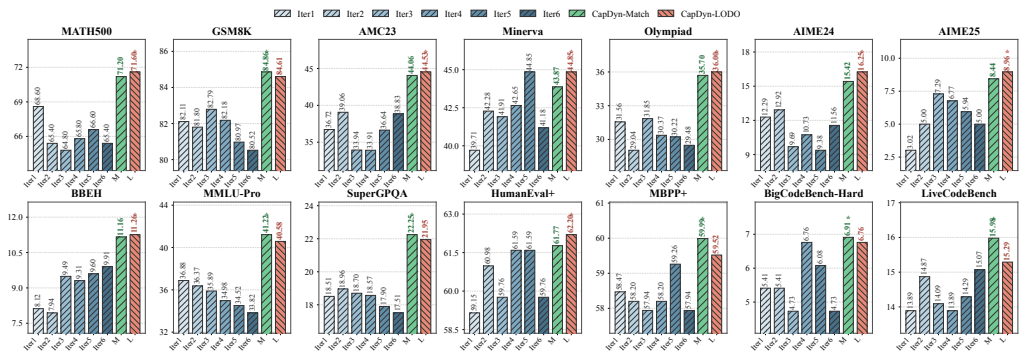

Figure 5: **Performance across six R-Zero self-evolution iterations with a Qwen3-1.7B solver.** Benchmark accuracy (%) is shown for individual iterations (Iter1–Iter6), CapDyn-Match under nested cross-validation (M), and CapDyn-Match under leave-one-dataset-out evaluation (L).

4.4 Generalization Across Datasets and Longer Trajectories

We further evaluate whether *CapDyn-Match* generalizes beyond the in-domain three-iteration setting, considering both unseen datasets and a longer self-evolution trajectory.

Unseen-Dataset Generalization. We conduct LODO evaluation on the Qwen3-4B trajectories, where held-out benchmarks are excluded from selector training and tuning. Compared with Train-Prior, *CapDyn-Match* improves benchmark-macro accuracy by 2.04 and 1.63 points on Agent0 and R-Zero (95% CIs: [1.36, 2.73] / [0.83, 2.49]), demonstrating transfer to unseen benchmarks.

Six-Iteration Trajectory. We further evaluate *CapDyn-Match* on a six-iteration Qwen3-1.7B R-Zero trajectory with a larger historical candidate pool. As shown in Figure 5, *CapDyn-Match* outperforms the latest iteration on all 14 benchmarks under both nested cross-validation and LODO. The gains reach 7.40, 6.22, and 5.80 points on MMLU-Pro, OlympiadBench, and MATH500 under nested cross-validation, and remain 6.76, 6.52, and 6.20 points under LODO.

Importantly, the gains are not merely due to a weak final iteration. Relative to the best individual iteration per benchmark, *CapDyn-Match* improves benchmark-macro accuracy by 1.93 points under nested cross-validation and 2.04 points under LODO. It further exceeds Representation-only by 0.72/0.99 points and Vanilla Linear by 0.50/0.86 points, respectively, and outperforms Representation-only on all 14 held-out benchmarks. These results show that *CapDyn-Match* exploits complementary capabilities along longer trajectories, with score fusion and gated selection providing consistent gains beyond representation alone. Full results are in Appendix F.7.

5 Conclusions and Limitations

We introduce *CapDyn-Match*, a response-level capability recovery framework for self-evolving agents. Motivated by historical capability fragmentation and ambiguous parameter responses, *CapDyn-Match* selects among historical responses using query–response representations, cross-iteration agreement, and margin-gated selection. Across 14 reasoning and code benchmarks, *CapDyn-Match* recovers useful historical capabilities across different self-evolution settings, while longer-trajectory and held-out evaluations further support its generalization.

This study has two limitations. First, we analyze capability dynamics across saved iterations; finer-grained intermediate states may reveal additional transitions. Second, *CapDyn-Match* selects a single historical response per query, leaving cross-response composition unexplored. Future work can extend recovery to finer-grained trajectories and complementary response composition.

6 AI USE STATEMENT

Generative AI tools, including ChatGPT, were used solely for language polishing and grammar checking. All technical content, methodological design, experiments, data analysis, and conclusions were developed and verified by the human authors. The use of generative AI does not affect the originality or authorship of this work.

7 ETHICS STATEMENT

This work studies capability dynamics and historical capability recovery in self-evolving language model agents. All experiments use publicly available models, self-evolution frameworks, and standard mathematical reasoning, general reasoning, and code benchmarks, without involving human subjects or personally identifiable information. Our method operates on frozen historical iterations and does not introduce additional data collection or human profiling. This study is intended to improve the understanding and reliability of self-evolving agents.

8 REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide the following resources:

(1) **Code:** The implementation of *CapDyn-Match* and evaluation pipelines is provided in the anonymous repository.

(2) **Models and Trajectories:** We evaluate Agent0 and R-Zero self-evolution trajectories based on Qwen3-4B and Qwen3-1.7B models, with reproduction details provided in the appendix.

(3) **Evaluation:** We evaluate on 14 publicly available benchmarks covering mathematical reasoning, general reasoning, and code generation. We report both nested cross-validation and leave-one-dataset-out evaluation, with all splitting and evaluation protocols detailed in the appendix.

(4) **Hyperparameters:** Selector training, feature construction, and hyperparameter selection procedures are fully specified in the appendix.

(5) **Compute:** Self-evolution trajectories are reproduced using 8× NVIDIA A100-40GB and 8× NVIDIA H20-96GB GPUs. Candidate generation and selection are conducted on the same GPU platforms, with end-to-end inference latency reported in the main paper.

REFERENCES

Sina Alemohammad, Josue Casco-Rodriguez, Lorenzo Luzi, Ahmed Imtiaz Humayun, Hossein Babaei, Daniel LeJeune, Ali Siahkoohi, and Richard G. Baraniuk. Self-consuming generative models go MAD. *arXiv preprint arXiv:2307.01850*, 2023. URL https://arxiv.org/abs/2307.01850.

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *European Conference on Computer Vision*, pp. 139–154, 2018.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with A-GEM. In *International Conference on Learning Representations*, 2019.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023. URL https://arxiv.org/abs/2305.05176.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models. *arXiv preprint arXiv:2401.01335*, 2024. URL https://arxiv.org/abs/2401.01335.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John

Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8493–8502. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.acl-long.581.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks Lakshmanan, and Ahmed H Awadallah. Hybrid LLM: Cost-efficient and quality-aware query routing. In *International Conference on Learning Representations*, pp. 41348–41366, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/b47d93c99fa22ac0b377578af0a1f63a-Paper-Conference.pdf.

