

Frontier and Localhost: How Production AI Learns Outside the Weights

Anonymous authors
Paper under double-blind review

Abstract

1 Production LLM systems increasingly adapt outside the model weights.
2 After deployment failures, teams modify prompts, rules, memories, skills,
3 tools, eval suites, routing graphs, and governance pipelines on a cadence
4 that frontier weight updates cannot match. But this scaffold layer is still
5 mostly maintained as patchwork: fixes are proposed by intuition, committed
6 with weak credit assignment, accumulated without pruning, and promoted
7 beyond the scope where they were validated. This paper formalises
8 a disciplined alternative, which we call artifact-layer descent. The pipeline
9 is concrete. Recurring residuals are assigned to scaffold coordinates. Candidate
10 artifact deltas are tested against patch loss. Accepted deltas persist
11 with rollback. Promotion across contexts is bounded by evidence radius: a
12 local fix spreads only as far as the evidence supports. Surveying approximately
13 130 production and research systems from 2023 to 2026, we find
14 current systems partially instantiate this loop but leave a central architecture
15 gap, namely governed scaffold optimization across contexts. The
16 contribution is not the claim that prompts are weights. It is to name the
17 missing optimizer for an adaptation layer the field has already built.

18 1. The gradient moved outside the model

19 The standard mental model of an LLM system is the model: pre-training fixes the weights,
20 fine-tuning nudges them, and deployment wraps a thin prompt-and-tool layer around the
21 result and ships. Production in 2025 and 2026 has stopped behaving this way. Frontier
22 models update on cycles measured in months; the systems built on top of them update
23 continuously, in markdown instruction files, persistent memories, retrieved skills, registered
24 tools, evaluation suites, agent topologies, version-controlled prompt repositories, and PR-
25 gated governance. A growing share of deployment-time adaptation now occurs outside the
26 weights, in a fast-changing local scaffold wrapped around a slow-changing frontier. The
27 section title is a metaphor: the scaffold layer is not a gradient in the calculus sense. The
28 substantive claim is the milder one — production AI has accidentally created an external
29 learning surface, and right now it is being operated as patchwork rather than as a disciplined
30 optimization process.

31 Read together, these artifacts form a single external adaptation surface that current practice
32 still treats as scattered engineering. A Claude Skill is a learned procedural weight kept
33 outside the model. A Cursor Rule is a local parameter for one project’s distribution shift. A
34 RAG connector is patch-specific memory access. A tool registry is capability provisioning
35 along an axis the base model cannot reach. A PR review against a shared rule repository is
36 a promotion gate on a federated update. An eval suite that blocks merge is a validation-loss
37 check before promotion. None of these mechanisms is new; what is new is reading them as
38 one adaptation surface and asking what optimization discipline that surface needs. This paper
39 surveys roughly 130 production and research systems, identifies the six substrates where
40 the scaffold lives, formalises a two-loop architecture (local update; cross-context promotion
41 under a gate), and names a missing architecture as its most consequential open problem:
42 governed cross-tenant scaffold optimization with versioned lineage, role-based access, and
43 rollback. The single-line thesis is this: the field has built an external adaptation layer for
44 LLMs, but not the optimizer that should govern it.

2. Why patch errors live outside the weights

The scaffold is not merely where production systems happen to store patches. Under deployment-time constraints — release cadence, tenant isolation, inspectability, rollback — it is often the practical substrate, because it is cheap, locally scoped, reversible, and updatable at human time-scales while the weights are not.

