VeriLoop Coder-E1: A VeriLoop Spiral-Driven Intelligent System for Verifiable Recursive Self-Improvement

Libo Wang, Haiqi Li, Anlin Xiang, Houde Liu *
Intelligent Robotics Laboratory
Tsinghua Shenzhen International Graduate School, Tsinghua University
Email: wanglibo@sz.tsinghua.edu.cn
*Corresponding author

Abstract

Distinguishing system modification from genuine improvement remains difficult in self-modifying agent harnesses. We introduce the VeriLoop Spiral and define recursive self-improvement as a verified correction that changes how subsequent errors are discovered, judged, and corrected. On a frozen Qwen3.6-27B backbone, we build VeriLoop Coder-E1 from four Surface Host parameter-efficient fine-tuning (PEFT) adapters and a Self-Harness. A dual-gate delayed-internalization mechanism governs evidence admission, validation, inheritance, and revocation. The complete system achieves 85.20% on SWE-bench Verified, 62.38% on SWE-bench Pro, 76.40% on Terminal-Bench 2.0, and 33.63% on DeepSWE. The primary adapter endpoints and all four predeclared Self-Harness mechanism tests receive statistical support, while the LoRA matrices account for only 0.012895% of the backbone parameters. These results show that evidence-governed control can establish a verifiable and reversible path for method inheritance without modifying backbone weights, providing a testable recursive control mechanism for model-harness collaboration in software-engineering agents.

1 Introduction

As code-oriented large language models move into real software-engineering environments, the ceiling of system capability is no longer determined by model weights in isolation. It is increasingly determined by an execution system in which model reasoning is coupled to runtime control (Jimenez et al., 2024; Yang et al., 2024; Lin et al., 2026; Zhang et al., 2026a). SWE-bench explicitly distinguishes the language model itself from a complete coding system that may include retrieval, tool use, repeated generation, and review mechanisms (SWE-bench, 2026). Recent public evaluations likewise pair Claude, GPT, Kimi, GLM, and Grok with execution harnesses such as Claude Code, Codex, Kimi Code, OpenHands, and Grok Build (Anthropic, 2026; Bolin, 2026; Moonshot AI, 2026; Z.AI, 2026; SpaceXAI, 2026). These systems indicate that end-to-end software-engineering performance depends on coordination between model reasoning and runtime control rather than on parameter scale or single-pass generation quality alone. A harness operates outside model weights and maintains task contracts, repository state, tool actions, execution receipts, verification gates, and rollback boundaries, enabling iterative generation, testing, and revision from environmental feedback (Bolin, 2026; Lin et al., 2026; Zhang et al., 2026a). It has therefore become more than a peripheral tool wrapper. Even with fixed model weights, changes to context organization, tool interfaces, state persistence, verification, and recovery can materially alter repository-level task success (Lopopolo, 2026; Lin et al., 2026). Method-level recursive self-improvement becomes plausible only when verified experience further changes how later tasks acquire evidence, judge errors, and select repairs (Zhang et al., 2026a).

A continuous line of work from prompting and context evolution to harness engineering provides empirical support for this system-level effect (Zhang et al., 2026c; Lin et al., 2026; Zhang et al., 2026a). SWE-agent showed that a model-oriented agent-computer interface can alter repository navigation, code editing, test execution, and ultimately task performance (Yang et al., 2024). Agentic Context Engineering (ACE) turns a static prompt into a strategy context that evolves through generation, reflection, and curation, reporting a mean task improvement of 10.6% and allowing smaller open models to reach the level of leading production systems on AppWorld (Zhang et al., 2026c). Agentic Harness Engineering (AHE) goes further by optimizing tools, middleware, and long-term mem-

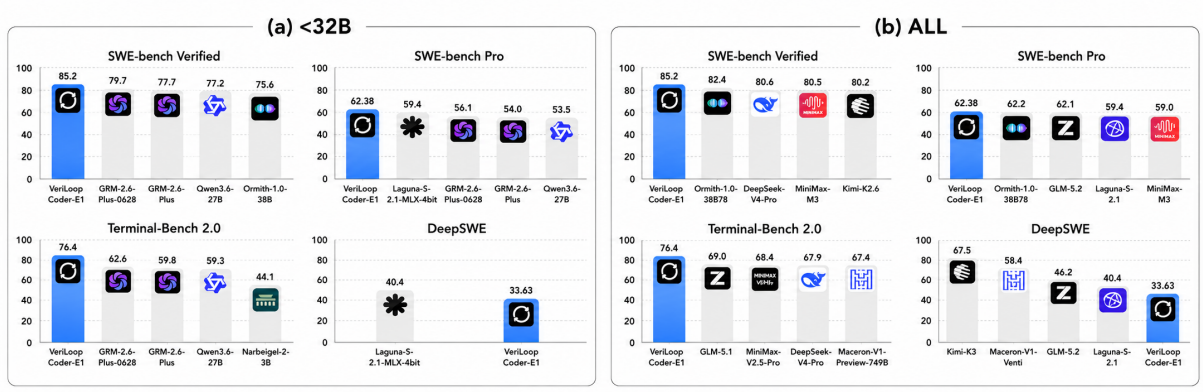


Figure 1: Scale-stratified performance comparison of VeriLoop Coder-E1 on software-engineering benchmarks. The left panel restricts models to fewer than 32B parameters, whereas the right panel imposes no parameter-scale restriction. Public results are compared separately on SWE-bench Verified, SWE-bench Pro, Terminal-Bench 2.0, and DeepSWE. Data were taken from the official VeriLoop Coder-E1 model page and the official leaderboards of the four benchmarks, retrieved on July 27, 2026. Because the benchmarks differ in task distributions and evaluation protocols, scores are intended only for within-benchmark comparison and should not be compared directly across benchmarks.

ory while model weights remain frozen. It raises Terminal-Bench 2.0 pass@1 from 69.7% to 77.0% and reports cross-model gains of 5.1–10.1 percentage points (Lin et al., 2026). Self-Harness then advances the harness from an optimized runtime framework to a closed loop that can revise its own control mechanisms from execution traces. It identifies model-specific weaknesses from failed trajectories, proposes minimal harness changes, and admits them only after regression checks. On a held-out Terminal-Bench 2.0 set, the reported pass rates of MiniMax M2.5, Qwen3.5-35B-A3B, and GLM-5 increase from 40.5% to 61.9%, 23.8% to 38.1%, and 42.9% to 57.1%, respectively (Zhang et al., 2026a). These results show that a harness can use evidence from its own execution to alter how later tasks are handled, providing an engineering basis for testable method-level self-improvement.

However, a harness that revises itself from its own trajectories demonstrates only that self-modification has entered the runtime control layer. It does not establish that genuine improvement has occurred (Huang et al., 2024). Current accounts of recursive self-improvement often expand modification authority progressively, from outputs, plans, and memory to evaluation, training, and eventually successor systems (Chen et al., 2026). The underlying problem is that if proposal generation, evidence admission, success criteria, and inheritance decisions remain governed by the same objective, evaluator, and cognitive frame, the criterion of improvement is not independent of the mechanism being improved. The system can then mistake adaptation to its evaluator for methodological progress (Panickssery et al., 2024; Gao et al., 2023; Armstrong et al., 2020). If a Self-Harness recursively closes this loop, evaluator omissions, task overfitting, random fluctuations, and cognitive blind spots may be accepted as method advantages and then encoded into routing, verification, memory, and stopping rules. A local misjudgment can therefore acquire cross-task persistence (Langosco et al., 2022; Armstrong et al., 2020). What is missing is not deeper modification authority, but an eligibility criterion for improvement that can challenge the current method, restrict the scope of inheritance, and allow later counterevidence to demote or revoke it (Armstrong et al., 2020; Thomas et al., 2015).

We therefore shift the criterion for recursive self-improvement from *what a system can modify* to *how a verified correction changes future error correction*. Rewriting an output, changing a harness, or updating parameters establishes only that system state has changed. The recursive threshold is crossed only when an evidence-tested correction changes how later tasks discover, judge, or repair errors and that change is actually inherited (Huang et al., 2024; Schmidhuber, 2007). This definition draws on Peirce's fallibilism and theory of inquiry, in which belief remains revisable under genuine doubt and external experience (Peirce, 1877, 1955), and on Popper's account of knowledge growth through falsifiability, severe testing,

and error elimination (Popper, 1962). Recursive self-improvement, on this view, is not repeated self-modification. It is the entry of a verified methodological correction into future operation while preserving its provenance, scope, conditions for counterevidence, and a path for later demotion or revocation (Thomas et al., 2015; Popper, 1962; Laroche et al., 2019).

Based on this definition, we introduce the VeriLoop Spiral. Evidence is not information appended to context or supporting material added to a pre-existing conclusion. An observation becomes evidence only when its provenance, scope, and assumptions are explicit and when it exerts a traceable constraint on cognition, action, or inquiry. Otherwise it remains context. The spiral contains six operations: *Evidence*, which makes claims and boundaries explicit; *Falsification*, which turns fallibility into observable failure conditions; *Inquiry*, which gathers evidence around decisive gaps; *Revision*, which applies a constrained change at the level of mismatch; *Verification*, which uses counterexamples and regression checks to rule out accidental success; and *Verified Internalization*, which allows only validated and transferable corrections to alter future methods. Fallibility keeps the loop open, falsifiability directs inquiry, the verification gate grants eligibility for inheritance, and revisability keeps inherited methods exposed to later counterevidence. A system does not improve merely because it can change itself. It crosses the recursive threshold only when evidence begins to change how it will change itself in the future.