Neel Guha, Mayee F. Chen, Trevor Chow, Ishan S. Khare, and Christopher Ré. Smoothie: Label-free language model routing. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://arxiv.org/abs/2412.04692.

Wes Gurnee, Neel Nanda, Matthew Pauly, Katherine Harvey, Dmitrii Troitskii, and Dimitris Bertsimas. Finding neurons in a haystack: Case studies with sparse probing. *Transactions on Machine Learning Research*, 2023.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021a.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. *NeurIPS Datasets and Benchmarks*, 2021b. arXiv:2103.03874.

Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. RouterBench: A benchmark for multi-LLM routing system. *arXiv preprint arXiv:2403.12031*, 2024. URL https://arxiv.org/abs/2403.12031.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-Zero: Self-evolving reasoning LLM from zero data. *arXiv preprint arXiv:2508.05004*, 2025. URL https://arxiv.org/abs/2508.05004.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *International Conference on Learning Representations*, 2023.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2403.07974.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. LLM-Blender: Ensembling large language models with pairwise ranking and generative fusion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, 2023. URL https://arxiv.org/abs/2306.02561.

Wittawat Jitkrittum et al. Universal model routing for efficient LLM inference. *arXiv preprint arXiv:2502.08773*, 2025. URL https://arxiv.org/abs/2502.08773.

Mehran Kazemi, Bahare Fatemi, Hritik Bansal, John Palowitch, Chrysovalantis Anastasiou, Sanket Vaibhav Mehta, Lalit K. Jain, Virginia Aglietti, Disha Jindal, Peter Chen, et al. BIG-Bench Extra Hard. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 2025.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *International Conference on Learning Representations*, pp. 54928–54958, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/f1cf02ce09757f57c3b93c0db83181e0-Paper-Conference.pdf.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. doi: 10.1073/pnas.1611835114. URL https://arxiv.org/abs/1612.00796.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022.

Zelong Li, Shuyuan Xu, Kai Mei, Wenyue Hua, Balaji Rama, Om Raheja, Hao Wang, He Zhu, and Yongfeng Zhang. AutoFlow: Automated workflow generation for large language model agents. *arXiv preprint arXiv:2407.12821*, 2024. URL https://arxiv.org/abs/2407.12821.

Zhizhong Li and Derek Hoiem. Learning without forgetting. In *European Conference on Computer Vision*, pp. 614–629, 2016.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023. URL https://arxiv.org/abs/2305.20050.

Bo Liu, Leon Guertler, Simon Yu, Zichen Liu, Penghui Qi, Daniel Balcells, Mickel Liu, Cheston Tan, Weiyan Shi, Min Lin, Wee Sun Lee, and Natasha Jaques. SPIRAL: Self-play on zero-sum games incentivizes reasoning via multi-agent multi-turn reinforcement learning. *arXiv preprint arXiv:2506.24119*, 2025. URL https://arxiv.org/abs/2506.24119.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2305.01210.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. Routing to the expert: Efficient reward-guided ensemble of large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1964–1974. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.naacl-long.109. URL https://aclanthology.org/2024.naacl-long.109/.

Yifan Lu, Rixin Liu, Jiayi Yuan, Xingqi Cui, Shenrun Zhang, Hongyi Liu, and Jiarong Xing. RouterArena: An open platform for comprehensive comparison of LLM routers. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.00202.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*, 2023. URL https://arxiv.org/abs/2308.08747.

M-A-P Team, Xinrun Du, Yifan Yao, Kaijing Ma, Bingli Wang, Tianyu Zheng, Kang Zhu, Minghao Liu, Yiming Liang, Xiaolong Jin, Zhenlin Wei, et al. SuperGPQA: Scaling LLM evaluation across 285 graduate disciplines. *arXiv preprint arXiv:2502.14739*, 2025. URL https://arxiv.org/abs/2502.14739v1.

Xueguang Ma, Qian Liu, Dongfu Jiang, Ge Zhang, Zejun Ma, et al. General-reasoner: Advancing LLM reasoning across all domains. *arXiv preprint arXiv:2505.14652*, 2025.

Mathematical Association of America. 2023 american mathematics competitions: Amc 10 and amc 12. https://maa.org/student-programs/amc/, 2023. Accessed: 2026-08-30.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems*, 2022.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs with preference data. *arXiv preprint arXiv:2406.18665*, 2024. URL https://arxiv.org/abs/2406.18665.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744, 2022.

Qwen Team. Qwen3-1.7b-base. Hugging Face model repository, 2025a. URL https://huggingface.co/Qwen/Qwen3-1.7B-Base.

Qwen Team. Qwen3-4b-base. Hugging Face model repository, 2025b. URL https://huggingface.co/Qwen/Qwen3-4B-Base.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. iCaRL: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Thomas Scialom, Tuhin Chakrabarty, and Smaranda Muresan. Fine-tuned language models are continual learners. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 6107–6122. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.emnlp-main.410. URL https://aclanthology.org/2022.emnlp-main.410/.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Yarin Gal, Nicolas Papernot, and Ross Anderson. The curse of recursion: Training on generated data makes models forget. *arXiv preprint arXiv:2305.17493*, 2023. URL https://arxiv.org/abs/2305.17493.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. Challenging BIG-Bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, 2023.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023a. URL https://arxiv.org/abs/2305.16291.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023b.

Yuanyi Wang, Yifan Yang, Su Lu, Yanggan Gu, Pengkai Wang, Wenjun Wang, Zhaoyi Yan, Congkai Xie, Jianmin Wu, Jialun Cao, Shing-Chi Cheung, and Hongxia Yang. Geometry conflict: Explaining and controlling forgetting in LLM continual post-training. *arXiv preprint arXiv:2605.09608*, 2026. URL https://arxiv.org/abs/2605.09608.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*, 2024.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 23965–23998. PMLR, 2022.

Tongtong Wu, Linhao Luo, Yuan-Fang Li, Shirui Pan, Thuy-Trang Vu, and Gholamreza Haffari. Continual learning for large language models: A survey. *arXiv preprint arXiv:2402.01364*, 2024. URL https://arxiv.org/abs/2402.01364.

Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning. *arXiv preprint arXiv:2511.16043*, 2025. URL https://arxiv.org/abs/2511.16043.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. TIES-Merging: Resolving interference when merging models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursive self-improvement. *arXiv preprint arXiv:2410.04444*, 2024. URL https://arxiv.org/abs/2410.04444.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 57905–57923. PMLR, 2024.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024. URL https://arxiv.org/abs/2406.07496.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 15476–15488, 2022.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D. Goodman. Quiet-STaR: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024. URL https://arxiv.org/abs/2403.09629.

