FROM EXISTING METHODS TO NEW RESEARCH IDEAS: TWO PERSPECTIVES ON RECURSIVE SELF-IMPROVEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Recursive self-improvement (RSI) offers a route to AI systems that participate in their own continued development. Recent methods allow language agents to retain changes to their parameters, memory, skills, and code for subsequent execution and improvement. We examine how these methods can inform new research ideas through two complementary perspectives. First, we introduce a unified taxonomy with nine design dimensions organized into four groups: update targets, update organization, update schedule and scope, feedback and selection. This taxonomy places existing methods in a shared design space, making it possible to identify unexplored combinations of their design choices. Second, we analyze how RSI methods draw on traditional deep learning, spanning optimization, backpropagation, reinforcement learning, meta-learning, distillation, and regularization. These correspondences reveal mechanisms that can guide the design of self-improvement procedures. We analyze two recent studies and their reported results to demonstrate the value of these perspectives for generating new research ideas in RSI. Together, the two perspectives provide a systematic basis for generating new RSI research ideas by recombining design choices and adapting learning mechanisms.

1 INTRODUCTION

Recursive self-improvement (RSI) is a process in which an AI system participates in modifying itself, retains these modifications, and uses the revised system in subsequent execution and improvement. Its appeal lies in the possibility that an improvement makes the system better at producing further improvements. A growing share of AI development could then be carried out by the systems being developed. This possibility underlies Good's suggestion that a sufficiently capable machine might be the last invention humans need to make (Good, 1965).

Recent research has produced numerous RSI methods, including those that update model parameters, revise memory and skills, and modify agent code (Zweiger et al., 2025; Zhang et al., 2026; Li et al., 2026a; Zhang et al., 2025). Several surveys now organize this literature by improvement targets, autonomy levels, or the degree of loop closure (Duan et al., 2026; Chen et al., 2026; Li et al., 2026b). These classifications help describe the field, but assigning a method to a broad category does not fully explain its design. Two methods that revise agent code, for example, can differ in who proposes changes, how earlier versions are reused, and what feedback determines which changes are retained. To use existing work as a source of new research ideas, we need to make these choices explicit and examine how they can be recombined. Existing surveys also discuss connections to earlier learning paradigms, but offer less systematic guidance on using specific mechanisms from traditional deep learning to generate new RSI research ideas. This calls for an analysis of how these mechanisms are adapted in RSI and which principles can be reused.

We view existing RSI methods as combinations of design choices. Separating these choices allows design elements from different methods to be recombined. This view leads to a high-dimensional design space: each dimension captures an aspect of the improvement process, and each method occupies a point determined by its choices along these dimensions. Once the space is characterized, we can locate existing work and identify unexplored combinations that can inspire new research ideas for RSI. To characterize this space, we introduce a unified taxonomy with four groups of

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.

design dimensions: *update targets*, *update organization*, *update schedule and scope*, and *feedback and selection*. These groups encompass nine dimensions, which we use to describe representative methods, allowing methods developed for different settings to be compared within the same space.

Traditional deep learning provides a second source of research ideas for RSI. It has developed mechanisms for choosing updates, learning from feedback, and controlling overfitting, all of which are relevant to recursive self-improvement. Recent work already draws on these connections: SkillAdam adapts ideas from Adam to skill revision; TextGrad applies the idea of backpropagation to textual feedback; and RRSI regularizes changes to agent code (Li et al., 2026a; Yüksekgönül et al., 2025; Xia et al., 2026). We study these correspondences to understand which principles carry over to RSI and how other established learning mechanisms can inspire new research ideas for RSI.

Our two contributions are **a unified taxonomy** and **insights from traditional deep learning**, which together help researchers generate new research ideas for RSI.

2 Background

We define *recursive self-improvement* by the presence of a self-improvement loop: the agentic system participates in modifying itself, retains the changes, and continues execution and improvement from its updated state. The system includes its models and the harness that coordinates them, together with context, memory, skills, and tools. Retained changes can affect task execution or the procedure that produces later updates.

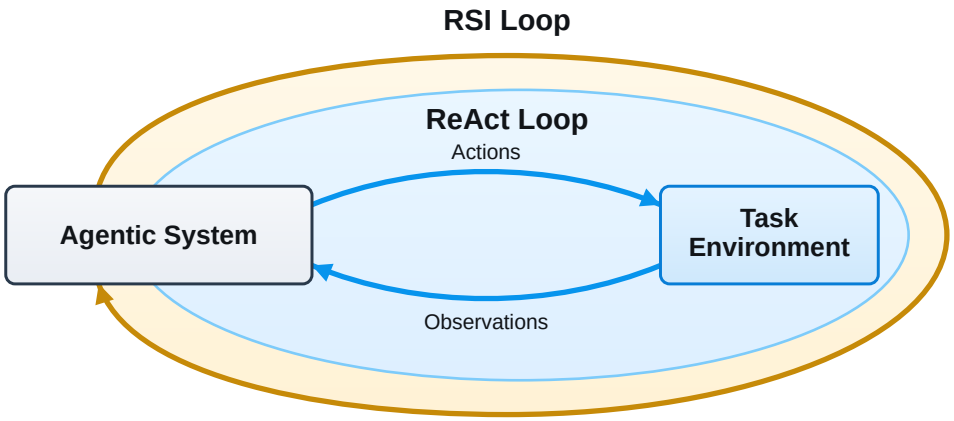

Figure 1: Nested execution and self-improvement loops. The inner ReAct Loop connects the agentic system to its task environment through actions and observations. The outer RSI Loop returns to the same system, which retains modifications for subsequent execution and improvement.

