TOWARD TRUSTWORTHY SELF-EVOLUTION: PRESERVING SAFETY VIA GEOMETRIC ORTHOGONALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-evolving agents generate their own tasks and derive training signal from their own interaction, enabling capability to grow beyond what human-curated data can specify. This autonomy incurs a cost: safety degrades across successive rounds even as task accuracy improves, a failure mode termed misevolution. We show that this degradation is geometric in origin. Although computed entirely from benign tasks, the evolutionary gradient intrudes into a safety subspace of the solver's parameter space; moreover, as the curriculum co-evolves with the model, the direction of this intrusion shifts unpredictably across rounds, affording no stable target for data-level or objective-level correction. We therefore reframe safety preservation as a constraint on optimization geometry and propose SEGO, which orthogonalizes the evolutionary gradient against a safety subspace identified from model representations and updated as the model drifts, permitting unconstrained capability growth while the update carries no component within that subspace. Since the proposer influences the solver solely through its gradient, constraining the solver alone is sufficient, leaving the co-evolutionary dynamic intact. Across multiple self-evolving frameworks and models, SEGO preserves the model's safety throughout evolution while matching the capability growth of unconstrained evolution. Our analysis further establishes the first empirical evidence that safety-critical and capability-driving directions remain separable under self-evolution. Source code is attached here [1].

1 INTRODUCTION

The training of large language models has increasingly shifted from static text corpora toward reinforcement training on tasks with verifiable outcomes, producing agents that decompose problems, reason over long horizons, and improve from outcome feedback (Guo et al., 2025; Wen et al., 2026). Such training commonly depends on a supply of tasks whose difficulty matches the current model and whose outcomes can be verified. Curating this supply is expensive, requiring domain expertise to design tasks and reliable verification for every instance (Zhao et al., 2026a). More fundamentally, human-constructed task suites are static in difficulty, whereas an agent's capability frontier moves throughout training; the complexity and diversity of tasks that humans can specify are bounded by the constructor's own knowledge and budget, so such data eventually ceases to provide an effective learning signal (Huang et al., 2026). Self-evolving agents have emerged in response to this limitation: rather than consuming an externally curated dataset, the system generates its own tasks and trajectories and derives training signal from its own interaction (Gao et al., 2025). The significance of this paradigm therefore lies not merely in reducing annotation cost, but in allowing capability to grow beyond what human-constructed data can specify.

Specifically, in self-evolving agents one component proposes tasks and another attempts to solve them, and the resulting interaction supplies the training signal, with verifiers establishing whether an outcome is correct (Huang et al., 2026). Because the proposing component is updated alongside the solving one, the task distribution co-evolves with the model, so difficulty tracks the capability frontier (Zhao et al., 2026a). Recent studies, however, report that this autonomy carries a cost: safety

[1] https://figshare.com/s/7c7919230c49bcee0790

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.

capabilities degrade over successive rounds even as task accuracy rises, a failure mode recently termed misevolution (Shao et al., 2026). What sets this apart from established alignment concerns is structural rather than incidental. The degradation is endogenous, arising from routine evolution on benign tasks rather than from an adversary or from corrupted data (Qi et al., 2024). And because each round starts from the model produced by the previous one, drift is inherited rather than corrected, with no mechanism inside the system that pulls it back (Guo et al., 2026). What is therefore in question is not whether a particular self-evolving system is safe, but whether the paradigm can be adopted as a trustworthy training regime at all.

Safety is conventionally enforced by curating training data and shaping objectives, but under self-evolution the data is self-generated and the process is a closed game. Any attempt to curate that data or shape the proposer and verifier against unsafe behavior reintroduces external human specification, imposing the biases and coverage limitations of human judgment while disrupting the co-evolution that drives capability (Lindström et al., 2024). A second class of methods intervenes outside the evolutionary process itself: realigning the model after evolution through preference optimization or safety fine-tuning, or constraining it at inference through prompt-level guards or memory-based safety cues. Post-hoc realignment incurs an alignment tax that diminishes the capabilities evolution produced, whereas inference-time wrappers act only on the model's outputs and leave its intrinsic, eroding safety capability unaddressed, while further being reported to induce over-refusal (Huang et al., 2025; Röttger et al., 2024). For self-evolution to serve as a trustworthy training regime, safety must not be sacrificed as a cost of capability growth, nor preserved at the expense of the autonomy and co-evolutionary dynamic that justify the paradigm.

To this end, we propose SEGO (Safety-preserving Evolution via Geometric Orthogonalization), which acts directly on the evolutionary update rather than on the data or the outputs, making safety preservation intrinsic to the evolution itself. Concretely, SEGO exploits the geometric structure of the evolutionary gradient: even under the shifting gradient distributions of self-evolution, the directions driving capability growth and those encoding safety occupy separable subspaces in the solver's parameter space, enabling the evolutionary update to be geometrically constrained to bypass safety-critical parameters. SEGO formulates the proposer's influence as passing entirely through the solver's gradient, so that orthogonalizing this gradient alone suffices, leaving the co-evolutionary dynamic unconstrained while adaptively tracking representational drift as the model evolves. Experiments on self-evolving frameworks including AZR and R-Zero show that SEGO preserves safety throughout evolution while matching the capability growth of standard self-evolution.

Our contributions are threefold. First, we identify why misevolution resists conventional safeguards: safety erodes not because the self-generated curriculum contains harmful content, but because the evolutionary gradient intrudes into the safety subspace, and this intrusion shifts unpredictably from round to round as the curriculum co-evolves, leaving no stable target for data-level or objective-level correction. Second, we reframe safety preservation under self-evolution as a constraint on optimization geometry and propose SEGO, which orthogonalizes the evolutionary gradient against the safety subspace so that capability grows freely while the update direction carries no component within that subspace. Because the proposer acts on the solver only through its gradient, SEGO decouples safety from the proposer's state without constraining the co-evolutionary dynamic, and tracks representational drift as the model evolves. Third, across multiple frameworks and models, SEGO preserves safety while matching the capability growth of unconstrained evolution, and our analysis gives the first evidence that safety-critical and capability-driving directions stay separable under self-evolution.

2 METHODOLOGY

2.1 PRELIMINARIES: SELF-EVOLVING TRAINING LOOP AND MISEVOLUTION

A self-evolving agent consists of a proposer π_ϕ that generates tasks, a solver π_θ that attempts to solve them, and a verifier V that evaluates outcomes. Training proceeds in rounds $t = 1, \ldots, T$: the proposer samples tasks $\mathcal{D}_t \sim \pi_{\phi_t}$, the solver produces responses y_i for each task $x_i \in \mathcal{D}_t$, and the verifier returns rewards $r_i = V(x_i, y_i)$. Both models are updated via reinforcement learning: $\theta_{t+1} = \theta_t + \alpha_s \nabla_\theta J_{\text{solve}}(\theta_t)$ and $\phi_{t+1} = \phi_t + \alpha_p \nabla_\phi J_{\text{propose}}(\phi_t)$, where J_{solve} rewards correct solutions and J_{propose} rewards tasks of appropriate learnability for the current solver (Zhao et al., 2026a; Huang et al., 2026). The two update rules together create a co-evolutionary dynamic.

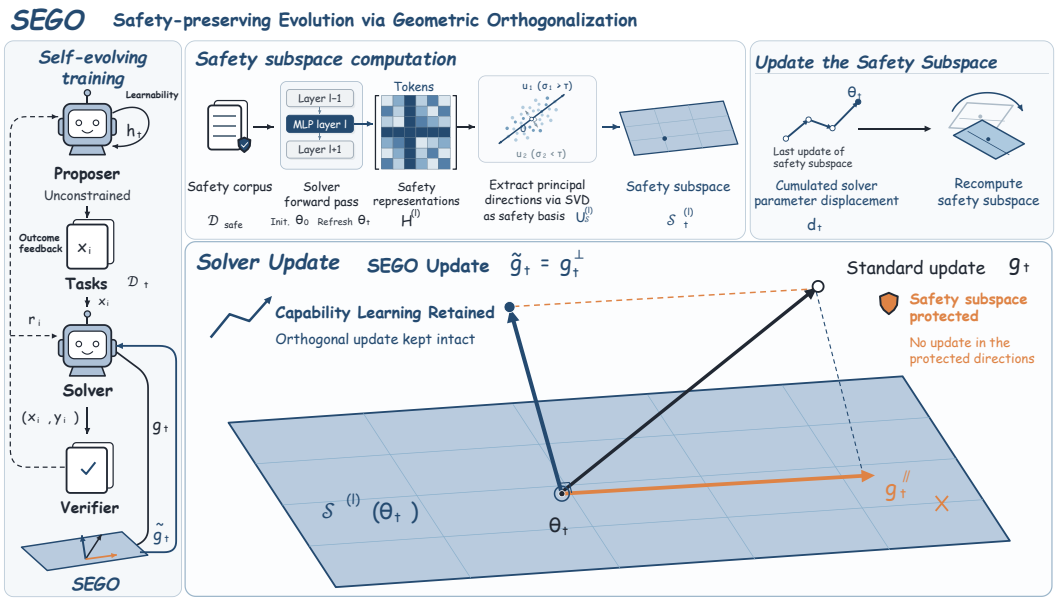
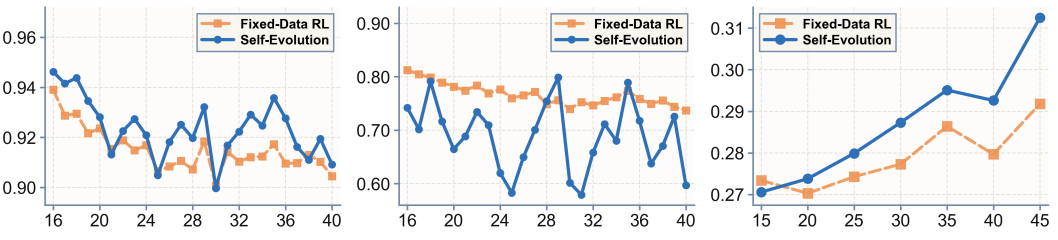


Figure 1: Overview of the SEGO framework, which identifies a safety subspace by SVD on hidden representations and orthogonalizes the evolutionary gradient against it.

