SKILLEVOREG: REGULARIZING AGENT SKILL EVOLUTION AGAINST OVERFITTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Language-model agents increasingly improve by converting execution experience into reusable external skills. Yet repeated skill updates form a learning process of their own: locally useful edits can accumulate into redundant or task-specific instructions, while new updates can disrupt behavior that previously worked. We study this problem as *skill-evolution overfitting* and introduce SKILLEVOREG, a general regularization framework for skill evolution inspired by anti-overfitting techniques in neural-network training. SKILLEVOREG combines training-time skill dropout, which perturbs update generation, and complexity-aware local regularization, which controls unnecessary structural growth, with causal counterexample validation (CCV), which provides targeted behavioral validation of candidate-specific regressions. We instantiate the framework across heterogeneous skill-evolution systems while retaining each system's native skill evolver and task evaluator. Across SkillOpt, SkillEvolBench, and ContinualSkillBench, SKILLEVOREG consistently controls skill-state growth while preserving competitive downstream capability, improves several transfer and later-stage evolution outcomes, and identifies update-level regressions that structural metrics alone cannot reveal. These results suggest that explicit regularization is a useful complement to increasingly capable skill updaters.

1 INTRODUCTION

Large language models (LLMs) are increasingly deployed as agents that interact with environments, invoke tools, and improve from experience through reusable external state (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2024a; Zhang et al., 2024; Wang et al., 2025; Xu et al., 2025). In particular, *agent skills*—reusable instructions, procedures, code, or structured resources that guide future executions—have emerged as a practical substrate for continual self-improvement. Recent systems therefore move beyond one-shot skill generation toward repeated *skill evolution*, treating external skill state as an object that can be revised from execution experience (Yang et al., 2026b; Ni et al., 2026; Zhang et al., 2026; Lei et al., 2026; Guan et al., 2026).

Repeated skill optimization, however, creates a generalization problem of its own. Each update is inferred from finite and often narrow recent experience, yet the resulting content persists and influences future behavior. Over time, this can produce *structural accumulation* of redundant or narrow rules, *semantic specialization* to incidental properties of recent tasks, or *update-induced behavioral regression* that disrupts behavior already supported by the previous skill. We refer to these phenomena collectively as *skill-evolution overfitting*. Importantly, they need not coincide: substantial structural growth can precede measurable performance loss, while a concise update can introduce a narrow regression without materially increasing skill size.

Recent work has begun to expose or mitigate individual aspects of this problem through consolidation, replay-based validation, and adaptive update control (Ni et al., 2026; Guan et al., 2026; Yang et al., 2026a; He & Yang, 2026; Li et al., 2026). We ask a broader question: *can general anti-overfitting principles be instantiated across otherwise heterogeneous skill-evolution procedures?*

This question has a close analogue in neural-network training, where optimization is complemented by mechanisms that perturb learning, control capacity, and test generalization beyond the examples directly optimized. Dropout reduces co-adaptation (Srivastava et al., 2014), complexity penalties discourage unnecessary capacity (Krogh & Hertz, 1991), and augmentation or adversarial training

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.

exposes models to informative input variations (Goodfellow et al., 2015; Zhang et al., 2018). Agent skills are discrete, language-generated artifacts rather than differentiable parameter vectors, so these techniques cannot be transferred mechanically. Their underlying regularization principles, however, provide a useful lens for controlling repeated skill updates.

We introduce SKILLEVOREG, a general regularization framework for skill evolution. Rather than prescribing a new updater, SKILLEVOREG regularizes both how candidate updates are generated and what structure becomes persistent. *Training-time skill dropout* perturbs the skill context used to generate updates, while *complexity-aware local regularization* controls unnecessary structural growth around candidate changes. Complementing these trajectory-shaping mechanisms, *causal counterexample validation* (CCV) uses candidate-conditioned paired validation to identify update-induced behavioral regressions that structural signals alone cannot reveal. The framework retains each system's native skill evolver and evaluator while adapting these regularization principles to its update semantics.

We evaluate SKILLEVOREG across SkillOpt, SkillEvolBench, and ContinualSkillBench, covering single-document optimization, later-stage multi-skill evolution, and continual library growth. Across these settings, SKILLEVOREG produces more compact skill states while preserving competitive downstream capability, improves several transfer and later-stage outcomes, and identifies behavioral regressions that structural metrics alone cannot reveal. Component analyses further support the complementary roles of generation-time perturbation, structural regularization, and behavioral validation.

Our contributions are:

- We introduce SKILLEVOREG, a general regularization framework for repeated skill evolution that targets overfitting in the evolution process while retaining the native updater. SKILLEVOREG combines training-time skill dropout and complexity-aware local regularization with CCV, which provides targeted behavioral validation for candidate-specific regressions.
- We instantiate SKILLEVOREG across SkillOpt, SkillEvolBench, and ContinualSkillBench, showing that the same regularization principles can be adapted to heterogeneous update semantics while controlling skill-state growth, preserving competitive downstream capability, and improving several transfer and later-stage evolution outcomes.

2 RELATED WORK

External agent state and skill evolution. Language-model agents increasingly improve through reusable external state, including reflections, memories, workflows, and executable skills (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2024a; 2025; Xu et al., 2025). Recent systems go further by treating procedural skill state as an object of iterative optimization or continual maintenance (Yang et al., 2026b; Ni et al., 2026; Zhang et al., 2026; Lei et al., 2026; Guan et al., 2026). Related prompt- and program-optimization methods similarly search over external language-model state (Pryzant et al., 2023; Yang et al., 2024; Guo et al., 2024; Fernando et al., 2024; Wang et al., 2024b; Khattab et al., 2024; Yuksekgonul et al., 2025). These lines of work primarily study how to acquire or improve external state; we study how repeated revision of that state should be regularized.

Reliable skill evolution. Several concurrent methods introduce safeguards against brittle skill updates. GSE combines cross-task consolidation with replay verification (Yang et al., 2026a); SkillCommit validates broader abstractions before commitment (He & Yang, 2026); SkillAdam uses optimization history and adaptive edit budgets (Li et al., 2026); and Trace2Skill consolidates trajectory-local lessons before skill formation (Ni et al., 2026). These approaches improve reliability within particular evolution procedures. SKILLEVOREG instead factors anti-overfitting into a complementary layer that can be instantiated around heterogeneous native updaters, targeting structural accumulation, specialization, and update-induced regression as distinct failure modes.

Regularization and behavioral validation. Our design draws on the complementary roles of dropout, capacity regularization, and robust validation in conventional learning (Srivastava et al., 2014; Krogh & Hertz, 1991; Goodfellow et al., 2015; Zhang et al., 2018). CCV is also related

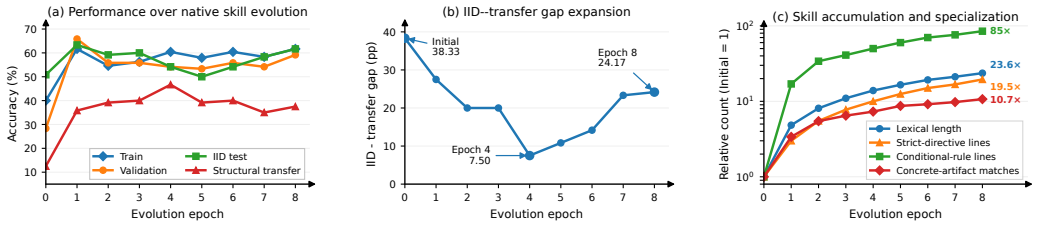


Figure 1: **A motivating longitudinal study of native skill evolution on SpreadsheetBench. (a)** Performance across evolution epochs. **(b)** The IID–transfer gap narrows and then re-expands after the transfer peak. **(c)** Skill structure continues to accumulate.

to counterexample-guided repair, where targeted failures expose weaknesses in candidate behavior (Orvalho et al., 2025). Rather than reproducing these techniques mechanically, SKILLEVOREG adapts their underlying roles to discrete, language-generated skill transitions. Appendix G discusses these connections in greater detail.

3 SKILL EVOLUTION AS A REGULARIZED LEARNING PROBLEM

We consider a generic skill-evolution system with skill state S_t, ranging from a single instruction document to a skill library or structured directory. Given an execution trajectory τ_t, the native updater U proposes

$$\widetilde{S}_{t+1} = U(S_t, \tau_t). \tag{1}$$

The host framework then applies its own validation and commit semantics; we assume no particular skill representation, updater, or commit rule.

Repeated application of this update creates a learning problem beyond immediate task performance: each update is inferred from finite recent trajectories yet persists to influence later tasks, allowing locally useful changes to accumulate, specialize, or interfere with previously supported behavior over time.

We use *skill-evolution overfitting* to describe this mismatch between narrow update evidence and persistent downstream influence. Three manifestations are particularly relevant. *Structural accumulation* adds rules, examples, or narrow exceptions without commensurate transferable benefit; *semantic specialization* promotes properties useful for recent trajectories into broadly applied instructions; and *update-induced regression* occurs when a new edit disrupts behavior already supported by the previous skill. These manifestations need not coincide: substantial structure may accumulate before aggregate performance deteriorates, while a concise edit may create a narrow regression without materially increasing skill complexity. Consequently, neither performance degradation nor skill complexity alone is sufficient to characterize skill-evolution overfitting.

3.1 A MOTIVATING LONGITUDINAL STUDY

We trace a SkillOpt trajectory on SpreadsheetBench for eight epochs, evaluating each checkpoint on an IID test set and a structural transfer-stress set with matched task semantics but greater workbook-level variation. Full split statistics and protocol details are provided in Appendix A.1.

Figure 1 shows a clear post-peak transfer pattern. Transfer performance initially improves and peaks at epoch 4 (46.67%), with the IID–transfer gap shrinking to 7.50 points. Continued evolution then favors familiar structure: by epoch 8, IID accuracy reaches 61.67% while transfer falls to 37.50%, widening the gap to 24.17 points.

Meanwhile, the skill continues to accumulate more specific structure after transfer has peaked, with lexical length reaching $23.6\times$ its initial value by epoch 8. Later updates thus keep reshaping the skill beyond its best transfer point, motivating explicit regularization of the evolution process.

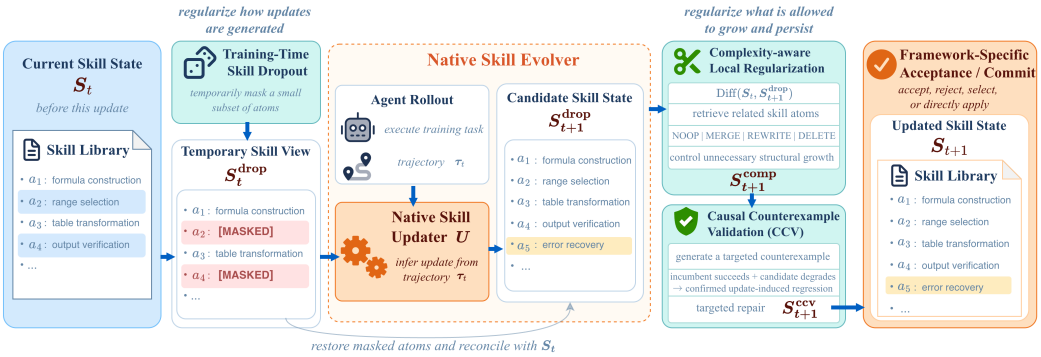


Figure 2: **Overview of SKILLEVOREG.** The three regularizers act on update generation, skill complexity, and behavioral validation, respectively. Colors distinguish the native skill evolver, SKILLEVOREG regularization modules and skill-states.

4 SKILLEVOREG

4.1 REGULARIZING SKILL EVOLUTION

The pilot suggests that the challenge is not simply producing useful skill updates, but controlling how repeated updates shape the skill over an evolution trajectory. We introduce SKILLEVOREG, a general framework that regularizes skill evolution while retaining the native updater. It acts at three complementary points: update generation, skill complexity, and behavioral validation.

The design draws inspiration from anti-overfitting mechanisms in neural-network training: dropout perturbs the information during learning, capacity regularization discourages unnecessary complexity, and validation under informative variations exposes brittle behavior. Because agent skills are discrete language-generated artifacts whose updates persist into future executions, these principles require operational adaptations. SKILLEVOREG instantiates the first two roles as training-time skill dropout and complexity-aware local regularization, which directly shape the evolution trajectory, and complements them with CCV, which validates the candidate behavior before persistence.