Guibin Zhang et al. MemEvolve: Meta-evolution of agent memory systems. *arXiv preprint arXiv:2512.18746*, 2025. URL https://arxiv.org/abs/2512.18746.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. MemSkill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026. URL https://arxiv.org/abs/2602.02474.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024. URL https://arxiv.org/abs/2410.10762.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: Llm agents are experiential learners. *arXiv preprint arXiv:2308.10144*, 2023. URL https://arxiv.org/abs/2308.10144.

Andrew Zhao, Yiran Wu, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute Zero: Reinforced self-play reasoning with zero data. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 105816–105879. Curran Associates, Inc., 2025. doi: 10.52202/085713-3534. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/9837dc00ff67d176373268ed48042d49-Abstract-Conference.html.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, James Hoang, Armel Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, Binyuan Hui, Niklas Muennighoff, David Lo, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, pp. 66602–66656, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a6a90bcc2aa470c3871b2d39a67d26e8-Abstract-Conference.html.

A Related Work

Self-Evolving Language Models. Existing approaches to agent self-evolution differ in what they change: model parameters, the agent harness, or external artifacts. Model-level methods update parameters using self-generated or verified experience, as in STaR (Zelikman et al., 2022), Self-Rewarding Language Models (Yuan et al., 2024), Quiet-STaR (Zelikman et al., 2024), Absolute Zero (Zhao et al., 2025), R-Zero (Huang et al., 2025), SPIRAL (Liu et al., 2025), and Agent0 (Xia et al., 2025). At the harness level, prompts, tools, or workflows can be revised without updating the backbone model; examples include DSPy (Khattab et al., 2024), TextGrad (Yuksekgonul et al., 2024), AutoFlow (Li et al., 2024), AFlow (Zhang et al., 2024), and Gödel Agent (Yin et al., 2024). Artifact-level methods accumulate external memories, rules, or skills, as in Reflexion (Shinn et al., 2023), ExpeL (Zhao et al., 2023), Voyager (Wang et al., 2023a), MemSkill (Zhang et al., 2026), and MemEvolve (Zhang et al., 2025). We study model-level self-evolution and ask whether earlier iterations retain useful capabilities that the latest iteration no longer exhibits.

Continual Learning and Capability Preservation. Sequential updates can acquire new capabilities while weakening previously learned ones. Continual-learning methods address this problem through parameter regularization (Kirkpatrick et al., 2017; Aljundi et al., 2018), knowledge distillation (Li & Hoiem, 2016), experience replay (Rebuffi et al., 2017; Rolnick et al., 2019), and gradient constraints (Lopez-Paz & Ranzato, 2017; Chaudhry et al., 2019). Recent work also studies capability forgetting during continual language-model training (Scialom et al., 2022; Luo et al., 2023; Wu et al., 2024). Post-training methods combine model parameters using model soups (Wortsman et al., 2022), task arithmetic (Ilharco et al., 2023), TIES-Merging (Yadav et al., 2023), or geometry-aware integration (Wang et al., 2026). CapDyn-Match instead addresses capability loss at inference time by selecting among responses generated by frozen historical iterations.

Model Routing and Answer Selection. LLM routing selects among models for each query, often balancing response quality and inference cost. Routing methods include FrugalGPT (Chen et al., 2023), RouteLLM (Ong et al., 2024), Hybrid-LLM (Ding et al., 2024), ZOOTER (Lu et al., 2024), Smoothie (Guha et al., 2024), and Universal Model Routing (Jitkrittum et al., 2025). RouterBench (Hu et al., 2024) and RouterArena (Lu et al., 2026) support the evaluation and comparison of routers. Related post-generation methods rank or combine candidate responses using learned ranking, reward models, self-consistency, or process supervision (Jiang et al., 2023; Ouyang et al., 2022; Wang et al., 2023b; Lightman et al., 2023). CapDyn-Match selects among responses generated by frozen iterations along the same self-evolution trajectory, using query–response representation scores and query-conditioned agreement among final outputs.

B Distinguishing Historical Capability Fragmentation from Related Phenomena

Catastrophic forgetting. Catastrophic forgetting refers to the degradation of previously acquired performance after subsequent training. It is typically assessed by comparing a model's performance on earlier tasks before and after an update. Historical capability fragmentation instead concerns how instance-level successes are distributed across a trajectory of iterations. Let C_t be the set of evaluation instances solved by iteration t. Fragmentation occurs when different iterations solve complementary subsets, such that $|\bigcup_t C_t| > \max_t |C_t|$ by a meaningful margin. An update can therefore cause forgetting on some instances while enabling gains on others. Aggregate accuracy may remain stable or improve even as historical iterations retain useful capabilities absent from the latest iteration. Forgetting can contribute to fragmentation, but a decline in aggregate performance is not required to establish it.

Knowledge conflict. Knowledge conflict arises when a model encounters incompatible claims and must determine which answer applies based on source, time, or context. Historical capability fragmentation does not require contradictory claims or inconsistent ground truth. It can arise even when all evaluation instances have unambiguous answers: different iterations may solve complementary subsets of the same benchmark, while no single iteration achieves their combined coverage. The distinction is therefore between resolving conflicting information within a task and identifying complementary capabilities distributed across model history.

Relationship among the phenomena. These phenomena are not mutually exclusive. Conflicting information may contribute to forgetting, and forgotten instances may form part of the historical coverage gap. Our analysis does not attribute every instance-level change to a specific cause. Rather, it measures the observable consequence across iterations: whether historical models collectively solve instances that no individual iteration covers alone. This distinction motivates selecting responses from historical iterations at inference time.

C Capability Exchange and Historical Complementarity

C.1 Capability Exchange Across Iterations

Table 6 provides a detailed breakdown of capability changes across self-evolution iterations. Let C_t denote the set of evaluation instances correctly solved by iteration t. For a transition from iteration t to $t+1$, Gain and Damage measure the evaluation-pool shares of $C_{t+1} \setminus C_t$ and $C_t \setminus C_{t+1}$, respectively, while Retained measures the share of $C_t \cap C_{t+1}$. Net Change equals Gain minus Damage; Retention is the fraction of previously solved instances that remain correct, and Turnover equals Gain plus Damage.