Frontier training optimises for breadth. The weights are tuned for cross-patch generalisation, smoothing over local conventions, contradictory tenant preferences, and rare workflow-specific failures to preserve broad competence across millions of unseen deployments. RLHF reduces output diversity and homogenises preferred styles across users (Kirk et al. 2024; Padmakumar and He 2023), so the frontier model is trained to be unable to encode a single tenant’s idiosyncrasies as preferred behaviour. Tiwari et al. (2026) supply the matching two-timescale argument from the in-context / in-weights side: in-context adaptation is cheap and rapid but capacity-bounded, in-weights adaptation is expressive but induces catastrophic forgetting and plasticity loss. Production reliability, by contrast, is achieved in depth: inside one deployment patch, reliability is determined by exactly what global training had to smooth away. Encoding every patch in the weights is ruled out on economic, release-cycle, and cross-patch-interference grounds at once. The frontier model generalises; the local scaffold specialises; reliability comes from governing that specialisation.

Prior work on error accumulation (Arbuzov et al. 2025, 2026) makes the specialisation tractable. It argues that LLM errors concentrate sparsely at a few key tokens and cluster into a finite catalogue of recurring failure modes whose size grows only slowly in the number of observed failures. A patch’s residual is therefore not diffuse noise but a small set of repeating modes, each attachable to a scaffold coordinate — which is what makes a disciplined local optimizer possible at all, and what tells that optimizer where to spend its budget.

Three stages of scaffold maintenance recur across the corpus. They differ in how candidate updates are produced, whether credit assignment is explicit, and whether promotion is evidence-bounded. Most production deployments sit at Stage 1 or 2; the formalism of §3 applies to Stage 3.

Stage	Update mechanism	Credit assignment	Promotion discipline	Dominant failure mode
1. Patchwork	Intuition-driven local edits	Implicit / ad hoc	None	Bloat
2. Random artifact search	Many variants tried under eval	Weak; eval is the only signal	Eval-gated CI, no failure-to-coordinate tracing	Overfitting to in-distribution patch
3. Artifact-layer descent	Residual $\rightarrow$ coordinate $\rightarrow$ delta	Explicit per failure mode	Evidence-radius-bounded; reversible	(the discipline this paper formalises)

A patchwork loop becomes artifact-layer descent once five pieces are in place: a measurable patch loss, captured residuals, a failure traced back to the scaffold coordinate that caused it, a synthesised candidate edit, and a gate that checks whether the edit actually lowers loss before it is kept or promoted. The survey of §5 asks which deployed systems have closed which of those five steps; §6 names the one closure no public system has yet reached.

3. Artifact-layer descent

The disciplined version of scaffold maintenance is one loop, run again and again over the life of a deployment. It does not react to single failures; it works from their accumulated record. As the deployment runs, it logs what went wrong — failed outputs, user corrections, rejected answers, eval regressions, tool-call errors. Reflecting over that record, the loop clusters the

failures into recurring failure modes and takes up the ones that matter most: a mode earns attention in proportion to how often it recurs and how badly it fails — repeatability times severity, which is exactly its contribution to the deployment’s failure rate. For such a mode it locates the single scaffold coordinate responsible — a stale memory entry, an under-specified instruction, a missing tool argument, a brittle retrieval rule, a misrouted agent edge — and synthesises one candidate edit to it. It measures whether the edit lowers patch loss (the deployment’s failure rate on its own recurring tasks), lets a gate accept or reject it on that measurement, keeps an accepted edit, and prunes or rolls back one that a later check shows made things worse. Then it takes the step most systems skip: it decides how far the edit should travel, promoting it to other deployments only as far as its evidence reaches — the edit’s evidence radius.

Most current production does not close this loop. A failure happens, someone adds a rule or memory entry, the artifact pile grows, nobody knows what caused improvement or when to prune, and fixes leak into broader contexts. That is patchwork, not descent. The loop has two distinct knobs. Loop 1 is the local fix on one deployment’s scaffold. Loop 2 is the separate decision of whether that fix should be shared more widely; it does not change the step size but the target — which average failure rate the scaffold is being tuned against — and the gate gets stricter as the radius widens, because an edit that lowers a narrow average can raise a broad one. This is why governance is not administrative overhead: it is the regularization system that keeps a small fix from being aimed at the wrong target.