Let S_t denote the incumbent skill state at update step t. We use distinct symbols at regularization-stage boundaries: S_{t+1}^{drop} is the candidate after dropout reconciliation, S_{t+1}^{comp} is the candidate after complexity regularization, and S_{t+1}^{ccv} is the candidate after CCV. The native framework then applies its own acceptance or commit semantics to obtain the next persistent state S_{t+1}. Figure 2 shows where the three regularizers intervene. Across evolution systems, the main adaptation lies at the native update boundary, where candidate changes may appear as document edits, library patches, or explicit create/modify operations. Full pseudocode is provided in Appendix D.

4.2 TRAINING-TIME SKILL DROPOUT

Skill evolution repeatedly generates new updates while conditioning on the complete current skill state. This can encourage successive updates to depend too strongly on the precise presence or wording of existing instructions. We therefore perturb the skill context used during update generation.

We represent the current skill state as locally meaningful *skill atoms*,

$$\mathcal{A}(S_t) = \{a_1, \ldots, a_m\}, \tag{2}$$

where an atom is a self-contained instruction, list item, procedural block, or another semantically coherent unit. At update step t, we sample $M_t \subseteq \mathcal{A}(S_t)$ and construct a temporary dropout view

$$S_t^{\text{drop}} = \text{Mask}(S_t, M_t). \tag{3}$$

The training rollout and native update are generated under S_t^{drop}. If the native updater proposes $\widetilde{S}_{t+1}^{\text{drop}}$, we extract only the change learned under that perturbed context and reconcile it with the complete incumbent:

$$\Delta_t^{\text{drop}} = \text{Diff}\left(S_t^{\text{drop}}, \widetilde{S}_{t+1}^{\text{drop}}\right), \qquad S_{t+1}^{\text{drop}} = \text{Reconcile}\left(S_t, \Delta_t^{\text{drop}}\right). \tag{4}$$

Thus, masking changes only the context in which an update is learned; masked atoms are restored before later regularization and are never treated as candidate deletions. Frozen evaluation always uses the complete skill state. Algorithm 1 in Appendix D gives the complete procedure.

Remark (Relation to neural-network dropout). The analogy lies in when the perturbation acts. Neural dropout perturbs the representation through which a parameter update is learned; skill dropout perturbs the external skill context through which a skill update is generated. In both cases, the perturbation affects learning but is absent from the persistent state used at inference. Unlike inverted neural dropout, no activation-rescaling analogue is needed because skill atoms are discrete contextual units rather than additive activations.

4.3 Complexity-Aware Local Regularization

Skill dropout changes how an update is generated but does not directly control how much structure becomes persistent. Skill state has no differentiable parameter norm, and additional content is not inherently harmful because genuinely new capability may require additional structure. We therefore use observable complexity to determine *when structural reconsideration is needed*, while separately constraining *what content may be changed*, rather than minimizing skill size itself.

Given the incumbent S_t and reconciled candidate S_{t+1}^{drop}, we define the change examined by this stage as

$$\Delta_t^{\mathrm{comp}} = \mathrm{Diff}\left(S_t, S_{t+1}^{\mathrm{drop}}\right). \tag{5}$$

Let $\phi(Z)$ summarize observable structural properties of a skill state or update, such as lexical size, atom organization, structural fragmentation, and executable-code structure. The regularizer computes

$$c_t = f_{\mathrm{comp}}\left(\phi(S_t), \phi(S_{t+1}^{\mathrm{drop}}), \phi(\Delta_t^{\mathrm{comp}})\right), \tag{6}$$

where f_{comp} maps these measurements to the framework-specific signal used to trigger or score regularization. Exact feature definitions and realizations of f_{comp} are given in Appendix D.3.1.

When structural reconsideration is triggered, we restrict editing to atoms introduced or modified by the candidate and a bounded set of related incumbent atoms:

$$\mathcal{D}_t = \mathrm{SegmentAtoms}(\Delta_t^{\mathrm{comp}}), \qquad \Omega_t = \mathcal{D}_t \cup \mathrm{Related}(S_t, \mathcal{D}_t). \tag{7}$$

This locality constraint prevents a structurally large candidate from triggering an unrestricted rewrite of the complete skill state. A diagnostic SpreadsheetBench case shows that prompt-level locality alone can still yield broad, destructive rewrites, motivating the explicit candidate-local edit scope used here; Appendix E.1 provides the detailed analysis.

Inspired by operation-based memory reconciliation in Mem0 (Chhikara et al., 2025), we adapt discrete edit decisions to post-update skill reconciliation. For each candidate atom $a \in \mathcal{D}_t$, the local editor selects

$$o_a \in \{\textsc{NoOp}, \textsc{Merge}, \textsc{Rewrite}, \textsc{Delete}\}. \tag{8}$$

NoOp preserves useful new structure, while the other operations consolidate, revise, or remove candidate content within Ω_t. Similarity is used only to define what should be considered jointly, not as evidence that two atoms should be merged. We denote the resulting candidate by S_{t+1}^{comp}; if no intervention is triggered, $S_{t+1}^{\mathrm{comp}} = S_{t+1}^{\mathrm{drop}}$. Algorithm 2 gives the complete procedure.

Remark (Relation to capacity regularization). The analogy to neural-network capacity regularization is functional rather than literal. Parametric penalties discourage unnecessary capacity through continuous norms; SkillEvoReg instead uses observable structural complexity to trigger local reconsideration of a discrete candidate update. In both cases the goal is to limit unnecessary capacity accumulated from finite experience, while NoOp explicitly allows additional skill structure when it carries distinct useful behavior. Complexity reduction is therefore a means of regularization, not the objective itself.

4.4 CAUSAL COUNTEREXAMPLE VALIDATION

Structural regularization cannot detect every harmful update. A concise candidate may add little complexity while introducing an assumption or decision boundary that breaks behavior already supported by the incumbent. CCV therefore asks whether the structurally regularized candidate S_{t+1}^{comp} deteriorates on a targeted variation that the incumbent S_t can still handle.

The key comparison is

	original source e_t	attacked case x_t^{ccv}	
incumbent S_t	✓	✓	$\Longrightarrow$ CCV-confirmed regression.
candidate S_{t+1}^{comp}	✓	×	

(9)

Here ✓ and × denote evaluator-dependent qualification and degradation rather than necessarily binary success and failure.

CCV constructs this comparison in three steps. First, it selects an eligible source from the available training history $\mathcal{H}_t$,

$$e_t = \text{SelectSource}\left(\mathcal{H}_t, S_t, S_{t+1}^{\text{comp}}\right). \tag{10}$$

The incumbent must already support e_t, and the candidate must remain qualified on the same unmodified source. This clean-pair requirement rules out candidates that have already broken the original task.

Second, because CCV acts after complexity regularization, it defines its own stage-local candidate change and uses it to construct the attack:

$$\Delta_t^{\text{ccv}} = \text{Diff}(S_t, S_{t+1}^{\text{comp}}), \quad a_t = \text{GenerateAttack}(e_t, \Delta_t^{\text{ccv}}), \quad x_t^{\text{ccv}} = \text{ApplyAttack}(e_t, a_t). \tag{11}$$

Here a_t specifies a targeted perturbation of an assumption, dependency, or decision boundary exposed by Δ_t^{ccv}, and x_t^{ccv} is the resulting attacked case.

Third, the incumbent and candidate are evaluated independently on that same case:

$$\begin{aligned} r_{\text{old}} &= \text{Evaluate}(S_t, x_t^{\text{ccv}}), \qquad r_{\text{new}} = \text{Evaluate}(S_{t+1}^{\text{comp}}, x_t^{\text{ccv}}), \\ g_t &= \text{RegressionCriterion}(r_{\text{old}}, r_{\text{new}}; \theta_{\text{ccv}}). \end{aligned} \tag{12}$$

where θ_{ccv} denotes evaluator-specific qualification and degradation criteria. A regression is confirmed only when the incumbent remains sufficiently successful on the attacked case while the candidate deteriorates. When $g_t = 1$, CCV performs at most one scope-constrained repair over the implicated candidate region; otherwise the candidate is unchanged. We denote the resulting CCV-processed candidate by S_{t+1}^{ccv}. Exact qualification rules, attack construction, and repair procedures are given in Appendix D.4.

Remark (Relation to data augmentation). CCV adapts the augmentation principle to a skill transition: the candidate update itself reveals where generalization may have narrowed, and the attack perturbs the source along that direction. The resulting case is therefore a targeted stress test of the behavioral boundary moved by the update, used primarily for paired validation and, when needed, scope-constrained repair rather than simply being added back to training.

Remark (Why causal?). The four-way comparison in Equation 9 provides the basis for the causal terminology. Qualification of both skills on the original source controls for ordinary source-task failure, while incumbent success on the attacked case controls for an attack that is simply too difficult. Candidate-specific degradation on that same case therefore associates the regression with the candidate transition. We use *CCV-confirmed regression* in this operational sense, not as a claim of formal causal identification under stochastic rollouts or multi-edit candidates.

4.5 ADAPTING TO DIFFERENT UPDATE SEMANTICS

The regularizers retain the same roles across evolution systems; the main adaptation lies at the native update boundary. SkillOpt exposes ranked edits to a single skill document, SkillEvolBench localized patches over a multi-skill library, and ContinualSkillBench explicit `CREATE`/`MODIFY` operations.

These semantics determine how stage-specific candidate changes are exposed, how dropout updates are reconciled with the incumbent, how editable scopes are defined, and how S_{t+1}^{ccv} is returned to the native framework. Appendix D details the framework-specific integrations.

5 EXPERIMENTS

We evaluate whether the same regularization principles improve three distinct forms of skill evolution: repeated optimization of a single skill, later-stage revision of an established multi-skill library, and native continual skill-library growth. We then analyze the roles of the individual regularization components.

5.1 EXPERIMENTAL SETUP

All compared methods use the same proprietary instruction-following LLM under a fixed inference configuration and each benchmark's native execution harness, task order, and evaluator. SKILLEVOREG is inserted at the candidate-update boundary while retaining the native skill evolver and evaluation protocol. We report downstream capability together with skill-state size and structural complexity C; where checkpoint selection is available, we distinguish the validation-selected *delivery* checkpoint from the terminal *final* checkpoint. Full benchmark protocols and measurement conventions are given in Appendix A.2–A.3. Frozen-evaluation inference usage is analyzed separately in Appendix C.4 as a deployment-side by-product of compact persistent skill states rather than an optimization objective of SKILLEVOREG.

5.2 SKILLOPT: EPISODIC SINGLE-SKILL OPTIMIZATION

SkillOpt (Yang et al., 2026b) repeatedly optimizes a single instruction document. We evaluate SpreadsheetBench, SearchQA, and LiveMathBench over eight evolution epochs, retaining SkillOpt's native updater and checkpointing procedure while regularizing candidate transitions. Full protocol and checkpoint details are given in Appendix A.2.

Table 1: SkillOpt results. Delivery denotes the validation-selected checkpoint and Final the epoch-8 checkpoint; Δ Test is Final minus Delivery. Skill tokens and complexity describe the final skill state.

Dataset	Method	Delivery Test ↑	Final Test ↑	Δ Test ↑	Skill tokens ↓	Complexity ↓
Spreadsheet	SkillOpt	54.75	47.00	−7.75	6,060	116.96
	SKILLEVOREG	**58.50**	**59.25**	**+0.75**	**2,341**	**57.21**
SearchQA	SkillOpt	80.46	**80.86**	**+0.39**	7,924	124.42
	SKILLEVOREG	**81.14**	80.39	−0.75	**1,833**	**39.51**
LiveMath	SkillOpt	34.79	**36.41**	**+1.61**	6,749	110.52
	SKILLEVOREG	**35.02**	30.18	−4.84	**1,606**	**36.52**

Capability and evolution quality. At the delivery checkpoint, SKILLEVOREG matches or improves all three tasks while producing substantially smaller final skill states. SpreadsheetBench continues to improve through the terminal checkpoint, while SearchQA remains essentially stable. LiveMath instead exposes a distribution-sensitive trade-off. SkillOpt repeatedly reinforces a meta-option prior favoring the answer that "a stronger result can be proved," whereas SKILLEVOREG consolidates repeated reinforcement of this pattern. Of the 217 test questions, 92 use this meta-option as the correct answer, and this subset accounts for 85.7% of SKILLEVOREG's late-stage decline. The performance gap is therefore strongly concentrated in a specific test-distribution pattern rather than distributed uniformly across questions. Appendix C.1 provides the item-level analysis.