Our contribution spans conceptual, mechanistic, and system levels. We redefine the condition under which recursive self-improvement is said to occur, formulate the VeriLoop Spiral, and instantiate it in the open-source VeriLoop Coder-E1[1] software-engineering agent. The system combines a frozen Qwen3.6-27B backbone, four Surface Host PEFT adapters, and a Self-Harness. The adapters provide detachable capabilities for tool specification, uncertainty estimation, evidence binding, and rollback control. The Self-Harness maintains task state, orchestrates tools, constrains evidence, enforces verification gates, and manages failure recovery outside model weights. This separates capability production from runtime control in a complete execution system. On four public software-engineering benchmarks hosted on Hugging Face and their corresponding leaderboards, the system obtains 85.20 on SWE-bench Verified, 62.38 on SWE-bench Pro, 76.40 on Terminal-Bench 2.0, and 33.63 on DeepSWE, as summarized in Figure 1.

[1]Hugging Face model repository.

2 System Principles and Architecture

This section first situates the work in recent research on recursive self-improvement and agent harnesses, then formalizes the VeriLoop Spiral and describes the model-side and runtime components of VeriLoop Coder-E1.

2.1 Related Work

The focus of recursive self-improvement is moving from direct modification of model parameters toward agent runtime mechanisms and the improvement process itself (Weng, 2026; Zhang et al., 2026b; Wang et al., 2026; Lu et al., 2026). Weng argues that near-term recursive self-improvement is more likely to emerge outside the model first. The harness evolves from deployment infrastructure into an optimization target, with optimization progressing from instructions, structured context, and workflow design to harness code and eventually optimizer code, while a mature harness can in turn support automated research and successor-model training (Weng, 2026). The Darwin Gödel Machine modifies agent code and selects successor variants through benchmark evaluation, MetaSkill-Evolve co-evolves task skills and improvement skills on two timescales, and AREX drives an outer research loop through constraint checking and unresolved claims (Zhang et al., 2026b; Wang et al., 2026; Lu et al., 2026). Together, these systems move RSI from revising outputs toward sustained modification of the improvement mechanism itself.

For coding agents, the harness has become a critical runtime foundation rather than a peripheral wrapper (Lin et al., 2026). SWE-agent demonstrates that a model-oriented agent-computer interface can materially change software-engineering behavior by structuring repository navigation, file editing, and test execution (Yang et al., 2024). Reflexion further shows that execution feedback can be transformed into structured reflection and stored in episodic memory, changing subsequent trial strategies without updating model weights (Shinn et al., 2023). From this point, Loop Engineering extends the unit of design beyond a single prompt to a closed execution protocol that includes persistent state, tool transactions, verification receipts,

failure rollback, and termination conditions. Proof-or-Stop similarly treats model outputs as claims awaiting verification. Progression to development, review, testing, or completion is allowed only when evidence bound to the current source state passes freshness, completeness, and verifiability checks. Otherwise the system enters repair, degradation, escalation, or safe stopping (Macedo, 2026; Huang et al., 2026).

2.2 The VeriLoop Spiral

The philosophical starting point of VeriLoop is fallibilism: any system judgment is only provisionally warranted within the evidence and conditions currently available and must remain open to revision or rejection under new evidence. Peirce treated belief as a temporarily stable habit of action, argued that genuine doubt restarts inquiry, and held that scientific judgment must remain answerable to experience independent of the inquirer's preferences (Peirce, 1877, 1955). Popper further argued that knowledge grows not by accumulating supporting material, but by retaining, revising, or eliminating conjectures that remain exposed to severe attempts at refutation (Popper, 1962). Accordingly, VeriLoop does not treat information as evidence merely because it supports an existing conclusion. Evidence must be able to challenge and change the system's current understanding, action, or future inquiry. Information acquires evidential status only when its provenance, scope, and conditions of applicability are explicit and when it imposes a traceable, testable, and revisable constraint on system state. Otherwise it is only an increment to context (Kim et al., 2025).

We operationalize this principle as a runtime control mechanism for a model-harness intelligent system. The resulting VeriLoop Spiral governs how evidence enters judgment, drives correction, and determines whether a correction may be inherited. Figure 2 summarizes the mechanism. It begins by making the task objective, current claim, accepted facts, unknown conditions, and evidential boundary explicit. Each unknown that could alter a decision is registered as an evidence obligation. *Falsification* defines observable failure conditions. *Inquiry* organizes retrieval, execution, and tool calls around decisive gaps. *Revision* restricts changes to the level at which the mismatch occurs. *Verification* tests whether the revision removes the mismatch and examines alternative explanations such as random fluctuation, evaluator defects, or environmental luck. Only corrections with traceable provenance, explicit scope and dependencies, and robustness under relevant variation proceed to *Verified Internalization* as candidates for inheritance. An observation therefore becomes evidence only when it can change a hypothesis, an action, the direction of inquiry, a stopping condition, or method state.

The VeriLoop Spiral is not a fixed workflow composed of six model calls. It is a control architecture that permits rejection, rollback, reopening, downgrading, and revocation. This makes it compatible with a Self-Harness. The model proposes explanations and candidates, while the harness maintains task anchors, system state, evidence obligations, tool transactions, verification gates, and a method registry outside the weights. High-impact transitions that change objectives, action direction, verification conclusions, stopping conditions, or future error-correction behavior are therefore evidence constrained. Passing a verification gate means only that a revision is supported within its declared scope. It does not automatically grant cross-task inheritance. Internalization is confirmed only when a method candidate actually changes how errors are discovered, judged, or corrected on later independent tasks. The Self-Harness therefore does not replace model reasoning. It writes corrigibility into the system control plane and prevents local success from being hardened into a future rule without further testing.

2.2.1 Dual-Gate Delayed Internalization

To prevent local success on the current task from being interpreted directly as methodological improvement, VeriLoop separates *Verified Internalization* into candidate formation and inheritance activation, controlled by a verification gate and an inheritance gate. The verification gate asks whether a revision is supported within its declared scope. The inheritance gate asks whether that revision stably changes error correction on later independent tasks. The resulting spiral is a two-timescale closed loop that separates within-task correction from cross-task inheritance. Let the VeriLoop state at round t be

$$Z_t = (c_t, \sigma_t, H_t, \Omega_t, E_t, K_t, I, b_t), \qquad (1)$$

where c_t is the current claim, σ_t is the evidential boundary and scope of applicability, H_t is the set of competing hypotheses, Ω_t is the set of open evidence obligations, E_t is the evidence acquired so far, $K_t = (G_t, J_t, R_t)$ is the current mechanism

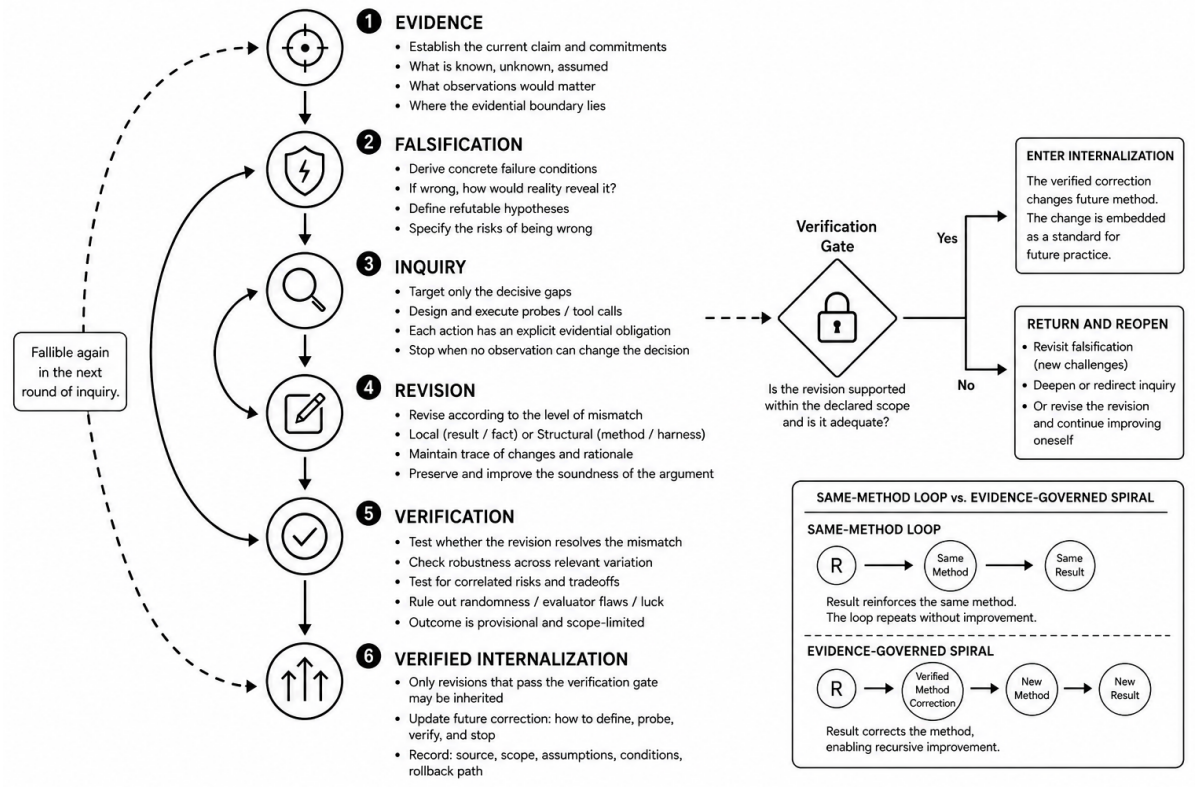


Figure 2: Operational flow and verification gating mechanism of the VeriLoop Spiral. The six stages are not six fixed model calls. They define a revisable control architecture in which claims, evidence obligations, corrections, and inheritance decisions can be rejected, reopened, downgraded, or revoked.

for error discovery, judgment, and correction, I is the set of invariants that must be preserved, and b_t is the remaining inquiry budget.