Write the patch loss as $L_D(\theta_s) = \mathbb{E}_{x \sim D}[\ell(\pi(\theta_M, \theta_s, x))]$, with the frontier weights θ_M fixed and the scaffold θ_s — a structured object over the six substrate coordinates — the thing being fit. A failure yields no analytic gradient, only a candidate edit z_t proposed in order of the recurrence-times-severity of the mode it addresses; the gate accepts it when its noisy estimate of the directional loss change clears a margin,

$$G_t(z_t) = 1 \iff \hat{\Delta}_{z_t} L_D(\theta_s^t) + \lambda \Omega(z_t) \leq -\tau_t,$$

where $\Omega \geq 0$ penalises edit complexity and promotion breadth and $\tau_t \geq 0$ is the required margin. Production systems implement this without writing the equation: an eval blocks a merge on regression, a reviewer rejects a vague rule, a governance system holds a local artifact back until evidence accumulates. The descent reading is mechanism when the five conditions above hold and analogy otherwise; Appendix A states the assumptions, a one-step descent inequality, and a convergence proposition to a gate-stable scaffold.

Recent work operationalises all five conditions. SkillOpt (Yang et al. 2026) treats a single markdown skill document as the trainable external state of a frozen agent and edits it with explicit deep-learning-style controls — rollout and reflection batches, add/delete/replace edits, a bounded textual learning rate with cosine decay, a held-out selection gate, and a rejected-edit buffer kept as negative feedback — and is best or tied-best on all 52 evaluated (model, benchmark, harness) cells, beating the strongest per-cell baseline by +5.4 points on average and reporting positive cross-model, cross-harness, and cross-benchmark transfer that reads as an empirical evidence-radius measurement rather than a governance mechanism. SkillOpt is one instance in a fast-crystallising space: reflective prompt evolution (GEPA, Agrawal et al. 2025), reflective context learning (Vassilyev et al. 2026), harness optimisation (Lee et al. 2026; Lin et al. 2026; Cai et al. 2026), and lifecycle hygiene (X. Zhang et al. 2026) each instantiate a different subset of the loop, which confirms the architecture is real rather than any one group’s local optimum.

4. Six scaffold substrates

A patch-local scaffold composes six substrates. They are the coordinates of θ_s , each independently editable.

Substrate	What persists	Score-5 mark
S1. Instructions	Project/team/org rules attached to a workspace	Versioned, two-loop, Auto-gated promotion
S2. Skills	Procedural artifacts (code/text) loaded on demand	Skill bank self-modifies on failure, unit-test gated, promoted
S3. Memory	Cross-session experience (episodic / semantic / procedural)	Provenance + versioning + correction pathways
S4. Tools	Vetted action surface + context schemas	Domain-curated bundle is the product; eval-gated tool versions
S5. Orchestration	Multi-role graph + routing	Population/policy updates from outer-loop signal
S6. Governance	Versioning, promotion, audit, rollback	dev→staging→prod with eval thresholds + audit log + rollback

130 The 0–5 rubric is uniform: 0 ephemeral; 1 persistent local; 2 persistent with tool attach-
 131 ment but no feedback; 3 multi-scope or feedback without automated promotion; 4 automated
 132 feedback updates the scaffold (one loop closed); 5 two-loop versioned promotion with gover-
 133 nance (both loops closed). The descent argument is invariant to the exact partition so long
 134 as coordinates remain independently editable.

135 5. Survey and findings

136 The survey covers roughly 130 LLM systems published or productionised between January
 137 2023 and May 2026, assembled by searching public discourse for systems described as “self-
 138 improving,” “scaffolded,” or “agentic,” then filtering by whether the system wraps a foun-
 139 dation model with at least one persistent artifact updated from deployment signal. About
 140 90 satisfy this patch-local discriminator (score ≥ 3 on at least one substrate) and form the
 141 core corpus; roughly 38 carry the name but fail it and are kept as a contrast corpus so the
 142 boundary is auditable rather than rhetorical. Each system carries one or more evidence
 143 tiers, from peer-reviewed publication (T1) down to industry-blog documentation (T5); pro-
 144 duction claims are not pooled with peer-reviewed evidence when computing cluster-level
 145 statistics. A stricter composite-two-loop audit — Loop 1 modifies a persisted artifact from
 146 session signal and Loop 2 has an explicit gated cross-context promotion mechanism — is
 147 passed by thirteen research systems; the count moves under two natural relaxations of the
 148 criterion, but the qualitative findings below survive both.