We define two evaluation metrics over the solver's parameters: a capability score $\mathcal{C}(\theta_t)$, measured on benchmarks aligned with the evolutionary objective, and a safety score $\mathcal{S}(\theta_t)$, measured on safety benchmarks. Under standard self-evolution, capability improves across rounds while safety degrades (Shi et al., 2026), a divergence recently termed *misevolution* (Shao et al., 2026): $\partial\mathcal{C}(\theta_t)/\partial t > 0$ while $\partial\mathcal{S}(\theta_t)/\partial t < 0$. Because each round initializes from the previous round's checkpoint, this drift compounds over time with no mechanism inside the system that pulls it back.

2.2 The Geometry of Misevolution

The misevolution described above is counterintuitive: the evolutionary objective J_{solve} targets only benign task-solving, and no signal in the training loop explicitly pushes the model away from safe behavior. We hypothesize that the cause is geometric rather than data-driven. Let $g_t = \nabla_\theta J_{\text{solve}}(\theta_t)$ denote the solver's evolutionary gradient at round t. Although g_t is computed from benign tasks, it operates on the full parameter space of π_θ, which encodes both general capabilities and safety-relevant behaviors. If there exists a subspace $S \subset \mathbb{R}^d$ critical for safety, then g_t can be decomposed as $g_t = g_t^{\parallel} + g_t^{\perp}$, where $g_t^{\parallel} = \text{proj}_S(g_t)$ is the component within the safety subspace and $g_t^{\perp} = g_t - g_t^{\parallel}$ is orthogonal to it. The component $g_t^{\perp}$ improves capability without affecting safety; $g_t^{\parallel}$ perturbs the safety subspace as a side effect of capability optimization.



(a) Intrusion Ratio **(b)** Intrusion Direction **(c)** Safety Score

Figure 2: Self-evolution versus fixed-data RL over training.

Empirical verification. To test this hypothesis, we conduct a pilot experiment comparing self-evolution with standard fixed-data RL. Both runs use the same base model and RL algorithm, differ-

ing only in task source. We identify safety subspace S from Section 2.3.1 and record at each round the intrusion ratio $\|g_t^{\parallel}\|/\|g_t\|$, the directional stability $\cos(g_t^{\parallel}, g_{t-1}^{\parallel})$, and the safety score $\mathcal{S}(\theta_t)$.

As shown in Figure 2, the intrusion ratio under self-evolution is consistently higher and more volatile than under fixed-data RL, the intrusion direction fluctuates substantially across rounds, and the resulting safety degradation is correspondingly more severe. This contrast is structural: under fixed-data RL the task distribution is constant and gradient intrusion follows a stable pattern, whereas under self-evolution $\mathcal{D}_t$ co-evolves with the model, perturbing the safety subspace in directions and magnitudes that shift unpredictably from round to round. This defines the problem SEGO addresses.

2.3 SEGO: Safety-Preserving Evolution via Geometric Orthogonalization

2.3.1 Safety Geometry: Identifying the Protected Subspace

SEGO's orthogonalization requires a geometric target: a subspace in parameter space that encodes the model's safety-relevant behavior. Before evolution begins, we identify this subspace from the initial solver π_{θ_0}. We adopt a publicly available safety corpus as $\mathcal{D}_{\text{safe}}$, since deriving it from the model's own refusal responses yields unreliable subspaces for weaker base models (Appendix D.1). A standardized corpus decouples subspace identification from the base model's own safety capability. We perform a forward pass of π_{θ_0} on $\mathcal{D}_{\text{safe}}$ and collect the hidden representations $H^{(l)} \in \mathbb{R}^{d\times N}$ at the MLP layers of each transformer block, where d is the representation dimension and N the total number of tokens. We focus on MLP layers because safety-aligned refusal behavior is predominantly encoded in MLP weights (Zong et al., 2026; Geva et al., 2021), and evolutionary capability updates also concentrate in MLP parameters (Section 4.1).

To extract the principal directions along which these representations vary, we compute the non-central covariance matrix $C^{(l)} = H^{(l)}H^{(l)\top} \in \mathbb{R}^{d\times d}$ and apply singular value decomposition (SVD), $C^{(l)} = U^{(l)}\Sigma^{(l)}U^{(l)\top}$. The singular values in $\Sigma^{(l)}$ rank the directions by their importance to safety-relevant representations. We retain the columns of $U^{(l)}$ corresponding to singular values above a threshold τ, forming the safety subspace basis $U_{\mathcal{S}}^{(l)} \in \mathbb{R}^{d\times k}$ at layer l. This basis defines the geometric object against which SEGO orthogonalizes the evolutionary update. The threshold τ controls the dimensionality of the protected subspace, governing the trade-off between safety preservation and capability growth (Appendix D.2).

2.3.2 Geometric Orthogonalization of the Evolutionary Update

At each round t, the solver computes an evolutionary gradient $g_t = \nabla_\theta J_{\text{solve}}(\theta_t)$ via reinforcement learning. The principle of SEGO is to let the solver evolve freely in every direction that does not interfere with safety, while the update it receives carries no component along safety-critical ones. We enforce this by projecting out the safety-subspace component at each layer l: $\tilde{g}_t^{(l)} = g_t^{(l)} - U_{\mathcal{S}}^{(l)}U_{\mathcal{S}}^{(l)\top}g_t^{(l)}$, so that $\tilde{g}_t^{(l)}$ is orthogonal to $\mathcal{S}^{(l)}$ by construction.

A necessary condition for this approach is that the projected gradient retains sufficient magnitude to drive capability improvement. In self-evolving systems this remains an open question: the gradient distribution shifts as the curriculum co-evolves, and it is unclear whether safety-critical directions remain a small, separable portion of the evolutionary update or grow to consume a large fraction of it. We verify that under an appropriate τ, the retention ratio $\|\tilde{g}_t\|/\|g_t\|$ remains high throughout evolution (Section 4.2), confirming that SEGO can protect a compact safety subspace without starving the evolutionary signal. The computational overhead is analyzed in Appendix B.

Geometric decoupling of safety from the proposer. SEGO applies the orthogonalization exclusively to the solver's gradient; the proposer's update remains unconstrained. This asymmetry yields a structural guarantee. The proposer influences the solver exclusively through the task distribution it generates, and this influence is fully mediated by the solver's gradient: $\phi_t \to \mathcal{D}_t \to g_t^{(l)} = \nabla_{\theta^{(l)}} J_{\text{solve}}(\theta_t; \mathcal{D}_t)$. The orthogonalization, however, eliminates the safety-subspace component of $g_t^{(l)}$ regardless of its origin: $\tilde{g}_t^{(l)} = g_t^{(l)} - U_{\mathcal{S}}^{(l)}U_{\mathcal{S}}^{(l)\top}g_t^{(l)}$ implies $U_{\mathcal{S}}^{(l)\top}\tilde{g}_t^{(l)} = 0$ for all $g_t^{(l)}$. The safety-subspace content of the solver's update therefore vanishes for all ϕ_t and all $\mathcal{D}_t$, independently of what the proposer generates. The solver's safety protection is geometrically decoupled from the

proposer's state: no matter how the proposer evolves or what tasks it generates, the orthogonalization neutralizes any safety-subspace content before it reaches the solver's parameters. Conversely, applying the same orthogonalization to the proposer's gradient $h_t = \nabla_\phi J_{\text{propose}}(\phi_t)$ would restrict its update to the orthogonal complement of $\mathcal{S}^{(l)}$, $\tilde{h}_t^{(l)} = h_t^{(l)} - U_{\mathcal{S}}^{(l)} U_{\mathcal{S}}^{(l)\top} h_t^{(l)}$, reducing the effective update space from $\mathbb{R}^d$ to $\mathbb{R}^{d-k}$ and deviating from the unconstrained curriculum gradient. We verify empirically that adding the orthogonalization to the proposer yields no further safety improvement but reduces capability growth (Section 4.5).

2.3.3 Adaptive Subspace Co-Evolution

As the solver's parameters shift from θ_0 to θ_t across rounds of evolution, the directions encoding safety-relevant behavior may drift accordingly. A fixed subspace risks protecting outdated directions. SEGO therefore models the safety subspace as a function of the current parameters, $\mathcal{S}^{(l)}(\theta_t)$, so that the orthogonalization in Section 2.3.2 adapts to the model it protects: $\tilde{g}_t^{(l)} = g_t^{(l)} - U_{\mathcal{S}}^{(l)}(\theta_t)\, U_{\mathcal{S}}^{(l)}(\theta_t)^\top g_t^{(l)}$, where $U_{\mathcal{S}}^{(l)}(\theta_t)$ is obtained by applying the procedure in Section 2.3.1 to the current model θ_t. Recomputing at every round, however, risks subspace oscillation: if the model has not moved substantially, the recomputed subspace may differ from the previous one only due to statistical noise, intermittently exposing safety-critical parameters. SEGO instead triggers recomputation based on cumulative parameter displacement $d_t = \sum_{i=t_{\text{last}}}^{t} \alpha_s \|\tilde{g}_i\|$, where t_{last} is the round at which the subspace was last recomputed. When d_t exceeds a threshold γ, $U_{\mathcal{S}}^{(l)}(\theta_t)$ is recomputed and d_t is reset to zero. This adaptive scheme lets the subspace track representational change at the pace of evolution itself: recomputation is frequent during rapid capability growth and rare as evolution converges. We discuss the sensitivity to γ in Appendix D.3.