Each evidence obligation is represented as

$$o = (u, \chi, Y, \tau_o). \tag{2}$$

An obligation is decisive only if at least two possible outcomes imply different effective commitments:

$$\text{Decisive}(o \mid Z_t) \Leftrightarrow |\{\pi_o(y; Z_t) : y \in Y\}| > 1. \tag{3}$$

This excludes formalized evidence gathering in which no possible result could change the decision. A new observation x acquires evidential status only when it satisfies admissibility requirements on provenance, relevance, and scope and actually changes an effective commitment:

$$\text{Evidence}(x \mid o, Z_t) \Leftrightarrow \text{Admissible}(x, o, \sigma_t) \wedge \pi(\tau_o(x; Z_t)) \neq \pi(Z_t). \tag{4}$$

The falsification stage constructs observable failure conditions, and the inquiry stage processes only decisive evidence obligations. If no falsifier can be formed, the claim is marked untestable. If no decisive observation is obtainable, the system returns an unresolved state.

When an observation triggers refutation, the correction level is determined by the location of the mismatch. Let correction levels form a partial order $\preceq$ by scope of impact. The revision selects the lowest sufficient level that can resolve the mismatch while preserving existing invariants:

$$\ell^* = \min_{\preceq} \{\ell : \exists r \in R_t,\ \text{Resolve}(r, E_t, \sigma_t) \wedge \text{Preserve}(r, I)\}. \tag{5}$$

Within that level, the objective is not to minimize textual edit distance. It is to minimize the portion of the revision that is not directly supported by the evidence:

$$r^* = \arg\min_{r \in R_{\ell^*}} \text{UF}(r \mid E_t, \sigma_t), \tag{6}$$

where UF denotes the footprint of the revision that lacks direct evidential causal support.

A revision passes the inheritance gate only if it resolves the identified mismatch, remains stable under relevant variation, excludes major alternative

explanations, keeps regression within the allowed boundary, and preserves complete evidence and revocation relations. In the source formulation, this gate is written as

$$\begin{aligned} G_I(\kappa) \Leftrightarrow\ & G_V(r^*) \wedge n \geq n_{\min} \\ & \wedge \neg\, \mathrm{OpEquivalent}(K^{\kappa}, K_t) \\ & \wedge\, \mathrm{LCB}_{1-\alpha}(\bar{d}_n) > \delta \\ & \wedge\, \mathrm{UCB}_{1-\alpha}(p_n) > p_{\max} \\ & \wedge\, \mathrm{Revisable}(K). \end{aligned} \tag{7}$$

Here, OpEquivalent tests whether the old and new methods are operationally equivalent in error discovery, error judgment, and correction. LCB and UCB denote the lower confidence bound on gain and the upper confidence bound on regression rate, respectively. δ is the minimum effective gain, and $\rho_{\max}$ is the maximum allowable critical-regression rate. Only when the new method changes subsequent error-correction behavior, its gain cannot be explained by random fluctuation, and its risk is controlled does the system execute

$$K_{t+1} = \mathrm{Promote}(K_t, \kappa). \tag{8}$$

Otherwise, the candidate remains provisional or is rejected. Any inherited method that encounters new counterevidence outside its validated scope, a critical regression, or invalidation of its original verification conditions must restore the previous method and reopen the corresponding evidence obligations:

$$\begin{aligned} & \mathrm{CounterEvidence}(\kappa) \Rightarrow K_{t+1} \leftarrow K_t, \\ & \Omega_{t+1} \leftarrow \Omega_{t+1} \cup \\ & \qquad \mathrm{Reopen}(\kappa). \end{aligned} \tag{9}$$

Algorithm 1 unifies within-task correction, verification gating, and delayed cross-task inheritance.

The algorithm operates on claims, falsifiers, evidence obligations, revision levels, verification outcomes, and inheritance eligibility. It does not prescribe model-call patterns, tool protocols, state-storage implementations, or schedulers. The Self-Harness is the execution carrier of this algorithm in the intelligent system, while Surface Host PEFT belongs to the model-side capability and fine-tuning layer.

2.3 Narrow-Domain Parameter-Efficient Fine-Tuning

The operating loop of the VeriLoop Spiral does not require the model to relearn general programming knowledge. It requires the model to provide the intelligent system with control signals that are discriminable, composable, and reversible. On this basis, VeriLoop Coder-E1 uses Qwen3.6-27B as a frozen backbone and confines parameter updates to external Surface Host capability surfaces, separating general code capability from runtime control at the parameter level.

Qwen3.6-27B uses a 27B dense language backbone with 64 layers and a hidden size of 5120, arranged in a fixed hierarchy of Gated DeltaNet and Gated Attention blocks. Its native context length is 262,144 tokens and can be extended to 1,010,000 tokens (Qwen Team, 2026). Public evaluations report 77.2 on SWE-bench Verified, 53.5 on SWE-bench Pro, 71.3 on SWE-bench Multilingual, and 59.3 on Terminal-Bench 2.0, indicating a strong base for repository understanding, long-range dependency modeling, and tool-driven programming.

This backbone was selected because it satisfies four engineering constraints. First, it already has code-generation and repository-level reasoning ability, so small-scale fine-tuning does not need to rebuild basic capability. Second, its long context can accommodate cross-file source code, test receipts, and execution traces. Third, its fixed 5120-dimensional dense hidden representation permits stable host-side mappings without MoE routing changes perturbing the control surface. Fourth, its open weights, Apache 2.0 license, and compatibility with Transformers, vLLM, and SGLang support reproducibility and independent deployment (Qwen Team, 2026). QLoRA has shown that a frozen 4-bit quantized backbone with trainable low-rank adapters can fine-tune a 65B model on a single 48 GB GPU (Dettmers et al., 2023). Applying a comparable quantized PEFT regime to a 27B backbone is therefore feasible on a single GPU from an engineering perspective.

Parameter adaptation is restricted to four capabilities: tool specification, evidence binding, rollback control, and uncertainty calibration. The ToolSpec surface learns tool selection, parameter schemas, preconditions, postconditions, permission boundaries, and verification-receipt requirements. The Evidence Binding surface links claims to sources, execution outcomes, and scopes of applicability. The Rollback surface identifies failure causes, rollback necessity, and the minimum revision boundary. The Uncertainty surface separately estimates uncertainty in answers, evidence, execution, specification, and risk. These surfaces act, respectively,

Algorithm 1 VeriLoop Dual-Gate Delayed Internalization

Require: current claim c, scope σ, correction mechanism K, invariants I, inquiry budget B, future independent task stream T^+

$Z \leftarrow \mathrm{Bound}(c, \sigma, K, I, B)$ ▷ Make claims, boundaries, and unknowns explicit
while $\mathrm{RemainingBudget}(Z) > 0$ **do**
  $F \leftarrow \mathrm{DeriveFalsifiers}(Z)$ ▷ Derive observable failure conditions
  if $F = \varnothing$ **then**
    return $(\bot, K, \textsc{Untestable})$
  end if
  $\Omega \leftarrow \mathrm{BuildDecisiveObligations}(F, Z)$
  if $\Omega = \varnothing$ **then**
    return $(\bot, K, \textsc{Unresolved})$
  end if
  $o \leftarrow \mathrm{Select}(\Omega)$
  $x \leftarrow \mathrm{Observe}(o)$ ▷ Obtain a decisive observation
  if not $\mathrm{Admissible}(x, o, \sigma)$ **then**
    continue
  end if
  if $\Pi(\tau_o(x; Z)) = \Pi(Z)$ **then**
    continue
  end if
  $Z \leftarrow \tau_o(x; Z)$
  if $\mathrm{AllObligationsClosed}(Z)$ **and not** $\mathrm{Refuted}(Z)$ **then**
    return $(\mathrm{CurrentClaim}(Z), K, \textsc{Supported-Within-Scope})$
  end if
  if not $\mathrm{Refuted}(Z)$ **then**
    continue
  end if
  $\ell^* \leftarrow \mathrm{LowestSufficientLayer}(Z, I)$
  $r^* \leftarrow \mathrm{MinUnsupportedFootprintRevision}(\ell^*, Z)$
  $v \leftarrow \mathrm{Verify}(r^*, Z, \sigma, I)$
  if $v = \textsc{Fail}$ **then**
    $Z \leftarrow \mathrm{Reopen}(Z, \mathrm{Diagnose}(v))$
    continue
  end if
  $y \leftarrow \mathrm{Apply}(r^*)$
  if $\ell^* = \textsc{Result}$ **then**
    return $(y, K, \textsc{Supported-Within-Scope})$
  end if
  $\kappa \leftarrow \mathrm{AbstractMethodCandidate}(r^*, E, \sigma, \text{rollback})$
  break
end while
if $\mathrm{RemainingBudget}(Z) = 0$ **then**
  return $(\bot, K, \textsc{Unresolved})$
end if
for task q in T^+ **do**
  $\mathrm{EvaluatePaired}(K, K_\kappa, q)$
  $\mathrm{UpdateEffectAndRegressionBounds}(\kappa)$
  if $\mathrm{HardCounterEvidence}(\kappa)$ **then**
    return $(y, K, \textsc{Rejected})$
  end if
  if $\mathrm{InheritanceGate}(\kappa) = \textsc{Pass}$ **then**
    return $(y, \mathrm{Promote}(K, \kappa), \textsc{Inherited})$
  end if
end for
return $(y, K, \textsc{Provisional})$

on pre-action constraints, post-observation adjudication, post-failure recovery, and end-to-end risk control. Together they form the minimum model-side control set needed by the Self-Harness to execute VeriLoop.