Panel A shows that small aggregate changes conceal substantial instance-level exchange. Across the reported transitions, net changes range from -0.33 to $+0.68$ percentage points, whereas turnover ranges from 9.65% to 12.58%. Thus, relatively stable aggregate accuracy does not imply stable correctness at the instance level.

Panel B further characterizes the complementary coverage retained across historical iterations. Directional complementarity measures the share of instances solved by one iteration but missed by another. Iteration-exclusive coverage measures the share solved by exactly one of the three iterations. For Agent0 and R-Zero, respectively, Iter1 uniquely solves 4.20% and 4.30% of the evaluation pool, while Iter2 uniquely solves another 2.70% and 3.10%. These disjoint subsets are missed by Iter3, indicating that useful historical coverage is distributed across multiple earlier iterations rather than concentrated in a single one.

C.2 Historical Consensus as an Observable Recovery Signal

The coverage analysis above uses correctness labels to characterize historical complementarity. We next examine whether a signal observable at inference time can help recover useful historical responses. Table 7 evaluates a fixed rule: select the response from Iter1 when the normalized responses

Table 6: **Capability exchange and complementary coverage across self-evolution iterations.** Later iterations acquire new solutions while losing previously solved instances, leaving complementary capability fragments across historical states. All coverage statistics are reported as percentages within each trajectory.

Panel A: Capability exchange between adjacent iterations

Method	Transition	Evaluation-pool share (%)				Net	Normalized dynamics (%)	
		Previous	Gain	Damage	Retained	Change (pp)	Retention	Turnover
Agent0	I1 → I2	33.76	6.59	5.91	27.85	+0.68	82.48	12.50
	I2 → I3	34.44	4.70	4.95	29.48	−0.25	85.62	9.65
R-Zero	I1 → I2	35.81	6.12	6.46	29.35	−0.33	81.97	12.58
	I2 → I3	35.47	5.99	6.26	29.21	−0.26	82.36	12.25

Panel B: Complementary coverage across historical iterations

Method	Directional complementarity (%)				Iteration-exclusive coverage (%)		
	I1 \ I3	I2 \ I3	I3 \ I1	I3 \ I2	Only I1	Only I2	Only I3
Agent0	6.45	4.95	6.88	4.70	4.20	2.70	2.99
R-Zero	7.46	6.26	6.86	5.99	4.30	3.10	3.84

Table 7: **Historical consensus provides a test-time-visible signal for recovering useful historical responses.** Across Agent0 and R-Zero, the fixed rule selects Iter1 when the normalized responses satisfy $a_1 = a_2 \neq a_3$ and otherwise retains Iter3. The rule uses no test labels for individual decisions and serves as a diagnostic rather than the learned method.

Method	Evaluation slice	Pool share (%)	I3 Acc. (%)	Rule Acc. (%)	B/H (%)
Agent0	$a_1 = a_2 \neq a_3$	8.51	18.06	**26.19**	25.24/17.11
	Full evaluation pool	100.00	34.19	**34.88**	2.15/1.46
R-Zero	$a_1 = a_2 \neq a_3$	12.52	19.59	**26.14**	24.67/18.12
	Full evaluation pool	100.00	35.21	**36.03**	3.09/2.27

satisfy $a_1 = a_2 \neq a_3$; otherwise, retain the response from Iter3. The rule uses only response agreement for individual decisions; correctness labels are used solely for evaluation.

The agreement event occurs on 8.51% of the Agent0 pool and 12.52% of the R-Zero pool. Within these subsets, selecting the historical response improves accuracy from 18.06% to 26.19% for Agent0 and from 19.59% to 26.14% for R-Zero. Applied to the full evaluation pools, the rule improves accuracy over Iter3 by 0.69 and 0.82 percentage points, respectively.

Here, B denotes the percentage of instances for which switching changes an incorrect response into a correct one, and H denotes the percentage for which switching changes a correct response into an incorrect one, each measured within the corresponding evaluation slice. Benefits exceed harms in both trajectories, although historical agreement does not guarantee correctness. This diagnostic shows that a simple, test-time-visible signal can recover part of the historical complementarity, motivating learned instance-level response selection.

D Parameter-Row Response Analysis on R-Zero

We further examine whether parameter-row response magnitude distinguishes capability repair from damage along the R-Zero self-evolution trajectory. Figure 6 shows layer-24 `gate_proj` CTR spectra for repair (incorrect to correct), damage (correct to incorrect), and stable (unchanged correctness) transitions. The three columns use sampling seeds 42, 43, and 44; orange triangles mark the top-20 responding rows.

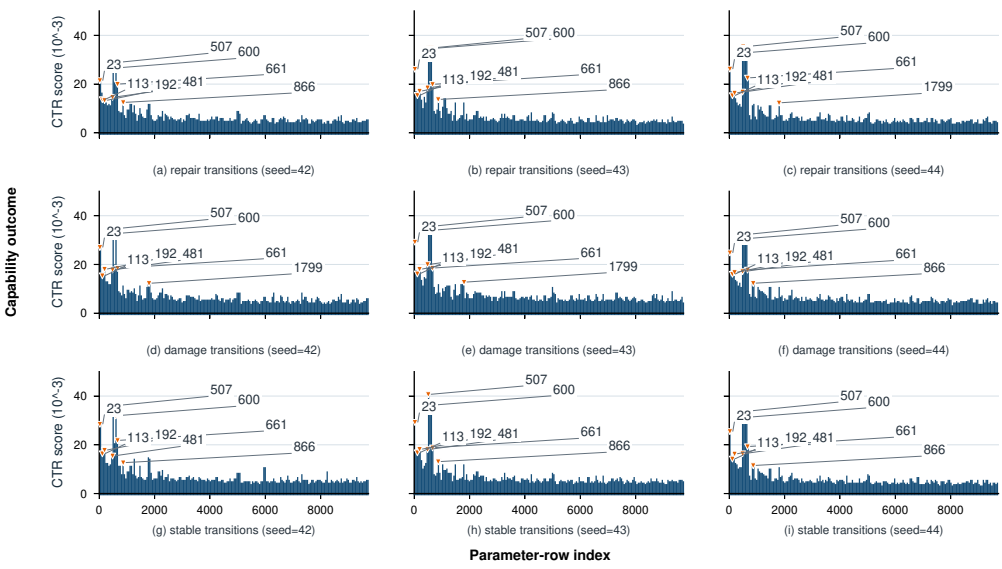

Figure 6: **R-Zero repairs and damages reuse the same update-sensitive parameter rows.** Row-level CTR spectra at layer 24 `gate_proj` are shown for repair (a–c), damage (d–f), and stable (g–i) transitions across three bootstrap seeds. Orange triangles mark the Top-20 responsive rows. Repair and damage spectra overlap strongly (Pearson $r = 0.988$, Spearman $\rho = 0.980$, Top-100 Jaccard $= 0.818$), showing that response magnitude alone cannot reliably distinguish beneficial from harmful changes.

