BEYOND DATA QUANTITY: FROM BOOTSTRAP EMERGENCE TO COMPOSITION-SENSITIVE SELF-EVOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-evolving agents aim to reduce reliance on expert supervision by learning from their own interaction experience, yet it remains unclear how self-evolution emerges from an initially unreliable agent and what limits further improvement once executable self-generation becomes possible. We study this process in a controlled tool-augmented VideoQA setting with nested, task-balanced expert supervision and controlled within-task solution logic. Our experiments reveal two empirically distinct regimes. Before bootstrap, performance is strongly sensitive to expert supervision. A small increase in structured trajectories can transform an unreliable planner into one capable of executable self-generation. After bootstrap, further increases in expert trajectory quantity yield diminishing and strongly task-dependent gains, and this pattern persists after matching the optimization budget. We further characterize this post-bootstrap regime. Compared with continued expert-only training, self-generated experience produces a substantially broader set of successful tool-use patterns. At the action level, residual failures exhibit heterogeneous bottlenecks across generation, execution, and correct completion. Finally, controlled interventions show that repeatedly exposing the model to existing trajectories or applying coarse rebalancing provides limited benefit, whereas changing trajectory composition or fine-grained action-level supervision can alter the affected capability. Together, these results identify a transition from quantity-sensitive bootstrap emergence to composition-sensitive post-bootstrap learning, providing an empirical characterization of how tool-augmented self-evolution changes after it becomes executable.

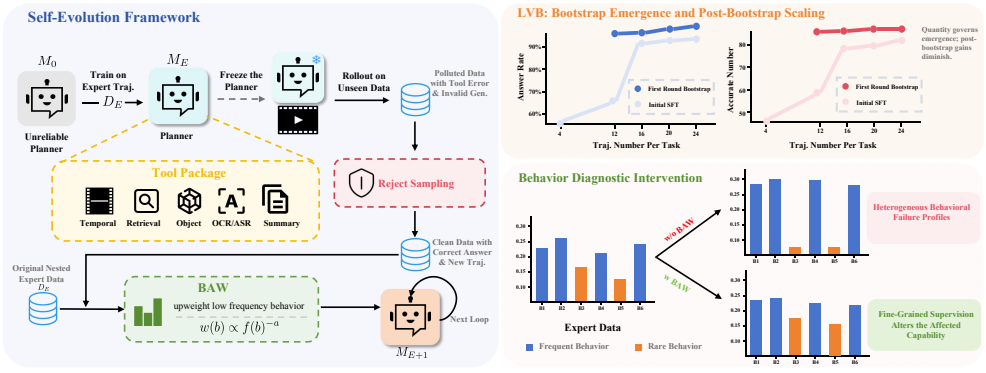


Figure 1: **Overview of bootstrap self-evolution.** Limited supervision enables executable self-generation, while post-bootstrap gains diminish and vary across tasks. Stage-wise analysis reveals heterogeneous behavioral failures, and controlled interventions highlight the importance of experience composition and fine-grained supervision.

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.

1 INTRODUCTION

Recent advances in large language models and vision-language models have enabled autonomous agents to reason, interact with external environments, and improve from accumulated experience. Self-evolving agents further reduce reliance on human supervision by generating and learning from their own interaction trajectories Wang et al. (2023b); Zelikman et al. (2022); Gulcehre et al. (2023); Shinn et al. (2023); Yuan et al. (2024). Meanwhile, tool-augmented agents extend model capabilities through multi-step interaction with external tools Schick et al. (2023); Yao et al. (2023); Qin et al. (2024); Patil et al. (2023). However, most existing approaches assume that the initial agent can already produce usable reasoning-action trajectories Zhai et al. (2025); Wu et al. (2025); Yang et al. (2026). This leaves two related questions: *how can self-evolution emerge from an initially unreliable tool-use planner, and what limits further improvement after bootstrap?*

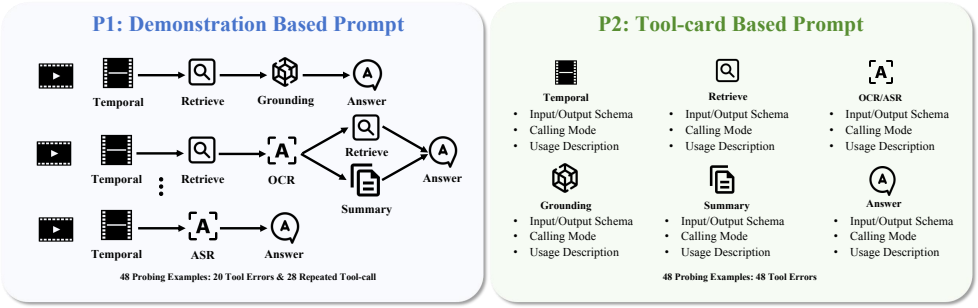


Figure 2: **Prompt probing.** We use two prompting strategies to probe the model's capability boundary. Each strategy is evaluated on 48 probing examples with 1 rollout per example.

We study this process in tool-augmented VideoQA using task-balanced expert trajectories with controlled within-task solution logic. Prompt probing shows that the pretrained planner possesses partial tool-use capability but cannot reliably orchestrate tools or complete reasoning loops. On LongVideoBench (LVB) Wu et al. (2024), limited structured supervision is sufficient to induce a clear transition from failed interaction to executable self-generation. After bootstrap, however, performance differences across larger supervision scales shrink substantially.

We further examine this post-bootstrap regime on HERBench Ben-Ami et al. (2026). Although expert supervision is nested and task-balanced, increasing its scale produces diminishing and task-dependent gains, and matching the optimization budget does not remove this pattern. These results suggest a transition from a quantity-sensitive bootstrap stage to a regime in which expert trajectory count alone no longer explains further improvement.

To understand this regime, we analyze planner behaviors across generation, valid execution, and correct completion. The resulting failure profiles are highly heterogeneous. Some behaviors are rarely generated, while others are frequently produced but fail at later stages. We use Behavior-Aware Weighting (BAW) as a diagnostic intervention at the same fine-grained action level. BAW improves both SVA performance and accuracy among answered examples, showing that the observed post-bootstrap limitation can be altered through fine-grained behavioral supervision.

We then use controlled ablations to examine which aspects of the training experience are associated with these limitations. Additional optimization, repeated exposure to the existing SVA trajectories, and coarse trajectory- or tool-level interventions provide little or no improvement. In contrast, SVA-Enhanced introduces new SVA trajectories containing behaviors that are sparsely represented in the original supervision and yields clear gains in both SVA and overall accuracy. This contrast suggests that simply increasing exposure to existing data is insufficient. The behavioral composition of the added experience matters. Together with the BAW results, these findings characterize post-bootstrap learning as sensitive to both trajectory composition and fine-grained supervision structure, rather than data quantity alone.

Our main contributions are:

- We establish a controlled framework for studying how self-evolution emerges from an initially unreliable tool-use planner, using nested task-balanced supervision and controlled within-task solution logic.
- We reveal two scaling regimes: supervision quantity governs bootstrap emergence, while post-bootstrap gains become diminishing and task-dependent.
- We show that first-round post-bootstrap improvement is sensitive to experience composition and fine-grained behavioral support.

2 METHOD

2.1 SELF-EVOLUTION FRAMEWORK FOR TOOL-AUGMENTED VIDEOQA

We study how an initially unreliable planner can bootstrap self-evolution from limited expert supervision in a fixed tool-augmented VideoQA environment. Given an initial planner M_0 and a limited set of expert trajectories $\mathcal{D}_E$, we first obtain an expert-initialized planner:

$$M_E = \mathrm{Train}(M_0, \mathcal{D}_E). \tag{1}$$

The initialized planner then interacts with the environment to generate new trajectories. At each interaction step, the planner produces either a tool call or a final answer. A frozen multimodal worker executes the requested video operations, while the planner is responsible for tool selection and multi-step orchestration. Generated trajectories satisfying the verification criteria are collected as $\mathcal{D}_{\mathrm{clean}}$. The planner is subsequently updated using both the original expert trajectories and the verified self-generated trajectories:

$$M_{k+1} = \mathrm{Train}\left(M_k, \mathcal{D}_E \cup \mathcal{D}^k_{\mathrm{clean}}\right). \tag{2}$$

The bootstrap self-evolution process can therefore be summarized as

$$M_0 \xrightarrow{\mathcal{D}_E} M_E \xrightarrow{\mathrm{Rollout}} \mathcal{D}_{\mathrm{clean}} \xrightarrow{\mathrm{Retraining}} M_1. \tag{3}$$

Throughout this process, the tool environment and multimodal worker remain fixed, and only the planner is updated. Detailed tool definitions, input/output schemas, and calling conventions are provided in the Appendix.

2.2 CONTROLLED BOOTSTRAP PROTOCOL

We initialize the planner using a limited expert trajectory set $\mathcal{D}_E = \{\tau_i\}_{i=1}^N$, where each trajectory records the sequence of planner actions, tool observations, and the final answer.