Figure 1 shows the ReAct Loop for task execution nested within the RSI Loop for self-improvement. In the inner *ReAct Loop*, the system acts in an environment and uses the resulting observations to continue the task (Yao et al., 2023). In the outer *RSI Loop*, the system uses execution records and feedback to propose, evaluate, and retain changes to itself. Each outer round can include many action–observation steps and several task executions. Both loops return to the same system, which can contain one agent, teacher–student roles, or multiple interacting versions. Updating the procedure that produces or selects later changes makes the recursive dependence explicit.

3 Related Work

Existing surveys organize self-improvement through autonomy levels (Duan et al., 2026), improvement targets and loop closure (Chen et al., 2026), or evolution targets within a Proposal–Feedback–Optimization loop (Li et al., 2026b). Gao et al. (2026) further distinguish what, when, how, and

where agents evolve. These taxonomies do not jointly represent update roles, version inheritance, update frequency, and acceptance criteria as separate dimensions of a common design space. Their category labels therefore offer limited guidance on which choices can be recombined across methods. Our taxonomy makes these choices explicit, allowing researchers to compare methods within the same category and identify unexplored combinations as starting points for new RSI research ideas.

Prior surveys also connect self-improvement to established learning mechanisms. Li et al. (2026b) discuss meta-learning and self-play, while Gao et al. (2026) describe how TextGrad draws on backpropagation. Madžar & Mekterović (2026) use neural architecture search to organize the automated design of agent scaffolds around largely frozen models. These connections are mainly examined within particular learning paradigms or method families, leaving limited guidance on how to transfer mechanisms across RSI update targets. We compare the problem each traditional deep learning mechanism addresses, its implementation in an RSI method, and the principle available for reuse. This analysis connects parameter updates with revisions to memory, skills, and code, providing a systematic basis for adapting established mechanisms to generate new RSI research ideas.

4 A Unified Taxonomy

Our taxonomy locates existing RSI methods in a shared design space. It comprises four groups of design dimensions: *Update targets*, *Update organization*, *Update schedule and scope*, and *Feedback and selection*. Together, these groups contain nine dimensions, shown in Figure 2. A method's choices along these dimensions describe its position in the space; comparing these positions helps identify combinations that can inspire new research ideas. A method can have several labels within a dimension when, for example, it updates several artifacts.

4.1 Update targets

Update targets describe the parts of an agentic system that are changed and retained for subsequent use. We characterize these targets through the *RSI artifact* dimension.

RSI artifact. This dimension distinguishes updates to model weights (*Parametric*) from updates to resources outside those weights (*Non-parametric*). Non-parametric targets are grouped by their role in the system: *Context* supplies information to model calls; *Memory* stores records for later retrieval; *Skill* packages reusable instructions and supporting resources; and *Harness code* coordinates model calls, tool execution, and control flow. *Other artifact* covers other retained resources.

Several artifacts can change together. SkillAdam maintains a skill and a record of earlier editing attempts (Li et al., 2026a), while SkillSmith co-evolves skills and tools (Wei et al., 2026). These methods therefore occupy more than one artifact category.

4.2 Update organization

Update organization describes how the system produces successive versions of itself. It combines two dimensions: *Artifact updater* identifies who forms a change, and *RSI topology* describes how a new version builds on earlier versions.

Artifact updater. This dimension describes who forms an update and distinguishes three categories: *Self*, *Teacher*, and *Joint*. *Self* denotes updates produced by the agent being improved. *Teacher* denotes updates produced by a separate teacher using the agent's execution records and feedback. *Joint* combines contributions from the agent and teacher: the agent proposes or revises a change, and the teacher further refines, curates, or selects it.