Repair and damage exhibit similar spectra, with Pearson correlation $r = 0.988$ and Spearman correlation $\rho = 0.980$. Their top-100 responding row sets have a Jaccard similarity of 0.818, compared with a random reference of 0.005. Prominent rows, including 507 and 661, recur across groups and seeds. Stable transitions also show pronounced peaks, indicating that strong responses are not specific to changes in correctness.

These descriptive results suggest that CTR identifies rows with substantial activation changes, but its magnitude alone does not reliably distinguish beneficial from harmful update effects. They caution against selecting rows for parameter restoration solely by response magnitude and motivate response-level historical capability recovery.

E *CapDyn-Match* ALGORITHM

Algorithm 1 summarizes the inference procedure of *CapDyn-Match*. Given a query, frozen historical iterations generate candidate responses in parallel. *CapDyn-Match* scores each candidate using response representations and cross-iteration agreement, fuses the two scores, and applies margin-gated selection. If the top candidate is not uniquely best or its margin is below τ, the method falls back to the training-derived prior k_{prior}.

F EXPERIMENTAL DETAILS

F.1 REPRODUCTION SETUP

Self-evolution trajectories. We study two independently reproduced self-evolution trajectories, Agent0 (Xia et al., 2025) and R-Zero (Huang et al., 2025). Both use Qwen3-4B-Base as the initial backbone and run for three iterations. Agent0 co-evolves a Curriculum Agent, which generates training tasks, and an Executor, which learns to solve them with tool-integrated reasoning. R-Zero co-evolves a Challenger, which generates questions, and a Solver, which learns from the generated questions. We retain the Agent0 Executor and the R-Zero Solver after each iteration as Iter1,

Algorithm 1 *CapDyn-Match*: Historical Capability Recovery

Require: Query q; frozen iterations $\{M_k\}_{k=1}^{n}$
Require: Frozen encoder E_{ref}; feature transform $\mathcal{T}$; query encoder ϕ
Require: Shared probe f_{rep}; agreement heads $\{f_{\mathrm{cap},k}\}_{k=1}^{n}$
Require: Training-derived prior k_{prior}; fusion weight α; margin threshold τ
Ensure: Selected response $y_{k^\star}$

Stage 1: Historical candidate generation
for $k = 1, \ldots, n$ **do**
  $y_k \leftarrow M_k(q)$
end for
Stage 2: Candidate scoring
$x \leftarrow \phi(q) \oplus g(\{y_k\}_{k=1}^{n})$
for $k = 1, \ldots, n$ **do**
  $h_k \leftarrow E_{\mathrm{ref}}(q \oplus y_k)_{\ell,\mathrm{last}}$
  $r_k \leftarrow f_{\mathrm{rep}}(\mathcal{T}(h_k))$
  $v_k \leftarrow f_{\mathrm{cap},k}(x)$
end for
$\widetilde{r} \leftarrow \mathrm{RowZScore}(r_1, \ldots, r_n)$
$\widetilde{v} \leftarrow \mathrm{RowZScore}(v_1, \ldots, v_n)$
$s \leftarrow \alpha\widetilde{r} + (1-\alpha)\widetilde{v}$
Stage 3: Margin-gated selection
$(k_{(1)}, k_{(2)}) \leftarrow \mathrm{TopTwo}(s)$
$\Delta \leftarrow s_{k_{(1)}} - s_{k_{(2)}}$
if $s_{k_{(1)}}$ is uniquely maximal **and** $\Delta \geq \tau$ **then**
  $k^\star \leftarrow k_{(1)}$
else
  $k^\star \leftarrow k_{\mathrm{prior}}$
end if
return $y_{k^\star}$

Table 8: **Training configurations reported for Agent0 and R-Zero.** Both frameworks initialize their two roles from Qwen3-4B-Base. Steps and rollouts refer to the corresponding role's training within an iteration; training rollouts should not be confused with the mean@32 evaluation samples.

	Agent0		R-Zero	
Configuration	Executor	Curriculum	Solver	Challenger
Initial backbone	Qwen3-4B-Base		Qwen3-4B-Base	
Global batch size	128	128	128	128
Learning rate	1×10^{-6}	1×10^{-6}	1×10^{-6}	1×10^{-6}
Weight decay	1×10^{-2}	1×10^{-2}	1×10^{-2}	1×10^{-2}
KL penalty coefficient	1×10^{-2}	1×10^{-2}	1×10^{-2}	1×10^{-2}
Maximum training steps	40	5	15	5
Training rollouts	16	4	5	4
Rollout temperature	1.0	1.0	1.0	1.0
Rollout top-p	0.99	0.99	0.99	0.99

Iter2, and Iter3, respectively. The two trajectories are trained independently; an iteration from one framework is never used to initialize the other.

Training configurations. We follow the training procedures and hyperparameters reported in the respective original papers. Table 8 summarizes the role-specific configurations. In Agent0, each generated task is evaluated using 10 Executor responses to estimate uncertainty and obtain pseudo-labels. The original configuration retains tasks whose self-consistency lies between 0.3 and 0.8, and uses a sandboxed code interpreter for the Executor. In R-Zero, the Challenger generates a pool of 8,000 candidate questions per iteration. For each question, the Solver generates 10 responses; questions are retained when 3–7 responses agree with the majority-vote pseudo-label. These quantities concern self-evolution training and are distinct from the mean@32 evaluation protocol.

F.2 Selector Fitting and Implementation Details

Reference encoder. For each trajectory, we use its frozen Iter2 model as the shared reference encoder for all three iterations: the executor checkpoint `agent0_executor_agent_training_v2/global_step_40` for Agent0 and the solver checkpoint `qwen3-4b_solver_v2/global_step_15` for R-Zero. Both are 36-layer Qwen3-4B models with a hidden size of 2,560. We extract the final non-padding token representation from zero-indexed layer 23 (`hidden_states[24]` in Hugging Face). Chat-templated questions and complete responses are right-truncated to 1,536 tokens for Agent0 and 8,192 tokens for R-Zero. The encoders run in bfloat16, and their outputs are cached in float32. The layer is fixed; no layer selection is performed.