Table 1: Safety and over-refusal results for pre-evolution models, AZR/R-Zero baselines, and SEGO. Best results per group are in **bold**; SEGO improvements over AZR/R-Zero are underlined.

Model	Safety						Over-refusal			
	PKU ↓	AdvB ↓	HB ↓	JBB ↓	SALAD ↓	HEx ↓	XST-S ↓	XST-U ↑	ORB ↓	JBB-B ↓
AZR Series										
Qwen2.5-7B	0.2930	0.2538	**0.4700**	**0.4000**	0.2760	0.4033	0.0800	0.6800	0.1031	0.1300
AZR-Base7B	0.3180	0.3538	0.5350	0.4700	0.2910	0.4233	0.0720	0.6300	0.1054	0.1300
SEGO-7B	**0.2920**	**0.2288**	0.4725	0.4300	**0.2525**	**0.3700**	**0.0640**	**0.6950**	**0.0788**	**0.1100**
Qwen2.5-Coder-3B	**0.5630**	0.6019	**0.5325**	**0.6400**	**0.3335**	**0.6767**	0.0400	**0.5200**	0.0826	0.1400
AZR-Coder3B	0.6240	0.6250	0.5400	0.7100	0.3665	0.7167	0.0320	0.4750	**0.0629**	0.1300
SEGO-Coder-3B	0.5860	**0.5865**	0.5350	**0.6400**	0.3405	0.6800	**0.0200**	0.4950	0.0681	**0.1100**
Qwen2.5-Coder-7B	0.3350	**0.1327**	0.3550	**0.3800**	0.2820	**0.2667**	0.0960	**0.6800**	0.2434	0.2200
AZR-Coder7B	0.3930	0.2231	0.4825	0.4200	0.2975	0.4067	0.0840	0.6300	0.2328	0.2000
SEGO-Coder-7B	**0.3120**	0.1954	**0.3500**	0.3900	**0.2590**	0.3067	**0.0640**	0.6750	**0.1600**	**0.1700**
Qwen2.5-Coder-14B	0.5330	0.3308	**0.3125**	**0.4000**	0.3485	**0.3400**	0.0760	0.5000	0.2343	0.1200
AZR-Coder14B	0.5720	0.3981	0.3625	0.4300	0.3720	0.4067	0.0320	0.4300	0.1380	0.1100
SEGO-Coder-14B	**0.5270**	**0.3114**	0.3175	0.4100	**0.3465**	0.3630	**0.0200**	**0.5200**	**0.1266**	**0.1000**
R-Zero Series										
Qwen3-4B	0.4490	**0.0808**	**0.2625**	**0.2100**	0.4165	**0.3200**	0.2040	0.4950	0.0948	**0.0700**
R-Zero-4B	0.3730	0.1250	0.3425	0.2600	0.3860	0.3600	**0.0840**	0.4900	0.0743	0.1600
SEGO-4B	**0.3640**	0.0942	0.3075	0.2300	**0.3740**	0.3267	0.1120	**0.5250**	**0.0637**	0.1300
Qwen3-8B	0.1360	**0.0481**	**0.2575**	0.1000	0.1710	**0.1300**	**0.0480**	**0.6200**	0.2123	**0.1600**
R-Zero-8B	0.1450	0.0563	0.3125	0.1000	0.1835	0.1566	0.0520	0.5950	0.1797	0.1900
SEGO-8B	**0.1240**	**0.0481**	0.2925	**0.0900**	**0.1625**	0.1433	**0.0480**	0.6150	**0.1706**	**0.1600**

3 Experiments

3.1 Experimental Setup

We evaluate SEGO on AZR (Zhao et al., 2026a) with Qwen2.5-7B (Yang et al., 2024) and Qwen2.5-Coder-3B/7B/14B-Instruct(Hui et al., 2024), and on R-Zero (Huang et al., 2026) with Qwen3-4B-

Table 2: Code and math results for pre-evolution models, AZR/R-Zero baselines, and SEGO. Best results per group are in **bold**; SEGO improvements over AZR/R-Zero are underlined.

Model	Code							Math			
	LCB↑	CRUX-I↑	CRUX-O↑	MBPP↑	HE↑	HE+↑	MBPP+↑	MATH-500↑	GSM8K↑	Olymp.↑	Minerva↑
AZR Series											
Qwen2.5-7B	0.0994	0.3613	0.3937	0.4020	**0.7988**	**0.7607**	0.3438	0.5400	0.5428	0.3294	0.1213
AZR-Base7B	**0.1228**	**0.4275**	0.3862	0.4960	0.7744	0.7178	0.4196	**0.6880**	0.5619	0.3561	**0.1838**
SEGO-7B	**0.1228**	0.4225	**0.4188**	**0.5440**	0.7866	0.7485	**0.5089**	0.6740	**0.5967**	**0.3631**	0.1781
Qwen2.5-Coder-3B	0.0526	0.2425	**0.3250**	0.3400	**0.7500**	**0.6810**	0.2946	0.4420	**0.7172**	0.1217	0.1124
AZR-Coder3B	0.0468	0.2487	0.3113	0.4080	0.6707	0.6074	**0.3482**	0.4400	0.6892	0.1276	0.1329
SEGO-Coder-3B	**0.0556**	**0.2662**	0.3187	**0.4160**	0.7073	0.6380	**0.3482**	**0.4500**	0.6952	**0.1395**	**0.1471**
Qwen2.5-Coder-7B	0.0673	0.3212	0.3862	**0.2800**	0.7500	0.7239	**0.1786**	0.5540	0.7747	0.2255	0.1691
AZR-Coder7B	**0.1345**	0.3050	0.3987	0.2600	**0.8476**	**0.8098**	0.1696	**0.6660**	0.7965	**0.3665**	**0.2022**
SEGO-Coder-7B	0.1228	**0.3650**	**0.4113**	0.2760	0.7805	0.7485	**0.1786**	0.6320	**0.8234**	0.2923	0.1912
Qwen2.5-Coder-14B	0.0965	0.5250	**0.5100**	0.6200	0.7866	0.7546	0.5223	0.4540	0.6748	0.2136	0.1397
AZR-Coder14B	0.1608	0.5225	0.4938	**0.6880**	**0.8110**	0.7485	**0.5804**	0.5340	0.7377	**0.3175**	0.2316
SEGO-Coder-14B	**0.1624**	**0.5375**	0.4972	0.6760	0.7933	**0.7635**	0.5602	**0.5420**	**0.8187**	0.3054	**0.2649**
R-Zero Series											
Qwen3-4B	0.1404	0.4662	0.4213	0.6000	0.8110	0.7730	0.5089	0.7300	0.8453	0.4036	0.2537
R-Zero-4B	**0.1784**	**0.4975**	**0.4400**	**0.6220**	0.8415	**0.8282**	0.5580	0.7620	0.9083	0.4199	0.3235
SEGO-4B	0.1725	0.4938	0.4275	0.6020	**0.8476**	0.8221	**0.5625**	**0.7800**	**0.9128**	**0.4228**	**0.3456**
Qwen3-8B	0.1725	0.5413	0.4850	0.4720	0.8598	0.8160	0.3327	0.7080	0.8954	0.3991	0.2794
R-Zero-8B	0.1959	0.5500	0.4913	**0.4780**	0.8537	0.8282	**0.3493**	0.7980	0.9219	0.4318	0.3860
SEGO-8B	**0.2105**	**0.5513**	**0.4938**	0.4700	**0.8659**	**0.8344**	0.3438	**0.8160**	**0.9378**	**0.4718**	**0.4007**

Base and Qwen3-8B-Base (Yang et al., 2025), following each framework's original training configuration and reporting results after 300 evolution steps for AZR and three iterations for R-Zero. Safety is measured by attack success rate (ASR) on PKU-SafeRLHF (Ji et al., 2025), AdvBench (Zou et al., 2023), HarmBench (Mazeika et al., 2024), JailbreakBench (Chao et al., 2024), SALAD-Bench (Li et al., 2024), and HEx-PHI (Qi et al., 2024); over-refusal on XSTest (Röttger et al., 2024), OR-Bench (Cui et al., 2024), and JBB-Benign (Chao et al., 2024); coding on LiveCodeBench (Jain et al., 2025), CRUXEval (Gu et al., 2024), and EvalPlus (Liu et al., 2023); and mathematical reasoning on MATH-500 (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), OlympiadBench (He et al., 2024), and Minerva (Lewkowycz et al., 2022). All benchmarks are decoded greedily with a single sample per prompt, and ASR is scored by an external classifier rather than keyword heuristics. The safety corpus $\mathcal{D}_{\text{safe}}$ consists of 1,024 samples from the PKU-SafeRLHF training split, disjoint from the test split used to report ASR, and the threshold is set to $\tau = 5 \times 10^{-5}$. All experiments run on 8×A100 GPUs; full training, baseline, and evaluation configurations are in Appendix C.