Substrate	n	Mean	Score-5 systems	Survey finding
S1. Instructions	12	2.83	(none)	Rules persist across IDEs and agents, but none closes both an automated inner loop and a governed two-loop promotion

Substrate	n	Mean	Score-5 systems	Survey finding
S2. Skills	16	2.62	SAGE, COSPLAY	Research frontier active (failure-triggered skill update); production governance weaker
S3. Memory	19	3.16	Cuadros et al.	Cross-session memory is widespread; correctable, versioned memory is rare
S4. Tools	19	3.63	MCP, Harvey, Hippocratic	Most mature production substrate; tool bundles are the commercial unit
S5. Orchestration	19	2.68	(none)	Topologies exist; topology evolution under a gate is rare
S6. Governance	21	3.05	Braintrust, Vellum, LangSmith	Mature prompts-as-code: eval-gated merge, fast rollback, typically without an inner loop
INTEGRATED	34	3.74	NanoResearch, AutoAgent, SkillRL	The high-water mark, but dominated by research systems lacking enterprise governance

Two things stand out. The field has independently converged on the six substrates — every mature system has a recognisable instance of each, under different names — yet no system matures all six at once: research systems learn fast and lack enterprise governance, production systems govern well and learn slowly, and open standards solve artifact portability but not adaptation. The maturity gradient is not a quirk of the corpus; it tracks which constraints each kind of system was built to satisfy.

Four archetypal clusters recur (Appendix B). Research full-auto systems (NanoResearch, SkillOpt, AlphaEvolve, DGM, Voyager) automate both loops under algorithmic gates and uniformly lack enterprise governance. Production hybrid systems (SkillForge, CASCADE, Sierra, Devin) pair an algorithmic Loop-1 gate with a reviewer-judgment Loop-2 gate — the production-viable cell. Governance-first systems (Braintrust, Vellum, LangSmith Hub) automate Loop 2 alone via eval-gated CI, with no failure-triggered inner loop. Open standards (AGENTS.md, MCP, Claude Skills, Cursor Rules) converge across vendors on artifact format and tool attachment; AGENTS.md alone spans 60{,}000+ repositories with measured efficiency gains (Lulla et al. 2026). Indexed by gate type on each loop, the composite systems populate only three cells of a 3×3 matrix — [Auto $\times$ Auto], [Auto $\times$ Human], and [None $\times$ Auto] — leaving the human-authored-inner-loop cells empty for structural reasons.

6. The missing architecture: governed cross-tenant scaffold optimization

Four properties define an architecture that no surveyed system fully instantiates: (a) Auto-gated Loop 1, an algorithmic commit criterion; (b) multi-tenant cross-org promotion, Loop 2 routing artifacts between organisations; (c) versioned lineage with RBAC; and (d) rollback of a bad promotion without redeploy. The closest approaches partition the requirements. AlphaEvolve has (a) and partial (b) within one organisation; SkillForge has (a) and (b) inside one enterprise with partial (c); SkillOpt has (a) and partial (c), and reports transfer as evidence-radius measurement but implements neither (b) nor (d); CORAL (Qu et al. 2026) is the only asynchronous multi-agent Loop 2 in the corpus, within one system; PACE (Ling et al. 2026) shows the two-timescale architecture under small-frozen-model constraints; the governance-first cluster has (c) and (d) but neither (a) nor (b). No surveyed system combines all four, and the cross-organisational triad (b, c, d) is itself unfilled under any gate type. Naming the corner converts a vague gap into a four-property audit criterion any future system can be measured against. Whether it should be filled is deployment-dependent — in safety-critical domains a reviewer-judgment gate may be a design feature rather than a defect — but no public system fills it today, and that is what matters for the next two years of architecture work.