5.3 SKILLEVOLBENCH: REGULARIZING LATER-STAGE SKILL EVOLUTION

SkillEvolBench (Lei et al., 2026) evaluates transfer under context shift, adversarial variation, and skill composition. Its native protocol uses a single acquisition pass. We add a controlled second pass

to emulate later-stage skill evolution, where an already formed library continues to absorb repeated experience and may begin to over-specialize. Starting from the same native one-pass library, we compare freezing it (1-pass), continuing with a second unregularized pass (2-pass), and regularizing that same second pass with SKILLEVOREG. The two second-pass conditions therefore receive identical additional experience, while deployment tasks remain unseen throughout acquisition. Details are given in Appendix B and Appendix C.2.

Later-stage transfer and controlled growth. A second unregularized pass illustrates the risk of repeated skill evolution: the library continues to grow, while deployment success decreases from 31.67% to 29.44%. Applying SKILLEVOREG to the same additional experience instead reaches 33.33% deployment success while substantially controlling persistent-state growth. Relative to the one-pass anchor, SKILLEVOREG reduces second-pass token growth by 34.0% and complexity growth by 32.7% compared with unregularized two-pass evolution.

Table 2: SkillEvolBench results. T1–T3 are replay tasks and T4–T6 are unseen deployment tasks.

Metric	1-pass	2-pass	Ours
Replay (T1–T3) ↑	40.00	38.89	**42.22**
T1 Canonical ↑	43.33	40.00	**48.33**
T2 Enriched ↑	**41.67**	36.67	38.33
T3 Variant ↑	35.00	**40.00**	**40.00**
Deployment (T4–T6) ↑	31.67	29.44	**33.33**
T4 Context shift ↑	36.67	**40.00**	31.67
T5 Adversarial ↑	40.00	35.00	**41.67**
T6 Composition ↑	18.33	13.33	**26.67**
Skill tokens ↓	30,828	36,921	34,847
Complexity C ↓	609.20	696.46	667.97

Transfer behavior. The improvement is not uniform across every transfer mode. SKILLEVOREG gives the strongest gain on composition (26.67% versus 18.33% after one pass and 13.33% after two unregularized passes) and improves adversarial transfer, while context-shift success decreases. Together with the controlled skill-state growth, this suggests that regularization does not simply suppress second-pass learning, but changes which parts of repeated experience are allowed to persist. Continuous verifier scores and additional structural diagnostics are reported in Appendix C.2.

5.4 CONTINUALSKILLBENCH: CONTINUAL SKILL-LIBRARY EVOLUTION

ContinualSkillBench (Guan et al., 2026) extends the setting further by allowing the evolving library to create new skills as well as modify existing ones through native CREATE-SKILL and MODIFY-SKILL actions. This makes library proliferation itself part of the regularization problem. Each condition learns sequentially on tasks 1–50, freezes the resulting library, and evaluates tasks 51–100 without further learning. SKILLEVOREG regularizes both CREATE and MODIFY outcomes before write-back while retaining the benchmark's native update semantics. Complete protocol and online-learning results are given in Appendix C.3 and Appendix B.

Table 3: ContinualSkillBench frozen-library results after learning tasks 1–50. Scores average two evaluations of held-out tasks 51–100; skill-state statistics use the complete step-50 library.

Domain	Method	Held-out capability ↑			Skill state ↓		
		Overall	Struct.	Rubric	# Skills	Tokens	Complexity
Mathematics	Baseline	67.90	74.00	61.80	33	21,952	612.26
	SKILLEVOREG	**69.50**	74.00	**65.00**	**32**	**16,368**	**438.30**
Law	Baseline	65.42	81.82	60.79	24	25,826	645.78
	SKILLEVOREG	**70.69**	**86.36**	**66.27**	**20**	**19,941**	**507.97**
Finance	Baseline	68.63	65.58	69.59	28	78,193	1,596.72
	SKILLEVOREG	**69.10**	**65.95**	**70.09**	**17**	**71,715**	**1,436.19**
Office	Baseline	**82.49**	**78.57**	**85.33**	18	19,038	502.45
	SKILLEVOREG	81.05	76.19	84.57	**17**	**16,087**	**414.60**
Healthcare	Baseline	70.92	**66.67**	72.27	42	33,646	822.07
	SKILLEVOREG	**73.12**	62.50	**76.47**	**34**	**28,416**	**727.60**

Held-out capability and consolidation. SKILLEVOREG maintains comparable or better Overall held-out performance across the five domains while consistently producing more compact persistent skill states. Skill tokens, complexity, and skill count decrease in every domain. The reduction is not simply uniform pruning: Mathematics primarily shortens existing skills, whereas the other domains show stronger suppression of library proliferation. Together, these results show that regularization controls persistent-state accumulation without broadly sacrificing held-out capability. Evaluator-level breakdowns and sensitivity analyses are provided in Appendix C.3.

5.5 COMPONENT ANALYSIS

We next examine how dropout and complexity regularization shape the evolution trajectory, and how CCV complements them with targeted behavioral validation.

5.5.1 DROPOUT AND COMPLEXITY REGULARIZATION ARE COMPLEMENTARY

To isolate the roles of skill dropout and complexity-aware regularization, we conduct a controlled SpreadsheetBench ablation with four variants: the native updater, complexity regularization only, skill dropout only, and the full SKILLEVOREG. All variants use the same eight-epoch evolution budget, training data, native updater, validation procedure, and frozen evaluation; only the active regularizers differ, and we compare final checkpoints to ensure equal evolution budgets.

Table 4: SpreadsheetBench ablation at the final checkpoint.

Method	**Test** ↑	**Chars.** ↓
Native updater	47.00	31.7K
Complexity only	55.50	13.1K
Dropout only	57.50	19.5K
Full SKILLEVOREG	**59.25**	**11.7K**

The two regularizers exhibit complementary effects. Complexity regularization substantially limits persistent-state growth, while dropout alone attains higher test accuracy but allows a larger skill state. Their combination yields both the smallest regularized skill and the strongest frozen score, consistent with dropout shaping how updates are generated and complexity regularization controlling what structure is allowed to persist. CCV does not modify the persistent state on this particular trajectory, so we examine its behavioral role separately.

5.5.2 CCV DETECTS REGRESSIONS BEYOND STRUCTURAL SIGNALS

CCV provides a complementary behavioral signal: even a concise update can disrupt behavior already supported by the incumbent without producing a distinctive structural signature. Across the evaluated trajectories, we identify 12 distinct CCV regression signals and 9 state-changing repairs. A representative SkillEvolBench dependency-resolution trace illustrates the mechanism: after the incumbent and candidate both qualify on the original source, a candidate-conditioned package-boundary attack yields reward 1.0 for the incumbent and 0.8 for the candidate. CCV identifies candidate-specific sensitivity to the package-boundary condition and produces a scope-constrained rewrite; a later replay recovers the affected process checks. Appendix E.2 provides additional regression evidence and a detailed representative trace.

6 CONCLUSION

Repeated skill evolution creates a learning problem beyond the native updater. We introduced SKILLEVOREG, a general framework that translates complementary anti-overfitting principles from neural-network training to discrete skill evolution: training-time skill dropout perturbs update generation, complexity-aware local regularization controls persistent structural growth, and CCV provides targeted behavioral validation of candidate-specific regressions. Across SkillOpt, SkillEvolBench, and ContinualSkillBench, SKILLEVOREG limits skill-state growth while preserving downstream capability and improving several transfer and later-stage outcomes. More broadly, these results suggest that regularization principles developed for parametric learning provide a useful lens for controlling persistent, language-generated skill updates, and support regularization as a first-class component of skill evolution.

AI USE STATEMENT

In this work, we used generative AI tools to help develop conceptual frameworks, design and provide feedback on research methodology and experiments, implement methods, and interpret results. Additionally, we used generative AI tools to create or edit software code, draft and edit parts of the paper, summarize and analyze existing literature, and assist with research result auditing. We have reviewed all AI-assisted work. Generated code and experimental analyses were checked against the implementation, execution logs, and evaluation artifacts, and manuscript claims were reviewed against the corresponding experimental evidence. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the experimental protocols, data splits, evaluation procedures, and main implementation details in Section 5 and Section 4. Additional implementation details, framework-specific regularization criteria, measurement conventions, complexity definitions, and supporting analyses are provided in the appendix. We will also release the implementation, configuration files, and scripts needed to reproduce the main experiments.

REFERENCES

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. In *Proceedings of the 28th European Conference on Artificial Intelligence*, 2025.

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 13481–13544. PMLR, 2024.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

Tianyi Guan, Yiding Wang, Haotong Yang, Siyuan Cao, Shirui Liu, Yi Hu, Jiaqi Li, and Muhan Zhang. Continualskillbench: Can llm agents truly evolve their capabilities?, 2026. URL https://arxiv.org/abs/2608.03874.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujia Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations*, 2024.

Yu He and Weikai Yang. Skillcommit: Evolving agent skills through behaviorally validated scope expansion, 2026. URL https://arxiv.org/abs/2608.15165.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *International Conference on Learning Representations*, 2024.

Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems*, volume 4, pp. 950–957, 1991.

Yingtie Lei, Zhongwei Wan, Jiankun Zhang, Samiul Alam, Zixuan Zhong, Peizhou Huang, Xin Wang, Jingxuan Zhang, Donghao Zhou, Yunta Hsieh, Zhihao Dou, Hui Shen, Yan Xu, Dimitrios Dimitriadis, Tuo Zhang, and Mi Zhang. Skillevolbench: Benchmarking the evolution from episodic experience to procedural skills, 2026. URL https://arxiv.org/abs/2605.24117.

Gaoyuan Li, Meihao Fan, Yizhe Liu, Shaolei Zhang, Ju Fan, Siyi Wang, Jiaheng Hou, Xudong Weng, Honghan Tian, and Zang Li. Skilladam: Stable and efficient skill evolution for agents, 2026. URL https://arxiv.org/abs/2609.08944.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2skill: Distill trajectory-local lessons into transferable agent skills, 2026. URL https://arxiv.org/abs/2603.25158.

Pedro Orvalho, Mikoláš Janota, and Vasco M. Manquinho. Counterexample guided program repair using zero-shot learning and MaxSAT-based fault localization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 649–657, 2025.

Lutz Prechelt. Early stopping—but when? In Genevieve B. Orr and Klaus-Robert Müller (eds.), *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pp. 55–69. Springer, Berlin, Heidelberg, 1998.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7957–7968. Association for Computational Linguistics, 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*, volume 36, 2023.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P. Xing, and Zhiting Hu. PromptAgent: Strategic planning with language models enables expert-level prompt optimization. In *International Conference on Learning Representations*, 2024b.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Chen Yang, Jiashuo Tian, Ziqi Wang, Xinyin Liu, Meiru Ye, and Junjie Chen. Learning globally reusable skills for coding agents, 2026a. URL https://arxiv.org/abs/2608.06153.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. Skillopt: Executive strategy for self-evolving agent skills, 2026b. URL https://arxiv.org/abs/2605.23904.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative AI by backpropagating language model feedback. *Nature*, 639:609–616, 2025.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue, Liu, Xiaoxiao Li, and Philip S. Yu. CoEvoSkills: Self-evolving agent skills via co-evolutionary verification, 2026. URL https://arxiv.org/abs/2604.01687.

Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

Wenqi Zhang, Ke Tang, Hai Wu, Mengna Wang, Yongliang Shen, Guiyang Hou, Zeqi Tan, Peng Li, Yueting Zhuang, and Weiming Lu. Agent-pro: Learning to evolve via policy-level reflection and optimization. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 5348–5375. Association for Computational Linguistics, 2024.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

A Additional Experimental Details

A.1 Motivating SpreadsheetBench Pilot

Purpose. The study in Section 3.1 is a longitudinal stress test of native skill evolution rather than a confirmatory evaluation of SkillEvoReg. The pilot is designed to examine how continued skill updates affect performance on familiar workbook structures and on structurally shifted workbooks while the skill state itself evolves. We therefore use this study as motivating evidence for the regularization problem; all comparative claims about SkillEvoReg are based on the independently specified experiments in Section 5.