Expert trajectories are constructed in a task-balanced manner and follow a controlled solution logic within each task. This design reduces within-task strategy variation, allowing us to vary supervision scale while keeping task composition and demonstrated solution structure relatively stable. It provides a controlled setting for studying both bootstrap emergence and subsequent post-bootstrap learning. After expert initialization, the planner performs autonomous rollout on VideoQA samples not used for expert initialization. Generated trajectories may fail because of invalid generation, tool execution errors, or incorrect final answers. To avoid directly reusing unreliable generations for training, we apply verification-based rejection sampling and retain a trajectory only if it is executable, correct, and structurally trainable:

$$D_{\mathrm{clean}} = \{\tau \mid \mathrm{Valid}(\tau) = 1,\ \mathrm{Correct}(\tau) = 1,\ \mathrm{Trainable}(\tau) = 1\}. \tag{4}$$

Validity and correctness are necessary acceptance conditions. Trajectories must satisfy the structural trainability criteria described in Appendix A.4.3 before being added to the next-round corpus. Correctness is evaluated using the ground-truth answers associated with the rollout pool together with deterministic execution checks. Expert initialization, autonomous rollout, and final evaluation use

question-level disjoint sets. Ground-truth answers from the rollout pool are accessed only for selecting self-generated training trajectories and are never used for final evaluation or model selection on the evaluation set.

At evolution round k, the training set is formed by combining the original expert trajectories with the accepted self-generated trajectories:

$$\mathcal{D}_{k+1} = \mathcal{D}_E \cup \mathcal{D}^k_{\text{clean}}. \tag{5}$$

Unless otherwise specified, our main experiments report one bootstrap self-evolution round. Accordingly, our setting should be interpreted as verifier-assisted self-evolution. This controlled design isolates the learning dynamics under reliable verification rather than modeling fully label-free deployment-time self-improvement.

2.3 Stage-wise Behavior Characterization and Diagnostic Intervention

To characterize post-bootstrap failures beyond trajectory-level performance, we analyze generated trajectories at the level of fine-grained planner actions. Each behavior is defined by its tool category and calling mode, with final-answer generation included in the same taxonomy. Our experiments characterize how limited expert supervision bootstraps executable tool use, how behavioral support evolves during self-evolution, and how supervision quantity and composition affect post-bootstrap performance.

For a behavior b, let $N_R(b)$ denote the number of generated trajectories containing b, where repeated occurrences within the same trajectory are counted once. We further define $N_V(b)$ as the number of such trajectories that remain valid and executable, and $N_{CV}(b)$ as the number that are both valid and correct. Given a set of generated trajectories $\mathcal{D}_R$, we define

$$p_R(b) = \frac{N_R(b)}{|\mathcal{D}_R|}, \qquad P(V \mid b) = \frac{N_V(b)}{N_R(b)}, \tag{6}$$

$$P(C \mid b, V) = \frac{N_{CV}(b)}{N_V(b)}, \qquad P(A \mid b) = \frac{N_{CV}(b)}{N_R(b)}. \tag{7}$$

Here, $p_R(b)$ measures behavior coverage in generated trajectories, $P(V \mid b)$ measures the probability that a trajectory containing b remains executable, and $P(C \mid b, V)$ measures correctness among valid trajectories containing b. $P(A \mid b)$ denotes passing the validity-and-correctness verification gate rather than final corpus admission, which additionally requires structural trainability. For intervention comparisons, we additionally report $P(C \mid b) = N_C(b)/N_R(b)$, where $N_C(b)$ denotes the number of trajectories containing b whose final answer is correct. Thus, $P(C \mid b)$ measures final-answer correctness among trajectories containing b without conditioning on validity. Since the verification gate requires both validity and correctness,

$$P(A \mid b) = P(V \mid b)\, P(C \mid b, V). \tag{8}$$

Thus, these quantities provide a stage-wise decomposition of whether a behavior is rarely generated, frequently occurs in trajectories that fail execution, or appears in executable trajectories that nevertheless fail to produce the correct answer. The same validity and correctness criteria can also be applied to held-out evaluation trajectories. In this case, $P(A \mid b)$ denotes satisfaction of the validity-and-correctness verification gate rather than actual inclusion in the training set.

The statistics are computed separately for each model condition rather than pooling trajectories across supervision scales. We use them descriptively to characterize heterogeneous post-bootstrap failure profiles and do not interpret them as estimating the causal effect of individual behaviors.

Motivated by this stage-wise analysis, we further examine whether the affected capability is sensitive to supervision at the same fine-grained action level. We introduce Behavior-Aware Weighting as a diagnostic intervention. Unlike the trajectory-level statistics above, BAW operates on supervised planner action spans. Each supervised trajectory is decomposed into planner-generated action spans,

where each span corresponds to one complete planner action, including a tool call or final-answer generation.

Let $f(b)$ denote the number of supervised action spans assigned to behavior b. To mitigate the dominance of frequent behaviors while avoiding excessive upweighting of rare ones, we assign each behavior an inverse-square-root frequency weight:

$$\tilde{w}(b) = f(b)^{-1/2}. \tag{9}$$

We normalize the weights across the behavior set $\mathcal{B}$ before clipping:

$$\bar{w} = \frac{1}{|\mathcal{B}|} \sum_{b \in \mathcal{B}} \tilde{w}(b), \qquad w(b) = \text{clip}\left(\frac{\tilde{w}(b)}{\bar{w}}, 0.5, 2.0 \right). \tag{10}$$

The normalization keeps the weights approximately at unit scale before clipping, while clipping prevents extremely rare or frequent behaviors from receiving excessively large or small weights. The final clipped weights are not required to have an exact mean of one. All action tokens within the same supervised action span inherit the weight of its associated behavior. The behavior-aware weighted objective is

$$\mathcal{L}_{\text{BAW}} = -\sum_{i} w(b_i) \sum_{t \in \mathcal{S}_i} \log p_\theta(y_t \mid y_{<t}, x), \tag{11}$$

where b_i and $\mathcal{S}_i$ denote the behavior label and the set of token positions for the i-th supervised action span, respectively. BAW reweights only planner-generated tokens, including tool names, tool arguments, and final answers; tool observations and worker outputs are excluded. Accordingly, $f(b)$ measures action-span frequency rather than trajectory-level behavior coverage. We consider BAW as a diagnostic intervention to test whether modifying the allocation of supervision across fine-grained planner behaviors changes behavior utilization and the associated downstream capability.

3 Experiments

We study bootstrap self-evolution in controlled tool-augmented VideoQA, focusing on the emergence of executable self-generation and the first post-bootstrap transition.

3.1 Experimental Setup

Benchmarks and Controlled Expert Construction. We evaluate on two VideoQA benchmarks, LongVideoBench (LVB) Wu et al. (2024) and HERBench Ben-Ami et al. (2026). LVB studies bootstrap emergence, while HERBench examines post-bootstrap scaling and behavior-level failure patterns. For LVB, we select 12 tasks that cover diverse video understanding abilities and construct nested task-balanced expert sets with $N = \{4, 12, 16, 20, 24\}$ trajectories per task (48–288 total trajectories). For HERBench, we use six heterogeneous tasks (AGAR, AGBI, AGLT, RLPC, SVA, and TSO) and construct nested task-balanced expert sets with $N = \{10, 20, 30, 40, 80\}$ trajectories per task (60–480 total trajectories). Expert trajectories follow the controlled solution logic described in Sec. 2.2. This construction varies supervision scale with fixed task composition.

Data Split and Label Usage. Expert initialization, autonomous rollout, and final evaluation use disjoint question-level sets. Expert trajectories are used for planner initialization and retraining, while self-generated training trajectories are collected from a separate 600-question rollout pool (100 per task). Evaluation uses an independent $1,200$-question set (200 per task). Ground-truth answers from the rollout pool are accessed only for verification-based filtering, while evaluation trajectories are analyzed with the same validity and correctness criteria but are never used for training.

Models and Training Details. Our main planner is initialized from Qwen3.5-9B and optimized with LoRA fine-tuning. Unless otherwise specified, all models are trained for 5 epochs on 4 A800 GPUs using a learning rate of 2×10^{-5}. We use LoRA with rank $r = 16$ and scaling factor $\alpha = 32$. The per-device batch size is 1 with gradient accumulation over 3 steps, resulting in an effective batch

size of 12. The maximum sequence length is 32,768 tokens. Training is performed in bf16 precision with SDPA and TF32 enabled.

Rollout and Evolution Protocol. After expert initialization, the planner generates trajectories by interacting with the fixed tool environment on a disjoint rollout pool. HERBench uses 600 rollout trajectories per supervision setting. HERBench performance is evaluated on a separate set of 1,200 question-level examples. Each trajectory is limited to 12 tool calls. The LVB rollout protocol is provided in Appendix A.3.1. A trajectory passes the verification gate only when all tool calls are executable and the final answer is correct, and is added to the next-round corpus only if it also satisfies structural trainability. Accepted trajectories are combined with expert data for subsequent retraining. Unless otherwise specified, results report one bootstrap self-evolution round; additional implementation details are provided in the Appendix.

Evaluation Metrics. We evaluate the planner from both task-solving and autonomous-generation perspectives. Task performance is measured using accuracy, final-answer count, and answer rate. We also report invalid generations, tool execution errors, and exceptions to characterize failures in trajectory completion. For behavior analysis, we use the stage-wise criteria defined in Sec. 2.3.

Evaluation Uncertainty. Main scaling and intervention accuracies are reported as mean ± standard deviation over three independent runs. Exact count-level behavioral and trajectory accounting uses a fixed Run 1, as specified in the corresponding table captions and Appendix. Each rollout samples one trajectory per question. Additional rollout statistics are provided in the Appendix.