RSI topology. We distinguish *Sequential*, *Tree*, and *Graph* topologies according to the dependencies between versions. *Sequential* improvement repeatedly updates the current version to form a single chain. A *Tree* allows archived versions to branch, with each child derived from one parent. A *Graph* allows a new version to draw on retained state, changes, or outcomes from several historical versions or lineages. DGM's branching archive and MGM's comparative evolution illustrate the latter two organizations. DGM selects an archived agent as the parent and modifies its code to produce a child; repeatedly branching from archived agents forms a version tree (Zhang et al., 2025). MGM also compares the target agent's execution trace with that of an agent from another lineage on the same

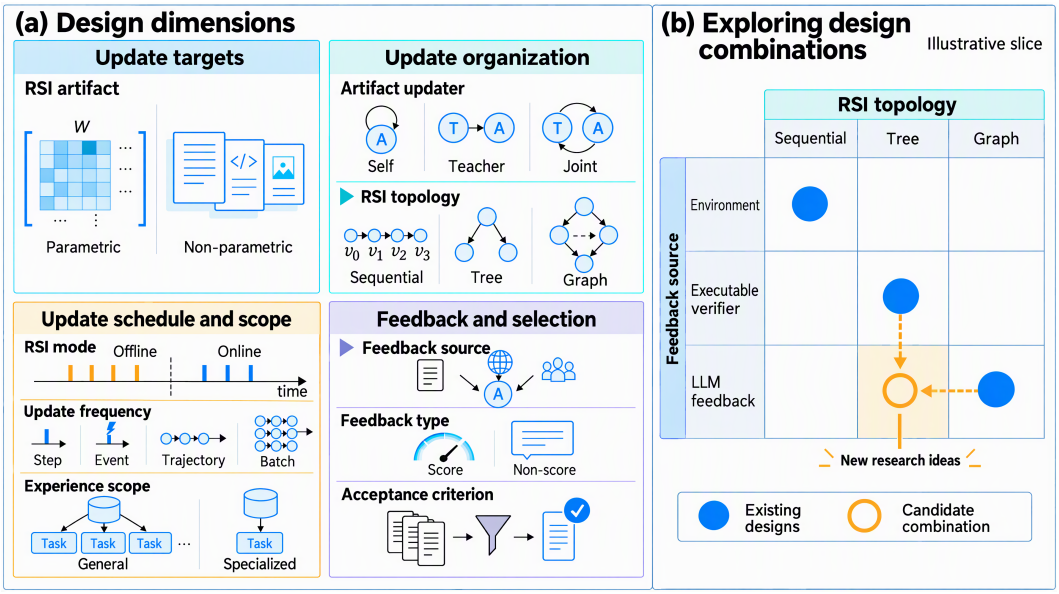

Figure 2: A unified taxonomy for identifying new research ideas. (a) Nine design dimensions are organized into four groups. (b) An illustrative slice across RSI topology and feedback source shows how choices from existing designs can form candidate combinations.

task, then uses this comparison to guide code changes, creating information dependencies across branches (Liu et al., 2026).

4.3 Update schedule and scope

Update schedule and scope describes when retained changes are made and which tasks they are intended to support. Three dimensions specify these choices: *RSI mode* places improvement relative to system use, *Update frequency* records when individual updates are triggered, and *Experience scope* describes the intended range of reuse.

RSI mode. This dimension distinguishes three modes according to when improvement occurs relative to system use: *Offline*, *Online*, and *Offline* $\rightarrow$ *Online*. *Offline* improvement occurs in a separate training or search phase before evaluation or deployment. *Online* improvement occurs during use, so later tasks can use the newly retained state. *Offline* $\rightarrow$ *Online* first builds a resource offline and continues to update it during use.

Update frequency. Within these modes, the timing of individual updates is described by four categories: *Step*, *Event*, *Trajectory*, and *Batch*. A *Step* update follows an action–observation step. An *Event* update is triggered by a specified condition, such as a detected failure. A *Trajectory* update follows a complete task execution, while a *Batch* update uses a group of trajectories or candidates. Together, mode and frequency can describe an online system that consolidates a batch of task results or an offline optimizer that revises an artifact after each trajectory.

Experience scope. This dimension describes the tasks for which an updated artifact is intended to be reused. It distinguishes *General* artifacts that support reuse across task types or domains from *Specialized* artifacts that serve a particular instance, topic, or task family. A system can retain both a general execution procedure and specialized skills.

4.4 Feedback and selection

Feedback and selection describes how the system obtains guidance for updates and decides which changes to retain. We separate three dimensions: *Feedback source* identifies where this guidance comes from, *Feedback type* describes its form, and *Acceptance criterion* specifies the basis for retaining a change.

Feedback source. We group the information used during improvement into six categories according to its origin. A *Train/dev set* provides task examples and reference outputs, while a *Benchmark* supplies results from a standardized evaluation suite. The *Environment* provides observations and action outcomes, and an *Executable verifier* produces test or validation results. *LLM feedback* supplies model-generated evaluations or suggestions; a *Human* can provide corrections, preferences, or guidance. A single update can draw on several of these sources.

Feedback type. This dimension groups feedback by the form in which it guides improvement, distinguishing *Score* and *Non-score* feedback. *Score* feedback summarizes an outcome or assessment as a *Binary* result or a *Non-binary* numerical or categorical evaluation. *Non-score* feedback provides content for revising the system: *Ground truth* supplies a reference answer or behavior, *LLM-as-a-judge* supplies a textual assessment, and *Other* covers observations or diagnostics. One source can provide both types: a model may return a numerical rating together with an explanation.