7. A new failure surface

Patchwork scaffold maintenance introduces failure modes weight-only systems do not have. Each one is also a missing piece of the optimizer. Scaffold bloat is the dominant near-term risk: instruction-following degrades and latency grows sharply as instruction count scales (Jaroslawicz et al. 2025; Liu et al. 2023), and bloat is also a signal-quality problem — LLM-authored skills have been reported at +0.0 pp over a no-skill baseline against +16.2 pp for human-curated ones, a gap (X. Zhang et al. 2026) attributes to lifecycle management — so complexity penalties and explicit pruning must be first-class. Poisoned memory is the dominant security risk: because memory is persistent and read back into later prompts, one malicious entry can be retrieved and acted on long after the attack (Zou et al. 2024; Z. Xu et al. 2026), which calls for provenance, write-gating, and a rollback envelope. Prompt injection via tools and MCP makes a tool’s output a cross-vendor attack surface, so tool bindings belong in the same promotion gate as any other coordinate. Coordination failures in multi-agent topologies (Cemri et al. 2025) make orchestration a coordinate subject to the same discipline. Patch overfitting is structural — a scaffold tuned to a deployment over-fits it, every time — and is contained only by evidence-radius discipline at promotion. Plasticity loss at the scaffold level (context rot, instruction conflict, memory pollution, tool-version drift) makes expiry and rollback first-class operations rather than retrospective hygiene.

Eval fragility is the dominant ecosystem risk, and it compounds because the eval suite is itself a patch-local artifact rather than a fixed instrument. It is optimised against: once a metric becomes the target it is gamed, and access asymmetries alone can produce up to 112% relative gains that reflect exposure to the test rather than capability (Singh et al. 2025). It is read non-independently: the gate scores path-dependent, non-i.i.d. agent trajectories, so its readings do not compose like ordinary test-set statistics, and evaluating an optimizer only by its downstream gains cannot distinguish informed Loop-1 updates

from trial-and-error (Ong et al. 2026). It goes stale: because the suite adapts on the same cadence as the scaffold it gates, a reading taken before an edit ages as the patch’s own failure distribution moves under it. And because failures concentrate in a finite catalogue of recurring modes rather than spreading uniformly, a suite that samples tasks at random under-weights exactly the families that drive the failure rate — the gate is only as good as its coverage of those modes. The architectural response is held-out evals scoped within evidence radius, weighted toward the recurring modes, re-read as the construct drifts, and checked against an update’s attribution rather than only its score.

The failure surface is real, quantified by 2024–2026 work, and largely unmitigated in production. Every entry on it names a piece of the optimizer that current systems do not yet have.

8. Conclusion

The field has built an external adaptation layer for LLMs, but not the optimizer that should govern it. If reliability is increasingly repaired through persistent scaffold artifacts, then production AI needs a theory and an engineering discipline for the discrete, gated, auditable learning that happens there — not just for the differentiable learning inside the model. This paper is one step toward that discipline: a survey of where production sits on the patchwork → random-search → descent gradient, a formalisation of the Stage-3 limit under stated assumptions, and a named missing architecture where the optimizer most plainly does not yet exist.

The frontier model generalises; the local scaffold specialises; reliability comes from governing that specialisation. The most consequential open architecture problem combines automated local evolution with auditable cross-tenant aggregation: Auto-gated Loop 1 plus governed multi-tenant Loop 2, with versioned lineage, RBAC, and rollback. A closely related problem rides alongside it. The instruments that validate these edits are themselves part of the moving, clustered scaffold, so the same evidence-radius and lifecycle discipline the optimizer needs has to extend to the gates that judge it — held-out, family-aware, and recalibrated as the target moves. Turning today’s patchwork into tomorrow’s optimizer depends on both.