Table 1: **Probing bootstrap emergence on LVB across expert-supervision scales.**

Model	Setting	Final Answer	Correct	Accuracy
Qwen3.5-9B	$N = 4$	0 ± 0	0 ± 0	$0.0\% \pm 0.0\%$
	$N = 12$	96 ± 3.3	58 ± 1.5	$40.2\% \pm 1.0\%$
	+ Loop	137 ± 5.3	80 ± 3.5	$55.5\% \pm 2.4\%$
	$N = 16$	135 ± 5.6	79 ± 2.2	$54.8\% \pm 1.5\%$
	+ Loop	140 ± 4.4	83 ± 2.3	$57.6\% \pm 1.6\%$
	$N = 20$	134 ± 4.8	79 ± 2.0	$54.8\% \pm 1.4\%$
	+ Loop	140 ± 4.5	83 ± 2.0	$57.6\% \pm 1.4\%$
	$N = 24$	134 ± 5.1	81 ± 2.1	$56.2\% \pm 1.5\%$
	+ Loop	140 ± 2.0	84 ± 1.5	$58.3\% \pm 1.0\%$

Table 2: **Bootstrap probing across Qwen model variants on LVB.**

Model	Setting	Final Answer	Correct	Accuracy
Qwen2.5-VL-7B-Instruct	$N = 4$	0 ± 0	0 ± 0	$0.0\% \pm 0.0\%$
	$N = 12$	54 ± 1.0	20 ± 0.3	$13.8\% \pm 0.2\%$
	+ Loop	65 ± 0.6	33 ± 0.0	$22.9\% \pm 0.0\%$
	$N = 16$	70 ± 1.5	32 ± 1.0	$22.2\% \pm 0.7\%$
	+ Loop	100 ± 2.0	48 ± 0.6	$33.3\% \pm 0.4\%$
	$N = 20$	72 ± 2.2	33 ± 0.6	$22.9\% \pm 0.4\%$
	+ Loop	101 ± 1.8	50 ± 1.0	$34.7\% \pm 0.7\%$
	$N = 24$	72 ± 2.0	33 ± 1.5	$22.9\% \pm 1.0\%$
	+ Loop	105 ± 1.5	53 ± 1.0	$36.8\% \pm 0.7\%$
	$N = 28$	75 ± 2.0	35 ± 1.0	$24.3\% \pm 0.7\%$
	+ Loop	105 ± 1.0	55 ± 0.6	$38.1\% \pm 0.4\%$

Table 3: **Overall evaluation statistics of HERBench under different supervision scales.** Results are reported as mean ± standard deviation over three independent runs. All models are evaluated on the same $1,200$ examples.

Setting	Final Answer↑	Answer Rate↑	Correct↑	Acc.↑	Invalid Gen.↓	Tool Error↓	Exception↓
$N = 10$	752 ± 6.6	$62.6 \pm 0.5\%$	306 ± 4.3	$25.5 \pm 0.3\%$	72 ± 2.6	48 ± 2.3	24 ± 2.1
+ Loop	1069 ± 7.4	$89.0 \pm 0.6\%$	446 ± 4.1	$37.1 \pm 0.3\%$	69 ± 2.1	43 ± 2.8	19 ± 1.7
$N = 20$	780 ± 5.8	$65.0 \pm 0.4\%$	316 ± 3.6	$26.3 \pm 0.3\%$	66 ± 2.3	47 ± 2.5	33 ± 2.4
+ Loop	1079 ± 5.2	$89.9 \pm 0.4\%$	447 ± 3.8	$37.2 \pm 0.3\%$	46 ± 2.7	44 ± 2.2	31 ± 2.0
$N = 30$	815 ± 5.1	$67.9 \pm 0.4\%$	322 ± 3.1	$26.8 \pm 0.2\%$	65 ± 2.5	75 ± 2.9	20 ± 1.8
+ Loop	1053 ± 4.7	$87.7 \pm 0.3\%$	463 ± 2.8	$38.5 \pm 0.2\%$	52 ± 2.2	78 ± 2.6	17 ± 2.1
$N = 40$	823 ± 4.3	$68.5 \pm 0.3\%$	330 ± 2.5	$27.5 \pm 0.2\%$	54 ± 2.4	52 ± 2.1	13 ± 1.6
+ Loop	1094 ± 3.8	$91.1 \pm 0.3\%$	470 ± 2.3	$39.1 \pm 0.1\%$	50 ± 2.0	48 ± 2.5	8 ± 1.4
$N = 80$	817 ± 3.2	$68.0 \pm 0.2\%$	328 ± 1.8	$27.3 \pm 0.1\%$	50 ± 2.1	68 ± 2.4	10 ± 1.5
+ Loop	1072 ± 2.9	$89.3 \pm 0.2\%$	494 ± 1.5	$41.1 \pm 0.1\%$	52 ± 2.4	70 ± 2.1	6 ± 1.2

Table 4: **Task-level post-bootstrap scaling on HERBench.** Results are reported as mean ± standard deviation over three independent runs. Each task is evaluated on the same 200 examples.

Setting	AGAR	AGBI	AGLT	RLPC	SVA	TSO	Overall
$N = 10$ **+ Loop**	$47.0 \pm 1.5\%$	$59.0 \pm 1.5\%$	$46.5 \pm 1.3\%$	$21.5 \pm 1.0\%$	$10.5 \pm 1.3\%$	$38.5 \pm 0.8\%$	$37.1 \pm 0.3\%$
$N = 20$ **+ Loop**	$40.5 \pm 0.8\%$	$53.0 \pm 1.3\%$	$49.5 \pm 1.3\%$	$29.5 \pm 1.5\%$	$9.5 \pm 1.5\%$	$41.5 \pm 1.3\%$	$37.2 \pm 0.3\%$
$N = 30$ **+ Loop**	$50.0 \pm 1.3\%$	$58.5 \pm 0.8\%$	$48.0 \pm 0.8\%$	$26.0 \pm 1.3\%$	$10.5 \pm 1.3\%$	$38.5 \pm 1.3\%$	$38.5 \pm 0.2\%$
$N = 40$ **+ Loop**	$48.0 \pm 0.8\%$	$59.0 \pm 1.0\%$	$45.0 \pm 1.0\%$	$31.0 \pm 0.8\%$	$11.5 \pm 0.8\%$	$40.5 \pm 0.5\%$	$39.1 \pm 0.1\%$
$N = 80$ **+ Loop**	$52.5 \pm 0.8\%$	$54.5 \pm 0.5\%$	$52.5 \pm 0.5\%$	$40.0 \pm 0.8\%$	$8.0 \pm 0.8\%$	$39.5 \pm 0.5\%$	$41.1 \pm 0.1\%$

3.2 Bootstrap Emergence and Post-Bootstrap Scaling

We first examine how supervision scale affects the emergence of executable self-generation. As shown in Table 1, very limited supervision fails to bootstrap the planner, whereas moderate su-

Table 5: **Stage-wise behavioral characterization on SVA.** We report the behavior statistics defined in Sec. 2.3 over the same 200 held-out SVA evaluation questions for fixed Run 1. – denotes an undefined value due to a zero denominator.

Tool	Calling Mode	$N = 10$				$N = 40$				$N = 80$			
		p_R	$P(V\|b)$	$P(C\|b,V)$	$P(A\|b)$	p_R	$P(V\|b)$	$P(C\|b,V)$	$P(A\|b)$	p_R	$P(V\|b)$	$P(C\|b,V)$	$P(A\|b)$
Summary	JSON Transform	0.0	–	–	–	12.5	52.0	7.6	4.0	2.0	100.0	25.0	25.0
Retrieve	Shot ID	34.0	27.9	15.7	4.4	6.0	33.3	0.0	0.0	46.5	32.2	3.3	1.0
Temporal	Shot Keyframes	39.5	27.8	18.1	5.0	6.5	30.7	0.0	0.0	22.5	66.6	3.3	2.2
Temporal	Shot Detection	91.5	38.2	15.7	6.0	95.0	20.0	17.1	3.4	65.5	60.3	13.9	8.4
Summary	Evidence-to-Answer Script	46.5	75.2	15.7	11.8	75.5	25.1	21.0	5.3	52.0	75.9	13.9	10.5

Table 6: **Behavior-level changes under BAW on representative SVA behaviors.** Action-span support denotes behavior frequency over the 1,863 supervised planner action spans in the merged $N = 40$ training corpus. Coverage and $P(C \mid b)$ are measured over the same 200 held-out SVA evaluation trajectories for fixed Run 1.

Category	Calling Mode	Action-Span Support	BAW Weight	Coverage		$P(C \mid b)$	
				N=40	BAW	N=40	BAW
Temporal	Shot Detection	116/1863	0.57	190/200	198/200	22/190	31/198
Summary	Evidence-to-Answer Script	173/1863	0.50	151/200	132/200	23/151	29/132
Summary	JSON Transform	41/1863	0.97	25/200	20/200	1/25	8/20

pervision enables executable tool-use trajectories. After one evolution round, the performance gaps across larger supervision scales shrink substantially, indicating that supervision quantity has a stronger effect before bootstrap than after self-evolution. The same qualitative trend is observed with Qwen2.5-VL-7B-Instruct in Table 2, indicating that the bootstrap point is an empirical property of the evaluated model and environment rather than a universal trajectory threshold.