Acceptance criterion. This dimension describes how the system decides whether to retain a proposed change. The options are *Artifact validation*, *Instance result*, *Benchmark score*, *Combined metrics*, and *No validation*. *Artifact validation* checks whether the modified artifact meets requirements such as successful compilation or interface compatibility. An *Instance result* uses performance on an individual task, while a *Benchmark score* uses aggregate performance on a benchmark. *Combined metrics* considers several measures, such as performance and cost. *No validation* denotes direct retention without a separate acceptance check.

Table 1 locates representative methods in this space using selected dimensions. Appendix A explains how to identify these choices in a method's improvement procedure. Comparing their combinations of choices helps identify well-studied designs and alternatives worth exploring. Choices that have proved useful in separate methods can then be considered together to generate new research ideas. Section 6 illustrates this use of the taxonomy through feedback sources and version topologies. Comparing the effectiveness of these designs also requires a common basis for evaluation.

4.5 Discussion: Toward unified evaluation

Recent research has introduced a growing number of RSI methods and benchmarks, but their results are often obtained under different evaluation conditions. Beyond the choice of tasks and datasets, studies differ in how improvement is combined with testing, how tasks are ordered, and what feedback is available to guide updates. These differences shape the improvement process and make reported gains difficult to compare across studies.

An important distinction concerns when self-improvement occurs. In *Online* evaluation, the system continues to improve while processing evaluation tasks (Wei et al., 2025). In *Offline* evaluation, improvement takes place in a separate training or search phase, followed by held-out testing with further self-improvement updates disabled (Ai et al., 2025). *Offline* $\rightarrow$ *Online* evaluation combines offline improvement with continued updates during evaluation. These settings measure different aspects of improvement: offline testing examines whether retained changes generalize to unseen tasks, while online testing also examines how improvement unfolds during use.

Even within one mode, task order and how feedback is obtained can affect results. Later tasks can benefit from knowledge and skills acquired on earlier ones, so changing the order can change performance on the same task set. Some evaluations provide reference answers or executable test results, while others rely on model-generated assessments. These signals provide different information for diagnosing failures and deciding what to revise.

Establishing shared benchmarks and evaluation protocols is therefore important for RSI research. Under comparable conditions, researchers could better assess which design choices improve performance and use these findings to guide subsequent work.

5 Insights from Traditional Deep Learning

Existing RSI methods already draw on ideas from traditional deep learning. We examine these connections through optimization, backpropagation, reinforcement learning, meta-learning, distillation, and regularization. For each connection, we explain the problem addressed by the traditional method

Table 1: Selected design choices in representative methods. Update organization lists the artifact updater followed by the version topology.

Method	RSI artifact	Update organization	RSI mode	Role of feedback
SkillAdam	Skill; Memory	Teacher; Sequential	Offline	Direction and edit budget
MemRL	Memory	Self; Sequential	Online	Retrieval utility
SEAL	Parametric	Self; Sequential	Offline	Post-update performance
DGM	Harness code; Context	Joint; Tree	Offline	Candidate execution
SIFT	Harness code; Context	Teacher; Tree	Offline	Pairwise comparison and execution
RRSI	Harness code; Context; Memory; Skill	Teacher; Graph	Offline	Performance, cost, and stability

and how an RSI method adapts it, identifying principles that can inspire new research ideas. Figure 3 summarizes these correspondences.

5.1 Optimization

Optimizers in traditional deep learning determine the direction and magnitude of parameter updates. Adam, for example, uses information accumulated from current and past gradients to compute its parameter updates (Kingma & Ba, 2015). Iterative skill revision in RSI faces a similar problem: deciding what to modify and how much to change at each iteration.

SkillAdam draws on this optimizer design to guide the revision of agent skills (Li et al., 2026a). It records identified problems, attempted fixes, and their outcomes, allowing later revisions to build on earlier attempts. It also compares task performance before and after a revision to determine how much of the skill may be changed next. When the gains vary substantially across evaluation cases, it reduces the amount of editing allowed in the next iteration; more consistent gains permit broader changes. These mechanisms use the history and results of previous revisions to guide both what to change and how much to change.

5.2 Backpropagation

Training a neural network requires using errors in its predictions to adjust parameters across its layers. Backpropagation works backward through the computation graph to calculate how the loss changes with each parameter, providing gradients for parameter updates (Rumelhart et al., 1986). RSI systems face a related problem: using feedback on a task result to guide changes to individual components.

TextGrad draws on this idea to propagate textual feedback through a system (Yüksekgönül et al., 2025). It represents operations and their input–output relationships as a computation graph, then passes feedback on the final result backward through that graph. At each step, a language model uses the local inputs, outputs, and feedback to generate revision suggestions for upstream variables. An optimizer then uses these suggestions to revise selected variables, such as instructions and code. For RSI, this suggests a way to turn task-level feedback into specific changes by following the steps that produced the result.

5.3 Reinforcement learning

Reinforcement learning uses rewards from interaction to improve decisions. Value-based methods estimate the expected return of available choices and use these estimates to guide selection (Sutton & Barto, 2018). An agent that retrieves memories for a task faces a related problem: selecting records that help it succeed. Task outcomes provide a signal for learning which memories to use.