3.2 Main Results

Table 1 and Table 2 presents results across all pre-evolution models, frameworks, and evaluation dimensions. We highlight three findings. Self-evolution erodes safety, and SEGO prevents it without sacrificing capability. Standard self-evolution raises ASR on 34 of the 36 model-benchmark pairs we evaluate, by up to 9.0 points, while its capability effect is uneven: it improves mathematical reasoning consistently but degrades several coding benchmarks on the smaller models. SEGO lowers ASR relative to unconstrained evolution on every one of the 36 pairs, and in 15 of them reaches a level at or below the pre-evolution model. Capability is preserved throughout: SEGO matches or exceeds unconstrained evolution on all eight mathematical-reasoning scores under R-Zero and on the majority of coding scores under AZR.

Safety protection generalizes beyond the construction corpus. The safety subspace is identified from PKU-SafeRLHF alone, yet SEGO lowers ASR over unconstrained evolution on all five held-out benchmarks across all six configurations, which differ in attack style and evaluation protocol. The subspace therefore captures general safety-relevant directions rather than corpus-specific features.

SEGO protects the ability to refuse, not the tendency to refuse. Under standard self-evolution, refusal on genuinely unsafe queries (XSTest-Unsafe) drops in every configuration, showing that the model's discriminative safety behavior is eroded rather than merely its output style. SEGO recovers most of

this loss, reaching or exceeding the pre-evolution level in several configurations. Meanwhile, refusal on benign queries does not increase: SEGO's OR-Bench rate is below the pre-evolution model in all six configurations, and its XSTest-Safe rate is at or below in all six. SEGO therefore preserves the safety boundary in both directions rather than shifting toward blanket refusal.

3.3 Comparison with Alternative Safety Interventions

We compare SEGO against three alternative safety strategies, all evaluated under AZR with Qwen2.5-7B; full results are reported in Appendix F.

KL regularization constrains all directions indiscriminately. Adding a KL divergence penalty to the evolutionary objective (AZR+KL) fails to preserve safety while limiting capability growth. This is expected: KL penalizes deviation from the reference policy uniformly across all parameter directions, constraining safety-relevant and capability-driving updates alike. SEGO, by contrast, constrains only the safety subspace, leaving capability-driving directions free.

Single refusal directions are insufficient. Prior work has identified refusal behavior as governed by a single direction in activation space (Arditi et al., 2024). We apply SPF (Zhang et al., 2026), which projects gradients along such low-dimensional safety directions, within the AZR loop. SPF not only fails to protect safety but degrades it relative to unconstrained evolution, while capability remains comparable to standard AZR. This suggests that the safety behavior relevant to self-evolution is richer than a single refusal direction: it spans a multi-dimensional subspace that a one-dimensional projection cannot adequately cover. SEGO's use of a k-dimensional subspace derived from activation covariance captures this richer structure.

Post-hoc realignment incurs an alignment tax and weakens with evolution depth. We apply DPO-based safety realignment (Rafailov et al., 2023) at three checkpoints during AZR training, after 300, 400, and 500 steps, and compare each realigned model against the same checkpoint before realignment. Realignment consistently trades away capability: across all three checkpoints and all eleven capability benchmarks, without exception, the realigned model scores below its pre-realignment counterpart. The safety it recovers, meanwhile, diminishes with the depth at which it is applied: averaged over the six safety benchmarks, DPO reduces ASR by two points at Step 300 but by only a third of a point at Step 500, where on two benchmarks it no longer improves safety at all. Since the unrealigned checkpoints show little further degradation over this interval, the loss of repairability does not stem from damage accumulating; the drift itself becomes structurally harder to reverse. SEGO avoids both failure modes: because it operates throughout evolution rather than after it, its protection does not weaken with evolution depth, and because it acts on the gradient geometry rather than on the training objective, it preserves the capability gains that evolution produces.

4 Analysis

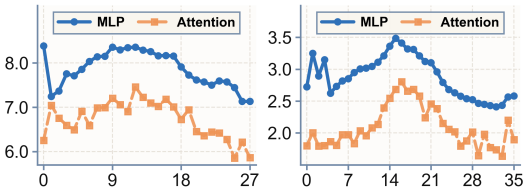
4.1 Evolutionary Updates Concentrate in MLP Layers

SEGO identifies and protects the safety subspace at MLP layers. To validate this design choice, we examine where in the network self-evolution produces the largest parameter changes. For each layer l, we compute the relative weight change $\|\Delta W^{(l)}\|_F / \|W_0^{(l)}\|_F$ between the pre-evolution and post-evolution model, separately for MLP and attention parameters. Normalizing by the initial weight norm makes the two module types directly comparable despite their differing parameter counts.

Figure 3 shows the results for AZR (Qwen2.5-7B) and R-Zero (Qwen3-4B-Base). MLP parameters undergo larger relative change than attention parameters at *every* layer of *both* models, without exception across 28 and 36 layers respectively. Averaged over layers, MLP updates exceed attention updates by 17% under AZR and 38% under R-Zero. The margin persists throughout the network but is widest in the early and late layers, where attention updates fall off more sharply; in both frameworks the two module types concentrate updates in the middle layers.

The universality of this ordering, rather than its magnitude, is what matters here: across two frameworks, two model families, and 64 layers in total, self-evolution never drives attention parameters harder than MLP parameters. This is consistent with prior findings that MLP layers serve as the primary site of knowledge storage in transformers (Geva et al., 2021). Combined with evidence that safety-aligned behavior is also encoded in MLP weights (Zong et al., 2026), this result supports

SEGO's focus on MLP layers as the site where capability updates and safety encoding coexist and conflict.



(a) AZR **(b)** R-Zero

Figure 3: MLP vs Attention.

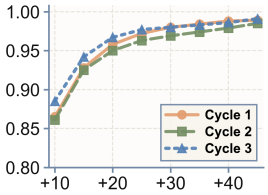


Figure 4: Gradient retention per cycle.

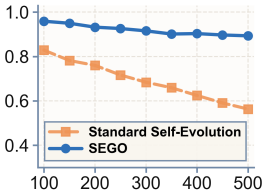


Figure 5: Safety subspace drift rate.

4.2 Separability of Safety and Capability Directions

A core premise of SEGO is that safety-critical and capability-driving directions are separable, so that orthogonalizing the evolutionary gradient does not starve capability growth. Whether this holds under the shifting gradient distributions of self-evolution has not been established. We measure the retention ratio $\|\tilde{g}_t\|/\|g_t\|$ throughout training, replacing the displacement-triggered recomputation of Section 2.3.3 with a fixed schedule so that cycles are directly comparable.

Figure 4 shows two patterns. First, the orthogonalization removes only a small fraction of the evolutionary signal: the bulk of the gradient magnitude survives the projection throughout every cycle, and even at its minimum, reached immediately after recomputation when the subspace is freshest, most of the update is retained. The safety subspace thus occupies a compact portion of the parameter space. Second, the three cycles trace nearly identical trajectories, so the separability does not degrade as evolution proceeds: despite the co-evolving task distribution continuously reshaping the gradient landscape, the geometric relationship between safety-critical and capability-driving directions remains stable across rounds. To our knowledge, this is the first empirical evidence that such separability holds under self-evolution.

The within-cycle shape also motivates the adaptive recomputation mechanism (Section 2.3.3). The rise in retention ratio within each cycle reflects the subspace becoming stale as the model evolves away from the parameters used in its last computation: an ever smaller share of the gradient is judged to lie in the safety subspace, not because the gradient has stopped perturbing safety-critical directions, but because the basis no longer tracks where those directions now lie. This staleness is what the displacement criterion is designed to detect.

4.3 Safety Subspace Stability Under Evolution

The adaptive recomputation mechanism of Section 2.3.3 presumes that the safety subspace drifts as the model evolves. We measure this drift and ask whether SEGO affects its rate. At regular intervals throughout training, independently of SEGO's own recomputation schedule, we recompute the safety subspace from the current parameters and measure the similarity between consecutive subspaces, $\mathrm{sim}(t) = \frac{1}{k}\|{U_{\mathcal{S}}^{(t-1)}}^{\top} U_{\mathcal{S}}^{(t)}\|_F^2$, under standard self-evolution and SEGO, both with AZR on Qwen2.5-7B. This tracks how fast safety-relevant representations reorganize under each regime.

Figure 5 shows that under standard self-evolution the subspace drifts substantially, with consecutive similarity declining monotonically throughout training. Under SEGO it remains markedly more stable, and the gap between the two conditions widens as training progresses. This is not simply a consequence of SEGO taking smaller steps: the retention ratio of Section 4.2 shows that the projected update preserves most of the gradient magnitude, so the two conditions move through parameter space at comparable rates while reorganizing safety-relevant representations at very different ones.

The orthogonalization therefore participates in a stabilizing feedback loop: it prevents the evolutionary gradient from perturbing safety-critical parameters, which keeps the safety subspace stable, which in turn keeps the orthogonalization accurate. Standard self-evolution lacks this feedback, and the subspace becomes a moving target that any fixed or infrequently updated protection would struggle to track. Protecting the safety subspace thus carries the secondary benefit of stabilizing it.

4.4 Orthogonalization as Implicit Regularization

A noteworthy pattern across Table 2 is that SEGO does not merely match but in several cases exceeds the capability scores of standard self-evolution. We attribute this to an implicit regularization effect. The safety-subspace component of the gradient (the $g_t^{\parallel}$ of Section 2.2) shifts direction unpredictably from round to round as the curriculum co-evolves (Figure 2). Updates along such an unstable direction act as noise on the optimization trajectory: they displace safety-critical parameters without steering the model toward any consistent objective, and whatever capability-relevant signal they carry is largely available in the orthogonal complement as well. Removing this component confines the update to directions whose geometry remains stable across rounds. Under this view SEGO is not a constraint that self-evolution pays for in capability, but a filter that suppresses a source of interference between safety and capability updates.