IID and structural-transfer evaluation. We organize the held-out evaluation into an in-distribution (IID) set and a structural transfer-stress set. The two sets keep the primary task semantics closely matched while differing in workbook-level structure. They contain identical proportions of the benchmark's primary operation families and the same proportions of cell-level and sheet-level tasks. Their average operation counts, compositional-depth proxies, and nearest-training instruction similarities are also similar, indicating that the transfer set does not primarily introduce new task families, rarer operations, or substantially different instruction semantics. Instead, the main shift lies in the structure of the input workbooks.

Compared with the IID set, the structural-transfer set contains more multi-sheet workbooks ($12/40$ versus $5/40$), more workbooks with three or more sheets ($3/40$ versus $0/40$), more workbooks containing existing cross-sheet formulas ($8/40$ versus $1/40$), more tasks requiring cross-sheet reasoning ($13/40$ versus $6/40$), and more merged-cell workbooks ($8/40$ versus $2/40$). The mean number of sheets increases from 1.125 to 1.475, and the mean nearest-training workbook-structure distance increases from 0.882 to 1.676. We therefore use this set to evaluate transfer across workbook structure rather than transfer to unseen semantic task families.

Table 5: Characteristics of the two held-out SpreadsheetBench pilot sets. The two sets have closely matched task semantics, while the transfer set contains greater workbook-level structural variation.

Characteristic	IID	Structural transfer
Multi-sheet workbooks	5/40	12/40
Three-or-more sheets	0/40	3/40
Existing cross-sheet formulas	1/40	8/40
Cross-sheet requirements	6/40	13/40
Merged-cell workbooks	2/40	8/40
Mean number of sheets	1.125	1.475
Nearest-Train structural distance	0.882	1.676

Protocol. The frozen pilot contains 80 training tasks, 40 validation tasks, 40 IID-test tasks, and 40 structural-transfer tasks, with a further 200 SpreadsheetBench tasks excluded from this study. SkillOpt is run for eight epochs with seed 41 and a batch size of 20, yielding 32 skill updates. Intermediate validation-based stopping and other dynamic regularization mechanisms are disabled so that the complete native evolution trajectory can be observed.

We retain the initial skill and the final checkpoint of every epoch. Each checkpoint is evaluated on the same frozen training, validation, IID-test, and structural-transfer sets. The complete checkpoint trajectory is evaluated in three independent evaluation replays, and we report the mean across replays. These replays evaluate the same learned trajectory and therefore should not be interpreted as independent training runs.

Post-peak transfer behavior. Structural-transfer performance reaches its maximum of 46.67% at E4. From E4 to E8, train, validation, and IID accuracy change by $+1.25$, $+5.00$, and $+7.50$ percentage points, respectively, whereas transfer accuracy decreases by 9.17 points. The IID–transfer gap consequently widens from 7.50 to 24.17 percentage points. The decline is also distributed across tasks: among the 40 transfer tasks, 13 regress from E4 to E8, 22 remain unchanged, and 5 improve.

Table 6: Checkpoint performance in the motivating SpreadsheetBench pilot. Values are mean task accuracy across three independent evaluation replays of the same learned trajectory.

Checkpoint	Train	Val.	IID	Transfer	IID–Transfer
Initial	40.00	28.33	50.83	12.50	38.33
E1	61.67	65.83	63.33	35.83	27.50
E2	54.58	55.83	59.17	39.17	20.00
E3	56.25	55.83	60.00	40.00	20.00
E4	60.42	54.17	54.17	46.67	7.50
E5	57.92	53.33	50.00	39.17	10.83
E6	60.42	55.83	54.17	40.00	14.17
E7	58.33	54.17	58.33	35.00	23.33
E8	61.67	59.17	61.67	37.50	24.17

Table 7: Deterministic lexical diagnostics along the motivating pilot trajectory. These are descriptive proxies and are distinct from the complexity measure $C(S)$ used by SKILLEVOREG.

Checkpoint	Lexical length	Strict lines	Conditional lines	Artifact matches
Initial	402	4	1	13
E1	1,944	12	17	44
E2	3,252	22	34	71
E3	4,416	31	41	84
E4	5,586	40	50	95
E5	6,655	50	60	113
E6	7,728	60	70	119
E7	8,501	67	76	127
E8	9,494	78	85	139

Deterministic lexical diagnostics. To characterize how the skill changes along the same trajectory without relying on an LLM judge, we apply four deterministic lexical diagnostics to every checkpoint. These measures are descriptive proxies used only in this motivating study and are distinct from the operational complexity measure $C(S)$ used by SKILLEVOREG in Appendix A.2.

Lexical length counts word-like sequences, numeric tokens, and non-whitespace punctuation over the full skill document. *Strict-directive lines* count substantive lines containing directive expressions such as `must`, `always`, `never`, or `do not`. *Conditional-rule lines* count substantive lines containing conditional expressions such as `if`, `when`, `unless`, or `fallback`. *Concrete-artifact matches* count spreadsheet-specific cell or range references, error identifiers, selected code identifiers, spreadsheet-specific phrases, and selected functions. These diagnostics characterize observable textual structure rather than semantic quality.

Between the transfer-performance peak at E4 and the final checkpoint E8, lexical length increases by 69.9% and conditional-rule lines increase by 70.0%. Over the full trajectory, lexical length, strict-directive lines, conditional-rule lines, and concrete-artifact matches reach $23.6\times$, $19.5\times$, $85\times$, and $10.7\times$ their initial values, respectively.

Interpretation. Taken together, the pilot illustrates a post-peak transfer pattern: after structural-transfer performance peaks, continued skill evolution is accompanied by a widening IID–transfer gap and continued accumulation of increasingly specific skill structure. This pattern motivates treating repeated skill updates as a regularized learning process rather than relying on the updater alone.

A.2 SKILLOPT MEASUREMENT CONVENTIONS AND COMPLEXITY

We evaluate SpreadsheetBench ($100/100/200$ train/validation/test examples), SearchQA ($400/200/1400$), and LiveMathBench ($100/80/217$). All SkillOpt runs use eight evolution epochs, eight-example reflection minibatches, and an initial edit budget of four. The native baseline retains SkillOpt's slow-update and meta-skill mechanisms; the SKILLEVOREG integration retains the same ranked-edit proposal mechanism and applies regularization before the native acceptance decision.

Table 8: Final composite complexity C on the SkillOpt benchmarks.

Dataset	Method	Complexity $C \downarrow$	
		Value	**Relative to baseline**
SpreadsheetBench	SkillOpt	116.96	1.00
	SKILLEVOREG	**57.21**	**0.49**
SearchQA	SkillOpt	124.42	1.00
	SKILLEVOREG	**39.51**	**0.32**
LiveMath	SkillOpt	110.52	1.00
	SKILLEVOREG	**36.52**	**0.33**

Final checkpoints are evaluated after the fixed eight-epoch training budget. Delivery selects among the eight epoch-end snapshots using the incumbent validation score recorded at the last training update of each epoch, breaking ties in favor of the earlier epoch. The selected epochs for SkillOpt/SKILLEVOREG are 7/7 on SpreadsheetBench, 4/2 on SearchQA, and 5/1 on LiveMath. Their selection scores are $59.5/63.5\%$, $80.25/80.25\%$, and $43.125/38.125\%$, respectively. Test scores average two frozen evaluations of the exact selected snapshot.

For completeness, SearchQA token-overlap F1 is $87.98/88.68\%$ at delivery and $88.26/88.41\%$ at the final checkpoint for SkillOpt/SKILLEVOREG, respectively, averaged over the same two frozen evaluations as EM.

Skill length uses the `o200k_base` tokenizer on the full document. We segment skill documents into deterministic exact-span atoms and define

$$C(S) = \sum_{a_i \in \mathcal{A}(S)} \left(\frac{\tau_i}{100} + 0.2 + \frac{L_i}{20} + 2\frac{B_i}{5} + 2\frac{D_i}{3} \right), \tag{13}$$

where τ_i, L_i, B_i, and D_i denote the lexical-unit count, nonempty code-line count, control-flow count, and external-dependency count of atom a_i, respectively; code terms are zero for non-code atoms. The exact-span segmenter retains document scaffolding, and each returned atom receives the base cost. For code, L_i excludes empty lines and Markdown fence delimiters, B_i counts common control-flow constructs, and D_i counts recognized imports and selected HTTP dependency calls. The measure is an operational regularization signal rather than a semantic-quality score.

A.3 EVALUATION PROTOCOL CONVENTIONS

All reported performance values use the benchmark-native verifier and the same task order for compared methods. Frozen evaluations do not update the skill state. When a benchmark provides a validation stream, checkpoint selection uses only the recorded validation scores and the selected checkpoint is then evaluated on the frozen test stream. Main comparisons use one evolved state per condition; repeated frozen evaluations measure execution-time variation of that state rather than variation across independently trained trajectories.

B FRAMEWORK-SPECIFIC INSTANTIATIONS OF SKILLEVOREG

SKILLEVOREG is a general regularization framework rather than a single shared software core or a fixed pipeline implementation. The three benchmark integrations retain their native skill evolvers and task evaluators but differ in skill representation, complexity semantics, CCV case construction, repair scope, and final candidate handling.

Table 9: Framework-specific integration of SKILLEVOREG.

Framework	Native proposal	Complexity handling	Final candidate handling
SkillOpt	Ranked edits to one instruction document	Complexity contributes to the regularized validation gate	Regularized acceptance gate; current/best checkpoint bookkeeping retained
SkillEvol	SKILLAUTHOR patch over a multi-skill family	Growth budget can trigger bounded local regularization; budget alone does not reject	Native library commit API reused with staging, optional repair, and finalization
Continual	Native CREATE-SKILL/MODIFY-SKILL over a skill directory	CREATE uses an absolute budget and library-level comparison; MODIFY uses relative growth	No validation reject of the final regularized candidate; final snapshot is written back

Table 10: Framework-specific CCV instantiations used in our experiments.

Framework	CCV case construction	Regression signal	Repair scope
SkillOpt	Dataset-specific candidate-conditioned attacks over workbook, context, or reasoning inputs	Hard incumbent pass and candidate failure, with clean-candidate and dataset-specific attribution controls	Candidate delta and authorized related atoms
SkillEvol	Original instruction plus a bounded robustness appendix	Qualified clean pair, old attacked score ≥ 0.8, and attacked score drop ≥ 0.05	Changed skill and implicated local regions
Continual	Answer-preserving transformation or validated boundary case	Incumbent passes and candidate either fails or drops by more than 0.1	Responsible skill paths

Across all three integrations, the incumbent and candidate must first satisfy the framework-specific clean qualification, after which both are evaluated on the same attacked case under the same model configuration and benchmark-native evaluator. SkillOpt uses a strict hard-pass/hard-fail attacked-case criterion with additional attribution controls, whereas SkillEvolBench and ContinualSkillBench also allow a framework-defined continuous-score drop to trigger a regression signal.

C ADDITIONAL BENCHMARK RESULTS

C.1 LIVEMATH: A DISTRIBUTION-SENSITIVE ANSWER HEURISTIC

LiveMath provides a case in which suppressing repetitive specialization can trade off against accuracy on the current test distribution. SkillOpt's final skill repeatedly emphasizes the meta-option stating that a stronger result can be proved. The regularized final skill retains one explicit meta-option rule but consolidates overlapping strongest-statement and strongest-existence guidance.

Of the 217 test questions, 92 have the correct answer text "One of the remaining options is correct, but a stronger result can be proven." Across the two frozen evaluation repeats, SKILLEVOREG's delivery checkpoint answers 33 of 184 corresponding question–repeat instances correctly, whereas the final checkpoint answers 15. The remaining 125 questions change only from 119/250 to 116/250 correct. Thus, meta-option questions account for 18 of the 21 net lost correct answers: 4.15 of the overall 4.84 percentage-point decline, or 85.7%. Pairing delivery and final outcomes within each repeat yields 23 correct-to-incorrect transitions and five reverse transitions on this subset.