MemRL applies value estimation to this selection process (Zhang et al., 2026). It first retrieves a pool of memories by semantic similarity, then selects from this pool using both relevance and learned utility values. After the agent executes the task, the resulting reward updates the utility values of the memories supplied to the agent. Memories associated with higher rewards are consequently more likely to be selected for later tasks. With the language model's parameters fixed, feedback improves how the agent selects from its accumulated records.

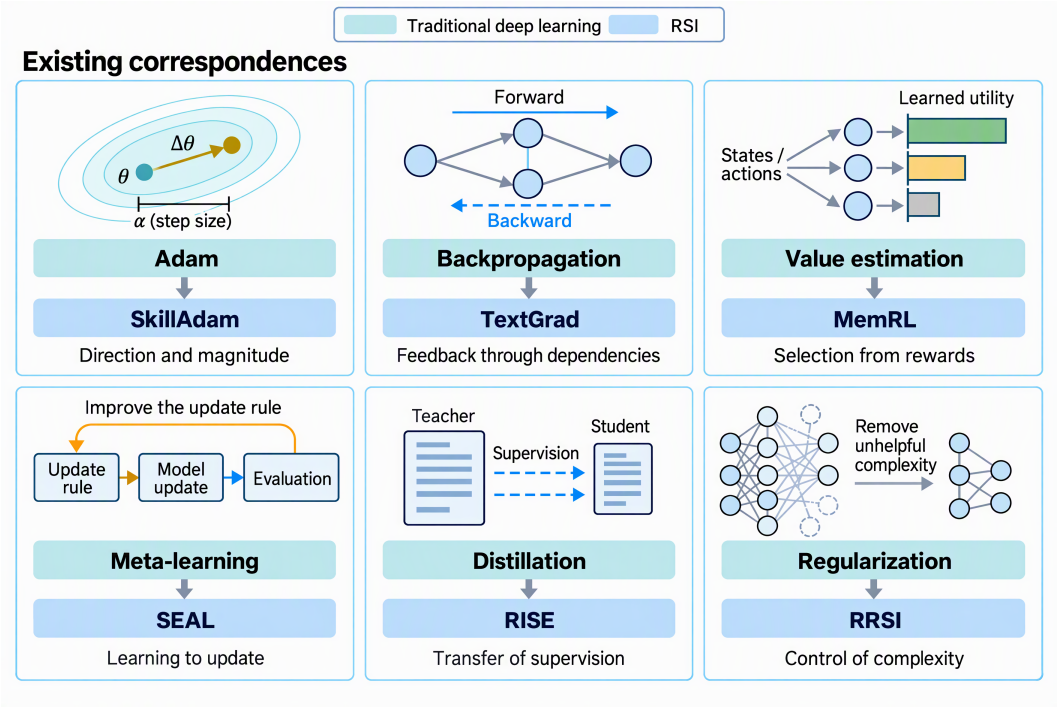

Figure 3: Six correspondences between traditional deep learning and RSI. Each panel links a learning mechanism to an RSI method and summarizes the principle it uses.

5.4 Meta-learning

Meta-learning uses performance after learning to improve how learning is carried out. For example, it can optimize a model's initialization or learn a parameter-update rule by evaluating how well the model performs after training (Finn et al., 2017; Andrychowicz et al., 2016). In RSI, this principle can guide the production of training data and update instructions: their quality can be assessed through the learning they produce.

SEAL implements this idea by training a model to generate its own self-edits (Zweiger et al., 2025). Given new information or a task, the model produces training materials or instructions for updating its parameters. The proposed update is applied through fine-tuning, and the updated model is evaluated on the relevant task. Its performance then serves as a reward for reinforcement learning of the self-edit generator. This creates two connected learning processes: one updates the model using generated materials, and the other improves how those materials and update instructions are produced.

5.5 Distillation

Knowledge distillation transfers behavior from a teacher model to a student by training the student to match the teacher's output probabilities (Hinton et al., 2015). These probabilities provide guidance about alternative outputs beyond a single target answer. A self-improving system can use this mechanism while also constructing and updating the source of its supervision.

RISE combines distillation with reinforcement learning using verifiable rewards (Li et al., 2026c). It compares the model before and after a reinforcement learning update, then constructs a teacher by extending the observed change farther in the same direction. This extrapolation can operate on model weights or output log-probabilities. The student is trained to match the resulting teacher's token probabilities. As reinforcement learning and distillation alternate, each new training update provides the basis for another teacher. RISE thus incorporates teacher construction into the improvement process, using the model's own training trajectory to produce supervision for subsequent learning.

5.6 Regularization

In neural network training, regularization can limit model complexity and reduce overfitting. For example, penalizing the number of nonzero parameters encourages sparse models (Louizos et al., 2018). RSI faces a related problem when an agent's harness is repeatedly modified and evaluated on a limited set of tasks: the harness can accumulate changes that fit those tasks but transfer poorly.