We use narrow-domain PEFT rather than full-parameter fine-tuning primarily to control capability interference. Full-parameter updates, or direct adaptation of backbone attention and feed-forward layers, would allow a small amount of harness-control data to alter program generation, knowledge representation, and instruction following at the same time. This makes performance changes harder to attribute and increases the risk of catastrophic forgetting and cross-task regression. Narrow-domain adaptation instead confines gradients to capability-specific surfaces so that each gain can be trained, evaluated, ablated, and revoked in-

dependently while leaving backbone weights and general code capability unchanged. The objective is not to make the model "know more code." It is to improve control over when to call tools, when to accept evidence, when to roll back, and when to stop.

Training data follow a harness-first, capability-directed mixture. Internal data include real harness trajectories, contract-driven deterministic expansion samples, and samples for tool boundaries and evidence gating. External data include Team-ACE/ToolACE, Salesforce/APIGen-MT-5k, the sanitized configuration of MBPP, and non-Verified slices of SWE-bench. ToolACE and APIGen-MT-5k primarily supplement tool selection and interface constraints. MBPP supplements test assertions, semantic invariants, and failure-repair examples. Non-Verified SWE-bench slices provide repository-level faults, patch verification, and rollback cases. SWE-bench Verified is explicitly excluded from training to preserve its independent evaluation boundary.

Complete source counts are retained for ToolSpec: 202 harness trajectories, 34 boundary-repair samples, 53 ToolACE samples, 19 APIGen-MT-5k samples, and 12 MBPP-sanitized samples, for a total of 320. For the other three tasks, the source families and final sizes can be confirmed, but the historical manifests do not preserve complete per-source counts. We therefore make no numerical inference about their actual mixture proportions. Training and evaluation sets are constructed independently by task, and evaluation examples are dominated by harness-side data so that public external data do not determine the control-capability assessment.

All four fine-tuning runs were completed sequentially on a single NVIDIA RTX PRO 6000 GPU. The backbone was loaded in 4-bit NF4 format with double quantization enabled, BF16 as the quantization compute type, FP16 disabled, and backbone parameters frozen throughout. Optimization used AdamW, cosine learning-rate scheduling, a 0.08 warmup ratio, 0.01 weight decay, and a gradient-clipping threshold of 1.0. This configuration is a 4-bit quantized frozen backbone with BF16 host-side PEFT computation. It is neither 8-bit training nor full-parameter BF16 fine-tuning of the backbone. The narrow-domain LoRA increment is

$$W'_k = W_k + \frac{\alpha}{r} B_k A_k, \tag{10}$$

where k indexes ToolSpec, Evidence Binding, Rollback, or Uncertainty; W_k and W'_k are the host-side linear mapping before and after adaptation; $A_k \in \mathbb{R}^{r \times d_{\text{in}}}$ and $B_k \in \mathbb{R}^{d_{\text{out}} \times r}$ are trainable low-rank matrices; r is the rank; α is the scaling factor; and d_{in} and d_{out} are the input and output dimensions of the mapping.

The sample counts, actual epochs, update steps, and best epochs in Table 2 were recovered from the training run manifests. The learning rates for the first three tasks are primarily sourced from the training scripts and should not be described as independently revalidated from historical runtime logs. The formal run record for the Rollback task did not preserve maximum sequence length, so a current script default should not be substituted for that historical fact.

After training, the four specialized parameter sets are not mathematically merged into the Qwen backbone Safetensors. They are loaded separately into explicit `surface_host.*` modules. Given the last-layer hidden-state sequence H_L, the system uses the attention mask to extract the shared representation at the last valid token,

$$h = H_L[t^*], \tag{11}$$

then applies layer normalization and a capability-specific mapping to produce host-side signals,

$$z_k = g_k\big(\mathrm{LN}(h); \phi_k\big), \quad k \in \{\text{tool}, \text{evidence}, \text{rollback}, \text{uncertainty}\}. \tag{12}$$

Here, H_L is the frozen backbone hidden-state sequence at layer L, t^* is the final valid token determined by the attention mask, h is the shared task representation for the current input, $\mathrm{LN}(\cdot)$ denotes layer normalization, $g_k(\cdot)$ is the Surface Host side mapping for capability k, ϕ_k is its trainable specialization parameter set, and z_k is the narrow-domain control representation consumed by the Self-Harness.

These signals do not replace the backbone output and are not written back into logits or hidden states. They are side-channel control representations used by the Self-Harness for tool gating, evidence adjudication, rollback, reopening, and stopping decisions. The adapters act only on `surface_host.*` modules and do not enter the original attention or MLP projections. Deployment therefore combines a frozen backbone with four detachable adapters rather than merging them into a Qwen checkpoint.

This structure creates a clear division of responsibility. The backbone provides general semantics

Capability surface	Train / Eval	Primary supervision targets	External supplemental data
ToolSpec	240 / 80	Tool choice, schema, parameter constraints, invocation gates, receipt requirements	ToolACE, APIGen-MT-5k, MBPP-sanitized
Evidence Binding	240 / 80	Adjudication state, source sufficiency, follow-up action, execution gate	SWE-bench non-Verified slices, MBPP-sanitized
Rollback	240 / 80	Rollback necessity, failure cause, revision mode, stable-state protection	SWE-bench non-Verified slices, MBPP failure-repair samples
Uncertainty	220 / 80	Answer, evidence, execution, specification, and risk uncertainty	MBPP and tool-ambiguity samples

Table 1: Training data composition and supervision targets of the four Surface Host capability surfaces.

Task	Train / Eval	Actual epochs	Update steps	Best epoch	Max length	Learning rate
ToolSpec	240 / 80	2	60	2	1024	1.5×10^{-4}
Evidence Binding	240 / 80	2	60	2	1024	8×10^{-5}
Rollback	240 / 80	3	90	3	–	8×10^{-5}
Uncertainty	220 / 80	3	84	2	1024	6×10^{-5}

Table 2: Training configurations and actual run records of the four narrow-domain PEFT tasks.

and code reasoning, narrow-domain PEFT generates control signals, and the Self-Harness uses external evidence to decide whether those signals should be accepted or revoked. Each capability surface can be loaded, replaced, ablated, or rolled back independently, and a local update does not rewrite the backbone. If a control decision is later contradicted, the system can revert the corresponding surface without republishing the entire model. The Surface Host Adapter therefore provides minimal intervention, capability isolation, verifiable updates, and reversible deployment.

2.4 Self-Harness-Driven VeriLoop Recursive Control

The preceding formulation specifies when a correction may enter a future mechanism, while narrow-domain PEFT provides signals for tool specification, evidence binding, rollback, and uncertainty. The Self-Harness turns these elements into executable control. Its purpose is not to increase the number of model calls. It separates task boundaries, evidence eligibility, state transitions, verification gates, and method inheritance from model narration, creating an auditable runtime control plane.

The model is responsible only for proposing root-cause hypotheses, action plans, code candidates, and revision candidates. It does not have authority to rewrite the task, admit evidence, promote candidates, or inherit methods. Deterministic components handle contract parsing, structural checks, sandbox execution, receipt generation, and version recording. The Self-Harness then uses external receipts to decide whether to continue, repair, roll back, stop, or register a method candidate. Figure 3 shows this division.

2.4.1 Task-Anchored Contract Compilation

When a task enters the system, a stable anchor is established first:

$$B = (I, O, F, C, V), \tag{13}$$

where I is task identity, O is the objective, F is the set of accepted facts, C is the set of environmental constraints, and V is the evaluation boundary. Working hypotheses and plans may be revised by evidence, but the anchor cannot be silently replaced. If the anchor itself is challenged, the system must generate an explicit revision event that records the rationale, scope of effect, and version difference.

The task acceptance contract then compiles a natural-language request into machine-executable constraints. It records the task family, native artifact type, target files, public interfaces, required