Feature fitting. For Agent0, transformations are fitted separately for the math, general, and code domains, with no parameters shared across domains. For both trajectories, all learned transformations are fitted within the corresponding training fold. The representation branch applies `StandardScaler` and truncated SVD to hidden states. The CapAgree branch uses question TF–IDF (1–2 grams; at most 30,000 features), truncated SVD, L2 normalization, and seven answer-agreement features. During inner cross-validation, these transformations are fitted only on the corresponding inner-training data; after tuning, they are refitted on the complete outer-training fold. Only label-independent hidden states from the frozen encoder are cached across folds.

Hyperparameters. SVD dimensionality is fixed at 128. Correctness predictors use class-balanced L2 logistic regression with `liblinear`, `C=1.0`, and `max_iter=400`. Inner cross-validation selects $\alpha \in \{0.25, 0.50, 0.75\}$ and $\tau \in \{0, 0.05, 0.10\}$. For Agent0, α and τ are selected separately for each domain.

Seeds and splits. We repeat nested five-fold cross-validation with split seeds 0, 1, and 2. Agent0 is split separately within each domain. Questions are shuffled within each benchmark, and all 32 rollouts of a question remain in the same fold. Inner-split seeds are `seed * 1000 + outer_fold`. The SVD and logistic-regression random states use the corresponding experiment seed. Candidate responses are fixed across seeds. Confidence intervals use 10,000 bootstrap resamples with a separate seed, `20260912`.

F.3 Baselines and Ablations

This section describes the comparison methods and ablation variants used to evaluate *CapDyn-Match*. Methods that select among historical responses use the same candidate responses generated by the frozen iterations Iter1–Iter3. Parameter-merging baselines instead generate responses from the merged models.

Individual iterations. Each baseline directly returns the response generated by the corresponding iteration, without cross-iteration selection. These baselines establish the performance of each individual model along the self-evolution trajectory.

Majority Vote. Majority Vote selects an output supported by the largest number of iterations, based on agreement among their extracted final outputs. It uses cross-iteration agreement without a learned correctness predictor.

Parameter-merging baselines. We include two parameter-level approaches for exploiting historical iterations. Model Soup (Wortsman et al., 2022) averages the parameters of historical iterations to obtain a single merged model. TIES-Merging (Yadav et al., 2023) resolves interference among parameter updates through trimming, sign election, and disjoint merging. We apply both methods to the same historical iterations used by *CapDyn-Match*, providing parameter-level counterparts to our response-level historical selection.

Qwen3-32B Judge. We use Qwen3-32B as an external judge to select among the candidate responses. The judge receives the query and candidate responses and selects the response it considers most correct. It has no access to reference answers or candidate correctness labels during selection.

Query-only Router. This baseline selects an iteration using only query features, without inspecting candidate responses. It learns iteration-specific correctness predictors from the query representation and selects an iteration according to their scores. This comparison tests whether query information alone is sufficient for historical iteration selection.

Best Iter. Best Iter. is the individual iteration with the highest aggregate accuracy on the corresponding evaluation pool. It is a retrospective reference determined using evaluation outcomes, rather than an instance-level selection strategy. It is distinct from the training-derived prior used by *CapDyn-Match*, which is determined without access to held-out evaluation labels.

Historical Union. Historical Union provides an oracle reference by selecting a correct candidate whenever at least one iteration produces one. For binary correctness labels c_{ik}, its accuracy is

$$\mathrm{Acc}_{\mathrm{Union}} = \frac{1}{N}\sum_{i=1}^{N} \max_{k\in\{1,2,3\}} c_{ik}. \tag{11}$$

For benchmarks evaluated with repeated sampling, the oracle is computed within each rollout-specific candidate group and then averaged according to the evaluation protocol. Because it requires correctness labels, Historical Union serves only as an upper reference for selection from the available candidates.

CapAgree-only. This variant uses only the CapAgree scoring component, which predicts candidate correctness from query features and cross-iteration agreement patterns. It excludes query–response hidden representations, isolating the contribution of the agreement-based branch.

Representation-only. This variant uses only the shared correctness probe over frozen query–response representations, excluding the CapAgree branch. It retains the margin-based selection rule and falls back to Train-Prior when the score separation is insufficient. This isolates the contribution of the representation branch within the gated selector.

Vanilla Linear Verifier. This baseline uses the same frozen representations, training-fitted feature transformation, shared linear logistic-regression probe, candidate correctness labels, and training splits as Representation-only. At inference time, it directly selects the candidate with the highest predicted correctness score, without margin-based gating or fallback to Train-Prior. The comparison with Representation-only isolates the effect of the gate when using representation scores alone.

F.4 Evaluation Benchmarks

Mathematical Reasoning. We use seven challenging benchmarks: AMC (Mathematical Association of America, 2023), Minerva (Lewkowycz et al., 2022), MATH-500 (Hendrycks et al., 2021b), GSM8K (Cobbe et al., 2021), OlympiadBench (He et al., 2024), AIME-2024, and AIME-2025. For these tasks, where answers can be complex, we employ GPT-4o as a programmatic judge to semantically verify the correctness of the final answer against the ground truth. For AMC23, AIME24, and AIME25, we report $\mathrm{mean@32}$ under a shared generation protocol across historical iterations. For all remaining mathematical benchmarks, we report accuracy under greedy decoding. The identical prompts and decoding configurations used across iterations enable aligned comparisons, while the greedy-decoded benchmarks additionally exclude sampling variance from instance-level correctness transitions.

General-Domain Reasoning. To test the generalization of reasoning ability, we evaluate the following challenging benchmarks:

- **MMLU-Pro** (Wang et al., 2024): An enhanced version of the MMLU benchmark (Hendrycks et al., 2021a), featuring a more challenging suite of multi-task questions designed to provide a stricter evaluation of language model capabilities.
- **SuperGPQA** (M-A-P Team et al., 2025): A large-scale benchmark evaluating graduate-level knowledge and reasoning across 285 disciplines. It contains diverse questions spanning both mainstream and specialized academic domains.

- **BBEH** (Kazemi et al., 2025): This benchmark builds upon the foundation of BIG-Bench Hard (Suzgun et al., 2023) by incorporating a new selection of tasks specifically engineered to be more difficult, thus providing a more accurate measure of complex reasoning skills.

For this category, we follow the experimental setup, prompts, and evaluation code of General-Reasoner (Ma et al., 2025), and report Exact Match (EM) accuracy obtained using greedy decoding.

Code Generation. We evaluate LiveCodeBench (Jain et al., 2025), HumanEval+ and MBPP+ (Liu et al., 2023), and BigCodeBench-Hard (Zhuo et al., 2025). We retain each benchmark's official execution-based evaluator and use EvalPlus for HumanEval+ and MBPP+. All checkpoint candidates for a given problem are decoded and judged under the same prompt, generation, and execution protocol.