RRSI applies regularization to the evolution of an agent's harness (Xia et al., 2026). It gradually reduces the number of independent edits allowed in each proposal and identifies components that repeatedly fail to yield gains as targets for removal. Candidate selection also weighs performance gains against execution cost. These mechanisms limit additions and remove unproductive components as the harness evolves, extending complexity control to the code and resources retained by an agent. Section 6.2 examines the reported effects of these constraints on transfer and execution cost.

Together, these connections provide concrete mechanisms for exploring the design space defined by the taxonomy. The next section uses SIFT and RRSI as case studies to illustrate how the two perspectives can guide RSI research, from identifying a problem to examining a design and its reported results.

6 Applying the Two Perspectives to RSI Research

We use SIFT and RRSI as case studies to illustrate how the two perspectives can guide RSI research. The taxonomy helps identify design choices relevant to a practical problem, while connections to traditional deep learning suggest mechanisms for addressing it. In each case, we connect the research problem to a concrete design and examine its reported results.

6.1 Ideas from the unified taxonomy

Applying the taxonomy starts with identifying a limitation and locating design choices that could address it. In code evolution, evaluating candidate harnesses through task execution can make search expensive (Fu et al., 2026). DGM, HGM, and SIFT all search over versions of a coding agent's harness. The taxonomy describes their shared choices as *Harness code* for the *RSI artifact* and *Tree* for the *RSI topology*, while allowing their sources of feedback to be compared separately.

This directs attention to *Feedback source*: cheaper signals can guide exploration before task evaluations finish. SIFT makes this choice by adding a language model that compares candidate patches in pairs. Their win–loss outcomes determine relative strength estimates, which help select versions for further modification and prioritize candidates for task evaluation. Through the taxonomy, this design can be understood as combining tree search over code with model-generated feedback. Table 2 shows how this choice differs from the feedback used in DGM and HGM.

Table 2: Feedback choices in tree-based harness search. The methods share an update target and version topology while using different signals to guide exploration. Sources: DGM (Zhang et al., 2025), HGM (Wang et al., 2025), and SIFT (Fu et al., 2026).

Method	Signal for choosing a branch	Role of task execution
DGM	Candidate performance and archive sampling	Evaluates modified agents
HGM	Performance of descendants estimates a branch's improvement potential	Supplies evidence for that estimate
SIFT	Pairwise judgments supplement execution scores	Focuses evaluation on promising candidates

With o3-mini as the execution model, SIFT reports Polyglot accuracy of 29.8% without a judge, 31.6% with GPT-5-mini, and 35.1% with GPT-5.4 (Fu et al., 2026). The 5.3-percentage-point gain with GPT-5.4 comes from changing the feedback source while keeping the update target, version topology, and execution model fixed, illustrating how a choice along one taxonomy dimension can improve an existing design.

6.2 Ideas from traditional deep learning

Connections to traditional deep learning can suggest mechanisms for addressing problems observed in RSI. Repeatedly revising a harness on the same task set can produce changes that fit those tasks but increase execution cost or fail to transfer. In neural network training, regularization addresses a related problem by controlling the complexity of what is learned. Applying this principle to harness evolution suggests constraining both the modifications proposed in each round and the changes retained for later use.

RRSI implements this approach at both stages (Xia et al., 2026). During proposal generation, it limits the number of independent edits and uses recorded outcomes to guide further changes, including the removal of unproductive components. During selection, it accounts for evaluation noise and weighs performance gains against execution cost. These choices translate complexity control into rules for modifying and retaining an agent's code and resources.

The workspace ablation in Xia et al. (2026) compares these constraints using the same initial harness, frozen execution model, and search budget. Table 3 summarizes the key results. RRSI improves the out-of-distribution average by 3.3 points over unregularized evolution and reduces execution tokens by 36.3%. Removing either group of constraints weakens transfer and raises token use. These results show how regularization can improve the transfer and efficiency of an evolved harness, even when unregularized evolution scores higher on the tasks used for search.

Table 3: Selected RRSI ablations from Xia et al. (2026), Table 2. Evolve is the Harvey LAB evolution score. OOD averages JobBench, GDPval, and APEX-Agents. Tokens count policy tokens per task trial with the resulting harness, in millions.

Variant	Evolve	OOD	Tokens
Unregularized evolution	92.8	40.3	3.80
Without proposal regularizers	90.7	41.9	2.69
Without acceptance regularizers	91.5	41.0	3.59
RRSI	90.5	43.6	2.42

These cases show how the taxonomy can help identify design alternatives and how traditional deep learning can supply mechanisms for addressing RSI problems. Researchers can use both perspectives to generate new research ideas, turn them into concrete design changes, and examine their effects through targeted comparisons.