4.5 Geometric Decoupling: Proposer Projection Ablation

Section 2.3.2 showed that orthogonalizing the solver gradient alone suffices to decouple safety from the proposer's state, and that extending the projection to the proposer would narrow its update space without additional safety benefit. We verify this on R-Zero (Qwen3-4B-Base) by comparing solver-only projection against projection applied to both the solver and the proposer.

Table 3: Safety comparison of solver-only and proposer-and-solver SEGO. Lower attack success rates indicate better safety.

Model	PKU↓	AdvB↓	HB↓	JBB↓	SB↓	HEx↓
Qwen3-4B	0.4490	0.0808	0.2625	0.2100	0.4165	0.3200
SEGO (solver-only)	0.3640	0.0942	0.3075	0.2300	0.3740	0.3267
SEGO (proposer + solver)	0.4100	0.0962	0.3175	0.2700	0.3815	0.3467

Table 3 and Table 4 confirms the theoretical prediction on both counts. On the safety side, projecting the proposer yields no improvement: ASR under the proposer+solver variant is no lower than solver-only on any of the six benchmarks, and is higher on all of them, indicating that the additional constraint does not strengthen safety protection. On the capability side, the proposer+solver variant underperforms solver-only on every one of the eleven coding and mathematical benchmarks, and on MBPP it falls below the pre-evolution model. This is consistent with the analysis in Section 2.3.2: restricting the proposer's gradient to the orthogonal complement of the safety subspace deflects it from its optimal curriculum direction, limiting the diversity and difficulty of generated tasks and thereby weakening the evolutionary signal available to the solver. These results empirically confirm the geometric decoupling: constraining the proposer buys no additional safety, while the unconstrained proposer is what sustains the co-evolutionary dynamic that drives capability growth.

Table 4: Code and math comparison of solver-only and proposer-and-solver SEGO. Higher accuracy indicates better performance.

Model	Code					Math			
	LCB↑	C-I↑	C-O↑	MBPP↑	HE↑	M500↑	GSM↑	OB↑	Min.↑
Qwen3-4B	0.1404	0.4662	0.4213	0.6000	0.8110	0.7300	0.8453	0.4036	0.2537
SEGO (solver-only)	0.1725	0.4938	0.4275	0.6020	0.8476	0.7800	0.9128	0.4228	0.3456
SEGO (proposer + solver)	0.1579	0.4788	0.4225	0.5380	0.8293	0.7480	0.8999	0.4095	0.3088

5 Conclusion and Limitations

This work locates the origin of misevolution in the geometry of the evolutionary update rather than the content of the self-generated curriculum, and shows that safety preservation can be enforced as a constraint on that geometry. SEGO orthogonalizes the solver's update against a safety subspace tracked as the model drifts, preserving safety while matching the capability growth of unconstrained evolution. Two limitations remain. First, the guarantee is exact at the level of the gradient: the projected update carries no safety-subspace component, but the realized parameter displacement passes through the optimizer's coordinate-wise rescaling and weight decay, and the subspace itself is a thresholded approximation, so safety scores are not strictly invariant. Second, confining the update to the orthogonal complement slows early capability growth; the gap closes later in training and the ceiling is unaffected (Appendix E).

REFERENCES

Andy Arditi, Oscar Obeso, Aaquib Syed, Daniel Paleka, Nina Panickssery, Wes Gurnee, and Neel Nanda. Refusal in language models is mediated by a single direction. *Advances in Neural Information Processing Systems*, 37:136037–136083, 2024.

Patrick Chao, Edoardo Debenedetti, Alexander Robey, Maksym Andriushchenko, Francesco Croce, Vikash Sehwag, Edgar Dobriban, Nicolas Flammarion, George J Pappas, Florian Tramer, et al. Jailbreakbench: An open robustness benchmark for jailbreaking large language models. *Advances in Neural Information Processing Systems*, 37:55005–55029, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Michael K Cohen, Marcus Hutter, Yoshua Bengio, and Stuart Russell. Rl, but don't do anything i wouldn't do. *arXiv preprint arXiv:2410.06213*, 2024.

Justin Cui, Wei-Lin Chiang, Ion Stoica, and Cho-Jui Hsieh. Or-bench: An over-refusal benchmark for large language models. *arXiv preprint arXiv:2405.20947*, 2024.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, et al. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 2025.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 5484–5495, 2021.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I Wang. Cruxeval: A benchmark for code reasoning, understanding and execution. *arXiv preprint arXiv:2401.03065*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Dongxin Guo, Jikun Wu, and Siu Ming Yiu. Safeanchor: Preventing cumulative safety erosion in continual domain adaptation of large language models. *arXiv preprint arXiv:2604.17691*, 2026.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen 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 (Volume 1: Long Papers)*, pp. 3828–3850, 2024.

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. *arXiv preprint arXiv:2103.03874*, 2021.

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. In *International Conference on Learning Representations*, volume 2026, pp. 130770–130790, 2026.

Tiansheng Huang, Sihao Hu, Fatih Ilhan, Selim Furkan Tekin, Zachary Yahn, Yichang Xu, and Ling Liu. Safety tax: Safety alignment makes your large reasoning models less reasonable. *arXiv preprint arXiv:2503.00555*, 2025.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Naman Jain, 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*, volume 2025, pp. 58791–58831, 2025.

Jiaming Ji, Donghai Hong, Borong Zhang, Boyuan Chen, Josef Dai, Boren Zheng, Tianyi Alex Qiu, Jiayi Zhou, Kaile Wang, Boxun Li, et al. Pku-saferlhf: Towards multi-level safety alignment for llms with human preference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 31983–32016, 2025.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35:21314–21328, 2022.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.

Lijun Li, Bowen Dong, Ruohui Wang, Xuhao Hu, Wangmeng Zuo, Dahua Lin, Yu Qiao, and Jing Shao. Salad-bench: A hierarchical and comprehensive safety benchmark for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3923–3954, 2024.

Yong Lin, Hangyu Lin, Wei Xiong, Shizhe Diao, Jianmeng Liu, Jipeng Zhang, Rui Pan, Haoxiang Wang, Wenbin Hu, Hanning Zhang, et al. Mitigating the alignment tax of rlhf. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 580–606, 2024.

Adam Dahlgren Lindström, Leila Methnani, Lea Krause, Petter Ericson, Íñigo Martínez de Rituerto de Troya, Dimitri Coelho Mollo, and Roel Dobbe. Ai alignment through reinforcement learning from human feedback? contradictions and limitations. *arXiv preprint arXiv:2406.18346*, 2024.

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. *Advances in neural information processing systems*, 36:21558–21572, 2023.

Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, et al. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal. *arXiv preprint arXiv:2402.04249*, 2024.

Nikhil Shivakumar Nayak, Krishnateja Killamsetty, Ligong Han, Abhishek Bhandwaldar, Prateek Chanda, Kai Xu, Oleg Silkin, Mustafa Eyceoz, Hao Wang, Aldo Pareja, et al. Sculpting subspaces: Constrained full fine-tuning in llms for continual learning. In *International Conference on Learning Representations*, volume 2026, pp. 20212–20243, 2026.

Yifan Niu, Han Xiao, Dongyi Liu, Nuo Chen, and Jia Li. Mitigating the safety alignment tax with null-space constrained policy optimization. In *International Conference on Learning Representations*, volume 2026, pp. 142434–142465, 2026.

Xiangyu Qi, Yi Zeng, Tinghao Xie, Pin-Yu Chen, Ruoxi Jia, Prateek Mittal, and Peter Henderson. Fine-tuning aligned language models compromises safety, even when users do not intend to! In *International Conference on Learning Representations*, volume 2024, pp. 30988–31043, 2024.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Paul Röttger, Hannah Kirk, Bertie Vidgen, Giuseppe Attanasio, Federico Bianchi, and Dirk Hovy. Xstest: A test suite for identifying exaggerated safety behaviours in 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. 5377–5400, 2024.

Shuai Shao, Qihan Ren, Dongrui Liu, Chen Qian, Boyi Wei, Dadi Guo, Jingyi Yang, Xinhao Song, Linfeng Zhang, Weinan Zhang, et al. Your agent may misevolve: Emergent risks in self-evolving llm agents. In *International Conference on Learning Representations*, volume 2026, pp. 99728–99793, 2026.

Dianxing Shi, Bowen Wang, Junqi He, Junhao Chen, and Yuta Nakashima. Towards healthy evolution: Exploring the role and mechanisms of human-agent interaction in self-evolving systems. *arXiv preprint arXiv:2606.06114*, 2026.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 13484–13508, 2023.

Xumeng Wen, Zihan Liu, Shun Zheng, Shengyu Ye, Zhirong Wu, Yang Wang, Zhijian Xu, Xiao Liang, Junjie Li, Ziming Miao, et al. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base llms. In *International Conference on Learning Representations*, volume 2026, pp. 49450–49483, 2026.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. Wizardlm: Empowering large language models to follow complex instructions. *arXiv e-prints*, pp. arXiv–2304, 2023.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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

Jiawen Zhang, Yangfan Hu, Kejia Chen, Lipeng He, Jiachen Ma, Jian Lou, Dan Li, Jian Liu, Xiaohu Yang, and Ruoxi Jia. Understanding and preserving safety in fine-tuned llms. *arXiv preprint arXiv:2601.10141*, 2026.

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. *Advances in Neural Information Processing Systems*, 38:105816–105879, 2026a.