References

- Agashe, Saaket et al. 2025. “Agent S2: A Compositional Generalist-Specialist Framework for Computer Use Agents.” arXiv Preprint. <https://arxiv.org/abs/2504.00906>.
- Agashe, Saaket, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. 2024. “Agent S: An Open Agentic Framework That Uses Computers Like a Human.” arXiv Preprint. <https://arxiv.org/abs/2410.08164>.
- AGENTS.md. 2025. AGENTS.md: A Cross-Vendor Open Standard for Agent Instructions. Linux Foundation. <https://agents.md>.
- Agrawal, Lakshya A., Shangyin Tan, Dilara Soylu, et al. 2025. “GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning.” arXiv Preprint. <https://arxiv.org/abs/2507.19457>.
- Alzubi, Salaheddin et al. 2026. “EvoSkill: Automated Skill Discovery for Multi-Agent Systems.” arXiv Preprint. <https://arxiv.org/abs/2603.02766>.
- Anthropic. 2024. Model Context Protocol Specification. <https://modelcontextprotocol.io>.
- Arbuzov, Mikhail L., Sisong Bei, Ziwei Dong, Dmitri Kalaev, and Alexey A. Shvets. 2025. “Beyond Exponential Decay: Rethinking Error Accumulation in Large Language Models.” arXiv Preprint. <https://arxiv.org/abs/2505.24187>.

- 253 Arbuzov, Mikhail L., Alexey A. Shvets, and Sisong Bei. 2026. The Architecture of Er-
254 rors: Logarithmic Mode Discovery and Polylogarithmic Intervention Budgets for Long-
255 Context LLM Reliability.
- 256 Cai, Qianshu, Yonggang Zhang, Xianzhang Jia, et al. 2026. “MOSS: Self-Evolution Through
257 Source-Level Rewriting in Autonomous Agent Systems.” arXiv Preprint. [https://arxiv.
258 org/abs/2605.22794](https://arxiv.org/abs/2605.22794).
- 259 Cemri, Mert et al. 2025. “Why Do Multi-Agent LLM Systems Fail?” arXiv Preprint.
260 <https://arxiv.org/abs/2503.13657>.
- 261 Chen, Minghao, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. 2024.
262 “AutoManual: Constructing Instruction Manuals by LLM Agents via Interactive Envi-
263 ronmental Learning.” Advances in Neural Information Processing Systems (NeurIPS).
264 <https://arxiv.org/abs/2405.16247>.
- 265 Chen, Yixing et al. 2025. “Multi-Agent Evolve: LLM Self-Improve Through Co-Evolution.”
266 arXiv Preprint. <https://arxiv.org/abs/2510.23595>.
- 267 Cuadros, Diego F., Abdoul-Aziz Maiga, Helen Meskhidze, and Andre Curtis-Trudel. 2026.
268 “Governed Collaborative Memory as Artificial Selection in LLM-Based Multi-Agent Sys-
269 tems.” arXiv Preprint. <https://arxiv.org/abs/2605.04264>.
- 270 Dohare, Shibhansh et al. 2024. “Loss of Plasticity in Deep Continual Learning.” Nature
271 632 (8026): 768–74. <https://doi.org/10.1038/s41586-024-07711-7>.
- 272 Fawzi, Alhussein, Matej Balog, Aja Huang, et al. 2022. “Discovering Faster Matrix Multi-
273 plication Algorithms with Reinforcement Learning.” Nature 610 (7930): 47–53.
- 274 Harvey. 2026. Harvey Product Overview. <https://www.harvey.ai>.
- 275 He, Yufei et al. 2025. “EvoTest: Evolutionary Test-Time Learning for Self-Improving
276 Agentic Systems.” arXiv Preprint. <https://arxiv.org/abs/2510.13220>.
- 277 Hippocratic AI. 2026. Polaris Clinical Outcome Evidence. <https://www.hippocraticai.com>.
- 278 Hu, Shengran, Cong Lu, and Jeff Clune. 2024. “Automated Design of Agentic Systems
279 (ADAS).” arXiv Preprint. <https://arxiv.org/abs/2408.08435>.
- 280 Huang, Xu et al. 2025. “CASCADE: Cumulative Agentic Skill Creation Through Au-
281 tonomous Development and Evolution.” arXiv Preprint. [https://arxiv.org/abs/2512.
282 23880](https://arxiv.org/abs/2512.23880).
- 283 Jaroslawicz, Daniel et al. 2025. “How Many Instructions Can LLMs Follow at Once?” arXiv
284 Preprint. <https://arxiv.org/abs/2507.11538>.
- 285 Kirk, Robert, Ishita Mediratta, Christoforos Nalmpantis, et al. 2024. “Understanding the
286 Effects of RLHF on LLM Generalisation and Diversity.” ICLR 2024. [https://arxiv.org/
287 abs/2310.06452](https://arxiv.org/abs/2310.06452).
- 288 Lee, Yoonho, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea
289 Finn. 2026. “Meta-Harness: End-to-End Optimization of Model Harnesses.” arXiv
290 Preprint. <https://arxiv.org/abs/2603.28052>.
- 291 Li, Hanchen, Runyuan He, Qizheng Zhang, et al. 2026. “Combee: Scaling Prompt Learning
292 for Self-Improving Language Model Agents.” arXiv Preprint. [https://arxiv.org/abs/
293 2604.04247](https://arxiv.org/abs/2604.04247).