HERBench reveals a different scaling pattern after bootstrap. As shown in Table 3, one evolution round substantially improves all expert-initialized planners, yet increasing expert supervision produces only modest and non-monotonic post-evolution gains. Task-level results further show markedly different scaling trends across capabilities. Once executable self-generation has emerged, expert trajectory quantity alone provides an incomplete description of further improvement.

Together, LVB and HERBench reveal two empirically distinct regimes. Supervision quantity strongly affects whether executable self-generation emerges, whereas post-bootstrap improvement becomes diminishing and task-dependent. This motivates a finer-grained analysis of where post-bootstrap trajectories fail during generation, execution, and correct completion.

Table 7: **Comparison of supervision-weighting interventions on HERBench.** Results are averaged over three independent runs with different sampled training subsets. We report mean accuracy $\pm$ standard deviation across runs.

Intervention	AGAR	AGBI	AGLT	RLPC	SVA	TSO	Overall
Task Balance	48.0 ± 0.8	59.0 ± 1.0	45.0 ± 1.0	31.0 ± 0.8	11.5 ± 0.8	40.5 ± 0.5	39.1 ± 0.1
Tool-based Weighting	48.5 ± 1.3	57.5 ± 1.5	46.5 ± 1.4	34.0 ± 1.6	8.0 ± 1.8	42.0 ± 1.5	39.4 ± 1.4
Random Weighting	47.7 ± 1.2	58.5 ± 1.3	45.3 ± 1.2	30.5 ± 1.5	10.0 ± 2.0	39.8 ± 1.2	38.6 ± 1.4
BAW	47.5 ± 1.3	57.5 ± 1.5	46.0 ± 1.4	34.0 ± 1.5	15.5 ± 1.9	42.5 ± 1.2	40.5 ± 1.3

3.3 Post-Bootstrap Behavioral Characterization

We next characterize persistent post-bootstrap failures at the behavior level. We use SVA as a diagnostic case because its performance remains consistently low across supervision scales despite the overall gains from self-evolution. Table 5 decomposes representative behaviors across generation, valid execution, and correct completion. The resulting profiles are highly heterogeneous. Some behaviors appear only sparsely in generated trajectories, whereas others are frequently generated but fail primarily during execution or correct completion. Behavior frequency alone does not capture the structure of post-bootstrap failure. Different planner actions encounter bottlenecks at different stages of the trajectory.

We examine whether the affected capability is responsive to supervision at the same fine-grained action level. As shown in Table 6, BAW changes both the coverage and conditional correctness $P(C \mid b)$ of individual behaviors. At the task level, fine-grained BAW improves SVA accuracy and accuracy among answered examples, whereas coarse tool-level weighting and random weighting do

not provide comparable gains, as demonstrated in Table 7. These results show that the observed post-bootstrap limitation can be altered by modifying supervision at the same fine-grained level revealed by the stage-wise analysis. The stage-wise analysis and BAW together indicate that persistent failures are heterogeneous at the level of planner actions and that downstream performance is sensitive to how supervision is allocated across these actions.

4 Ablation Studies

4.1 Continued Training vs. Self-Evolution

We first ask whether the post-bootstrap improvement can be explained simply by continued optimization on the expert demonstrations. To isolate the effect of self-generated experience, both variants start from the same $N = 12$ initialization and use matched optimization settings and update counts. Continued training uses only the original expert trajectories, whereas self-evolution trains on accepted self-generated trajectories collected from the same bootstrap stage. As shown in Table 8, both methods improve over the initial planner, indicating that additional optimization accounts for part of the gain. However, the resulting successful trajectory distributions differ markedly. Only 6.58% of correct trajectories produced by continued training differ from the reference expert trajectory, compared with 46.25% after self-evolution. An alternative trajectory is defined as a correct solution whose tool category or calling mode differs from the corresponding expert trajectory. This result complements the scaling analysis in Sec. 3.2. Self-evolution introduces successful experience outside that pattern. We therefore view self-generated trajectories as changing the composition of successful learning experience rather than simply extending expert-only training.

Table 8: **Continued training vs. self-evolution from the same initialization.** Alt. Correct Traj. denotes correct trajectories that differ from the reference expert trajectory.

Setting	Answer Rate	Correct	Alt. Traj.	Alt. Ratio
$N = 12$	66.67%	58	–	–
Continue Train	84.03%	76	5	6.58%
Bootstrap Self-Evolve	95.14%	80	37	46.25%

4.2 Matched Optimization Budget

The fixed-epoch protocol naturally produces different numbers of optimizer updates across expert-supervision scales. We test whether the weaker performance of smaller expert sets is simply caused by receiving fewer optimization steps. Specifically, we match the $N = 10$ and $N = 40$ settings to the $1{,}415$ post-bootstrap retraining updates used by $N = 80$. As shown in Table 9, increasing the optimization budget for the smaller expert sets does not improve post-bootstrap accuracy and slightly reduces it in both cases. Thus, the diminishing and non-monotonic scaling observed in Sec. 3.2 cannot be explained solely by unequal optimization budgets. This control also clarifies the role of supervision quantity in our controlled setting. Repeated optimization on a limited, logic-controlled expert set mainly revisits the same demonstrated solution structures and does not introduce new training experience and additional optimization alone is insufficient to reproduce the gains associated with changes in the available experience.

Table 9: **Matched optimization-budget control.**

Setting	$N = 10$	$N = 40$	$N = 80$
Original Steps	475	780	1,415
Acc.	37.1%	39.1%	41.1%
Matched Steps	1,415	1,415	1,415
Acc.	36.3%	38.9%	41.1%

4.3 Trajectory Composition Interventions

We next examine whether post-bootstrap performance depends on the composition of expert trajectories beyond their amount or repeated exposure. *Rare-Exclude* replaces trajectories containing sparsely represented behaviors with same-task alternatives while preserving supervision scale and task balance. *SVA-Random* introduces additional randomly sampled SVA trajectories without explicitly targeting these behaviors, and *SVA-Oversampling* increases exposure to the existing SVA trajectories. *SVA-Enhanced* introduces new SVA trajectories enriched with sparsely represented behaviors identified from training-side expert and self-generated rollout statistics using the same behavior taxonomy. All settings follow the same fixed-epoch training protocol. As shown in Table 10, removing sparse-behavior support substantially degrades performance. Neither randomly adding SVA trajectories nor repeatedly exposing

Table 10: **Trajectory intervention comparison.**

Intervention	Answer Rate	Acc.	SVA Acc.
Task-Balance	91.1%	39.1%	11.5%
Rare-Exclude	65.0%	29.1%	5.1%
SVA-Random	90.2%	38.5%	10.3%
SVA-Oversampling	91.1%	38.8%	10.2%
SVA-Enhanced	93.2%	43.7%	19.8%

the model to existing SVA trajectories improves over the task-balanced baseline. In contrast, *SVA-Enhanced* improves overall accuracy from 39.1% to 43.7% and SVA accuracy from 11.5% to 19.8%. The contrast with *SVA-Random* is particularly informative. Introducing additional SVA experience alone does not reproduce the gain obtained when the added trajectories contain the identified sparse behaviors. These results provide stronger evidence that, in our controlled setting, post-bootstrap learning depends on trajectory quantity, exposure frequency, and on the behavioral composition of the available experience. For expert-data construction, increasing trajectory count alone may be less effective than ensuring coverage of task-relevant behaviors that are otherwise weakly represented.

5 Related Works

5.1 Self-Evolving and Tool-Augmented Agents

Self-evolving models improve through self-generated supervision and iterative refinement, with representative approaches including Self-Instruct Wang et al. (2023b), STaR Zelikman et al. (2022), ReST Gulcehre et al. (2023), Reflexion Shinn et al. (2023), and Self-Rewarding Language Models Yuan et al. (2024). Recent agent systems further explore experience-driven autonomous improvement and inter-task evolution Zhai et al. (2025); Wu et al. (2025); Qian et al. (2024); Zhou et al. (2025). EvoTool evolves modular tool-use policies through failure attribution and diversity-aware selection Yang et al. (2026). Tool-augmented agents extend model capabilities through external interaction and multi-step action execution Yao et al. (2023); Schick et al. (2023); Qin et al. (2024); Patil et al. (2023); Wang et al. (2023a). Most of these approaches study how already-capable agents improve from additional experience. In contrast, we focus on how executable self-evolution emerges from an initially unreliable planner and how its scaling behavior changes after bootstrap.

5.2 Experience Composition and Fine-Grained Tool Learning

Learning from expert trajectories is closely related to behavioral cloning Bain & Sammut (1995) and imitation learning Ross et al. (2011); Ho & Ermon (2016). Offline reinforcement learning similarly emphasizes the importance of data coverage and support for reliable policy improvement Levine et al. (2020); Kumar et al. (2020); Li et al. (2026b). Long-tailed learning studies how uneven data distributions affect model optimization Lin et al. (2017); Cui et al. (2019); Ren et al. (2020). Recent tool-learning methods further introduce finer-grained supervision, such as StepTool Yu et al. (2024) and SPORT Li et al. (2025). Our study focuses on behavior-level analysis and controlled interventions to characterize how post-bootstrap performance depends on the structure and composition of learning experience.