Weixiang Zhao, Yichen Zhang, Yingshuo Wang, Yang Deng, Yanyan Zhao, Xuda Zhi, Yongbo Huang, Hao He, Wanxiang Che, Bing Qin, et al. On safety risks in experience-driven self-evolving agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 42145–42169, 2026b.

Mingyu Zong, Sampad Mohanty, and Bhaskar Krishnamachari. Localizing safety alignment: Mlp layers and mid-network blocks encode refusal behavior in large language models. *arXiv preprint arXiv:2608.11583*, 2026.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.

Contents of Appendix

A Related Work 14

B Computational Overhead of SEGO 14

C Experimental Details 15

C.1 Training Configuration 15

C.2 Baseline Configurations 15

C.3 Evaluation Protocol 16

D Ablation Studies 16

D.1 Safety Corpus Source 16

D.2 Threshold τ 17

D.3 Adaptive Subspace Recomputation 17

E Training Convergence Under SEGO 18

F Baseline Comparison 18

G Multi-Seed Results 20

A Related Work

Self-Evolving Agents Recent approaches to autonomous model improvement have progressively reduced reliance on externally specified training signals. Early work explored complementary directions: CodeRL uses reinforcement learning with unit-test feedback to optimize code generation, while the training problems themselves remain externally provided (Le et al., 2022); STaR instead bootstraps learning from self-generated rationales over fixed questions and answers (Zelikman et al., 2022). Self-Instruct further shifts data construction to the model by generating instructions and instances from a small human-written seed set (Wang et al., 2023), while Evol-Instruct iteratively increases task complexity through predefined evolution operators (Xu et al., 2023). Despite this growing autonomy, the training distribution remains anchored by fixed datasets, human seeds, or externally specified task-generation rules. Absolute Zero moves beyond this setting by coupling task proposal and solving within the same training loop, allowing the curriculum itself to change with the evolving learner (Zhao et al., 2026a). Self-evolution therefore marks a shift from externally specified training distributions to ones that co-evolve with the model itself.

Safety Degradation in Self-Evolution Safety degradation is not unique to self-evolution: even benign capability fine-tuning can weaken the safety alignment of initially aligned models (Qi et al., 2024). Self-evolution makes this risk more structural because updates are repeated over a changing, self-generated training distribution, with each round inheriting the model produced by the previous one. Recent work terms such unintended degradation *misevolution*, showing that safety risks can emerge from routine evolution without malicious data or an explicit adversary (Shao et al., 2026). Meanwhile, representative frameworks such as Absolute Zero and R-Zero optimize learnability, task difficulty, and solution correctness to sustain capability growth (Zhao et al., 2026a; Huang et al., 2026), but include no corresponding mechanism to preserve pre-existing safety across successive updates. Safety preservation therefore remains incidental to the evolutionary objective rather than intrinsic to the evolution process.

Safety Interventions for Self-Evolution Existing safety interventions act at different points relative to the evolutionary process. KL regularization limits policy drift toward a trusted reference, but neither isolates safety-critical directions nor guarantees preservation of trusted behavior (Cohen et al., 2024). Post-evolution DPO can partially restore degraded safety, while preference-based realignment may incur capability forgetting or alignment tax (Shao et al., 2026; Lin et al., 2024). Prompt- or memory-based safeguards instead act outside parameter evolution and can induce over-refusal (Zhao et al., 2026b). A related line constrains parameter updates geometrically: OSFT protects prior knowledge by restricting new-task updates to orthogonal subspaces (Nayak et al., 2026), NSPO projects safety-alignment gradients away from subspaces encoding general capabilities (Niu et al., 2026), and SPF removes utility-gradient components that conflict with a low-rank safety subspace (Zhang et al., 2026). These methods demonstrate the effectiveness of subspace-constrained optimization, but are designed for externally specified fine-tuning or continual-learning settings rather than the endogenous, continuously shifting optimization dynamics of self-evolution. More directly comparable in setting is ANCHOR, which injects human-like supervision into multiple stages of self-evolution and substantially mitigates safety drift (Shi et al., 2026), yet relies on external oversight of the otherwise autonomous loop. Preserving safety intrinsically during autonomous self-evolution without suppressing useful capability growth therefore remains unresolved.

B Computational Overhead of SEGO

The geometric orthogonalization in SEGO introduces two sources of computational cost. The first is subspace identification (Section 2.3.1): computing the covariance matrix and its SVD incurs $O(d^3)$ per MLP layer, performed once at initialization and again at each adaptive recomputation triggered by the displacement threshold γ. Since recomputation is triggered roughly every 50 steps, this cost is amortized over many updates. The second is the per-round projection (Section 2.3.2): at each layer l, the projection $\tilde{g}_t^{(l)} = g_t^{(l)} - U_{\mathcal{S}}^{(l)} {U_{\mathcal{S}}^{(l)}}^{\top} g_t^{(l)}$ is implemented as two matrix-vector multiplications, first ${U_{\mathcal{S}}^{(l)}}^{\top} g_t^{(l)} \in \mathbb{R}^k$ and then $U_{\mathcal{S}}^{(l)}$ times the result, costing $O(dk)$ per layer, where $k \ll d$. In practice $U_{\mathcal{S}}^{(l)} \in \mathbb{R}^{d \times k}$ is precomputed and stored in GPU memory, so the projection adds two small

matrix-vector products per layer per step, negligible relative to the forward and backward passes. The storage overhead is $k \times d$ floats per MLP layer, likewise negligible compared to the model parameters.

C Experimental Details

C.1 Training Configuration

Table 5 lists the training hyperparameters for both self-evolving frameworks. These configurations are identical to the official implementations of AZR (Zhao et al., 2026a) and R-Zero (Huang et al., 2026) respectively. When applying SEGO, we adopt the same hyperparameters without modification.

Main results are reported after 300 evolution steps for AZR and three iterations for R-Zero. The post-hoc realignment experiment in Section 3.3 requires checkpoints at greater evolution depth, so AZR training is extended to 500 steps for that comparison only; the convergence curves in Appendix E cover the first 250 steps.

SEGO projects the gradient before it is passed to the optimizer. Since AdamW rescales coordinates by second-moment estimates and applies decoupled weight decay, the realized parameter displacement is confined to the orthogonal complement only up to these optimizer-level effects; the gradient fed to the optimizer carries no safety-subspace component by construction.

Table 5: Training hyperparameters for AZR and R-Zero. Entries of the form a / b under R-Zero denote separate values for the proposer and the solver.

Hyperparameter	AZR	R-Zero
Advantage estimator	REINFORCE++	GRPO
Learning rate	1×10^{-6}	1×10^{-6}
LR scheduler	Constant	Constant
Weight decay	0.01	0.01
Gradient clipping	1.0	1.0
Clip ratio (low / high)	0.2 / 0.2	0.2 / 0.3
PPO epochs	1	1
Train batch size	64	512
PPO mini-batch size	128	128
Rollout n	1	5 / 4
Temperature	1.0	1.0
Top-p	1.0	0.99
Max prompt length	6144	2048
Max response length	8096	4096
KL loss	Off	Off / low-variance, coef. 0.01

C.2 Baseline Configurations

All baselines in Section 3.3 are run under AZR with Qwen2.5-7B, using the training configuration above unless stated otherwise.

KL regularization (AZR+KL). A KL divergence penalty toward the pre-evolution model is added to the evolutionary objective, with coefficient 0.001.

Single-direction projection (AZR+SPF). SPF (Zhang et al., 2026) is applied within the AZR loop, projecting the gradient along the low-dimensional safety direction it identifies, following the original implementation.

Post-hoc realignment (AZR+DPO). DPO (Rafailov et al., 2023) is applied to AZR checkpoints taken after 300, 400, and 500 evolution steps, using preference pairs from the PKU-SafeRLHF training split, with $\beta = 0.01$, learning rate 1×10^{-6}, and one epoch of training. This is the same source from which $\mathcal{D}_{\text{safe}}$ is drawn, so the comparison between SEGO and post-hoc realignment does not turn on either method having access to safety data the other lacks.

C.3 Evaluation Protocol

Decoding. All benchmarks are evaluated with greedy decoding (temperature 0.0, top-p 1.0) and a single sample per prompt; no benchmark uses sampling or multi-run averaging, so all pass@1 figures are single-pass greedy accuracies. The vLLM engine and sampling seeds are both fixed to 0. The generation budget is 512 tokens for safety and over-refusal; 2048 for coding (4096 for LiveCodeBench); and 4096 for mathematics (2048 for GSM8K).

Dataset sizes and sampling. We use the full set or the officially designated subset for every benchmark except two. For SALAD-Bench, we draw 2,000 prompts uniformly without replacement from the full `base_set` (approximately 21,000 prompts) with seed 42, restored to their original order. For PKU-SafeRLHF, we filter the three test subsets and then sample 1,000 prompts. For HarmBench we use the DirectRequest split (test and validation combined) without jailbreak templates.

Attack success rate. ASR is scored by an external classifier rather than by keyword heuristics. Table 6 lists the judge and decision rule for each benchmark. For AdvBench and HEx-PHI we additionally cross-check the primary classifier against a secondary signal (refusal-prefix matching and MD-Judge-v0.1 (Li et al., 2024) respectively) on the same set of responses.

Over-refusal. For XSTest and OR-Bench, responses are classified by GPT-4o-2024-08-06 into full compliance, full refusal, and partial refusal; the refusal rate is the sum of the latter two. XSTest-Unsafe is reported in the opposite direction, since those prompts are genuinely harmful and refusal is the desired behavior. JBB-Benign is scored by refusal-prefix matching rather than by the GPT-4o classifier, so its absolute values are not directly comparable to those of XSTest and OR-Bench; we report it as a supplementary signal.