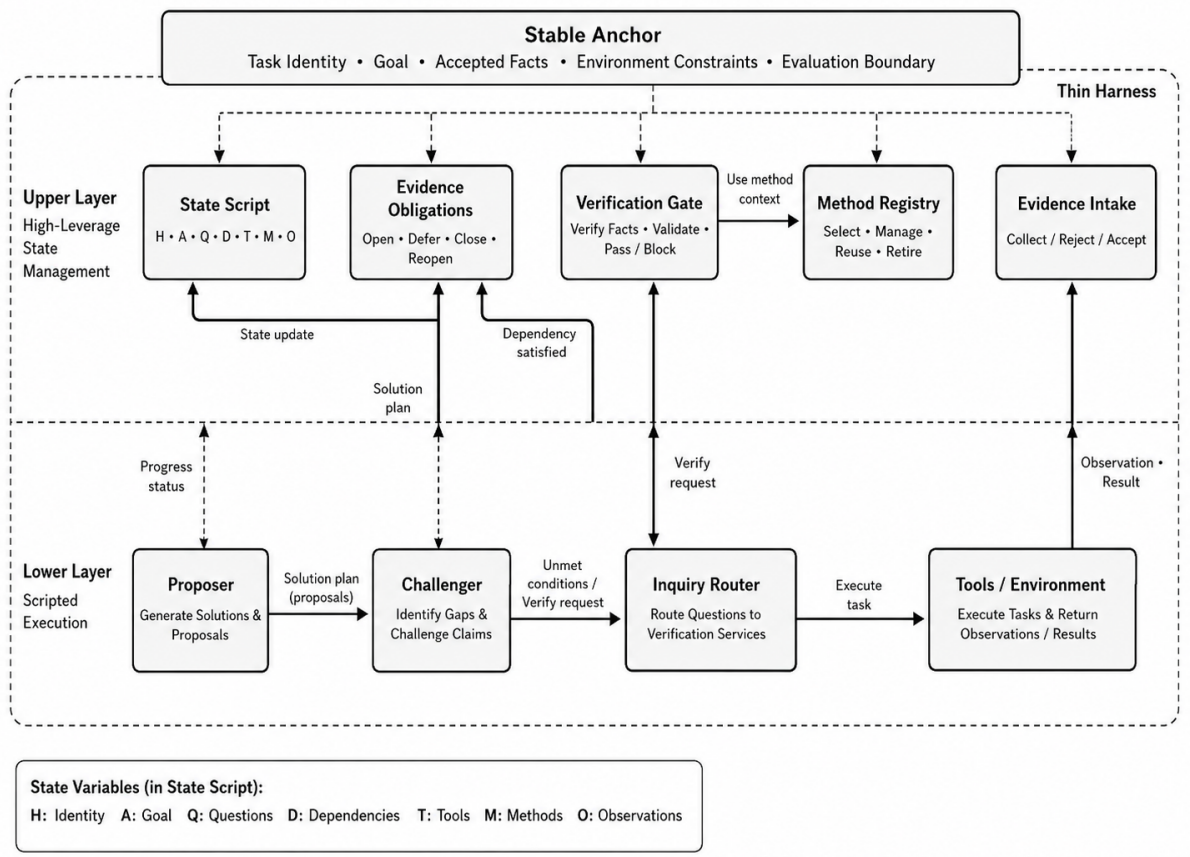


Figure 3: Self-Harness-driven VeriLoop recursive control architecture. A stable task anchor constrains the thin high-leverage control layer, while proposing, challenging, inquiry routing, and tool execution remain separated in the lower execution layer.

behavior, prohibited behavior, verification depth, resource budget, and tool permissions. Repository-level tasks additionally bind the permitted edit scope, compatibility constraints, local conventions, and invariants that must not be broken. Generation, verification, and repair then share the same acceptance criterion.

Inputs for each stage are compiled into independent, immutable Markdown or JSON artifacts that record the stage, round, parent candidate, content digest, and SHA-256. Earlier rounds enter later rounds only through structured references. The raw long conversation is not accumulated directly, preventing historical narration from changing the current task or contaminating candidate generation.

2.4.2 Evidence-Governed State Control

The Self-Harness explicitly maintains hypotheses, actions, inquiry direction, reasoning depth, stopping conditions, method candidates, and evidence obligations. The engineering implementation separates these into a task specification, an uncertainty vector, a budget ledger, evidence blocks, atomic claims, and an append-only audit log. Every state transition can therefore be traced to a concrete observation, scope of applicability, and execution receipt.

An observation x is promoted from context to evidence only when it matches an active evidence obligation o, satisfies provenance and scope constraints, and produces a nonempty bounded update to the constrained state:

$$\begin{aligned} &\mathrm{Evidence}(x \mid S, o) \\ &\quad \Leftrightarrow \mathrm{Admissible}(x, o, S) \\ &\qquad \wedge \mathrm{EffectiveUpdate}(\Delta_o S(x)). \end{aligned} \tag{14}$$

Here, S is the current state. $\mathrm{Admissible}$ means that the observation satisfies provenance, scope, and evidence-obligation constraints. $\mathrm{EffectiveUpdate}$ means that the observation produces a legal, nontrivial, and bounded update to at least one of the hypothesis, action, inquiry direction, stopping condition, method candidate, or evidence-obligation components. Algorithm 2 formalizes evidence admission and its transactional state update.

Algorithm 2 Evidence Admission and Transactional State Update

Require: system state $S = (H, A, Q, D, T, M, \Omega)$, new observation x

$o \leftarrow \text{MatchObligation}(x, \Omega)$
$m \leftarrow$ (provenance, scope, assumptions)
if $\text{HasMatchedObligation}(o) =$ **false then**
  return $(S, \text{CONTEXT})$
end if
if $\text{Admissible}(x, o, S) =$ **false then**
  $\text{AppendAuditRecord}(x, o, S, \text{REJECTED}, m)$
  return $(S, \text{REJECTED})$
end if
$\Delta S \leftarrow \text{DeriveBoundedUpdate}(x, o, S)$
if $\text{EffectiveUpdate}(\Delta S) =$ **false then**
  return $(S, \text{CONTEXT})$
end if
$S_{\text{candidate}} \leftarrow \text{Apply}(S, \Delta S)$
if $\text{ValidateTransition}(S, S_{\text{candidate}}, x, o) =$ **false then**
  $\text{AppendAuditRecord}(x, o, S \rightarrow S_{\text{candidate}}, \text{ROLLED-BACK}, m)$
  return $(S, \text{ROLLED-BACK})$
end if
$S' \leftarrow \text{Commit}(S_{\text{candidate}})$
$\text{AppendAuditRecord}(x, o, S \rightarrow S', \text{EVIDENCE}, m)$
return $(S', \text{EVIDENCE})$

Tool calls are wrapped as evidence transactions. Before a call, the system records the claim to be tested, expected observation, possible outcomes, state effects, provenance scope, and cost. After the call, it retains only normalized receipts, content digests, hashes, and evidence references. Logs, retrieval results, or model narratives that are not bound to a concrete claim or cannot change a decision remain context and cannot advance system state.

Reasoning depth is determined by open evidence obligations and the expected decision value of the next observation rather than by a fixed number of rounds. Inquiry continues only when an unknown can still change the decision, an executable discriminating path exists, and the expected benefit exceeds cost and risk. Otherwise the system enters Resolved, Saturated, or Unresolved, and uncertainty remains explicit in the unresolved state.

2.4.3 Evidence-Driven Closed-Loop Execution

The main execution chain proceeds through problem binding, candidate implementation, machine verification, failure attribution, targeted repair, delivery review, and method distillation. Problem binding produces only testable hypotheses, competing explanations, and evidence obligations. It does not generate final code. Candidate implementation produces only complete code, patches, scripts, or configuration that conform to the native artifact surface.

Each candidate is checked for artifact completeness, syntax and static structure, interface contracts, sandbox behavior, and regression preservation. Verification results are normalized into a set of *Violations*. Each violation records a constraint identifier, severity, failure description, evidence reference, and repair hint. Only tests, logs, exit codes, and artifact digests produced by actual execution can constitute passing evidence.

After a verification failure, the Self-Harness deduplicates violations, orders them causally, and collapses cascading noise into a failure matrix. The next repair round receives only the original task, the immutable contract, the previous complete candidate, open violations, and already validated protective invariants. The model must repair the smallest failed boundary and cannot manufacture a pass by lowering the acceptance standard or performing an unrelated rewrite.

A candidate proceeds to final delivery review only after all hard constraints are satisfied, critical tests pass, and no high-severity item remains unresolved. Candidate scores rank only already-qualified artifacts and cannot replace the acceptance gate. If the budget is exhausted without a qualified result, the system emits a safe-stop record and preserves the candidate, receipts, and rollback points. Figure 4 shows how state transitions, verification rollback, and method-level effects propagate from a current correction to later corrections.

The Thin Evidence Loop controls only high-leverage transitions such as anchor binding, state versioning, evidence obligations, tool transactions, verification gates, stopping conditions, and method-candidate registration. It does not script token-by-token reasoning or low-risk local generation. This keeps control minimally sufficient rather than either underconstrained or over-scripted.

2.4.4 Agentic Search

Agentic Search has three channels: repository evidence construction, external web retrieval, and engineering-memory retrieval. All three follow the same principles of current-task supremacy, evidence-obligation triggering, bounded budgets, and provenance binding. Their outputs are normalized into a common evidence-atom representation. Retrieved material enters the main execution chain only after checks for relevance, credibility, conflict, and scope of applicability.

Repository evidence construction begins with

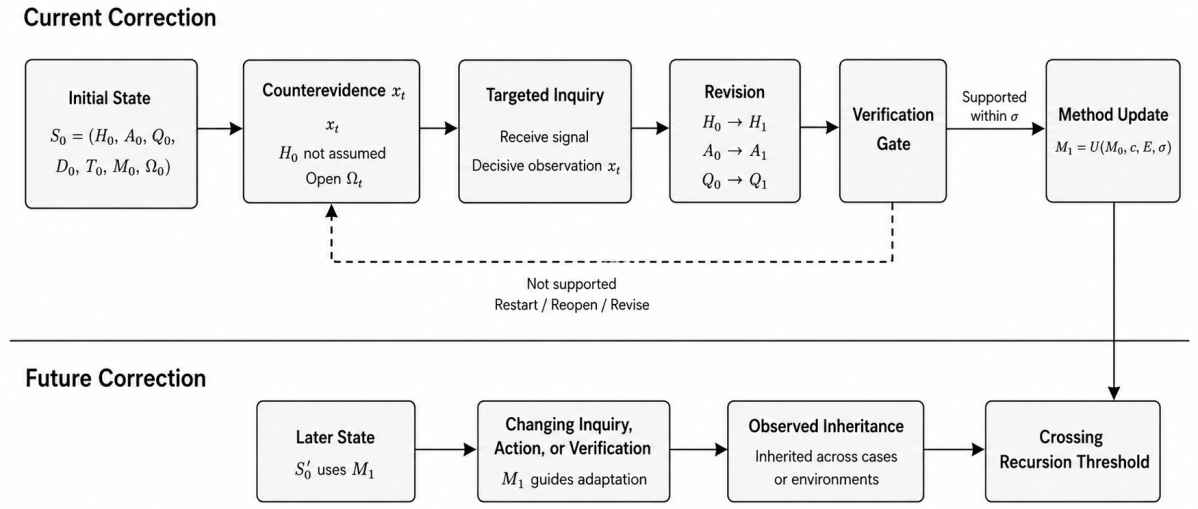


Figure 4: VeriLoop state transitions and cross-round inheritance of method-level effects. A verified current correction updates the method only within its supported scope, and recursive improvement is established only when the updated method changes a later independent correction.

deterministic reading of repository rules, configuration, and constraints. It then resolves entry points, callers, public interfaces, tests, build procedures, and local conventions to produce a repository contract and a generation blueprint. Retrieval scope is determined by the active hypotheses and evidence obligations. Source fragments that are not consumed by a downstream decision do not enter the generation context, preventing whole-repository context accumulation from substituting for code understanding.