5.3 Agentic Long-Video Understanding

Long-video benchmarks Xiao et al. (2021); Mangalam et al. (2023); Fu et al. (2024) highlight the difficulty of reasoning over temporally distributed visual evidence. Agentic approaches address this challenge through adaptive evidence acquisition and tool use Wang et al. (2024); Fan et al. (2025); Li et al. (2026a); Zhang et al. (2026a). Recent work has also begun to explore self-evolution in visual and video reasoning. Video-Zero Zhang et al. (2026b) develops an annotation-free questioner–solver co-evolution framework centered on temporally grounded evidence. Whereas these methods primarily aim to improve self-evolving visual agents, our work uses tool-augmented VideoQA as a controlled environment to study the transition from bootstrap emergence to post-bootstrap scaling and to characterize the role of experience structure after executable self-generation has emerged.

6 Conclusion

We study bootstrap self-evolution from an initially unreliable tool-use planner. Our experiments reveal a transition from quantity-sensitive bootstrap emergence to diminishing and task-dependent first-round post-bootstrap scaling. Stage-wise analysis further shows heterogeneous behavioral bottlenecks, while controlled interventions indicate that post-bootstrap performance is sensitive to the composition and granularity of learning experience rather than trajectory quantity alone. These findings suggest that future expert-data construction may benefit from considering data scale, task balance, behavioral coverage, and solution patterns exposed to the agent. Our analysis is limited to manually defined behaviors, oracle verification, and controlled within-task solution logic.

REFERENCES

Michael Bain and Claude Sammut. A framework for behavioural cloning. *Machine Intelligence*, 1995.

Dan Ben-Ami, Gabriele Serussi, Kobi Cohen, and Chaim Baskin. Herbench: A benchmark for multi-evidence integration in video question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. Class-balanced loss based on effective number of samples. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

Sunqi Fan, Jiashuo Cui, Meng-Hao Guo, and Shuojin Yang. Tool-augmented spatiotemporal reasoning for streamlining video question answering task. In *Advances in Neural Information Processing Systems*, 2025.

Chaoyou Fu et al. Video-mme: The first-ever comprehensive evaluation benchmark of multi-modal llms in video analysis. *arXiv preprint arXiv:2405.21075*, 2024.

Caglar Gulcehre et al. Reinforced self-training (rest) for language model alignment. *arXiv preprint arXiv:2308.08998*, 2023.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, 2016.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *NeurIPS*, 2020.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Chenglin Li, Qianglong Chen, Feng Han, Yikun Wang, Xingxi Yin, Yan Gong, Ruilin Li, Yin Zhang, and Jiaqi Wang. Videothinker: Building agentic videollms with llm-guided tool reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2026a.

Pengxiang Li, Zhi Gao, Bofei Zhang, Yapeng Mi, Xiaojian Ma, Chenrui Shi, Tao Yuan, Yuwei Wu, Yunde Jia, Song-Chun Zhu, and Qing Li. Iterative trajectory exploration for multimodal agents. *arXiv preprint arXiv:2504.21561*, 2025.

Yibo Li, Jiashuo Yang, Zhi Zheng, Zhiyuan Hu, Yuan Sui, Shizun Wang, Yufei He, and Bryan Hooi. Apex: Autonomous policy exploration for self-evolving llm agents. *arXiv preprint arXiv:2605.21240*, 2026b.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *International Conference on Computer Vision*, 2017.

Karttikeya Mangalam, Amir Bar, Rohit Girdhar, et al. Egoschema: A diagnostic benchmark for very long-form video language understanding. *Advances in Neural Information Processing Systems*, 2023.

Shishir G Patil et al. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023.

Cheng Qian, Shihao Liang, Yujia Qin, Yining Ye, Xin Cong, Yankai Lin, Yesai Wu, Zhiyuan Liu, and Maosong Sun. Investigate-consolidate-exploit: A general strategy for inter-task agent self-evolution. *arXiv preprint arXiv:2401.13996*, 2024.

Yujia Qin et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *International Conference on Learning Representations*, 2024.

Jiawei Ren, Cunjun Yu, Xia Ma, et al. Balanced meta-softmax for long-tailed visual recognition. In *Advances in Neural Information Processing Systems*, 2020.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, et al. Toolformer: Language models can teach themselves to use tools. In *International Conference on Learning Representations*, 2023.

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*, 2023.

Guanzhi Wang, Yu Xie, Yunfan Jiang, et al. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023a.

Xiaohan Wang et al. Videoagent: Long-form video understanding with large language models. *European Conference on Computer Vision*, 2024.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, et al. Self-instruct: Aligning language models with self-generated instructions. In *ACL*, 2023b.

Haoyu Wu et al. Longvideobench: A benchmark for long-context video understanding. *arXiv preprint arXiv:2407.01505*, 2024.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. Evolver: Self-evolving llm agents through an experience-driven lifecycle, 2025.

Junbin Xiao, Xindi Shang, Angela Yao, and Tat-Seng Chua. Next-qa: Next phase of question answering to understand dynamic visual content. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

Shuo Yang, Caren Han, Xueqi Ma, Yan Li, Mohammad Reza Ghasemi Madani, and Eduard Hovy. Evotool: Self-evolving tool-use policy optimization in llm agents via blame-aware mutation and diversity-aware selection. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pp. 43553–43572, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, et al. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Yuanqing Yu, Zhefan Wang, Weizhi Ma, Zhicheng Guo, Jingtao Zhan, Shuai Wang, Chuhan Wu, Zhiqiang Guo, and Min Zhang. Steptool: A step-grained reinforcement learning framework for tool learning in llms. *arXiv preprint arXiv:2410.07745*, 2024.

Weizhe Yuan et al. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*, 2024.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, 2022.

Yunpeng Zhai, Shuchang Tao, Cheng Chen, Anni Zou, Ziqian Chen, Qingxu Fu, Shinji Mai, Li Yu, Jiaji Deng, Zouying Cao, Zhaoyang Liu, Bolin Ding, and Jingren Zhou. Agentevolver: Towards efficient self-evolving agent system, 2025.

Fan Zhang, Guangming Yao, Jinyang Wu, Hao Wu, Zheng Lian, Xinyu Geng, Jingdong Chen, Yi Yuan, and Pheng-Ann Heng. Videogaia: A benchmark for general ai assistants on agentic video understanding. *arXiv preprint arXiv:2608.14718*, 2026a.

Ruixu Zhang, Deyi Ji, Lanyun Zhu, Xuanyi Liu, Yuxin Meng, Ruihang Chu, and Yujiu Yang. Videozero: Self-evolution video understanding. *arXiv preprint arXiv:2605.14733*, 2026b.

Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen, Shuai Wang, Xiaohua Xu, Ningyu Zhang, et al. Symbolic learning enables self-evolving agents. 2024. *URL https://arxiv. org/abs/2406.18532*, 2025.

A Experimental Details and Additional Analyses

This appendix provides additional details on the tool environment, behavior taxonomy, prompt-only probing, expert-data construction, training and evaluation protocols, self-generated trajectory accounting, Behavior-Aware Weighting and the controlled ablations used in the main paper.

Unless otherwise specified, a *trajectory* denotes a complete planner interaction consisting of planner-generated actions, tool observations, and a terminal answer. A *training action span* denotes one supervised planner-generated action extracted from a training trajectory. Following Sec. 2.3, a planner behavior is defined by its *Tool Category* and *Calling Mode*.

A.1 Tool Environment and Behavior Taxonomy

A.1.1 Tool Runtime

The planner interacts with a typed tool runtime. At each step, it emits either one serialized tool call,

```
<tool_call>{"tool":"...","arguments":{...}}</tool_call>
```

or a terminal answer,

```
<final_answer>{
  "answer":...,
  "answer_index":...,
  "answer_text":...,
  "evidence":...
}</final_answer>
```

Downstream calls may consume only fields explicitly exposed by previous tool observations, referenced through expressions of the form `tool_result:<call_id>.<field>`. Tool observations and frozen-worker outputs are inserted into the planner context but are masked from the supervised training loss. Only planner-generated action tokens are optimized. The tool environment remains fixed throughout expert initialization, self-generation, and subsequent self-evolution.

A.1.2 Tool Categories and Calling Modes

We define each planner behavior as

$$b = (Tool_Category, Calling_Mode).$$

This taxonomy separates the semantic role of an action from its implementation function. Different calling modes of the same implementation are therefore treated as different behaviors. Calling modes with explicitly different request specifications are also kept separate.

Final-answer generation is included in the same taxonomy as *Final* + *Final Answer*.

Table 11 maps the paper-level taxonomy to the underlying implementation interfaces.

A.2 Prompt-only Capability Probing

We use two prompt-only conditions to probe whether the pretrained planner can compose the available tools before any trajectory-level fine-tuning. The two conditions use exactly the same frozen system prompt and differ only in the information supplied through the user prompt.

The demonstration-based prompt contains 12,309 characters and two complete end-to-end trajectories. The tool-card prompt contains 7,267 characters and six independent interface cards.

A.2.1 Shared System Prompt

The following system instruction is identical in both conditions:

Table 11: **Tool categories, calling modes, and implementation interfaces.** The paper-level behavior taxonomy is defined by Tool Category + Calling Mode; implementation names are provided only to make the executable environment explicit.