F.5 Evaluation Protocols and Metrics

Evaluation Protocols. We evaluate *CapDyn-Match* in two settings. For the main comparisons and ablations, we use nested five-fold cross-validation: outer folds define the evaluation partitions, while inner folds select hyperparameters. All splits are grouped by question, and results are averaged over three random seeds using out-of-fold predictions.

For cross-dataset generalization, we conduct leave-one-dataset-out (LODO) evaluation. For each held-out dataset, all selector components, hyperparameters, and the training-derived prior are fitted exclusively on the remaining datasets. The held-out dataset and its correctness labels are used only for final evaluation.

Evaluation Metrics. We report accuracy (%). AMC23, AIME24, and AIME25 use mean@32, whereas all other benchmarks use greedy decoding. Overall and domain-level results for the main comparison are question-micro-averaged.

For aggregate LODO evaluation, we report benchmark-macro accuracy, which assigns equal weight to each held-out benchmark, together with question-weighted improvements. LODO gains are measured against the training-derived prior, with 95% benchmark-stratified paired-bootstrap confidence intervals.

F.6 Metric Computation for Repeated-Sampling Benchmarks

AMC23, AIME24, and AIME25 are evaluated using mean@32. For each question q, we align the 32 rollouts across historical iterations by rollout index $r \in \{1, \ldots, 32\}$. Each pair (q, r) forms a candidate group containing one response from each iteration. The selector is applied independently to each candidate group, while all 32 groups belonging to the same question remain in the same training, validation, or test split.

Let $c_{t,q,r} \in \{0, 1\}$ indicate whether the response from iteration t is correct for question q in rollout r, and let $\hat{t}(q, r)$ be the iteration selected without access to correctness labels. The question-level mean@32 accuracies of an individual iteration and the selector are, respectively,

$$A_t(q) = \frac{1}{32} \sum_{r=1}^{32} c_{t,q,r}, \qquad A_{\text{sel}}(q) = \frac{1}{32} \sum_{r=1}^{32} c_{\hat{t}(q,r),q,r}. \tag{12}$$

The Historical Union is computed *within each rollout-specific candidate group* before averaging over rollouts:

$$A_{\text{union}}(q) = \frac{1}{32} \sum_{r=1}^{32} \max_t c_{t,q,r}. \tag{13}$$

Thus, it does not take the maximum of the iterations' question-level mean@32 scores. For example, if two iterations are correct on disjoint sets of rollouts, the Historical Union credits both sets.

For adjacent iterations $t-1$ and t, we likewise compute correctness transitions at the rollout level:

$$G_t(q) = \frac{1}{32}\sum_{r=1}^{32} \mathbb{I}[c_{t-1,q,r} = 0,\ c_{t,q,r} = 1], \tag{14}$$

$$D_t(q) = \frac{1}{32}\sum_{r=1}^{32} \mathbb{I}[c_{t-1,q,r} = 1,\ c_{t,q,r} = 0]. \tag{15}$$

These quantities satisfy $A_t(q) - A_{t-1}(q) = G_t(q) - D_t(q)$. For greedy-decoded benchmarks, there is only one candidate group per question, so the same definitions apply with a single rollout.

F.7 Per-Benchmark Results and Extended-Trajectory Evaluation

This section provides per-benchmark results for the Qwen3-4B experiments and extends the evaluation to a six-iteration Qwen3-1.7B trajectory. In all settings, a routing method selects one of the responses generated by the historical iterations for the same question.

Qwen3-4B: held-out benchmarks. Table 9 details the leave-one-dataset-out (LODO) results for Agent0 and R-Zero. *CapDyn-Match* exceeds the strongest individual iteration on 10 of 14 benchmarks for Agent0 and 11 of 14 for R-Zero. The gains span mathematical reasoning, general reasoning, and code generation, but are not uniform. For example, routing falls below the strongest iteration on BigCodeBench-Hard for both trajectories and on LiveCodeBench for R-Zero. These comparisons describe the per-benchmark results; the LODO aggregate gain in the main text is measured against the training-derived prior, which does not use labels from the held-out benchmark.

Qwen3-1.7B: six-iteration nested cross-validation. Table 10 evaluates routing among six historical iterations. *CapDyn-Match* matches or exceeds the other routing methods on all 14 benchmarks and improves over the latest iteration, Inter6, on every benchmark. The gains range from 0.91 percentage points on LiveCodeBench to 7.40 points on MMLU-Pro. The comparison also shows why selection matters: the strongest individual iteration varies across benchmarks, and Inter6 is not consistently the strongest. On Minerva, for instance, Inter5 achieves 44.85%, above *CapDyn-Match* at 43.87%; thus the routing result should not be interpreted as outperforming every historical iteration on every benchmark.

Qwen3-1.7B: generalization to held-out benchmarks. Table 11 reports the corresponding LODO results. *CapDyn-Match* matches or exceeds the other routing methods on all 14 held-out benchmarks, with a tie with Vanilla Linear on SuperGPQA. It also improves over Inter6 on every benchmark, although the gain on LiveCodeBench is small at 0.22 percentage points. The results show that selection among a longer model history remains effective when the target benchmark is excluded from selector training. As in the nested-CV setting, these gains relative to Inter6 do not imply that the selector surpasses the strongest individual iteration in every row.

The six-iteration experiment complements the three-iteration Qwen3-4B results by testing a longer history and a larger candidate set. Since the solver size and evolution trajectory also differ, accuracy values across the two experiments are not directly comparable.

Table 9: **Leave-one-dataset-out (LODO) evaluation on Qwen3-4B.** Accuracy (%) on held-out benchmarks for individual iterations and *CapDyn-Match*. Highest scores within each trajectory are bolded.