7 Conclusion

We presented two perspectives on RSI: a unified taxonomy and insights from traditional deep learning. The taxonomy places existing methods in a shared design space, helping researchers compare design choices and identify unexplored combinations. Our analysis of traditional deep learning connects RSI mechanisms to established approaches for optimization, learning from feedback, and controlling overfitting. We analyzed two recent studies and their reported results to illustrate the value of these perspectives for RSI research. Together, the two perspectives provide a basis for generating new research ideas by recombining existing design choices and adapting learning mechanisms to self-improving systems.

AI use statement

Generative AI assisted literature search and organization, manuscript drafting, and figure generation. Three coauthors re-screened and verified the literature records. The authors reviewed and rewrote the initial draft.

REFERENCES

Qingyao Ai, Yichen Tang, Changyue Wang, Jianming Long, Weihang Su, and Yiqun Liu. MemoryBench: A Benchmark for Memory and Continual Learning in LLM Systems. *CoRR*, abs/2510.17281, 2025. doi: 10.48550/ARXIV.2510.17281. URL https://doi.org/10.48550/ARXIV.2510.17281.

Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 3981–3989, 2016. URL https://proceedings.neurips.cc/paper/2016/hash/fb87582825f9d28a8d42c5e5e5e8b23d-Abstract.html.

Mingguang Chen, Licheng Wang, and Bo Qu. Recursive Self-Improvement in AI: From Bounded Self-Refinement to Autonomous Research Loops. *CoRR*, abs/2607.07663, 2026. doi: 10.48550/ARXIV.2607.07663. URL https://doi.org/10.48550/ARXIV.2607.07663.

Yi Duan, Ying Liu, Zirui Tang, Haodong Chen, Jun Zhou, Yumou Liu, Bangrui Xu, Yukai Wu, Sidi Chen, Yuhan Zhou, Haoyu Wang, Xiaoyou Yu, Shaokun Han, Xuzhou Zhu, Le Zhou, Bolin Lu, Wei Zhou, Jiachen Liu, Nuozhou Fang, Jiaxin Tian, Ruoyu Chen, Yuxuan Li, Kai Zuo, Kaiyan Zhang, Qianyu Yang, Zijie Wang, Jiantao Qiu, Conghui He, Guoliang Li, Bowen Zhou, Zhiyuan Liu, Zhoufutu Wen, Jihua Kang, Xuanhe Zhou, and Fan Wu. The Last AI Built by Humans: Toward Genuine Recursive Self-Improvement. arXiv preprint arXiv:2609.11873, 2026. URL https://arxiv.org/abs/2609.11873.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1126–1135. PMLR, 2017. URL http://proceedings.mlr.press/v70/finn17a.html.

Xinghong Fu, Aravinth Kulanthaivelu, and Yutaro Yamada. Self improvement via fast tree-search. arXiv preprint arXiv:2609.19526, 2026. URL https://arxiv.org/abs/2609.19526.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A Survey of Self-Evolving Agents: What, When, How, and Where to Evolve on the Path to Artificial Super Intelligence. *Trans. Mach. Learn. Res.*, 2026, 2026. URL https://openreview.net/forum?id=CTr3bovS5F.

Irving John Good. Speculations Concerning the First Ultraintelligent Machine. *Adv. Comput.*, 6: 31–88, 1965. doi: 10.1016/S0065-2458(08)60418-0. URL https://doi.org/10.1016/S0065-2458(08)60418-0.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the Knowledge in a Neural Network. *CoRR*, abs/1503.02531, 2015. URL http://arxiv.org/abs/1503.02531.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

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. arXiv preprint arXiv:2609.08944, 2026a. URL https://arxiv.org/abs/2609.08944.

Hanjing Li, Qiguang Chen, Chenyuan Zhang, Qionglin Qiu, Fanqing Meng, Mengkang Hu, Libo Qin, and Min Zhang. Towards AI That Improves Itself: A Survey of Recursive Self-Improvement. Author-hosted manuscript, 2026b. URL https://lee1003-lee.github.io/Awesome-RSI-Research/assets/rsi-survey.pdf.

Yang Li, Semih Yavuz, and Shafiq Joty. RISE: Recursive improvement via self-extrapolating policy distillation. arXiv preprint arXiv:2609.05295, 2026c. URL https://arxiv.org/abs/2609.05295.

Changzhi Liu, Yilun Liu, Sikuan Yan, Volker Tresp, and Yunpu Ma. Mendel Gödel Machine: Recursive Self-Improving Coding Agents via Comparative Evolution. *CoRR*, abs/2608.07645, 2026. doi: 10.48550/ARXIV.2608.07645. URL https://doi.org/10.48550/ARXIV.2608.07645.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning Sparse Neural Networks through L_0 Regularization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=H1Y8hhg0b.

Maksim Madžar and Igor Mekterović. Automated Design of Agentic Systems: A Survey of Algorithms for Searching, Optimizing, and Evolving LLM Agents, Workflows, and Prompts. Preprints.org, 2026. URL https://www.preprints.org/manuscript/202606.0238. Preprint.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi: 10.1038/323533a0. URL https://doi.org/10.1038/323533a0.

Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction, 2nd Edition*. MIT Press, 2018. URL http://www.incompleteideas.net/book/the-book-2nd.html.

Wenyi Wang, Piotr Piekos, Nanbo Li, Firas Laakom, Yimeng Chen, Mateusz Ostaszewski, Mingchen Zhuge, and Jürgen Schmidhuber. Huxley-Gödel Machine: Human-Level Coding Agent Development by an Approximation of the Optimal Self-Improving Machine. *CoRR*, abs/2510.21614, 2025. doi: 10.48550/ARXIV.2510.21614. URL https://doi.org/10.48550/ARXIV.2510.21614.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. Evo-Memory: Benchmarking LLM Agent Test-time Learning with Self-Evolving Memory. *CoRR*, abs/2511.20857, 2025. doi: 10.48550/ARXIV.2511.20857. URL https://doi.org/10.48550/ARXIV.2511.20857.

Yangbo Wei, Zhen Huang, Shaoqiang Lu, Junhong Qian, Qifan Wang, Chen Wu, and Lei He. SkillSmith: Co-Evolving Skills and Tools for Self-Improving Agent Systems. *CoRR*, abs/2606.01314, 2026. doi: 10.48550/ARXIV.2606.01314. URL https://doi.org/10.48550/ARXIV.2606.01314.

Peng Xia, Rujun Han, Zifeng Wang, Yanfei Chen, Yufan Zhang, Yoonho Lee, Chengsong Huang, Han Yu, Zhongying CuiZhu, Yifei Ming, Huaxiu Yao, Burak Gokturk, Tomas Pfister, and Chen-Yu Lee. RRSI: Regularized recursive self-improvement of agent harnesses. arXiv preprint arXiv:2609.24972, 2026. URL https://arxiv.org/abs/2609.24972.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Mert Yüksekgönül, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative AI by backpropagating language model feedback. *Nat.*, 639(8055):609–616, 2025. doi: 10.1038/S41586-025-08661-4. URL https://doi.org/10.1038/S41586-025-08661-4.

Jenny Zhang, Shengran Hu, Cong Lu, Robert T. Lange, and Jeff Clune. Darwin Godel Machine: Open-Ended Evolution of Self-Improving Agents. *CoRR*, abs/2505.22954, 2025. doi: 10.48550/ARXIV.2505.22954. URL https://doi.org/10.48550/ARXIV.2505.22954.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Weinan Zhang, Ying Wen, Zhiyu Li, Feiyu Xiong, Yutao Qi, Bo Tang, and Muning Wen. MemRL: Self-Evolving Agents via Runtime Reinforcement Learning on Episodic Memory. *CoRR*, abs/2601.03192, 2026. doi: 10.48550/ARXIV.2601.03192. URL https://doi.org/10.48550/ARXIV.2601.03192.

Adam Zweiger, Jyothish Pari, Han Guo, Yoon Kim, and Pulkit Agrawal. Self-Adapting Language Models. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/6b41e04c41726e2a60e456d0a2b961ab-Abstract-Conference.html.

A Describing methods with the taxonomy

To place a method in the design space, we follow one round of its improvement procedure, from proposing a modification to retaining it for later use. This connects the dimensions in Section 4 to concrete operations in the method. The following examples explain how to identify its choices within each of the four groups.

Update targets. First, identify the resources that the method changes and carries into subsequent execution or improvement. Their roles determine the *RSI artifact* labels. For example, SkillAdam revises a skill used to execute tasks and maintains a record of problems and attempted solutions to guide later revisions (Li et al., 2026a). We therefore describe its targets as both *Skill* and *Memory*.

Update organization. Next, trace who forms the modification and which earlier versions inform it. For *Artifact updater*, we examine how the task agent and any teacher contribute to proposing, revising, or selecting the retained change. For *RSI topology*, we trace the state and feedback inherited from earlier versions. RRSI, for example, uses records of previous candidates, including rejected edits, to guide later proposals (Xia et al., 2026). Under our definition, these dependencies give it a *Graph* topology: a new proposal draws on both the current harness and information from other evaluated candidates.

Update schedule and scope. Then locate the artifact updates relative to task execution. *RSI mode* describes the phase in which improvement occurs, while *Update frequency* describes what triggers each update. For example, a method that revises a skill from a group of trajectories during a separate optimization phase is *Offline* with *Batch* updates. Continuing these batch updates during use gives an *Offline* $\rightarrow$ *Online* procedure, with *Batch* frequency in both phases. *Experience scope* describes the intended reuse of each artifact: a procedure intended for several task domains is *General*, while a skill for a particular task family is *Specialized*.

Feedback and selection. Finally, follow the information used to guide modifications and retain candidates. A model-generated pairwise preference has *LLM feedback* as its source and *Score* as its type. If the model also provides a diagnosis used to revise the candidate, the procedure additionally uses *Non-score* feedback. The *Acceptance criterion* records the decision that makes a change persistent. A compilation check corresponds to *Artifact validation*; a rule that weighs task performance against execution cost corresponds to *Combined metrics*.