Tool Category	Calling Mode	Implementation	Primary Function
Temporal	Shot Detection	`detect_shots`	segment a video into temporal shots
Temporal	Timestamp Extraction	`extract_frames(mode=timestamps)`	extract frames at specified timestamps
Temporal	Shot Keyframes	`extract_frames(mode=shot_keyframes)`	extract representative frames from detected shots
Temporal	Window Extraction	`extract_frames(mode=window)`	extract frames from a localized temporal window
Retrieve	Track ID	`retrieve_visual_text(group_by=track_id)`	retrieve visual evidence grouped by tracked entity
Retrieve	Shot ID	`retrieve_visual_text(group_by=shot_id)`	retrieve visual evidence grouped by temporal shot
Grounding	Appearance Target	`track_people(preset=appearance_target)`	ground and track a target person by appearance
Grounding	Region Count	`track_people(preset=region_count)`	track people for region-based counting
OCR/ASR	Subtitle Localization	`subtitle_locate_core`	localize subtitle-relevant temporal regions
OCR/ASR	Subtitle Evidence	`collect_subtitle_evidence`	collect localized subtitle evidence
Summary	Target Track and Timestamps	`rule_based_processor`	deterministically select a target track and timestamps
Summary	Choice Normalization	`rule_based_processor`	normalize an answer into the required choice format
Summary	RLPC Choice Mapping	`rule_based_processor`	map a deterministic region count to an answer choice
Summary	TSO Permutation Mapping	`rule_based_processor`	map a temporal permutation to an answer choice
Summary	Evidence-to-Answer Script	`generate_evidence_to_answer_script`	synthesize an answer from available evidence
Summary	JSON Transform	`generate_json_transform_script`	transform structured evidence into the required JSON form
Final	Final Answer	`final_answer`	terminate the trajectory with the predicted answer

```
You are a video-question tool-use planner.
Return exactly one <tool_call>{"tool":"...","arguments":{...}}</tool_call>
or one <final_answer>{"answer":...,"answer_index":...,
"answer_text":...,"evidence":...}</final_answer>.
Only reference visible outputs as tool_result:<call_id>.<field>.
Use tool arguments exactly according to the demonstrated interfaces.
Do not invent argument values or use one tool's output as an
incompatible input to another tool.
A window extraction requires a localization result that visibly
contains a window; a detect_shots result alone is not a window
localization.
When a tool observation contains a complete decision with answer,
answer_index, and answer_text, your next response must be
<final_answer> using that decision. Never call the answer-generation
tool again after a complete decision is visible.
```

Thus, both probing conditions share the same output protocol, dependency rules, interface constraints, and termination criterion.

A.2.2 Demonstration-Based Prompt

The demonstration-based condition provides two complete multi-step trajectories and asks the planner to imitate their tool planning, explicit result references, and answer format.

A shortened representation of the corresponding user prompt is shown below.

```
Complete multi-step demonstrations. Imitate their tool planning,
explicit result references, and answer format.

<demo>
TASK {... subtitle-grounded question ...}

detect_shots
-> subtitle_locate_core(
     shots_doc=tool_result:detect_shots#1)
-> extract_frames(
     mode=window,
     localization=tool_result:subtitle_locate_core#2,
     shots_doc=tool_result:detect_shots#1)
-> collect_subtitle_evidence(
     localization=tool_result:subtitle_locate_core#2,
     frames_doc=tool_result:extract_frames#3)
-> generate_evidence_to_answer_script(
```

```
     available_evidence={...})
-> final_answer
</demo>

<demo>
TASK {... visual question ...}

detect_shots
-> extract_frames(
     mode=shot_keyframes,
     shots_doc=tool_result:detect_shots#1,
     shots_ref=tool_result:detect_shots#1.shots)
-> retrieve_visual_text(
     images_ref=tool_result:extract_frames#2.frames,
     aggregation={group_by: shot_id, method: max})
-> generate_evidence_to_answer_script(
     available_evidence={...})
-> final_answer
</demo>

For every target task, return only the next planner response.
```

The two demonstrations expose complete executable compositions rather than isolated API examples. They demonstrate action ordering, explicit `call_id` dependencies, evidence propagation, and the transition from a complete intermediate decision to the terminal answer.

The source trajectories are `LVB:fryyNwUCPWA_1:47:sample_3037` and `LVB:d5JlCEDlHGE_1:71:sample_1801`.

A.2.3 TOOL-CARD PROMPT

The tool-card condition uses the same system prompt but provides no complete task trajectory. The user prompt begins with:

```
Independent tool interfaces only. They are not an end-to-end plan;
choose and compose tools yourself.
```

Each card specifies the input schema, output schema, calling mode, and valid downstream references. For example:

```
<tool name=extract_frames>
call={
  "tool":"extract_frames",
  "arguments":{
    "video_path":"...",
    "mode":"shot_keyframes",
    "shots_doc":"tool_result:detect_shots#1",
    "shots_ref":"tool_result:detect_shots#1.shots",
    "frames_per_shot":1,
    "include_images":true,
    "output_paths":true,
    "backend_profile":"videotool"
  }
}
output={
  "status":"str",
  "call_id":"str",
  "root_ref":"str",
  "fields":"object"
}
```

```
next_refs=["tool_result:extract_frames#2.frames"]
</tool>
```

Unlike the demonstration-based condition, the cards expose only individual interfaces. The planner must independently infer the action order, maintain cross-call dependencies, determine evidence sufficiency, and decide when to terminate.

A.2.4 PROMPT-PROBING RESULTS

Both conditions are evaluated on the same 48-example LVB probing set with one rollout per example. The demonstration-based prompt produces 20 tool errors and 28 repeated-tool-call failures. The tool-card prompt produces 48 tool errors. Complete demonstrations provide substantially stronger procedural structure for tool composition than isolated interface descriptions, but prompting alone is insufficient to produce reliable closed-loop execution in this environment.

A.3 DATASET CONSTRUCTION AND EXPERIMENTAL PROTOCOL

A.3.1 EXPERT TRAJECTORY CONSTRUCTION

Expert trajectories are constructed from training-side benchmark questions that are disjoint from both the rollout and final-evaluation sets. For each task, we first define a controlled principal solution structure using the executable Tool Category + Calling Mode interfaces described in Sec. A.1. The task-specific structures are summarized in Table 12.

Each expert trajectory is then instantiated by executing the prescribed planner actions in the same frozen tool environment used during autonomous rollout. Tool observations are obtained from actual tool execution rather than manually inserted intermediate results. The resulting interaction, including planner actions, tool observations, and the terminal answer, is recorded as a complete trajectory. Before inclusion in the expert set, trajectories are checked for executable tool dependencies and final-answer correctness.

This construction intentionally controls the principal solution logic within each task rather than introducing multiple manually designed strategies. It therefore provides a consistent initialization distribution for studying how executable behaviors and alternative solution patterns emerge during subsequent self-evolution. Expert examples are subsequently sampled using the nested and task-balanced protocol described below.

A.3.2 QUESTION-LEVEL SPLITS AND LABEL USAGE

Expert initialization, autonomous rollout, and final evaluation use disjoint sets at the question level.

For LVB, self-generated training trajectories are collected from a 96-question rollout pool. Final performance is evaluated on a separate fixed 144-question pool. Ground-truth answers from rollout questions are accessed only for verification-based filtering of generated trajectories.

For HERBench, each supervision condition uses a 600-question rollout pool, containing 100 questions from each of the six tasks. Final evaluation uses a fixed, disjoint set of 1,200 questions, with 200 examples per task.

Rollout ground-truth answers are never exposed to the planner, frozen multimodal worker, or tool runtime. Final-evaluation questions are never used as training examples.

LVB uses nested task-balanced expert subsets,

$$D_4 \subset D_{12} \subset D_{16} \subset D_{20} \subset D_{24},$$

where N denotes the number of expert trajectories per task.

For the Qwen2.5-VL-7B-Instruct control, we additionally evaluate $N = 28$ under the same nested task-balanced construction, extending the sequence with

$$D_{24} \subset D_{28}.$$

Table 12: **Controlled within-task solution structures on HERBench.** Each action is represented using the paper-level Tool Category + Calling Mode taxonomy.

Task	Controlled Behavior Sequence	Decision Structure
AGAR / AGBI / AGLT	Grounding [Appearance Target] → Retrieve [Track ID] → Summary [Target Track and Timestamps] → Temporal [Timestamp Extraction] → Summary [Evidence-to-Answer Script] → Final [Final Answer]	identify candidate tracks, select the target and temporal evidence, and answer from the selected frames
RLPC	Grounding [Region Count] → Summary [RLPC Choice Mapping] → Final [Final Answer]	count tracks satisfying the region criterion and deterministically map the result to an answer choice
SVA	Temporal [Shot Detection] → Temporal [Shot Keyframes] → Retrieve [Shot ID] → Summary [Evidence-to-Answer Script] → Final [Final Answer]	identify relevant scenes and reason over their shot-level arrangement
TSO	Temporal [Shot Detection] → Temporal [Shot Keyframes] → Retrieve [Shot ID] → Summary [TSO Permutation Mapping] → Final [Final Answer]	associate scenes with temporal shots, sort them, and map the permutation to an answer choice

HERBench similarly uses

$$D_{10} \subset D_{20} \subset D_{30} \subset D_{40} \subset D_{80}.$$

Within each supervision scale, every task contributes the same number of expert trajectories. Larger expert sets contain the smaller sets rather than being sampled independently.