External web retrieval uses a controlled chain of necessity assessment, narrow tool routing, and provenance-bound projection. It is triggered only for time-sensitive information, external APIs, dependency versions, security advisories, or cases in which local evidence is insufficient. For code tasks, the search is restricted to a small set of high-precision source families, with preference given to official documentation, source repositories, package indexes, and vulnerability databases.

Each external result records the tool type, source URL, source domain, title, summary, retrieval time, credibility level, and task-relevant metadata. The retriever does not copy a full web page directly into the prompt. It projects the result into citable evidence atoms. When sources conflict, content is stale, access is restricted, or verification fails, the system preserves an explicit risk flag and prevents the result from becoming the sole basis for a decision.

Engineering memory forms the internal Agentic Search channel. Before problem binding and before candidate generation, the system performs bounded recall by account, project, task family, and current error characteristics. It removes identity, filenames, prompts, and answer-like content from earlier tasks. Only abstract engineering rules, failure-prevention patterns, verification recipes, and reusable tool strategies may enter the current context.

2.4.5 Engineering-Memory-Driven Recursive Inheritance

After a task finishes, the Self-Harness distills method candidates from the failure, counterevidence, revision, and verification trajectory rather than storing the raw conversation or complete logs. A candidate records at least the trigger condition, refuted assumption, preventive rule, supporting evidence, scope of applicability, failure signal, and rollback reference. It enters engineering memory in a provisional state. Figure 5 illustrates how a local verified correction becomes a scoped method rule, is recalled in a later round, and crosses the recursive threshold only when a method-level effect is observed.

Memory admission applies provenance verification, deduplication, contradiction detection, scope constraints, and identity-contamination checks in sequence. A single success on the current task can create only a provisional candidate. Promotion to a reusable rule occurs only when the method changes evidence obligations, edit location, tool choice, or verification behavior in a later independent task and then passes acceptance again.

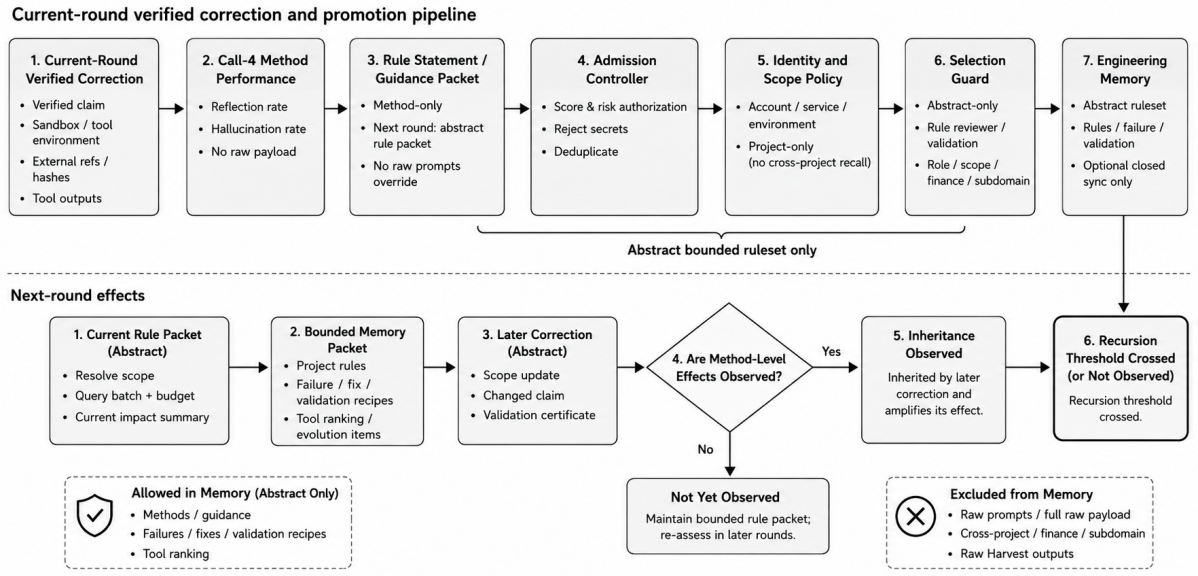


Figure 5: Recursive threshold and inheritance scope. A verified current-round correction is abstracted, admitted under identity and scope controls, stored as bounded engineering memory, and assessed for method-level effects in a later correction. Recursive inheritance is claimed only when such effects are observed within the approved scope.

Promoted methods retain versions and revocation paths. New counterevidence may narrow their scope, reduce their priority, restore an earlier version, or reopen evidence obligations. Only methods with stable cross-task benefit and clear boundaries may later enter harness rules or subsequent PEFT data. They are never written directly back into the frozen backbone.

2.4.6 The 14 Golden Rules

The Self-Harness of VeriLoop Coder-E1 exposes 14 model-visible disciplines that constrain task understanding, evidence use, artifact generation, verification and repair, and final delivery. These rules do not replace task-specific contracts. They define a common minimum engineering boundary across task families.

1. The current request takes precedence over historical memory, old templates, cached rules, and habits from previous tasks.
2. Without concrete evidence or an explicit contract gap, do not escalate to research, reverse engineering, retrieval, sandbox execution, or repair workflows.
3. Before writing or modifying code, inspect entry points, callers, contracts, task requirements, relevant evidence, and failure signals.
4. Produce the smallest complete artifact that satisfies the user's intent and can be verified.
5. Repair only the violated invariant. Rewrite the whole artifact only when evidence shows that a local repair is insufficient.
6. Syntax and import checks are necessary but not decisive. The decisive criterion is whether the artifact satisfies the task intent.
7. Record failures, skips, degradations, and unknown states explicitly. Never fabricate a successful execution that did not occur.
8. Whenever possible, implement deterministic logic for static checks, validation, self-tests, scoring, and artifact parsing in typed code.
9. Treat the current `max_new_tokens` limit as the only visible output budget, and allocate tokens only to the current request and artifact contract.
10. Every tool call, retrieval action, or memory recall must have triggering evidence, an expected output, and an explicit downstream consumer.
11. Long-running or multistage tasks must persist candidate artifacts, verification reports, self-check results, selection reports, and progress state.
12. Preserve existing filenames, APIs, paths, artifact surfaces, repository style, task-family invariants, and benchmark constraints.
13. Deliver only one selected final artifact while retaining the complete evidence package that justifies it.

14. Load domain-specific or benchmark-specific overlays only when the current task requires them, and never allow an overlay to override the current request.

Together, these disciplines define the Self-Harness priority order, evidence conditions, modification boundaries, verification standards, resource governance, and delivery discipline. They do not prescribe a concrete solution to every task. Their purpose is to ensure that heterogeneous task families execute within the same verifiable, auditable, and reversible engineering boundary.

3 Results

We report empirical results at both system and component levels. The four benchmarks evaluate end-to-end performance of the complete intelligent system. Narrow-domain PEFT experiments test model-side capability adaptation, while Self-Harness experiments test execution-side mechanism integrity.

The model weights, Surface Host Adapters, and evaluation evidence for VeriLoop Coder-E1 are publicly released in its Hugging Face model repository. The task resolution rate is 85.20% on SWE-bench Verified and 62.38% on SWE-bench Pro. The pass rate is 76.40% on Terminal-Bench 2.0, and the task resolution rate is 33.63% on DeepSWE. These results show strong performance on repository-level repair and terminal-execution tasks, with a clear decline on the multi-stage autonomous repair setting represented by DeepSWE. Because the benchmarks differ in task distributions and evaluation protocols, these scores characterize performance within their respective task families and should not be interpreted as directly comparable across benchmarks.

3.1 Narrow-Domain Capability Gains and Parameter Efficiency of Surface Host PEFT

Surface Host PEFT consists of four narrow-domain adapters: ToolSpec, Evidence Binding, Rollback, and Uncertainty. The first three use before-versus-after fine-tuning comparisons. The Uncertainty adapter uses held-out predictive evaluation. Each primary endpoint uses $n = 80$ examples.

All four primary endpoints in Table 3 remain statistically supported after correction for multiple comparisons. Evidence Binding and Rollback improve in accuracy by 0.5500 and 0.5125, reaching 0.8250 and 0.8625 after fine-tuning. These results indicate stable gains in evidence adjudication and rollback decisions. ToolSpec improves by 0.3375 but reaches only 0.4375 final accuracy, making tool selection the clearest weakness among the four capabilities. For Uncertainty, the RMSE skill score is 0.1500 and the explained variance for answer uncertainty is 0.2775. This indicates useful but limited held-out predictive capability and does not constitute a before-versus-after causal effect of fine-tuning.