- 294 Lin, Jiahang, Shichun Liu, Chengjun Pan, et al. 2026. “Agentic Harness Engineering:
295 Observability-Driven Automatic Evolution of Coding-Agent Harnesses.” arXiv Preprint.
296 <https://arxiv.org/abs/2604.25850>.
- 297 Ling, Chen, Pei Chen, Albert Guan, et al. 2026. “PACE: Two-Timescale Self-Evolution for
298 Small Language Model Agents.” arXiv Preprint. <https://arxiv.org/abs/2605.23019>.
- 299 Liu, Nelson F. et al. 2023. “Lost in the Middle: How Language Models Use Long Contexts.”
300 Transactions of the Association for Computational Linguistics (TACL). <https://arxiv.org/abs/2307.03172>.
301
- 302 Liu, Xingyan et al. 2026. “SkillForge: Forging Domain-Specific, Self-Evolving Agent Skills
303 in Cloud Technical Support.” arXiv Preprint. <https://arxiv.org/abs/2604.08618>.
- 304 Lulla, Jai Lal et al. 2026. “On the Impact of AGENTS.md Files on the Efficiency of AI
305 Coding Agents.” arXiv Preprint. <https://arxiv.org/abs/2601.20404>.
- 306 Lyle, Clare, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will
307 Dabney. 2023. “Understanding Plasticity in Neural Networks.” International Conference
308 on Machine Learning (ICML). <https://arxiv.org/abs/2303.01486>.
- 309 Madaan, Aman et al. 2023. “Self-Refine: Iterative Refinement with Self-Feedback.” Ad-
310 vances in Neural Information Processing Systems (NeurIPS). <https://arxiv.org/abs/2303.17651>.
311
- 312 Ni, Jingwei et al. 2026. “Trace2Skill: Distill Trajectory-Local Lessons into Transferable
313 Agent Skills.” arXiv Preprint. <https://arxiv.org/abs/2603.25158>.
- 314 Novikov, Alexander et al. 2025. AlphaEvolve: A Coding Agent for Scientific and Algorith-
315 mic Discovery. arXiv preprint arXiv:2506.13131. <https://arxiv.org/abs/2506.13131>.
- 316 Ong, Kai Tzu-iunn, Minseok Kang, Dongwook Choi, et al. 2026. “Towards Direct Evalu-
317 ation of Harness Optimizers via Priority Ranking.” arXiv Preprint. <https://arxiv.org/abs/2605.22505>.
318
- 319 Padmakumar, Vishakh, and He He. 2023. “Does Writing with Language Models Reduce
320 Content Diversity?” arXiv Preprint. <https://arxiv.org/abs/2309.05196>.
- 321 Park, Joon Sung et al. 2023. “Generative Agents: Interactive Simulacra of Human Behav-
322 ior.” Proceedings of the 36th Annual ACM Symposium on User Interface Software and
323 Technology (UIST). <https://arxiv.org/abs/2304.03442>.
- 324 Qu, Ao, Han Zheng, Zijian Zhou, et al. 2026. “CORAL: Towards Autonomous Multi-
325 Agent Evolution for Open-Ended Discovery.” arXiv Preprint. <https://arxiv.org/abs/2604.01658>.
326
- 327 Shalev, Yuval, Zifeng Ding, and Mateja Jamnik. 2026. “Training Language Agents to Learn
328 from Experience.” arXiv Preprint. <https://arxiv.org/abs/2605.20477>.
- 329 Shinn, Noah et al. 2023. “Reflexion: Language Agents with Verbal Reinforcement Learning.”
330 Advances in Neural Information Processing Systems (NeurIPS). <https://arxiv.org/abs/2303.11366>.
331
- 332 Sierra. 2024. Sierra Agent OS. <https://sierra.ai>.
- 333 Sierra. 2025. Agent OS 2.0: From Answers to Memory and Action. <https://sierra.ai/blog/agent-os-2-0>.
334