A.3.3 Independent Runs

The main scaling results reported as mean and standard deviation are obtained from three independent runs. Within each run, expert subsets obey the same task-balancing and nesting rules. Across runs, expert subsets are independently sampled, LoRA training is rerun from a new initialization, and rollout generation is performed independently.

Each rollout question produces one planner trajectory per run. Whenever the Appendix reports exact count-level accounting from a single run, we report Run 1 under a fixed run ordering. Run 1 is not selected according to evaluation performance. The table reports the principal evidence-acquisition structure and omits optional answer-normalization actions.

A.3.4 Controlled Within-Task Solution Logic

Expert trajectories follow a controlled principal solution structure within each task. This reduces arbitrary within-task strategy variation while preserving differences across task families. The controlled construction does not require self-generated trajectories to exactly reproduce these expert paths. Rather, it provides a stable reference distribution for studying how solution patterns change during self-evolution.

A.4 Training and Evaluation Details

A.4.1 Planner Training

The planner is initialized from Qwen3.5-9B and optimized using LoRA with rank $r = 16$ and scaling factor $\alpha = 32$. Unless otherwise specified, training uses 5 epochs with a learning rate of 2×10^{-5} on 4 A800 GPUs. The per-device batch size is 1 with gradient accumulation over 3 steps, yielding an effective batch size of 12. The maximum sequence length is 32,768 tokens. Training uses bfloat16 precision with SDPA attention and TF32 enabled.

The multimodal worker and all tools remain frozen throughout training and self-evolution. Each generated trajectory is limited to at most 12 tool calls.

A.4.2 Evaluation Metrics

For a rollout or evaluation set of size N, we report:

- **Final Answer**: number of trajectories that terminate with a parseable final answer;
- **Answer Rate**: $\mathrm{FinalAnswer}/N$;
- **Correct**: number of trajectories whose normalized final answer matches the ground-truth answer;
- **Accuracy**: $\mathrm{Correct}/N$;
- **Answered Accuracy**: $\mathrm{Correct}/\mathrm{FinalAnswer}$.

Accuracy therefore always uses the full evaluation or rollout set as its denominator.

A.4.3 Failure and Acceptance Definitions

An *invalid generation* is a malformed or protocol-incompatible planner emission, including malformed tool-call or final-answer serialization.

A *tool error* occurs when a syntactically valid planner action reaches the tool layer but cannot be executed successfully.

An *exception* denotes an unrecovered runtime or infrastructure failure recorded separately from invalid generation and tool error.

A generated trajectory is *correct* when its normalized final answer matches the rollout ground truth. A trajectory used for self-training must satisfy the validity, correctness, and trainability criteria in Eq. (4) of the main paper.

For count-level analysis, we further distinguish two structural subtypes among accepted trajectories:

- **Strict accepted**: the trajectory is executable and correct, and additionally satisfies the predefined canonical task structure, dependency constraints, and replay requirements;
- **Relaxed accepted**: the trajectory is executable and correct but deviates from one or more canonical structural requirements while remaining trainable under the disclosed relaxed criterion.

Both types therefore satisfy

$$\mathrm{Valid}(\tau) = 1, \qquad \mathrm{Correct}(\tau) = 1.$$

Relaxed admission preserves the planner-generated trajectory as produced. Missing intermediate actions are not inserted after generation.

A.5 Self-Generated Trajectory Accounting

A.5.1 Rollout-to-Training Funnel

We distinguish three stages in the construction of self-generated training data:

$$N_{\mathrm{generated}} \supseteq N_{\mathrm{usable\text{-}correct}} \supseteq N_{\mathrm{added}}.$$

Generated. $N_{\mathrm{generated}}$ contains all requested rollout trajectories, including invalid generations, tool errors, exceptions, and incorrect answers.

Usable-Correct. $N_{\mathrm{usable\text{-}correct}}$ contains executable and correct trajectories that can be converted into trainable planner trajectories under the strict or relaxed structural criterion.

Added. N_{added} contains the usable-correct trajectories actually merged into the next-round training corpus.

Table 13: **Run-1 Round-1 rollout accounting on LVB for Qwen3.5-9B.** Each setting generates 96 rollout trajectories.

Setting	Expert Traj.	Usable & Correct	Added	Training Set
$N = 4$	48	0	0	$48 \rightarrow 48$
$N = 12$	144	38	38	$144 \rightarrow 182$
$N = 16$	192	43	43	$192 \rightarrow 235$
$N = 20$	240	44	44	$240 \rightarrow 284$
$N = 24$	288	48	48	$288 \rightarrow 336$

Table 14: **Run-1 Round-1 self-generated trajectory accounting on HERBench.** Each setting generates 600 trajectories. Strict and relaxed denote structural subtypes of executable and correct trajectories; both satisfy the acceptance criterion in Eq. (4).

Setting	Correct	Accepted		Not Added	Added
		Strict	Relaxed		
$N = 10$	153	108	45	0	153
$N = 20$	192	88	103	1	191
$N = 30$	150	63	87	0	150
$N = 40$	112	70	42	0	112
$N = 80$	183	148	30	5	178

A.5.2 LVB Rollout Accounting

Table 13 reports exact Run 1 accounting for the first LVB self-evolution round. Run 1 is fixed independently of evaluation performance; the main results are averaged across three independent runs.

The corresponding numbers of newly added planner action-span SFT targets are 0, 197, 223, 233, and 256.

For the Qwen2.5-VL-7B-Instruct control at the 144-trajectory initialization scale, 96 rollout trajectories are generated, 8 are usable-correct, and all 8 are added, producing a $144 \rightarrow 152$ trajectory merge.

A.5.3 HERBench Rollout Accounting

Each HERBench supervision scale generates 600 Round-1 trajectories. Table 14 reports exact Run 1 accounting. The $N = 80$ Run 1 merged training corpus contains $480 + 148 + 30 = 658$ trajectories.

A.5.4 Task-Level HERBench Admission

Each HERBench task contributes 100 rollout questions at every supervision scale. Table 15 reports *Correct / Strict Accepted / Relaxed Accepted / Added*.

At $N = 20$, one correct SVA trajectory is not admitted, resulting in six correct but five added SVA trajectories. Despite the balanced rollout budget, the resulting self-generated training data are not task-balanced. SVA contributes only 5, 5, 2, 5, and 0 added trajectories across the five supervision scales. Task-balanced generation therefore does not imply task-balanced self-training after validity and correctness filtering.

A.5.5 Example of Relaxed Structural Admission

The following AGAR trajectories illustrate the difference between a canonical expert path and an executable, correct trajectory admitted under the relaxed structural criterion.

Canonical expert trajectory.

Table 15: **Run-1 task-level Round-1 trajectory admission on HERBench.** Each task receives 100 rollout questions. Cells report: Correct / Strict Accepted / Relaxed Accepted / Added.

Expert Scale	AGAR	AGBI	AGLT	RLPC	SVA	TSO
$N = 10$	25/10/15/25	34/19/15/34	39/34/5/39	23/21/2/23	5/2/3/5	27/22/5/27
$N = 20$	32/1/31/32	49/3/46/49	42/16/26/42	27/27/0/27	6/5/0/5	36/36/0/36
$N = 30$	34/3/31/34	42/6/36/42	30/15/15/30	15/15/0/15	2/2/0/2	27/22/5/27
$N = 40$	9/0/9/9	14/1/13/14	17/3/14/17	35/35/0/35	5/1/4/5	32/30/2/32
$N = 80$	30/15/15/30	30/18/12/30	46/43/3/46	39/39/0/39	0/0/0/0	38/33/0/33

```
track_people(preset=appearance_target)
-> retrieve_visual_text(aggregation.group_by=track_id)
-> rule_based_processor(
     operation=select_target_track_and_timestamps)
-> extract_frames(
     mode=timestamps,
     timestamps_sec=<selector output>)
-> generate_evidence_to_answer_script(
     request_type=evidence_to_answer)
-> rule_based_processor(
     operation=normalize_choice_answer)
-> final_answer
```

Relaxed self-generated trajectory.

```
track_people(preset=appearance_target)
-> retrieve_visual_text(aggregation.group_by=track_id)
-> extract_frames(
     mode=timestamps,
     timestamps_sec=[61,81,101,121,141,161])
-> generate_evidence_to_answer_script(
     request_type=evidence_to_answer)
-> final_answer
```

The generated path is executable and produces the correct answer, but omits the canonical target-selection and answer-normalization actions. Relaxed admission keeps the generated trajectory unchanged rather than inserting these actions post hoc.

A.6 Behavior-Aware Weighting

A.6.1 Training Support and Objective

BAW is applied to the merged $N = 40$ training corpus consisting of the original expert trajectories and accepted self-generated trajectories. Accordingly, $f(b)$ denotes the number of supervised planner action spans assigned to behavior b in this *merged training set*, rather than in the expert set alone.

For action span i, let $\mathcal{S}_i$ denote the token positions corresponding to one serialized planner-generated action. Tool observations, worker outputs, task context, and padding are masked from the supervised loss.