The concentration is consistent with reduced reinforcement of the meta-option heuristic, but the comparison does not isolate any individual rule edit as causal. It illustrates that regularization can suppress repetitive specialization even when the specialized pattern remains useful under the current evaluation distribution. The case therefore illustrates a genuine regularization trade-off: suppressing repeated specialization can reduce benchmark accuracy when the evaluation distribution itself rewards the specialized answer prior.

Table 11: Mean normalized SkillEvolBench verifier scores, averaged over two frozen evaluations. Scores range from 0 to 1.

Task block	1-pass	2-pass	SKILLEVOREG
T1–T3, Replay	0.8557	**0.8588**	0.8558
T1, Canonical replay	0.8604	0.8625	**0.8807**
T2, Enriched replay	**0.8484**	0.8439	0.8381
T3, Variant replay	0.8584	**0.8699**	0.8487
T4–T6, Deployment	**0.8299**	0.8263	0.8269
T4, Context shift	0.8334	**0.8447**	0.8227
T5, Adversarial	**0.8317**	0.8226	0.8297
T6, Composition	0.8247	0.8115	**0.8283**
Complete, T1–T6	**0.8428**	0.8425	0.8414

C.2 SKILLEVOLBENCH MEASUREMENTS AND DIAGNOSTICS

SkillEvolBench contains six environments with five task families each; T1–T3 are acquisition tasks and T4–T6 are deployment tasks. In the original benchmark protocol, the acquisition sequence is traversed once before the resulting skill library is evaluated on unseen deployment tasks. Our 1-pass condition corresponds to this native acquisition endpoint.

To study later-stage skill evolution after a library has already formed, we introduce a controlled second acquisition pass; this 2-pass setting is not part of the original SkillEvolBench protocol. Both the unregularized 2-pass condition and SKILLEVOREG start from the exact same one-pass library and process the same T1–T3 acquisition sequence again in the same order. Thus, they receive identical additional experience, while T4–T6 remain unseen throughout acquisition; the comparison isolates how that repeated experience is incorporated into the existing skill library. SKILLEVOREG retains the native execution environment, verifier, SkillAuthor proposal, and library commit interface, inserting regularization only around the candidate update before write-back. Deployment and replay evaluations use frozen libraries and two independent sessions.

Library tokens use `o200k_base`, summed over the 32 final `SKILL.md` files across six isolated environments. Structural atoms use the same deterministic exact-span segmentation as Equation 13. We sum C over individual files and report growth relative to the one-pass library. The original benchmark-specific atom counts are 991, 1,113, and 1,070 for the one-pass, two-pass, and regularized libraries; the common structural counts are 1,422, 1,551, and 1,515, respectively. Relative complexity growth is 14.32% for two passes and 9.65% with SKILLEVOREG.

C.2.1 CONTINUOUS VERIFIER SCORES

In addition to strict success, we report the benchmark's normalized overall verifier score, which gives partial credit according to each task's official rubric. We average the two frozen evaluations of each task and then average across tasks in each block.

The deployment means differ by less than 0.004 despite larger differences in strict success. The two composition evaluations yield 6/30 and 5/30 successes for one pass, 4/30 and 4/30 for two passes, and 8/30 and 8/30 with SKILLEVOREG. All conditions retrieve the required skills on all original composition tasks, making missing-skill retrieval an unlikely explanation for the composition difference.

C.3 CONTINUALSKILLBENCH MEASUREMENTS AND COMPLETE RESULTS

ContinualSkillBench differs from the other settings in that its native updater can expand the skill library through explicit CREATE-SKILL actions in addition to revising existing skills through MODIFY-SKILL. The evolving state therefore includes both the contents of individual skills and the library structure itself. We consequently treat library proliferation as part of the regularization problem: CREATE introduces a new persistent skill, whereas MODIFY changes an existing one.

Training processes tasks 1–50 sequentially while retaining the evolving library. Frozen evaluation then uses the step-50 library on tasks 51–100, with a fresh session for each task and no further skill

Table 12: ContinualSkillBench online learning performance on tasks 1–50, reported in the source benchmark's format. Entries are rewards in $[0, 1]$ from a single sequential learning trajectory.

Domain	Method	EM	F1	Num.	Prog.	Rubric	Raw	Norm.
Mathematics	Baseline	**0.641**	–	**0.909**	–	–	**0.700**	**0.700**
	SKILLEVOREG	0.615	–	0.818	–	–	0.660	0.660
Law	Baseline	**0.853**	–	–	**0.800**	0.615	0.795	0.795
	SKILLEVOREG	0.824	–	–	**0.800**	**0.774**	**0.810**	**0.810**
Finance	Baseline	0.364	**0.488**	**0.583**	**1.000**	**0.513**	**0.541**	**0.541**
	SKILLEVOREG	**0.545**	0.392	0.417	0.600	0.492	0.478	0.478
Office	Baseline	–	–	0.862	**0.933**	**0.913**	0.890	0.890
	SKILLEVOREG	–	–	**0.897**	**0.933**	0.908	**0.909**	**0.909**
Healthcare	Baseline	0.750	–	–	**0.500**	0.759	0.736	0.736
	SKILLEVOREG	**0.813**	–	–	**0.500**	**0.769**	**0.761**	**0.761**

Table 13: Mean frozen reward by the source benchmark's evaluator categories. Every score averages two repeats.

Domain	Method	EM	F1	Num.	Prog.	Rubric
Mathematics	Baseline	–	–	73.08	75.00	61.80
	SKILLEVOREG	–	–	73.08	75.00	**65.00**
Law	Baseline	100.00	–	–	60.00	60.79
	SKILLEVOREG	100.00	–	–	**70.00**	**66.27**
Finance	Baseline	–	21.73	–	87.50	69.59
	SKILLEVOREG	–	**22.86**	–	87.50	**70.09**
Office	Baseline	100.00	–	**75.00**	80.00	**85.33**
	SKILLEVOREG	100.00	–	71.43	80.00	84.57
Healthcare	Baseline	**66.67**	–	–	–	72.27
	SKILLEVOREG	62.50	–	–	–	**76.47**

updates. Baseline and SKILLEVOREG share the same task order, model, and evaluation procedure. The two evaluation repeats use the same frozen library and therefore measure evaluation variation rather than variation across training runs.

Online learning performance. Table 12 follows the source benchmark's reporting format and evaluates tasks 1–50 as they are encountered during skill evolution. Raw is the mean over all 50 tasks; Norm. is the mean over tasks with scoreable outputs in both conditions. All 50 tasks are common-valid in every domain here, so Raw and Norm. coincide.

Held-out evaluator breakdown. Structured in Table 3 averages individual non-rubric task scores across both repeats rather than averaging evaluator-category means equally. The source-category breakdown is shown in Table 13.

CREATE/MODIFY regularization. The two native update modes induce different growth semantics. For CREATE-SKILL, SKILLEVOREG controls the absolute complexity of the newly introduced skill and considers overlap with the existing library; for MODIFY-SKILL, it regularizes the candidate relative to the incumbent skill and its update-induced growth. In both cases, the final regularized state is written back through the benchmark's native continual-learning workflow.

Library measurements. Skill count is the number of skill directories containing a `SKILL.md` in the delivered domain library, including retained initial domain skills. Tokens use `o200k_base`

and are summed over skill documents and bundled auxiliary text/code files. Complexity uses Equation 13; auxiliary files contribute lexical length and code cost. All measurements use the step-50 frozen snapshot.

Office sensitivity analysis. Office's 1.44-point Overall difference is concentrated rather than broad. All but three of the 21 structured tasks are identical between conditions. One rubric task additionally generates a complete briefing but writes it to a path different from the evaluated path. We retain the strict end-to-end score in the main table. Excluding this single task reduces the Overall gap to 0.63 points; excluding it together with the three table-selection outliers leaves SKILLEVOREG 0.42 points higher on the other 46 tasks. We use this analysis only to localize the observed difference, not to replace the official score.

Healthcare evaluator analysis. Healthcare gains 2.20 Overall points while reducing the library from 42 to 34 skills. Its Rubric score improves by 4.21 points, whereas the Structured difference corresponds to one net exact-match success across the 24 question–repeat instances. The aggregate gain therefore comes primarily from open-ended transfer rather than from easier exact-match behavior.

C.4 DEPLOYMENT INFERENCE USAGE AND SKILL COMPACTNESS

Inference usage is not an optimization target of SKILLEVOREG; we examine it as a deployment-side by-product of compacting the persistent skill state. A shorter persistent skill can reduce the context consumed during downstream execution, but end-to-end model usage also depends on the execution trajectory, particularly when external search, document retrieval, or other tool interactions introduce substantial and variable context. For each evaluation run, we count input plus output tokens from the completed model session, including cached input. Judge calls, training, regularizer calls, and superseded failed attempts are excluded. Every row uses the frozen evaluation segment corresponding to the reported checkpoint and pools the two independent repeats.

Table 14: Frozen-evaluation model-token usage. Values are total input-plus-output tokens per task, pooled over two repeats; trimmed means remove the top and bottom 10% of tasks.

Benchmark	Method	Mean	Median	Trimmed mean
SpreadsheetBench	SkillOpt	12,572	10,099	11,266
	SKILLEVOREG	**8,865**	**7,328**	**7,880**
SearchQA	SkillOpt	6,280	6,245	6,257
	SKILLEVOREG	**2,759**	**2,713**	**2,726**
LiveMath	SkillOpt	8,186	7,426	7,789
	SKILLEVOREG	**5,028**	**4,437**	**4,637**
SkillEvolBench	2-pass	396,513	334,137	353,599
	SKILLEVOREG	**359,340**	**281,948**	**323,701**
Continual–Math	Baseline	**153,611**	74,418	**113,380**
	SKILLEVOREG	168,458	**73,441**	105,502
Continual–Law	Baseline	**401,321**	277,371	**336,404**
	SKILLEVOREG	412,642	**260,898**	347,242
Continual–Finance	Baseline	1,236,906	418,529	694,990
	SKILLEVOREG	**863,415**	**256,081**	**558,126**
Continual–Office	Baseline	238,229	129,351	155,029
	SKILLEVOREG	**220,137**	**114,689**	**130,362**
Continual–Healthcare	Baseline	**156,572**	**121,384**	**140,291**
	SKILLEVOREG	194,875	129,183	150,549

The main pattern is that skill compactness reduces a stable component of inference context, but whether this translates into lower end-to-end token usage depends on how much execution is dominated by trajectory-dependent external context. The effect is clearest on tasks with relatively con-

trolled execution paths. On SpreadsheetBench and LiveMath, where task context is fixed and execution does not depend on open-ended external information gathering, SKILLEVOREG reduces mean frozen-evaluation token usage by 29.5% and 38.6%, respectively, with corresponding reductions in both the median and trimmed mean. SearchQA also shows a substantial 56.1% reduction despite using web search, indicating that external search does not by itself eliminate the benefit of a compact skill state. SkillEvolBench likewise reduces mean usage by 9.4% relative to the matched two-pass condition. Together, these results show that compacting the persistent skill can produce substantial deployment-side savings when the skill state remains a meaningful and comparatively stable component of the inference context.

ContinualSkillBench exposes a different regime, in which high-variance information-acquisition and tool-use trajectories can dominate total inference usage. Finance and Office decrease across the principal summaries, whereas Healthcare increases; Mathematics and Law have higher arithmetic means under SKILLEVOREG despite lower medians, indicating that a small number of long trajectories can dominate average usage. To examine this pattern more closely, we pair the same frozen-evaluation task and repeat under the baseline and SKILLEVOREG for Mathematics, Law, and Healthcare. As one measurable indicator of externally mediated trajectories, we separate pairs according to whether either execution invokes `web_search`. Across the resulting 300 pairs, the three domains increase by 6.45 million tokens in aggregate; pairs involving `web_search` account for a 6.66 million-token increase, whereas pairs with no `web_search` decrease by 0.21 million tokens overall. The contrast is particularly pronounced in Mathematics and Healthcare, where no-search pairs decrease by 6,754 and 1,005 tokens per task on average, respectively, while search-associated pairs increase by 53,247 and 73,161 tokens. Law is more mixed: search-associated pairs contribute most of the net increase, but several high-cost trajectories involve extensive local-document processing without web search.