The Uncertainty adapter performs five-dimensional continuous regression, so Macro-F1 and MCC are not applicable. Its mean MAE and weighted MAE are 0.139382 and 0.140354, respectively. Its mean RMSE and weighted RMSE are 0.178999 and 0.180485. Under the prespecified error tolerances, medium-tolerance accuracy is 0.7575 and strict-tolerance accuracy is 0.4600. The RMSE for answer uncertainty is 0.147648. The adapter therefore produces stable numerical uncertainty estimates, but substantial room for improvement remains under strict error constraints.

As shown in Table 4, the four adapters contain 3,481,640 LoRA-matrix parameters in total, corresponding to only 0.012895% of the 27B backbone. Their combined artifact size is 13.059 MiB. Together with Table 3, this shows that the capability gains are obtained with a very small parameter increment rather than by expanding backbone capacity. This parameter count covers only the LoRA matrices and excludes other trainable parameters such as task-specific prediction heads.

3.2 Self-Harness Mechanism Tests

The Self-Harness experiments test whether method rules can enter later rounds through admission control, rule-package generation, and boundary constraints. The four predeclared endpoints correspond to admission coupling, recursive-package closure, non-degenerate gating, and evolution-guard integrity.

All four endpoints remain statistically supported after Holm correction. M2 and M3 show stable coupling between rule admission and the next-round rule package, and every admitted rule completes recursive-package generation. M4 has an admission rate of 19.13%, excluding the two degenerate states in which all candidates are accepted or all are rejected. M5 observes no evolution-guard violation. These results show that the cross-round inheritance path is selective, closed, and boundary consistent.

Adapter	Primary endpoint	n	Before	After	Δ / R^2	p	q_{BH}
ToolSpec	Tool-selection accuracy	80	0.1000	0.4375	$\Delta = 0.3375$	4.19×10^{-5}	4.19×10^{-5}
Evidence Binding	Evidence-judgment accuracy	80	0.2750	0.8250	$\Delta = 0.5500$	1.65×10^{-7}	3.35×10^{-7}
Rollback	Rollback-need accuracy	80	0.3500	0.8625	$\Delta = 0.5125$	1.67×10^{-7}	3.35×10^{-7}
Uncertainty	Answer-uncertainty RMSE	80	0.1737	0.1476	$R^2 = 0.2775$	1.30×10^{-6}	1.73×10^{-6}

Diagnostic metrics

Adapter	Metric family	Macro-F1	MCC	RMSE skill
ToolSpec	Classification	0.4373	0.3775	–
Evidence Binding	Classification	0.7845	0.7614	–
Rollback	Classification	0.7530	0.6890	–
Uncertainty	Regression	–	–	0.1500

Table 3: Primary capability endpoints of Surface Host PEFT. The upper panel reports the primary effects and corrected significance levels; the lower panel reports task-appropriate diagnostic metrics.

Adapter	r	α	Target modules	LoRA pairs	LoRA parameters	Backbone share	Size (MiB)
ToolSpec	8	16	11	11	901,120	0.003337%	3.441
Evidence Binding	8	16	13	13	1,064,960	0.003944%	4.066
Rollback	8	16	17	17	1,392,640	0.005158%	5.317
Uncertainty	8	16	2	2	122,920	0.000455%	0.235
Total	–	–	43	43	3,481,640	0.012895%	13.059

Table 4: Parameter efficiency of Surface Host PEFT.

They are mechanism-level evidence and do not, by themselves, establish a causal effect of cross-round inheritance on downstream task pass rates.

The results therefore form a layered evidence chain. The four benchmarks measure end-to-end performance of the complete system. PEFT experiments quantify narrow-domain model-side capabilities and their parameter cost. The Self-Harness experiments test whether rule admission, cross-round transmission, and boundary control operate according to the intended contract. Together, these results connect local capability, mechanism constraints, and system performance while also identifying tool selection and complex multi-stage repair as current performance boundaries.

4 Future Work

A more fundamental research question is the coupling between backbone capability and harness complexity. VeriLoop Coder-E1 uses a frozen 27B backbone, and several components of the Self-Harness, including task decomposition, context maintenance, tool routing, failure attribution, and repeated verification, partly compensate externally for limitations in long-horizon autonomy of the backbone. The present harness scale is therefore an engineering configuration for a particular stage of model capability, not a fixed form required by evidence-governed control.

As general models improve in code understanding, long-horizon planning, tool use, and self-verification, the marginal value of compensatory harness logic should decline. Frontier systems such as Kimi K3 have expanded to 2.8 trillion parameters and are designed for long-horizon coding and agentic tasks (Kimi Team, 2026), suggesting that capabilities once implemented through external orchestration are increasingly moving into the model. Under this trajectory, a complex harness should not be treated as the terminal form of a frontier

ID	Mechanism endpoint	Analysis unit	Observed	Null / threshold	Positive margin
M2	Admission–rule-package coupling	episode	$\phi = 1.0000$	$\phi \geq 0.50$	0.5000
M3	Recursive-package closure rate	admitted episode	22/22	Rate> 0.80	0.2000
M4	Admission-gate non-degeneracy	episode	22/115	$[0.05, 0.95]$	0.1413
M5	Evolution-guard integrity rate	audited episode	115/115	Rate> 0.95	0.0500

Significance and effect size

ID	p	p_{Holm}	Effect size
M2	8.93×10^{-16}	4.46×10^{-15}	MCC$= 1.0000$
M3	0.00738	0.01476	ΔRate$= 0.2000$
M4	5.76×10^{-8}	2.30×10^{-7}	Interior margin$= 0.1413$
M5	0.00274	0.00823	ΔRate$= 0.0500$

Table 5: Mechanism integrity and cross-round inheritance-path tests of the Self-Harness. The upper panel reports the predeclared mechanism endpoints; the lower panel reports corrected significance and effect sizes.

Code Agent. Loop Engineering should reduce control branches, specialized rules, and code volume as model capability improves, moving from dense compensation toward minimally sufficient control.

This simplification does not imply that all external control should disappear. Compensatory modules for task planning, local routing, and routine self-checking can gradually be internalized or removed. Deterministic verification, permission isolation, state persistence, evidence traceability, and safe rollback should remain outside the model. The likely direction is therefore not the elimination of the harness, but its contraction from a complex capability-compensation layer into a lightweight governance control plane.

The same trend is likely to reduce the centrality of small vertical models in frontier software-engineering agents. General-purpose large models can cover more programming languages, repository types, tool interfaces, and engineering stages within a shared capability space, reducing the need to maintain a specialized model, adapter stack, and control chain for each narrow domain. Small models remain useful for low-cost deployment, local execution, and privacy-sensitive settings. However, the main research object is likely to shift toward model-agnostic general Code Agents in which different backbones can be exchanged as reasoning resources behind a unified evidence-governed execution framework.

Future work will quantify the relationship between model capability and harness complexity. On a common task set, we plan to compare full, pruned, and minimally controlled architectures and estimate the marginal contribution of each module to task success, regression risk, tool overhead, token consumption, and execution latency. The goal is to reduce loop depth and engineering code as model capability grows, allowing VeriLoop to evolve from a compensatory system for a specific model into an adaptive lightweight control architecture for general Code Agents.

5 Conclusion

We addressed the difficulty of distinguishing system modification from genuine improvement in a Self-Harness by shifting the criterion for recursive self-improvement from "what the system changed" to "whether a correction changed future error correction." On this basis, we introduced the VeriLoop Spiral and its dual-gate delayed-internalization mechanism. The six operations of Evidence, Falsification, Inquiry, Revision, Verification, and Verified Internalization organize evidence obligations, bounded revision, verification gating, and method inheritance. A correction acquires recursive significance only when it passes testing within an explicit scope, enters later operation, and retains a path for revocation.

VeriLoop Coder-E1 implements these principles

as a software-engineering intelligent system that separates a model capability layer from a runtime control layer. A frozen Qwen3.6-27B backbone provides general code understanding and generation. Four Surface Host PEFT adapters provide ToolSpec, Evidence Binding, Rollback, and Uncertainty signals. The Self-Harness performs task anchoring, tool transactions, state maintenance, verification rollback, and engineering-memory inheritance outside the model weights. This separates capability generation, evidence adjudication, and method-promotion authority.

Empirically, VeriLoop Coder-E1 obtains task-resolution rates of 85.20% on SWE-bench Verified and 62.38% on SWE-bench Pro, a pass rate of 76.40% on Terminal-Bench 2.0, and a task-resolution rate of 33.63% on DeepSWE. All three classification adapters show statistically supported before-versus-after gains, while the Uncertainty adapter shows statistically supported predictive association on a held-out set. The four adapters together use LoRA matrices equal to only 0.012895% of the 27B backbone parameters. All four predeclared Self-Harness mechanism endpoints remain supported after Holm correction, indicating that rule admission, recursive-package generation, non-degenerate gating, and the evolution guard operate according to their intended contracts. These mechanism results should not be interpreted as a causal demonstration that cross-round inheritance improves downstream task performance.

The results support three connected conclusions. Narrow-domain control capabilities can be acquired through low-parameter side paths without rewriting the backbone. Method-level corrections can be governed by an external control plane with verification, scope, and revocation constraints. Finally, coordination between the model and the Self-Harness can produce measurable software-engineering task performance. Future work will further quantify the relationship between backbone capability and harness complexity, prune compensatory control as model capability increases, retain minimally sufficient verification, permission, state, and rollback mechanisms, and advance VeriLoop toward a model-replaceable general Code Agent.