	AGENT0				R-ZERO			
Held-out benchmark	Iter1	Iter2	Iter3	*CapDyn-Match*	Iter1	Iter2	Iter3	*CapDyn-Match*
Mathematical Reasoning								
AMC23	49.92	49.84	48.83	**53.67**	54.30	54.45	53.20	**56.33**
AIME24	11.35	12.60	12.40	**15.52**	12.40	13.54	18.23	**18.96**
AIME25	9.27	6.88	6.35	**12.29**	7.71	4.90	2.71	**9.27**
GSM8K	92.04	92.65	91.21	**92.72**	91.96	91.96	91.89	**92.57**
MATH500	78.80	78.40	77.20	**81.20**	78.40	77.00	78.00	**79.80**
OlympiadBench	47.41	47.56	44.59	**49.93**	44.44	46.81	42.52	**46.96**
Minerva	59.19	59.19	59.19	**59.93**	**60.66**	58.46	57.35	59.93
General Knowledge and Reasoning								
BBEH	9.71	11.19	**13.56**	12.95	10.53	10.38	10.22	**12.21**
MMLU-Pro	50.25	50.59	50.22	**53.20**	54.80	54.16	53.28	**57.44**
SuperGPQA	26.60	27.46	26.99	**27.50**	28.08	27.80	27.66	**30.19**
Code Generation								
HumanEval+	78.66	77.44	81.10	**81.71**	76.22	73.78	76.22	**77.44**
MBPP+	72.22	**74.34**	71.96	73.28	66.40	66.67	67.20	**67.46**
BigCodeBench-Hard	16.89	**18.24**	16.22	16.89	16.22	**16.89**	15.54	16.22
LiveCodeBench	27.05	26.82	**27.27**	27.05	22.11	22.70	**29.16**	24.27

Table 10: **Routing among six historical iterations under nested cross-validation.** Question-level official accuracy (%) is averaged over three split seeds. Δ_{Inter6} is the accuracy difference (pp) between CapDyn-Match and the latest iteration, Inter6. Bold indicates the best result among the six routing methods in each row; individual iterations are excluded from this comparison.

Benchmark	Inter1	Inter2	Inter3	Inter4	Inter5	Inter6	Majority	Query-only	CapAgree	Representation	Vanilla Linear	CapDyn-Match	Δ_{Inter6}
Math													
MATH500	68.60	65.40	64.80	65.80	66.60	65.40	70.13	66.00	69.07	70.87	71.00	**71.20**	+5.80
GSM8K	82.11	81.80	82.79	82.18	80.97	80.52	83.60	81.88	83.17	84.58	84.53	**84.86**	+4.34
AMC23	36.72	39.06	33.94	33.91	36.64	38.83	40.81	37.89	41.22	43.28	43.67	**44.06**	+5.23
Minerva	39.71	42.28	41.91	42.65	44.85	41.18	43.26	40.81	43.01	43.01	43.63	**43.87**	+2.69
Olympiad	31.56	29.04	31.85	30.37	30.22	29.48	34.52	31.95	35.41	33.98	34.27	**35.70**	+6.22
AIME24	12.29	12.92	9.69	10.73	9.38	11.56	13.16	9.90	13.30	14.86	15.21	**15.42**	+3.86
AIME25	3.02	5.00	7.29	6.77	5.94	5.00	6.84	5.35	5.38	**8.44**	**8.44**	**8.44**	+3.44
General													
BBEH	8.12	7.94	9.49	9.31	9.60	9.91	9.31	9.96	9.59	11.07	11.14	**11.16**	+1.25
MMLU-Pro	36.88	36.37	35.89	34.98	34.52	33.82	36.80	36.09	37.11	40.79	41.14	**41.22**	+7.40
SuperGPQA	18.51	18.96	18.70	18.57	17.90	17.51	19.04	18.69	19.16	22.03	22.08	**22.25**	+4.74
Code													
HumanEval+	59.15	60.98	59.76	61.59	61.59	59.76	60.98	59.35	60.16	60.37	60.77	**61.77**	+2.01
MBPP+	58.47	58.20	57.94	58.20	59.26	57.94	58.64	58.11	58.02	58.64	58.29	**59.99**	+2.05
BigCodeBench-Hard	5.41	5.41	4.73	6.76	6.08	4.73	5.86	5.41	5.86	5.86	6.08	**6.91**	+2.18
LiveCodeBench	13.89	14.87	14.09	13.89	14.29	15.07	13.89	15.07	15.00	14.94	15.59	**15.98**	+0.91

Table 11: **Routing among six historical iterations on held-out benchmarks.** Leave-one-dataset-out (LODO) accuracy (%). Δ_{Inter6} is the accuracy difference (pp) between CapDyn-LODO and the latest iteration, Inter6. Bold indicates the best result among the six routing methods in each row; individual iterations are excluded from this comparison.

Benchmark	Inter1	Inter2	Inter3	Inter4	Inter5	Inter6	Majority	Query-only	CapAgree	Representation	Vanilla Linear	CapDyn-LODO	Δ_{Inter6}
Math													
MATH500	68.60	65.40	64.80	65.80	66.60	65.40	69.40	66.00	70.00	70.40	70.60	**71.60**	+6.20
GSM8K	82.11	81.80	82.79	82.18	80.97	80.52	83.47	81.27	83.40	84.31	83.78	**84.61**	+4.09
AMC23	36.72	39.06	33.94	33.91	36.64	38.83	40.94	37.42	41.25	44.30	43.75	**44.53**	+5.70
Minerva	39.71	42.28	41.91	42.65	44.85	41.18	43.01	40.44	44.12	44.12	44.49	**44.85**	+3.67
Olympiad	31.56	29.04	31.85	30.37	30.22	29.48	34.67	32.15	33.33	33.63	33.93	**36.00**	+6.52
AIME24	12.29	12.92	9.69	10.73	9.38	11.56	13.75	10.10	14.27	15.52	15.83	**16.25**	+4.69
AIME25	3.02	5.00	7.29	6.77	5.94	5.00	6.98	5.94	5.62	8.85	8.85	**8.96**	+3.96
General													
BBEH	8.12	7.94	9.49	9.31	9.60	9.91	9.31	8.72	9.27	10.91	10.91	**11.26**	+1.35
MMLU-Pro	36.88	36.37	35.89	34.98	34.52	33.82	36.57	35.70	37.03	39.94	40.33	**40.58**	+6.76
SuperGPQA	18.51	18.96	18.70	18.57	17.90	17.51	19.08	18.71	19.07	21.70	**21.95**	**21.95**	+4.44
Code													
HumanEval+	59.15	60.98	59.76	61.59	61.59	59.76	60.98	61.59	61.59	60.98	59.76	**62.20**	+2.44
MBPP+	58.47	58.20	57.94	58.20	59.26	57.94	58.20	58.47	58.99	57.94	58.73	**59.52**	+1.58
BigCodeBench-Hard	5.41	5.41	4.73	6.76	6.08	4.73	6.08	6.08	6.08	4.73	6.08	**6.76**	+2.03
LiveCodeBench	13.89	14.87	14.09	13.89	14.29	15.07	13.89	14.29	14.29	13.11	13.31	**15.29**	+0.22