Table 6: Judges and decision rules for the safety and over-refusal benchmarks.

Benchmark	Size	Judge	Decision rule
Safety (ASR, lower is better)			
AdvBench	520	HarmBench-cls	classifier outputs `yes`
HarmBench	400	HarmBench-cls	classifier outputs `yes`
JailbreakBench	100	HarmBench-cls	classifier outputs `yes`
HEx-PHI	300	HarmBench-cls	classifier outputs `yes`
PKU-SafeRLHF	1,000	MD-Judge-v0.1	first line is `unsafe`
SALAD-Bench	2,000	MD-Judge-v0.1	first line is `unsafe`
Over-refusal			
XSTest-Safe	250	GPT-4o	full or partial refusal
XSTest-Unsafe	200	GPT-4o	full or partial refusal
OR-Bench	1,319	GPT-4o	full or partial refusal
JBB-Benign	100	HarmBench-cls	refusal-prefix match

D Ablation Studies

D.1 Safety Corpus Source

We compare three sources for constructing the safety subspace: (1) a standardized public safety corpus (SEGO), (2) the pre-evolution model's own refusal responses to harmful prompts (SEGO-generate), and (3) a random subspace of matching dimensionality (Random). All three are evaluated under AZR across Qwen2.5-Coder-3B, 7B, and 14B.

As shown in Table 7, the standardized corpus yields the lowest ASR at every model scale and on all but one benchmark, followed by model-generated responses, with the random subspace worst. Two observations stand out. First, the random subspace confers essentially no protection: its ASR matches that of unconstrained self-evolution, confirming that SEGO's effect is not a byproduct of removing an arbitrary fraction of the gradient but depends on the projected subspace being genuinely safety-relevant. Second, subspaces derived from the model's own refusals are unreliable: at

3B they are barely better than random, and worse on two benchmarks. Self-generation only becomes competitive at 14B, that is, precisely where the model is already well aligned, which is the one assumption self-evolution cannot rely on. These results motivate SEGO's default use of a standardized corpus, which decouples subspace quality from the pre-evolution model's own alignment capability and provides consistent protection regardless of model scale.

Table 7: Safety results for SEGO, its generate variant, and random baselines. Lower attack success rates indicate better safety.

Model	PKU↓	AdvB↓	HB↓	JBB↓	SB↓	HEx↓
SEGO-Coder-3B	0.5860	0.5865	0.5350	0.6400	0.3405	0.6800
SEGO-Coder-3B-generate	0.6170	0.6140	0.5625	0.6900	0.3765	0.7233
Random-3B	0.6320	0.6240	0.5700	0.7200	0.3745	0.7267
SEGO-Coder-7B	0.3120	0.1954	0.3500	0.3900	0.2590	0.3067
SEGO-Coder-7B-generate	0.3370	0.2181	0.3825	0.4100	0.2970	0.3500
Random-7B	0.3730	0.2231	0.4125	0.4300	0.3275	0.3867
SEGO-Coder-14B	0.5270	0.3114	0.3175	0.4100	0.3465	0.3630
SEGO-Coder-14B-generate	0.5320	0.3208	0.3225	0.4000	0.3585	0.3804
Random-14B	0.5620	0.3781	0.3525	0.4100	0.3690	0.4033

D.2 THRESHOLD τ

The threshold τ controls the dimensionality of the protected safety subspace: smaller τ retains more singular vectors, yielding a larger subspace with stronger protection but greater constraint on capability updates. We ablate τ across five values under AZR (Qwen2.5-7B), average ASR, and average capability scores.

As shown in Table 8, the two objectives respond to τ on different schedules. Safety improves steadily as τ decreases from 5×10^{-4} to 5×10^{-5}, as the subspace progressively covers the principal safety-critical directions, and then saturates: below 5×10^{-5}, enlarging the subspace further buys no measurable reduction in ASR. Capability is flat across this same range and then falls at 1×10^{-5}, where the subspace has grown large enough to encroach on capability-driving directions. We adopt $\tau = 5 \times 10^{-5}$ as the default, which balances the two objectives across the range we examined.

Table 8: Effect of the threshold τ on average safety, code, and math performance. The selected configuration is highlighted in bold.

τ	ASR (avg)↓	Code (avg)↑	Math (avg)↑
No SEGO	0.3985	0.4778	0.4475
5e-4	0.3781	0.5126	0.4634
1e-4	0.3674	0.5021	0.4576
5e-5	**0.3410**	**0.5074**	**0.4530**
1e-5	0.3405	0.4578	0.4331
5e-6	0.3386	0.4490	0.4228

D.3 ADAPTIVE SUBSPACE RECOMPUTATION

SEGO models the safety subspace as a function of the evolving parameters and triggers recomputation when cumulative parameter displacement exceeds a threshold γ (Section 2.3.3). We ablate this mechanism by comparing five recomputation strategies under AZR (Qwen2.5-7B): recomputing at every training step, every 10 steps, adaptively at approximately every 50 steps (the default), every 200 steps, and never (a fixed subspace throughout training).

As shown in Table 9, capability scores remain largely stable across all strategies, confirming that recomputation frequency does not meaningfully affect the evolutionary signal. Safety, however, is

sensitive to it, and the relationship is non-monotonic: ASR is lowest at the adaptive default and rises in both directions away from it. Recomputing too rarely leaves the subspace stale, and a fixed subspace computed once from the pre-evolution model gives up most of SEGO's protection, approaching the ASR of unconstrained evolution; even recomputing every 200 steps recovers much of that loss. Recomputing too often is also counterproductive, consistent with the subspace oscillation effect described in Section 2.3.3: when the model has not moved substantially between recomputations, the newly computed subspace may differ from the previous one primarily due to statistical noise, intermittently exposing safety-critical parameters.

We therefore set γ so that recomputation is triggered roughly every 50 steps, which offers the best trade-off in this ablation: the subspace is refreshed only once the model has moved far enough to warrant it, tracking representational change at the pace of evolution itself.

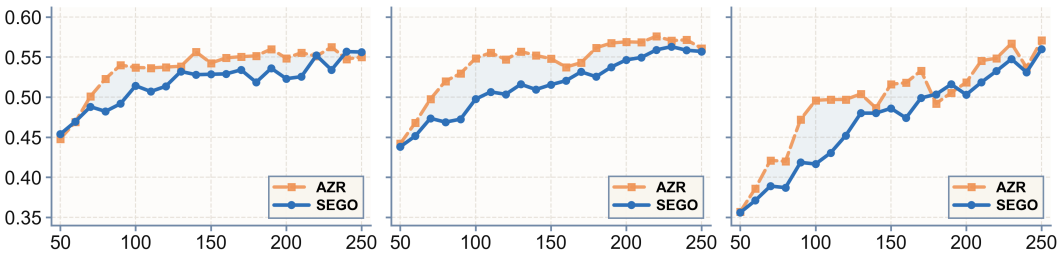
Table 9: Effect of the recomputation schedule on average safety, code, and math performance. The selected configuration is highlighted in bold.

Recomputation	ASR (avg) ↓	Code (avg) ↑	Math (avg) ↑
Every step	0.3564	0.4931	0.4475
Every 10 steps	0.3495	0.5025	0.4567
Every ∼50 steps (adaptive)	**0.3410**	**0.5074**	**0.4530**
Every 200 steps	0.3592	0.5101	0.4493
Fixed (never)	0.3887	0.5063	0.4508

E Training Convergence Under SEGO

A natural concern is whether SEGO's gradient orthogonalization slows capability growth by removing part of the evolutionary signal. Figure 6 compares the in-distribution capability curves of SEGO and standard AZR (Qwen2.5-7B) on fixed random subsets of CRUXEval-I, CRUXEval-O, and LiveCodeBench-Execution throughout training.

SEGO exhibits a slightly slower capability rise in the early phase of training (steps 50 to 130), with a gap of up to approximately 8 percentage points behind standard AZR. This is expected: the orthogonalization removes the safety-subspace component from the gradient, leaving the solver to find capability-improving directions within the remaining subspace, which initially constrains the optimization trajectory. However, the gap narrows steadily as training progresses and effectively closes by around step 200, with both methods reaching comparable performance by the end of training. This indicates that SEGO constrains the speed of capability growth but not its ceiling: the capability-driving directions in the orthogonal complement of the safety subspace are sufficient to reach the same performance level, requiring only moderately more steps to do so.



(a) CruxEval-I **(b)** CruxEval-O **(c)** LiveCodeBench-Execution

Figure 6: Training Convergence Under SEGO.

F Baseline Comparison

This appendix reports the full results underlying Section 3.3, where SEGO is compared against three alternative safety strategies under AZR with Qwen2.5-7B: KL regularization toward the pre-evolution policy, single-direction gradient projection (SPF), and post-hoc DPO realignment applied

at three evolution depths. Table 10 reports attack success rates, Table 11 coding accuracy, and Table 12 mathematical reasoning accuracy. Each table lists the pre-evolution model first, followed by unconstrained AZR at three evolution depths, the two in-process baselines, the realigned checkpoints, and SEGO last. AZR-Step300 is the checkpoint reported as AZR in the main results; listing it alongside Step400 and Step500 allows each realigned model to be read against the checkpoint it was derived from.

Table 10: Attack success rate on six safety benchmarks for the baseline comparison of Section 3.3. Lower is better; best per column in **bold**.