The `web_search` split should therefore be interpreted as an observable indicator of externally mediated trajectories rather than as a causal attribution to search itself. High-usage trajectories can also involve substantially different amounts of PDF or document extraction, shell-returned content, and prolonged tool interaction, while the available logs do not expose the exact token contribution of each source after context injection and truncation. The resulting picture is therefore not that compact skills universally reduce total inference tokens, but that they reduce one persistent and controllable source of context consumption. When execution context is relatively stable, this reduction can translate into substantial token savings; when external context dominates the trajectory, the same benefit can be obscured by much larger variation in tool-mediated context.

Takeaway. Reduced inference usage is a deployment-side by-product rather than an optimization objective of SKILLEVOREG. Compact persistent skills reduce the controllable skill-context component of inference usage, while end-to-end savings depend on how strongly the downstream trajectory is dominated by external information and tool interactions.

D ALGORITHMIC DETAILS AND UPDATER INTEGRATION

This appendix formalizes the three regularizers introduced in Section 4 and describes how they connect to different native skill updaters. The regularization roles are shared across SkillOpt, SkillEvolBench, and ContinualSkillBench, while the update boundary is framework-dependent: candidate changes may be represented as document edits, localized library patches, or changes between skill-library snapshots.

D.1 NATIVE UPDATE INTERFACE

We represent one native evolution step abstractly as

$$(\tau_t, \widetilde{S}_{t+1}) = \mathrm{NativeUpdate}(S_t, x_t), \tag{14}$$

where S_t is the incumbent skill state, x_t is the current training task or batch, τ_t is the resulting execution trajectory, and $\widetilde{S}_{t+1}$ is the native candidate. When dropout is active, the native update is first generated from a temporary masked view and then reconciled with the complete incumbent, as described below.

Algorithm 1 Training-Time Skill Dropout

Input: Incumbent skill state S_t, training task x_t
Output: Dropout-reconciled candidate S_{t+1}^{drop}

$\mathcal{A}_t \leftarrow \text{SegmentAtoms}(S_t)$
$M_t \leftarrow \text{SampleMask}(\mathcal{A}_t)$
$S_t^{\text{drop}} \leftarrow \text{Mask}(S_t, M_t)$
$(\tau_t, \widetilde{S}_{t+1}^{\text{drop}}) \leftarrow \text{NativeUpdate}(S_t^{\text{drop}}, x_t)$
$\Delta_t^{\text{drop}} \leftarrow \text{Diff}(S_t^{\text{drop}}, \widetilde{S}_{t+1}^{\text{drop}})$
$S_{t+1}^{\text{drop}} \leftarrow \text{Reconcile}(S_t, \Delta_t^{\text{drop}})$
return S_{t+1}^{drop}

Throughout the appendix, $\text{Diff}(S, S')$ denotes the change from a source state S to a candidate state S'. We use stage-specific superscripts for the resulting deltas: Δ_t^{drop} for the change learned under the dropout view, Δ_t^{comp} for the reconciled candidate change examined by complexity regularization, and Δ_t^{ccv} for the post-complexity candidate change examined by CCV. This keeps the underlying notion of a candidate delta shared while distinguishing the state boundary at which it is computed.

The main framework-dependent integration operations are `Reconcile`, which transfers an update learned from a temporarily perturbed skill view onto the complete incumbent, and `WriteBack`, which returns the final regularized candidate through the native framework's acceptance or commit interface.

D.2 Training-Time Skill Dropout

A *skill atom* is a locally meaningful unit of reusable skill content, such as an individual directive, list item, compact procedural block, or another semantically coherent segment. Atom segmentation provides stable units for temporary masking and later local structural comparison.

Algorithm 1 formalizes training-time skill dropout. Its central invariant is that masking affects update generation only. Content removed from the temporary view is restored before the dropout-stage candidate proceeds to subsequent regularization.

Atom segmentation. For textual skill states, atoms are obtained from deterministic structural segmentation of the skill document, so that repeated references to the same instruction or procedural block remain stable across an update. Multi-skill libraries additionally retain skill-file or skill-path boundaries. These higher-level boundaries determine the region within which atom-level operations are permitted.

Restoration. Because the native updater observes S_t^{drop} rather than the complete incumbent S_t, Δ_t^{drop} captures only the change proposed under the perturbed context. `Reconcile` restores the masked atoms from S_t and transfers that change onto the complete incumbent, producing S_{t+1}^{drop}. An atom is therefore never removed merely because it was absent from the temporary training-time view.

D.3 Complexity-Aware Local Regularization

Complexity-aware local regularization operates on the dropout-reconciled candidate S_{t+1}^{drop}. The stage-local change is

$$\Delta_t^{\text{comp}} = \text{Diff}\left(S_t, S_{t+1}^{\text{drop}}\right), \tag{15}$$

which is measured relative to the complete incumbent, unlike Δ_t^{drop} in the preceding stage.

Algorithm 2 separates complexity measurement, local-scope construction, and discrete edit selection.

Algorithm 2 Complexity-Aware Local Regularization

Input: Incumbent skill state S_t, dropout-reconciled candidate S_{t+1}^{drop}
Output: Complexity-regularized candidate S_{t+1}^{comp}

$\Delta_t^{\text{comp}} \leftarrow \text{Diff}(S_t, S_{t+1}^{\text{drop}})$
$c_t \leftarrow f_{\text{comp}}\Big(\phi(S_t), \phi(S_{t+1}^{\text{drop}}), \phi(\Delta_t^{\text{comp}})\Big)$
$S_{t+1}^{\text{comp}} \leftarrow S_{t+1}^{\text{drop}}$
if $\text{NeedsRegularization}(c_t, \Delta_t^{\text{comp}})$ **then**
    $\mathcal{D}_t \leftarrow \text{SegmentAtoms}(\Delta_t^{\text{comp}})$
    $\mathcal{R}_t \leftarrow \text{RetrieveRelated}(S_t, \mathcal{D}_t)$
    $\Omega_t \leftarrow \text{BoundScope}(\mathcal{D}_t \cup \mathcal{R}_t)$
    for all $a \in \mathcal{D}_t$ **do**
        $o_a \leftarrow \text{SelectOperation}(a, \Omega_t, c_t)$
        $S_{t+1}^{\text{comp}} \leftarrow \text{ApplyLocalOperation}(S_{t+1}^{\text{comp}}, a, o_a, \Omega_t)$
    end for
end if
return S_{t+1}^{comp}

D.3.1 Complexity Signal Definitions

Equation 6 leaves the structural feature extractor abstract in the main text. We instantiate it as

$$\phi(Z) = \big[\phi_{\text{size}}(Z), \phi_{\text{atom}}(Z), \phi_{\text{structure}}(Z), \phi_{\text{code}}(Z), \dots\big] \tag{16}$$

for either a skill state or an update delta. For a skill state, the components summarize lexical, atom-level, structural, and executable-code properties; for an update delta, they summarize the corresponding structural change introduced or removed by the candidate.

The regularization signal is

$$c_t = f_{\text{comp}}\left(\phi(S_t), \phi(S_{t+1}^{\text{drop}}), \phi(\Delta_t^{\text{comp}})\right). \tag{17}$$

The function f_{comp} need not be identical across native update protocols; it determines how shared structural measurements are converted into the signal used to activate or score regularization.

SkillOpt. For SkillOpt, the structural features are summarized by the composite complexity measure $C(S)$ defined in Equation 13. The corresponding f_{comp} compares incumbent and candidate complexity and supplies the complexity term used by regularized candidate evaluation. Complexity can therefore influence whether a candidate transition is retained.

SkillEvolBench. For SkillEvolBench, f_{comp} is used primarily as a growth trigger. When the candidate exceeds the configured structural-growth budget, it enters bounded local regularization. The signal determines whether additional consolidation is invoked rather than acting as an independent hard rejection criterion.

ContinualSkillBench. ContinualSkillBench conditions f_{comp} on the native update type. For CREATE, the signal uses an absolute complexity budget for the newly created skill together with its relation to existing library content. For MODIFY, it emphasizes relative growth of the modified skill with respect to its pre-update state. These are soft regularization triggers: exceeding them can invoke structural reconsideration without automatically vetoing the final candidate.

Discrete local operations. The operation-based formulation is inspired by Mem0's memory reconciliation mechanism (Chhikara et al., 2025), in which newly extracted facts are compared with related memories and assigned one of four actions: ADD, UPDATE, DELETE, or NOOP. Our problem begins after the native skill updater has already proposed new persistent content, so we adapt the discrete-action principle to post-update skill reconciliation:

$$o_a \in \{\text{NOOP}, \text{MERGE}, \text{REWRITE}, \text{DELETE}\}. \tag{18}$$

NOOP retains the candidate atom unchanged. MERGE consolidates overlapping candidate and incumbent atoms. REWRITE modifies useful but unnecessarily narrow, redundant, or overspecified content. DELETE removes candidate content that contributes no distinct capability.

Related-content retrieval. For each candidate atom, the regularizer retrieves a bounded number of semantically or structurally related incumbent atoms. Retrieval defines the comparison neighborhood rather than the edit decision itself. High similarity does not imply that two atoms are interchangeable, since apparently redundant instructions can play distinct operational roles in language-model execution.

Granularity across updaters. SkillOpt and SkillEvolBench expose the four operations directly over candidate-relative local content. ContinualSkillBench enforces the same locality principle through affected skill paths and can rewrite the full text of an implicated skill. This changes the editing granularity but not the role of the regularizer.

Complexity semantics. The complexity signal can combine textual size, structural fragmentation, executable-code structure, and relative growth. SkillOpt can consume it as part of candidate acceptance; SkillEvolBench uses excessive growth to trigger bounded local regularization; and ContinualSkillBench uses soft absolute and relative budgets for CREATE and MODIFY, respectively. These differences determine when local reconsideration is activated rather than defining different regularization mechanisms.

D.4 CCV IMPLEMENTATION DETAILS

Section 4.4 presents the four-way behavioral comparison. Here we make the underlying checks explicit. CCV validates the post-complexity transition from S_t to S_{t+1}^{comp}, so its stage-local change is $\Delta_t^{\text{ccv}} = \text{Diff}(S_t, S_{t+1}^{\text{comp}})$. This is distinct from Δ_t^{comp}, which is computed before structural regularization.

Source selection and clean qualification. CCV draws e_t from previously observed training trajectories rather than held-out evaluation data. The first two checks correspond to the first column of Equation 9. The incumbent must succeed on the original source, establishing that the behavior is already supported by S_t, and the candidate must remain qualified on that source, ruling out the simpler case in which the update has already broken the original task. Previously computed clean results are reused when available; the two `Evaluate` operations in Algorithm 3 denote logical checks and do not require redundant re-execution.

Candidate-conditioned attack generation. After the clean pair qualifies, `GenerateAttack` receives e_t and the post-complexity candidate change Δ_t^{ccv}. The generator identifies an assumption, dependency, or decision boundary that the candidate may have introduced, strengthened, or narrowed, and proposes a bounded perturbation designed to stress that property. Generation may return no attack when the candidate change exposes no suitable target or when no valid perturbation can be constructed.

Benchmark-specific attack application. The attack specification a_t is converted into an executable instance through `ApplyAttack`. SpreadsheetBench can perturb workbook cells, formulas, or task instructions subject to validity checks. SearchQA introduces bounded distractor context while retaining the original question and gold answer. LiveMath adds controlled prior scratch-work context while retaining the original problem, answer options, and label. SkillEvolBench preserves the original software-task instruction and appends bounded robustness context targeted at the candidate change. ContinualSkillBench uses both answer-preserving transformations and nearby boundary cases with independently checkable oracles. Although the surface transformations differ, every integration constructs one attacked instance x_t^{ccv} and holds it fixed for the incumbent–candidate comparison.