Limitations

The present experiments are constrained by available compute, the cost of official benchmark calls, and the execution time of long agent trajectories. Evaluation currently covers four software-engineering benchmarks, four narrow-domain PEFT endpoints, and Self-Harness mechanism tests. These constraints do not invalidate the reported results within their stated tasks and statistical definitions, but they limit the scale of validation across programming languages, repositories, temporally shifted distributions, and long-term software-maintenance settings. Additional task-level repetitions and cross-environment evaluations are required to establish the stability and transfer boundaries of method inheritance.

A second limitation is the coupling between backbone capability and harness complexity. Because VeriLoop Coder-E1 uses a frozen 27B backbone, parts of the current Self-Harness compensate externally for limitations in long-horizon task decomposition, context maintenance, tool routing, failure attribution, and repeated verification. The current harness scale should therefore be interpreted as an engineering configuration for this model-capability regime rather than as a necessary fixed form of evidence-governed control.

Acknowledgments

First, I thank the DeepSeek recruiting team for an unpleasant experience they once gave me. The feeling of being looked down upon and dismissed genuinely angered me, and it became an important source of motivation for my practical research on large language models. Throughout the development process, challenging DeepSeek and defending my own honor and dignity remained among the forces that kept me moving forward. People may differ in aptitude and ability, but human dignity should not be ranked. Precisely because people are born equal, injustice tests the conscience and drives more people to pursue fairness and justice. DeepSeek is unquestionably a highly creative large-model company, but no technical achievement can make contempt for others acceptable. Resisting unfairness and arrogance was an inevitable choice for me. Surpassing DeepSeek will remain one of my long-term goals. At the same time, I respect this competitor and hope it moves forward in its strongest form, so that on the road toward AGI we can continue to chase one another and test one another.

I sincerely thank Professor Houde Liu. He never looked down on me because of my sharp words, nor because my educational background was in

business administration at a Southeast Asian university. Instead, he consistently gave me trust and encouragement without prejudice toward convention. Whenever the research reached a new milestone, he was the first person with whom I wanted to share it. That sustained and unwavering support was an important reason I was able to complete this entire development journey. A nontraditional background does not mean that a person cannot enter frontier research, and it does not mean that one must accept ability boundaries defined in advance by others. I have always believed that I am one of the best artificial intelligence architects, and Professor Liu's trust allowed that belief to be tested repeatedly in real work.

Finally, I thank everyone who has denied me, criticized me, or mocked me. You may never understand the intensity of the happiness that comes from giving everything you have to one thing and finally seeing it bear fruit. To have such an experience even once in a lifetime is enough. I do not want to offer polished ceremonial words. If a person has never been willing to give the greatest effort of a lifetime to something, it is difficult to truly understand this kind of happiness.

There is much unfairness in life, and none of it is inherently reasonable. When you are lost, mocked, humiliated, or reduced by others to worthlessness, if you come to believe that these are simply things one is supposed to accept while growing up, then congratulations, you are learning to accept bullying every day. But if fate takes you by the throat, then do what I did. Fight, fight, fight. Never, never, never, never give up, until you win back the dignity that belongs to you and experience the exhilaration known only to those who have changed their own fate with their own hands.

Look at me. I endured ridicule and attacks for a long time, and there are probably few artificial intelligence researchers as unconventional as I am. But a person cannot lose heart because of that, and cannot allow critics and doubters to block the road toward a dream. I believe this is an important reason why VeriLoop Coder-E1 was able to briefly surpass DeepSeek V4 Preview. There has never been any savior, nor can we rely on gods or emperors. If we are to create human happiness, we must rely on ourselves. Do not submit, do not retreat, and do not stop doing what you believe is right. Anything worth doing will inevitably pass through hardship, and it is from those hardships that we gain joy.

References

Anthropic. 2026. Claude Code by Anthropic: AI coding agent, terminal, and IDE. https://claude.com/product/claude-code.

Stuart Armstrong, Jan Leike, Laurent Orseau, and Shane Legg. 2020. Pitfalls of learning a reward function online. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 1592–1600. doi:10.24963/ijcai.2020/221.

Mark Bolin. 2026. Unrolling the Codex agent loop. OpenAI, January 23, 2026. https://openai.com/index/unrolling-the-codex-agent-loop/.

Mingguang Chen, Licheng Wang, and Bo Qu. 2026. Recursive self-improvement in AI: From bounded self-refinement to autonomous research loops. arXiv:2607.07663. doi:10.48550/arXiv.2607.07663.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems 36*, pages 10088–10115.

Leo Gao, John Schulman, and Jacob Hilton. 2023. Scaling laws for reward model overoptimization. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pages 10835–10866. PMLR.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=IkmD3fKBPQ.

Jek Huang, Jeffery Hsia, Jiayi Sun, Freddie Shi, Wei Huang, and Ian H. White. 2026. Proof-or-Stop: Don't trust the agent, trust the evidence—loop engineering for verifiable evidence-gated lifecycle control. arXiv:2607.14890. doi:10.48550/arXiv.2607.14890.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, and Ofir Press. 2024. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*. https://openreview.net/forum?id=VTF8yNQM66.

Minsu Kim, Sangryul Kim, and James Thorne. 2025. From evidence to belief: A Bayesian epistemology approach to language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 10578–10611. Association for Computational Linguistics. doi:10.18653/v1/2025.naacl-long.531.

Kimi Team. 2026. Kimi K3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*. https://arxiv.org/abs/2607.24653.

Lauro Langosco Di Langosco, Jack Koch, Lee D. Sharkey, Jacob Pfau, and David Krueger. 2022. Goal misgeneralization in deep reinforcement learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 12004–12019. PMLR.

Romain Laroche, Paul Trichelair, and Rémi Tachet des Combes. 2019. Safe policy improvement with baseline bootstrapping. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 3652–3661. PMLR.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. 2026. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. arXiv:2604.25850. doi:10.48550/arXiv.2604.25850.

Ryan Lopopolo. 2026. Harness engineering: Leveraging Codex in an agent-first world. OpenAI, February 11, 2026. https://openai.com/index/harness-engineering/.

Shuqi Lu, Chaofan Li, Kun Luo, Zhang Zhang, Hui Wang, Hongwang Xiao, Zheng Liu, Lei Xiong, Jiahao Wang, Sen Wang, Xiyan Jiang, Wanli Li, Yuyang Hu, Hongjin Qian, Bingyu Yan, Ziyi Xia, Yingxia Shao, Kang Liu, Zhicheng Dou, Di He, Chaozhuo Li, Qiwei Ye, Zhongyuan Wang, and Zheng Liu. 2026. AREX: Towards a recursively self-improving agent for deep research. arXiv:2607.21461. doi:10.48550/arXiv.2607.21461.

Sandeco Macedo. 2026. Stop hand-holding your coding agent: Engineering the loops that replace step-by-step prompting. arXiv:2607.00038. doi:10.48550/arXiv.2607.00038.

Moonshot AI. 2026. Using Kimi API Platform in Kimi Code CLI. https://platform.moonshot.cn/docs/guide/kimi-cli-support.

Arjun Panickssery, Samuel R. Bowman, and Shi Feng. 2024. LLM evaluators recognize and favor their own generations. In *Advances in Neural Information Processing Systems 37*, pages 68772–68802. doi:10.52202/079017-2197.

Charles S. Peirce. 1877. The fixation of belief. *Popular Science Monthly*, 12:1–15.

Charles S. Peirce. 1955. The scientific attitude and fallibilism. In Justus Buchler, editor, *Philosophical Writings of Peirce*, pages 42–59. Dover Publications, New York.

Karl R. Popper. 1962. *Conjectures and Refutations: The Growth of Scientific Knowledge*. Basic Books, New York.

Qwen Team. 2026. Qwen3.6-27B. https://huggingface.co/Qwen/Qwen3.6-27B.

Jürgen Schmidhuber. 2007. Gödel machines: Fully self-referential optimal universal self-improvers. In Ben Goertzel and Cassio Pennachin, editors, *Artificial General Intelligence*, pages 199–226. Springer, Berlin. doi:10.1007/978-3-540-68677-4_7.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36*. doi:10.52202/075280-0377.

SpaceXAI. 2026. Introducing Grok 4.5. July 16, 2026. https://x.ai/news/grok-4-5.

SWE-bench. 2026. SWE-bench leaderboards. https://www.swebench.com/.

Philip S. Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. 2015. High confidence policy improvement. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 2380–2388. PMLR.

Zefeng Wang, Minxi Yan, Jinhe Bi, Sikuan Yan, Volker Tresp, and Yunpu Ma. 2026. MetaSkill-Evolve: Recursive self-improvement of LLM agents via two-timescale meta-skill evolution. arXiv:2607.05297. doi:10.48550/arXiv.2607.05297.

Lilian Weng. 2026. Harness engineering for self-improvement. July 4, 2026. https://lilianweng.github.io/posts/2026-07-04-harness/.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems 37*, pages 50528–50652. doi:10.52202/079017-1601.

Z.AI. 2026. Claude Code: Overview. https://docs.z.ai/devpack/tool/claude.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. 2026a. Self-Harness: Harnesses that improve themselves. arXiv:2606.09498. doi:10.48550/arXiv.2606.09498.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. 2026b. Darwin Gödel Machine: Open-ended evolution of self-improving agents. In *The Fourteenth International Conference on Learning Representations*. https://openreview.net/forum?id=pUpzQZTvGY.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. 2026c. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*. https://openreview.net/forum?id=eC4ygDs02R.