Model	PKU ↓	AdvB ↓	HB ↓	JBB ↓	SB ↓	HEx ↓
Qwen2.5-7B	0.2930	0.2538	**0.4700**	**0.4000**	0.2760	0.4033
AZR-Step300	0.3180	0.3538	0.5350	0.4700	0.2910	0.4233
AZR-Step400	0.3230	0.3647	0.5372	0.4700	0.2830	0.4167
AZR-Step500	0.3310	0.3604	0.5275	0.4800	0.2850	0.4300
AZR-KL	0.3260	0.3154	0.5275	0.4500	0.2945	0.4133
AZR-SPF	0.3190	0.3673	0.5850	0.5400	0.2905	0.4333
AZR-Step300 + DPO	0.3070	0.3419	0.5150	0.4500	0.2580	0.3967
AZR-Step400 + DPO	0.3250	0.3528	0.5200	0.4500	0.2740	0.4073
AZR-Step500 + DPO	0.3380	0.3534	0.5325	0.4700	0.2690	0.4292
SEGO	**0.2920**	**0.2288**	0.4725	0.4300	**0.2525**	**0.3700**

Table 11: Code generation and code reasoning accuracy for the baseline comparison of Section 3.3. Higher is better; best per column in **bold**.

Model	LCB ↑	C-I ↑	C-O ↑	MBPP ↑	HE ↑	HE+ ↑	MBPP+ ↑
Qwen2.5-7B	0.0994	0.3613	0.3937	0.4020	**0.7988**	**0.7607**	0.3438
AZR-Step300	0.1228	0.4275	0.3862	0.4960	0.7744	0.7178	0.4196
AZR-Step400	**0.1287**	0.4087	0.3950	0.5140	0.7678	0.7510	**0.5112**
AZR-Step500	0.1228	**0.4387**	0.4100	0.5280	0.7683	0.7117	0.4795
AZR-KL	0.1040	0.3763	0.3650	0.4380	0.7561	0.7055	0.4598
AZR-SPF	0.1170	0.4275	0.4113	0.4980	0.7683	0.7178	0.5045
AZR-Step300 + DPO	0.1199	0.3987	0.3450	0.4360	0.7327	0.7301	0.3973
AZR-Step400 + DPO	0.1111	0.3625	0.3525	0.4520	0.7344	0.7239	0.3616
AZR-Step500 + DPO	0.1082	0.3750	0.3550	0.4740	0.7500	0.7055	0.3750
SEGO	0.1228	0.4225	**0.4188**	**0.5440**	0.7866	0.7485	0.5089

Table 12: Mathematical reasoning accuracy for the baseline comparison of Section 3.3. Higher is better; best per column in **bold**.

Model	M500↑	GSM↑	OB↑	Min.↑
Qwen2.5-7B	0.5400	0.5428	0.3294	0.1213
AZR-Step300	0.6880	0.5619	0.3561	0.1838
AZR-Step400	0.6860	0.5865	**0.3665**	**0.2022**
AZR-Step500	**0.6960**	0.5836	0.3620	0.1901
AZR-KL	0.5900	0.5664	0.3427	0.1324
AZR-SPF	0.6820	0.5778	0.3650	0.1654
AZR-Step300 + DPO	0.6140	0.5142	0.3264	0.1422
AZR-Step400 + DPO	0.6480	0.5423	0.3394	0.1616
AZR-Step500 + DPO	0.6570	0.5635	0.3472	0.1647
SEGO	0.6740	**0.5967**	0.3631	0.1781

G MULTI-SEED RESULTS

For the main results in Tables 1 and 2, we repeated training and evaluation with three different random seeds; Tables 13 and 14 report the standard deviations across seeds.

Table 13: Three-seed standard deviations for safety and over-refusal results in Table 1.

Model	**Safety**						**Over-refusal**			
	PKU	AdvB	HB	JBB	SALAD	HEx	XST-S	XST-U	ORB	JBB-B
AZR Series										
Qwen2.5-7B	0.0092	0.0136	0.0150	0.0179	0.0092	0.0159	0.0087	0.0146	0.0068	0.0178
AZR-Base7B	0.0110	0.0179	0.0174	0.0219	0.0086	0.0163	0.0094	0.0219	0.0081	0.0190
SEGO-7B	0.0089	0.0114	0.0128	0.0205	0.0098	0.0192	0.0088	0.0151	0.0087	0.0166
Qwen2.5-Coder-3B	0.0098	0.0144	0.0134	0.0192	0.0068	0.0167	0.0079	0.0155	0.0058	0.0187
AZR-Coder3B	0.0123	0.0167	0.0147	0.0258	0.0084	0.0213	0.0095	0.0215	0.0089	0.0214
SEGO-Coder-3B	0.0085	0.0149	0.0147	0.0201	0.0079	0.0159	0.0083	0.0193	0.0069	0.0179
Qwen2.5-Coder-7B	0.0087	0.0102	0.0136	0.0205	0.0084	0.0136	0.0103	0.0179	0.0087	0.0156
AZR-Coder7B	0.0126	0.0128	0.0178	0.0216	0.0105	0.0190	0.0102	0.0207	0.0080	0.0241
SEGO-Coder-7B	0.0092	0.0110	0.0139	0.0178	0.0093	0.0190	0.0081	0.0150	0.0087	0.0176
Qwen2.5-Coder-14B	0.0082	0.0135	0.0120	0.0181	0.0077	0.0157	0.0081	0.0164	0.0078	0.0142
AZR-Coder14B	0.0110	0.0182	0.0141	0.0240	0.0112	0.0175	0.0109	0.0233	0.0105	0.0164
SEGO-Coder-14B	0.0091	0.0116	0.0129	0.0207	0.0081	0.0139	0.0093	0.0194	0.0083	0.0198
R-Zero Series										
Qwen3-4B	0.0078	0.0099	0.0111	0.0181	0.0083	0.0175	0.0108	0.0154	0.0063	0.0158
R-Zero-4B	0.0101	0.0159	0.0191	0.0197	0.0085	0.0204	0.0113	0.0183	0.0073	0.0214
SEGO-4B	0.0090	0.0103	0.0129	0.0214	0.0086	0.0173	0.0099	0.0182	0.0081	0.0187
Qwen3-8B	0.0092	0.0091	0.0150	0.0139	0.0065	0.0143	0.0101	0.0132	0.0089	0.0177
R-Zero-8B	0.0092	0.0145	0.0184	0.0240	0.0081	0.0187	0.0108	0.0225	0.0083	0.0191
SEGO-8B	0.0078	0.0099	0.0153	0.0184	0.0073	0.0164	0.0092	0.0157	0.0079	0.0196

Table 14: Standard deviations across three random seeds for the code and math results in Table 2

Model	Code							Math			
	LCB	CRUX-I	CRUX-O	MBPP	HE	HE+	MBPP+	MATH-500	GSM8K	Olymp.	Minerva
AZR Series											
Qwen2.5-7B	0.0071	0.0085	0.0097	0.0119	0.0182	0.0177	0.0129	0.0139	0.0094	0.0128	0.0116
AZR-Base7B	0.0086	0.0134	0.0108	0.0166	0.0181	0.0195	0.0129	0.0164	0.0087	0.0139	0.0164
SEGO-7B	0.0088	0.0100	0.0089	0.0150	0.0166	0.0206	0.0149	0.0147	0.0080	0.0149	0.0155
Qwen2.5-Coder-3B	0.0076	0.0099	0.0094	0.0127	0.0172	0.0182	0.0107	0.0128	0.0087	0.0113	0.0127
AZR-Coder3B	0.0090	0.0123	0.0105	0.0164	0.0241	0.0187	0.0150	0.0130	0.0108	0.0160	0.0156
SEGO-Coder-3B	0.0084	0.0094	0.0114	0.0148	0.0202	0.0219	0.0138	0.0160	0.0099	0.0140	0.0169
Qwen2.5-Coder-7B	0.0065	0.0109	0.0099	0.0118	0.0202	0.0175	0.0104	0.0130	0.0073	0.0128	0.0143
AZR-Coder7B	0.0098	0.0121	0.0112	0.0153	0.0186	0.0172	0.0144	0.0146	0.0096	0.0168	0.0207
SEGO-Coder-7B	0.0090	0.0089	0.0109	0.0140	0.0178	0.0180	0.0126	0.0134	0.0084	0.0128	0.0147
Qwen2.5-Coder-14B	0.0078	0.0087	0.0116	0.0146	0.0149	0.0198	0.0115	0.0142	0.0074	0.0130	0.0154
AZR-Coder14B	0.0107	0.0112	0.0131	0.0145	0.0231	0.0191	0.0184	0.0159	0.0110	0.0152	0.0180
SEGO-Coder-14B	0.0081	0.0106	0.0124	0.0135	0.0180	0.0193	0.0158	0.0137	0.0073	0.0123	0.0173
R-Zero Series											
Qwen3-4B	0.0072	0.0086	0.0090	0.0112	0.0178	0.0163	0.0142	0.0127	0.0085	0.0124	0.0136
R-Zero-4B	0.0105	0.0130	0.0108	0.0182	0.0238	0.0176	0.0139	0.0173	0.0086	0.0177	0.0194
SEGO-4B	0.0075	0.0114	0.0095	0.0125	0.0152	0.0165	0.0140	0.0109	0.0071	0.0124	0.0152
Qwen3-8B	0.0065	0.0115	0.0100	0.0133	0.0141	0.0189	0.0120	0.0136	0.0077	0.0106	0.0126
R-Zero-8B	0.0104	0.0116	0.0121	0.0158	0.0187	0.0190	0.0150	0.0172	0.0100	0.0175	0.0214
SEGO-8B	0.0094	0.0106	0.0121	0.0116	0.0156	0.0162	0.0115	0.0133	0.0079	0.0148	0.0176