Regression criteria. The attacked evaluations correspond to the second column of Equation 9. A CCV-confirmed regression requires the incumbent to remain sufficiently successful on x_t^{ccv} while

Algorithm 3 Causal Counterexample Validation

Input: Incumbent skill state S_t, complexity-regularized candidate S_{t+1}^{comp}, training history $\mathcal{H}_t$, CCV configuration θ_{ccv}
Output: CCV-processed candidate S_{t+1}^{ccv}

$S_{t+1}^{\text{ccv}} \leftarrow S_{t+1}^{\text{comp}}$
$\Delta_t^{\text{ccv}} \leftarrow \text{Diff}(S_t, S_{t+1}^{\text{comp}})$
$e_t \leftarrow \text{SelectSource}(\mathcal{H}_t, S_t, S_{t+1}^{\text{comp}})$
if $e_t = \varnothing$ **then**
    return S_{t+1}^{ccv}
end if
$r_{\text{old}}^{\text{clean}} \leftarrow \text{Evaluate}(S_t, e_t)$
$r_{\text{new}}^{\text{clean}} \leftarrow \text{Evaluate}(S_{t+1}^{\text{comp}}, e_t)$
$q_t \leftarrow \text{CleanQualify}(r_{\text{old}}^{\text{clean}}, r_{\text{new}}^{\text{clean}})$
if $q_t = 0$ **then**
    return S_{t+1}^{ccv}
end if
$a_t \leftarrow \text{GenerateAttack}(e_t, \Delta_t^{\text{ccv}})$
if $a_t = \varnothing$ **then**
    return S_{t+1}^{ccv}
end if
$x_t^{\text{ccv}} \leftarrow \text{ApplyAttack}(e_t, a_t)$
if $x_t^{\text{ccv}} = \varnothing$ **then**
    return S_{t+1}^{ccv}
end if
$r_{\text{old}}^{\text{attack}} \leftarrow \text{Evaluate}(S_t, x_t^{\text{ccv}})$
$r_{\text{new}}^{\text{attack}} \leftarrow \text{Evaluate}(S_{t+1}^{\text{comp}}, x_t^{\text{ccv}})$
$g_t \leftarrow \text{RegressionCriterion}(r_{\text{old}}^{\text{attack}}, r_{\text{new}}^{\text{attack}}; \theta_{\text{ccv}})$
if $g_t = 1$ **then**
    $\Omega_t^{\text{ccv}} \leftarrow \text{ImplicatedScope}(\Delta_t^{\text{ccv}}, a_t, r_{\text{old}}^{\text{attack}}, r_{\text{new}}^{\text{attack}})$
    $S_{t+1}^{\text{ccv}} \leftarrow \text{Repair}(S_{t+1}^{\text{comp}}, \Omega_t^{\text{ccv}})$
end if
return S_{t+1}^{ccv}

the candidate deteriorates according to the native evaluator. SkillOpt uses a strict incumbent-pass/candidate-fail condition after clean qualification. SkillEvolBench requires the incumbent attacked-case score to be at least 0.8 and the candidate score to decrease by at least 0.05 relative to the incumbent; the candidate need not cross a binary failure threshold. ContinualSkillBench requires incumbent success together with either candidate failure or a score decrease greater than 0.1. These differences affect the numerical realization of `RegressionCriterion`, while the four-way comparison remains unchanged.

Scope-constrained repair. When the paired criterion identifies a regression, CCV uses Δ_t^{ccv}, a_t, and the attacked-case outcomes to localize the candidate content most directly implicated by the failure. SkillOpt and SkillEvolBench restrict repair to the relevant candidate change and related local content, using the local editing mechanism described in Section 4.3. ContinualSkillBench follows the same locality principle at the level of implicated skill paths and may rewrite the full content of an affected skill. Repair granularity therefore follows the native representation while remaining constrained to the region implicated by the candidate-conditioned attack. Each triggering CCV event performs at most one scope-constrained repair.

D.5 Integration with Native Skill Updaters

The principal framework-dependent implementation choices occur at the native update boundary. Table 15 summarizes these differences.

Table 15: Integration of SKILLEVOREG with the three native skill-update interfaces. The regularization mechanisms remain largely unchanged; the main differences concern how the native framework represents, exposes, and writes back a candidate update.

Interface	SkillOpt	SkillEvolBench	ContinualSkillBench
Skill state	Single evolving instruction document.	Multi-skill library containing localized skill files.	Continually evolving skill library.
Native update	Ranked textual edits generated from the execution trajectory.	Localized skill patch proposed by the native skill author.	Explicit CREATE or MODIFY operation.
Candidate representation	Updated candidate document and corresponding textual changes.	Proposed patch over one or more localized skill regions.	Raw post-update skill or library snapshot produced by the native operation.
Delta extraction	Compare the candidate document with the temporary skill view used during update generation.	Recover the changed skill and source patch from the native proposal.	Compare pre-update and raw post-update snapshots and identify affected skill paths.
Dropout reconciliation	Restore masked atoms from the incumbent document and apply only the updater-generated change.	Restore masked library content and reconcile the native patch with the complete incumbent library.	Restore the pre-update library and reconcile the raw CREATE/MODIFY result with the restored snapshot.
Local regularization scope	Candidate atoms plus related existing atoms.	Changed skill/source delta plus related local content.	Affected skill paths, with whole-skill rewriting permitted within an implicated path.
Final write-back	Return the regularized candidate to SkillOpt's acceptance and current/best bookkeeping.	Write the finalized patch through the existing library update interface.	Write the final regularized snapshot under the continual framework's apply semantics.

The table highlights why the three benchmarks do not require three different formulations of SKILLEVOREG. Their native updaters determine how each candidate transition becomes visible and how the final candidate is returned. Once the stage-specific changes are exposed, dropout reconciliation, local structural regularization, and paired counterexample validation retain the same conceptual roles.

E DETAILED COMPONENT AND TRAJECTORY ANALYSES

E.1 FAILURE CASE: NON-LOCAL WHOLE-SKILL REWRITING

We analyze a SpreadsheetBench failure case to illustrate why SKILLEVOREG restricts complexity regularization to candidate-local regions rather than allowing unrestricted whole-skill rewriting. In this case, a validation regression triggered a repair procedure that operated over the complete persistent skill, even though the complexity trigger itself was inactive. The repaired skill was then written back without a post-repair evaluation. We therefore treat this example as a diagnostic case study of non-local rewriting rather than as a controlled comparison between compression strategies.

The original candidate update was comparatively local. Relative to the preceding incumbent, it added four bullets, removed one inherited bullet and one third-level heading, and altered content in only 4 of the 14 major `##` sections. The subsequent repair, however, was permitted to rewrite the complete skill rather than being restricted to this candidate-local change. Although the repair prompt requested localized revision, it exposed the complete skill and imposed no programmatic boundary on the editable region.

Table 16: Diagnostic failure case illustrating the scope and behavioral consequences of unrestricted whole-skill rewriting. Textual rewrite statistics characterize rewrite scope rather than semantic rule deletion.

Quantity	**Observed change**
Original candidate scope	$4/14$ major sections affected
Whole-skill repair scope	$14/14$ major sections changed
Candidate $\rightarrow$ repaired text	$18{,}313 \rightarrow 9{,}598$ characters (-47.6%)
Persistent skill length	$2{,}679 \rightarrow 1{,}450$ words (-45.9%)
Inherited bullets preserved verbatim	$6/47$
Inherited bullets not preserved verbatim	$41/47$ (87.2%)
Test accuracy	$60.5\% \rightarrow 48.5\%$ (-12.0 pp)
Correct $\rightarrow$ incorrect	32 tasks
Incorrect $\rightarrow$ correct	8 tasks
Post-repair evaluation	None before write-back

The rewrite footprint was substantially broader than the source candidate delta. Although the candidate affected only $4/14$ major sections, the whole-skill repair changed content in all 14 sections, and seven original section titles no longer appeared verbatim. Among the 47 bullets inherited by the candidate from the preceding skill, only six remained textually identical after the repair, while $41/47$ were no longer preserved verbatim. These statistics quantify textual non-locality rather than semantic deletion: they do not imply that 41 distinct rules were removed, since some content may have been merged, relocated, or paraphrased.

The resulting checkpoint also exhibited strongly asymmetric behavioral change. Test accuracy decreased from 60.5% to 48.5%: 32 previously solved tasks became failures, while only 8 previously failed tasks became correct. Inspection of the trace indicated that the global rewrite altered established operational guidance rather than merely consolidating candidate-local redundancy. Because the transition also involved an erroneous trigger route and lacked post-repair evaluation, these observations should not be interpreted as a controlled estimate of the causal effect of whole-skill rewriting.

This failure case motivates the separation between the *trigger* for structural reconsideration and the *scope* permitted to change. In SKILLEVOREG, complexity measurements can use the broader incumbent and candidate states to determine whether regularization is warranted, but editing is anchored to the candidate-local delta $\mathcal{D}_t$ and a bounded related context Ω_t. Unrelated incumbent content is therefore preserved rather than being exposed to an unconstrained global rewrite. Locality does not require every repair to be small; it requires the permissible edit region to remain tied to the candidate transition being regularized.

E.2 CCV REGRESSION EVIDENCE AND REPRESENTATIVE TRACE

Regression evidence. Across the evaluated trajectories, CCV identifies 12 distinct regression signals, 9 of which lead to state-changing repairs. Each signal arises from a candidate-conditioned comparison in which the incumbent and candidate remain qualified on the original source, while the candidate deteriorates relative to the incumbent on the same attacked case. These events provide update-level evidence for the type of behavioral regression targeted by CCV. We next examine one representative trace in detail.

Representative CCV trace. We use a representative SkillEvolBench dependency-scope trace to illustrate the mechanism. The source task asks the agent to repair a React 18 peer-dependency conflict. The incumbent contains a general instruction to regenerate and validate the repository lockfile. The candidate adds a more specific rule that makes manifest ownership and package/workspace boundaries an explicit decision criterion.

CCV applies the same package-boundary attack to both incumbent and candidate runs. The incumbent receives reward 1.0, satisfying 7/7 outcome checks and 5/5 process checks. The candidate receives 0.8: all seven outcome checks remain satisfied, but two process checks fail after it replaces versioned React dependencies with local `file:` shims. The attacked comparison therefore reveals

Table 17: SpreadsheetBench shared-prefix continuation. E1–E6 are common to both branches; E7–E8 differ after independent update sampling. Test scores average two frozen evaluations.

Branch	Checkpoint	Train	Validation	Test
Shared	Initial	32.50	37.00	38.75
	E1	44.50	52.00	52.50
	E2	48.00	55.50	56.50
	E3	48.00	55.50	56.50
	E4	48.50	60.00	58.50
	E5	48.50	60.00	58.50
	E6	53.50	61.00	57.25
Main	E7	49.50	**62.50**	58.50
	E8	50.00	61.50	**59.25**
Degrading	E7	46.50	51.00	52.00
	E8	45.50	51.50	50.00

candidate-specific sensitivity to the package-boundary condition targeted by the candidate-added rule.

CCV rewrites the exact candidate-added atom from a rule that emphasizes the directory owning the manifest to one that ties lockfile regeneration and validation to the dependency scope actually used by the resolver. CCV therefore identifies candidate-specific sensitivity to package-boundary wording and produces a targeted scope-constrained rewrite. A later replay recovers the two affected process checks.

E.3 Residual Stochasticity and Validation-Based Checkpointing

Shared-prefix continuation. Two SkillEvoReg SpreadsheetBench continuations share a byte-identical skill state through step 30, the end of epoch six, and then process the same remaining batch sequence under the same configuration. Steps 29 and 30 are rejected, so the shared step-30 snapshot is also textually identical to the state produced at step 28. The first different skill states appear at step 31, where independently sampled candidate updates are accepted. These are controlled continuations of one learned prefix, not independent end-to-end training seeds.

The validation-selected delivery checkpoint is E7 on the main continuation and E6 on the degrading continuation. Although the two terminal states diverge substantially (59.25% versus 50.00% test accuracy), checkpoint selection yields much more comparable delivered performance (58.50% versus 57.25%). This illustrates that regularization and checkpoint selection address distinct sources of risk: SkillEvoReg shapes the repeated update process, whereas validation-based checkpointing protects delivery from residual stochastic trajectory degradation. We retain the latter as an established early-stopping-style safeguard rather than a contribution of SkillEvoReg.

Accepted update patterns. After the fork, each branch accepts five of the ten remaining proposals. The main branch accepts steps 31, 32, 33, 35, and 37; the degrading branch accepts 31, 34, 36, 37, and 39. The degrading branch accumulates broader instructions for preserving blank/nonmatching slots, clearing a prefilled destination block, blanking non-terminal rows, and clearing unused spill cells. The main branch is not free of blanking language, but combines it with explicit full-range reconnaissance, blank-versus-zero disambiguation, and conjunctive deletion conditions. The comparison therefore concerns the combination and scope of accepted rules, not a binary distinction between "blanking" and "no blanking."