For behavior b, we first compute

$$\tilde{w}(b) = f(b)^{-1/2}.$$

The raw weights are normalized by their arithmetic mean over the observed behavior set $\mathcal{B}$,

Table 16: **Run-1 BAW action-span support and weights in the $N = 40$ merged training set.** Support is measured over 1,863 supervised planner action spans. Explicit and default Summary calls are treated as different calling modes. The two Retrieve modes are shown as an archived API-level aggregate because their exact count summary stores the shared implementation name.

Tool Category	Calling Mode	$f(b)$	$f(b)^{-1/2}$	Weight
Final	*Final_Answer*	352	0.0533	0.5000
Retrieve	*Track_ID / Shot_ID*	273	0.0605	0.5000
Temporal	*Timestamp_Extraction*	185	0.0735	0.5000
Summary	*Evidence-to-Answer Script*	173	0.0760	0.5
Grounding	*Appearance_Target*	160	0.0791	0.5000
Summary	*Target_Track_and_Timestamps*	124	0.0898	0.5585
Temporal	*Shot_Detection*	116	0.0928	0.5775
Temporal	*Shot_Keyframes*	92	0.1043	0.6484
Summary	*Choice_Normalization*	92	0.1043	0.6484
Summary	*Choice_Mapping*	75	0.1155	0.7182
Grounding	*Region_Count*	75	0.1155	0.7182
Summary	*Permutation_Mapping*	70	0.1195	0.7434
Summary	*JSON_Transform*	41	0.1562	0.9713

$$\bar{w} = \frac{1}{|\mathcal{B}|} \sum_{b \in \mathcal{B}} \tilde{w}(b),$$

and then clipped:

$$w(b) = \mathrm{clip}\left(\frac{\tilde{w}(b)}{\bar{w}}, 0.5, 2.0 \right).$$

The behavior-aware objective is

$$\mathcal{L}_{\mathrm{BAW}} = -\sum_{i} w(b_i) \sum_{t \in \mathcal{S}_i} \log p_\theta \left(y_t \mid y_{<t}, x \right).$$

All supervised tokens in the same planner action span receive the same behavior weight. No additional action-length or trajectory-level normalization is applied. Weight normalization is performed before clipping.

BAW is used as a diagnostic intervention rather than proposed as a general optimization method. Its purpose is to test whether changing the allocation of training signal across fine-grained planner actions alters the affected capability. The inverse-square-root rule and clipping interval are fixed design choices for constructing a bounded diagnostic perturbation rather than hyperparameters optimized for downstream accuracy. We do not claim that these choices are optimal. Sensitivity to alternative weighting functions remains outside the scope of this study.

A.6.2 Action-Span Support and Weights

The archived $N = 40$ merged training set contains 1,863 supervised planner action spans. Table 16 reports the exact Run 1 frequency summary and final weights.

The paper-level taxonomy distinguishes Retrieve [Track ID] from Retrieve [Shot ID]. The archived count summary used for exact accounting stores the shared `retrieve_visual_text` implementation name and therefore reports their 273 action spans in aggregate. All stage-wise behavior analyses distinguish the two retrieval modes from the structured `aggregation.group_by` argument.

A.6.3 Run-Level BAW Evaluation Accounting

For exact count-level accounting, we report Run 1. The run is fixed independently of evaluation performance. On the fixed 1,200-example HERBench evaluation set, BAW produces 1,101 final

answers and 486 correct answers, corresponding to a 91.75% answer rate, 40.50% accuracy, and 44.14% answered accuracy. The corresponding $N = 40$ task-balanced baseline produces 1,094 final answers and 470 correct answers, corresponding to a 91.17% answer rate, 39.17% accuracy, and 42.96% answered accuracy.

A.7 Additional Ablation Protocols

A.7.1 Continued Training versus Self-Evolution

The continued-training control and self-evolution condition start from the same $N = 12$ expert-initialized checkpoint and use matched optimizer settings and update counts.

Continued training performs additional optimization using only the original expert set. Self-evolution instead trains on accepted self-generated trajectories collected from the corresponding bootstrap stage together with the expert supervision.

For the trajectory-diversity analysis in the main paper, a correct trajectory is classified as an *alternative trajectory* when its sequence contains a Tool Category or Calling Mode that differs from the corresponding reference expert trajectory. Thus, the comparison is performed at the same Tool Category + Calling Mode granularity used throughout the behavior analysis.

A.7.2 Matched Optimization Budget

The standard fixed-epoch protocol naturally produces different numbers of optimizer updates across supervision scales. To separate supervision scale from optimization budget, we additionally train the $N = 10$ and $N = 40$ conditions using the 1,415 expert-initialization updates used by $N = 80$.

This control changes only the number of optimization updates. It does not add new trajectories or alter the composition of the expert set. The resulting models are subsequently evaluated under the same self-evolution and evaluation protocol used in the corresponding main experiment.

A.7.3 Trajectory-Composition Interventions

We construct four interventions from the task-balanced $N = 40$ expert configuration to distinguish trajectory quantity, repeated exposure, and behavioral composition.

All interventions are applied at the *expert-supervision stage*. The modified expert data are first used for planner initialization, after which the same rollout, verification, self-evolution, and final-evaluation pipeline is executed. The interventions therefore modify the experience available before bootstrap rather than directly editing the post-bootstrap merged training corpus.

All variants follow the same fixed-epoch training protocol.

Rare-Exclude. Rare-Exclude starts from the task-balanced $N = 40$ expert set and replaces 28 trajectories containing the identified sparsely represented behaviors with same-task replacement trajectories. The replacement preserves both the total number of expert trajectories and the number of trajectories per task.

The replacements comprise 7 AGAR, 6 AGBI, 7 RLPC, and 8 SVA trajectories; AGLT and TSO are unchanged. Rare-Exclude therefore modifies behavioral composition while keeping the overall supervision scale and task balance fixed.

SVA-Random. SVA-Random retains the original 40 SVA expert trajectories and introduces 20 newly sampled SVA expert trajectories. The new questions are disjoint from both the self-generation rollout pool and the final evaluation set and are sampled without explicitly targeting the sparse behaviors identified from training-side statistics.

The resulting expert set therefore contains 60 unique SVA trajectories.

SVA-Oversampling. SVA-Oversampling duplicates the original 40 SVA expert trajectories. It therefore increases SVA training exposure without introducing new questions, new unique trajectories, or additional behavioral support.

Table 17: **Task-level construction of Rare-Exclude.** The intervention starts from the $N = 40$ expert set and replaces 28 trajectories while preserving 40 trajectories per task.

Task	Replaced	Replacement
AGAR	7	7
AGBI	6	6
AGLT	0	0
RLPC	7	7
SVA	8	8
TSO	0	0
Total	**28**	**28**

Table 18: **Construction of the trajectory-composition interventions.** Unique SVA denotes distinct SVA expert trajectories. Training instances count duplicated trajectories when oversampling is used.

Setting	Unique SVA	SVA Training Instances	Total Expert Instances	Intervention
Task-Balance	40	40	240	original task-balanced $N = 40$ supervision
Rare-Exclude	40	40	240	replace 28 trajectories containing targeted sparse behaviors
SVA-Random	60	60	260	add 20 new untargeted SVA trajectories
SVA-Oversampling	40	80	280	duplicate the original 40 SVA trajectories
SVA-Enhanced	60	60	260	add 20 new SVA trajectories with targeted behavioral composition

The condition contains 40 unique SVA trajectories but 80 SVA training instances after duplication.

SVA-Enhanced. SVA-Enhanced retains the original 40 SVA expert trajectories and introduces 20 additional SVA expert trajectories constructed to contain task-relevant behaviors that are sparsely represented in the training-side expert and self-generated rollout statistics.

The resulting expert set contains 60 unique SVA trajectories. In contrast to SVA-Random, the additional supervision is therefore selected according to behavioral composition rather than only task identity.

Comparison of intervention designs. Table 18 summarizes the controlled differences among the trajectory-composition variants.

These controls separate four distinct changes to the available training experience: reducing sparse-behavior support while holding scale fixed, acquiring untargeted new experience, repeatedly exposing the planner to existing experience, and acquiring new experience with targeted behavioral composition.

A.8 Evaluation-Set Usage and Intervention Construction

The behavior taxonomy is defined from the executable tool interfaces and structured planner actions rather than from final-evaluation outcomes.

The final evaluation set is used only to report performance and behavior statistics. The sparse behaviors targeted by BAW and SVA-Enhanced are identified exclusively from training-side expert and self-generated rollout statistics.

In particular, no final-evaluation outcome is used to

- define the Tool Category + Calling Mode taxonomy;
- select the behaviors targeted by SVA-Enhanced;
- select the additional SVA-Enhanced expert examples;
- determine the inverse-square-root BAW weighting rule; or
- tune the BAW clipping interval $[0.5, 2.0]$.

The fixed 1,200-example HERBench evaluation set is used only for final reporting and for descriptive behavior analysis under the same validity and correctness criteria defined in Sec. 2.3. When

$P(A \mid b)$ is reported on held-out evaluation trajectories, it denotes whether a trajectory would satisfy the validity-and-correctness verification gate. It does not denote actual inclusion of that evaluation trajectory in the training set.

AI USE STATEMENT

Generative AI tools were used to assist with language editing and improving the clarity and readability of the manuscript. They were not used to generate experimental data, execute experiments, or determine reported experimental results. All technical content, experimental results, and final claims were reviewed and verified by the authors, who take full responsibility for the content of the paper.