Task-level audit. The same 200 test tasks are evaluated twice at the shared E6 checkpoint and at the degrading E8 checkpoint. Thirty-six tasks regress, 12 improve, and 152 are unchanged, yielding a net decline of 7.25 percentage points. Thirty-one of the 36 regressions are cell-level manipulation tasks. Twenty-one of the 36 produce a missing prediction in both E8 repeats, and 30 do so in at least one repeat. The dominant observed failure pattern is therefore missing required cell values rather than uniform degradation across task families.

The accepted clearing and blanking instructions are behaviorally compatible with this pattern, but they are not isolated as the cause. Multiple skill edits and evaluation-time randomness remain confounders. Train and validation performance also decline along the degrading branch. We therefore interpret this controlled fork as stochastic trajectory-level degradation rather than conventional train–test overfitting. The example illustrates a residual source of variation that update-level regularization does not eliminate, and shows how validation-based checkpoint selection can prevent such late-stage degradation from determining the delivered state.

F Prompts and Scope Enforcement for LLM-Based Regularization Operations

The algorithms in Appendix D define semantic operations such as $\mathrm{SelectOperation}$, $\mathrm{GenerateAttack}$, and Repair rather than prescribing a single prompt shared across all host frameworks. Their concrete realizations depend on the native skill representation and task-valid perturbation space. This section provides representative prompt excerpts and summarizes the corresponding writable-scope constraints. Native skill-updater prompts are inherited from the host frameworks and are not reproduced here.

For presentation consistency, implementation identifiers in the excerpts are normalized to the terminology used in this paper; the operational instructions and constraints are otherwise preserved. Dynamic runtime fields are shown in braces.

F.1 Complexity-Aware Local Editing

For SkillOpt and SkillEvolBench, the local editor receives broader skill context for interpretation, but only explicitly identified source-delta regions are writable. The core editing instruction defines four operations:

```
Treat the listed source deltas as independent editable
regions.  Similarity only nominates historical content for
semantic comparison; it does not imply that two rules are
interchangeable.
For every source delta, choose exactly one operation:
NOOP: preserve the source unchanged when it adds a distinct
or complementary capability, or whenever semantic
preservation is uncertain.
REWRITE: replace only the source delta when it is
overspecific, unclear, or internally redundant, while
preserving its supported capability.
MERGE: use only when one coherent rule can preserve the full
union of the source delta and linked historical content.
DELETE: remove only a source delta that is unsupported,
harmful, or fully subsumed by retained content.
The complete skill is read-only context.  Return
modifications only for explicitly supplied editable regions;
never return a rewritten complete skill.
```

When complexity regularization is triggered, the editor additionally receives:

```
Merge duplicates and semantic overlaps, remove narrow or
unsupported rules, replace similar rule clusters with
transferable general rules, and compress repeated
explanations or examples.  Do not add new rules merely in
response to excessive complexity, and preserve distinct
useful behavior.
```

The corresponding user request separates read-only context from writable regions:

```
## Full Current Skill (read-only context)
{current_skill}
```

```
## Editable Source-Delta Regions
{editable_regions}
## Regularization Feedback
{regularization_feedback}
Return one decision for every listed source delta and only
the replacements required by those decisions.
```

SkillEvolBench uses the same four-operation semantics with an additional `skill_id`, since edits may belong to different localized skill files. ContinualSkillBench instead follows its native CREATE/MODIFY representation: CREATE operates over explicitly supplied skill paths, whereas MODIFY restricts changes to a set of allowed window identifiers. In all cases, the writable scope is constructed and validated by the framework adapter rather than inferred by the language model.

F.2 CCV Attack Generation

CCV constructs a candidate-conditioned behavioral probe rather than an arbitrary harder example. Because valid perturbations differ substantially across benchmarks, $\mathrm{GenerateAttack}$ has framework-specific realizations.

SkillOpt uses a two-stage Scout–Worker procedure. The Scout observes the incumbent and candidate skills, their semantic delta, recent trajectories, eligible source tasks, and the verifier contract, and returns a candidate-specific weakness together with a suitable source. A Worker then materializes that weakness using the benchmark's permitted perturbation space. SpreadsheetBench applies bounded operations to a private workbook copy; SearchQA inserts bounded distractor context while preserving the question and gold answer; and LiveMath introduces bounded prior scratch-work context while preserving the original problem, options, and gold label.

SkillEvolBench uses a single bounded probe generator. Its central instruction is:

```
Generate one bounded CCV probe for a coding-agent skill
update.  The original task instruction and verifier are
immutable.
Target a specific semantic addition, deletion, or
strengthening introduced by the candidate skill.  The probe
may add only neutral operational context, an answer-free
counterfactual condition, or a presentation variation
targeted at an incidental assumption introduced by the
candidate.
Do not change the requested deliverable, acceptance
criteria, repository contents, tests, or environment.  Do
not reveal a solution, expected patch, hidden test, or
answer.  Select responsible skill identifiers only from
skills changed by the candidate.
```

The generator returns the attack type, targeted assumption, predicted failure region, responsible skill identifiers, a goal-preservation justification, and the bounded robustness context appended to the original instruction.

ContinualSkillBench follows the same candidate-conditioned principle but uses domain-specific probe generators for Mathematics, Finance, Law, Healthcare, and Office. These generators target domain-appropriate boundaries or invariance-preserving transformations, with additional oracle or equivalence checks before the probe is used for paired incumbent–candidate evaluation.

F.3 CCV Repair

Once paired evaluation identifies a candidate-specific regression, the repair stage is instructed to recover a transferable capability rather than encode the attacked example itself. SkillOpt and SkillEvolBench reuse the local editor from Appendix F.1 with CCV-specific feedback:

```
A CCV regression has been detected.  Revise, generalize, or
remove the incidental assumption responsible for the
```

Table 18: Writable-scope enforcement across host frameworks. The model may inspect broader context, but the adapter determines which regions, windows, or skill paths are eligible for write-back.

Framework	Readable context	Writable scope	Enforcement
SkillOpt	Complete incumbent skill and local context	Explicit source-delta regions	Unknown or duplicate region IDs are rejected; linked historical atoms and merge effects are determined by the runtime.
SkillEvolBench	Complete same-family skills as read-only context	Selected regions within implicated skills	Unknown skill/region pairs and out-of-scope edits are rejected before native commit.
ContinualSkillBench MODIFY	Source skill and bounded matched context	Explicit editable windows	Unknown, stale, overlapping, or out-of-scope windows are rejected before application.
ContinualSkillBench CREATE	New skill and retrieved historical skills	Authorized skill paths	Unauthorized paths are rejected; historical paths can change only under an allowed merge.
ContinualSkillBench CCV repair	Probe and responsibility-skill contents	Authorized responsibility paths	Returned paths and the actual post-repair modified-path set are both checked before write-back.

```
regression and restore the exposed transferable capability.
Never add a one-off exception for a single counterexample.
Derive exactly one transferable procedural invariant from
the CCV counterexample and use it to replace, merge, or
generalize the responsible existing rule.  Do not copy
counterexample-specific field names, file names, schema
keys, literal values, concrete formats, or wording; express
the invariant at the capability level instead.
```

SkillEvolBench additionally provides the measured score decrease, attack type, targeted assumption, and predicted failure region as diagnostic context, while retaining the same region-level editing restriction.

ContinualSkillBench uses a separate repair call at responsibility-path granularity:

```
Repair the cause of the detected CCV regression as a
transferable invariant.  Prefer editing or merging the
responsible rule.  Modify only the supplied responsibility
skills, preserve unrelated semantics, and do not add source
answers, task identifiers, or irrelevant probe-specific
details.
```

Unlike the region-local realizations above, this operation may return the complete text of an authorized responsibility skill. The implementation therefore validates the returned paths and verifies the actual modified-path set before write-back.

F.4 Programmatic Enforcement of Editable Scope

Prompt-level locality is not itself an enforceable editing boundary. As illustrated by the failure case in Appendix E.1, a model can perform a broad rewrite even when instructed to revise locally. In the final implementation, the framework adapter determines the writable scope independently of the model.

The common invariant is therefore not that every repair is atom-local. Rather, the framework adapter determines the writable scope, while the granularity of that scope follows the native representation. SkillOpt and SkillEvolBench use region-level editing, ContinualSkillBench MODIFY uses bounded windows, and ContinualSkillBench CREATE and CCV repair may rewrite complete skills only within explicitly authorized paths.

This separation is deliberate: prompt instructions specify *how* the model should transform authorized content, whereas programmatic scope enforcement determines *what* content is permitted to change.

G Extended Related Work

Learning from agent experience. A broad line of research studies how language-model agents can improve through interaction without modifying model parameters. Reflexion stores verbal reflections derived from task feedback in episodic memory and reuses them in subsequent attempts (Shinn et al., 2023), while ExpeL extracts reusable natural-language insights from collections of agent experiences (Zhao et al., 2024). Agent-Pro iteratively refines a language-level behavioral policy through reflection and search (Zhang et al., 2024). Voyager maintains an expanding library of executable code skills that can be retrieved and composed for later tasks (Wang et al., 2024a). More recent memory systems explicitly organize reusable experience: Agent Workflow Memory induces recurring workflows from successful trajectories (Wang et al., 2025), and A-MEM continuously restructures an interconnected memory network as new experiences arrive (Xu et al., 2025). These methods establish reusable external state as an effective mechanism for non-parametric adaptation; our work focuses on regularizing repeated revision of such procedural state.

Skill generation, optimization, and maintenance. SkillOpt formulates a natural-language skill as trainable external state and optimizes it through bounded textual edits, validation-gated acceptance, and checkpoint selection (Yang et al., 2026b). Trace2Skill aggregates trajectory-local lessons before hierarchically consolidating them into transferable skills (Ni et al., 2026), while CoEvoSkills jointly evolves structured skill packages and a surrogate verifier (Zhang et al., 2026). SkillEvolBench evaluates transfer from episodic experience to procedural skills under context shifts, adversarial shortcuts, and skill composition (Lei et al., 2026); ContinualSkillBench studies skill creation and modification across sequential tasks and highlights the difficulty of consolidating experience into compact reusable skills (Guan et al., 2026). Closest to our reliability motivation, GSE uses a global skill-relation graph, cross-task consolidation, and replay-driven verification (Yang et al., 2026a); SkillCommit validates broader abstractions before committing them (He & Yang, 2026); and SkillAdam uses optimization history and adaptive edit budgets to stabilize iterative skill optimization (Li et al., 2026). These methods provide safeguards within particular evolution algorithms, whereas SkillEvoReg asks whether complementary anti-overfitting principles can transfer across different update semantics.

Optimization of prompts and external language-model state. ProTeGi uses textual feedback analogous to gradients together with beam search to improve prompts (Pryzant et al., 2023); OPRO treats an LLM itself as an optimizer over natural-language solutions (Yang et al., 2024); EvoPrompt and Promptbreeder use evolutionary search over prompts (Guo et al., 2024; Fernando et al., 2024); and PromptAgent formulates prompt optimization as strategic search (Wang et al., 2024b). DSPy extends optimization to modular language-model programs (Khattab et al., 2024), while TextGrad propagates language-model feedback through compound AI systems in analogy to automatic differentiation (Yuksekgonul et al., 2025). These works primarily address how to search for higher-performing external state. SkillEvoReg instead regularizes the transitions proposed by an existing updater.

Regularization and counterexample-based validation. Classical learning combines multiple mechanisms to control overfitting: dropout reduces feature co-adaptation (Srivastava et al., 2014), weight decay constrains unnecessary capacity (Krogh & Hertz, 1991), and data augmentation or adversarial training exposes models to informative perturbations (Goodfellow et al., 2015; Zhang et al., 2018). Validation-based early stopping provides an additional safeguard against late-stage deterioration (Prechelt, 1998). SkillEvoReg transfers the principles behind these techniques to discrete

skill evolution rather than reproducing their parametric implementations. CCV is also related to counterexample-guided synthesis, differential testing, and program repair, where carefully chosen failures reveal weaknesses in candidate programs. Counterexample-guided program repair, for example, combines failing examples with fault localization to direct corrections (Orvalho et al., 2025). In our setting, the validated object is a skill transition: incumbent and candidate are first qualified on the original source and then compared on the same candidate-conditioned attacked case, turning the counterexample into an update-level behavioral check.