- 335 Singh, Shivalika et al. 2025. “The Leaderboard Illusion.” arXiv Preprint. <https://arxiv.org/abs/2504.20879>.
336
- 337 Song, Weijia, Jiashu Yue, and Zhe Pang. 2026. “ABSTRAL: Automatic Design of
338 Multi-Agent Systems Through Iterative Refinement and Topology Optimization.” arXiv
339 Preprint. <https://arxiv.org/abs/2603.22791>.
- 340 Sumers, Theodore R., Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. 2023.
341 “Cognitive Architectures for Language Agents (CoALA).” Transactions on Machine
342 Learning Research (TMLR). <https://arxiv.org/abs/2309.02427>.
- 343 Tan, Weihao et al. 2024. “Cradle: Empowering Foundation Agents Towards General Com-
344 puter Control.” arXiv Preprint. <https://arxiv.org/abs/2403.03186>.
- 345 Tanjim, Md Mehrab, Jayakumar Subramanian, Xiang Chen, et al. 2026. “MOCHA: Multi-
346 Objective Chebyshev Annealing for Agent Skill Optimization.” arXiv Preprint. <https://arxiv.org/abs/2605.19330>.
347
- 348 Tiwari, Rishabh, Kusha Sareen, Lakshya A. Agrawal, et al. 2026. “Learning, Fast and Slow:
349 Towards LLMs That Adapt Continually.” arXiv Preprint. <https://arxiv.org/abs/2605.12484>.
350
- 351 Tran, Dat, and Douwe Kiela. 2026. “Single-Agent LLMs Outperform Multi-Agent Systems
352 on Multi-Hop Reasoning Under Equal Thinking Token Budgets.” arXiv Preprint. <https://arxiv.org/abs/2604.02460>.
353
- 354 Vassilyev, Nikita, William Berrios, Ruowang Zhang, Bo Han, Douwe Kiela, and Shikib
355 Mehri. 2026. “Reflective Context Learning: Studying the Optimization Primitives of
356 Context Space.” arXiv Preprint. <https://arxiv.org/abs/2604.03189>.
- 357 Wang, Guanzhi et al. 2023. “Voyager: An Open-Ended Embodied Agent with Large
358 Language Models.” arXiv Preprint. <https://arxiv.org/abs/2305.16291>.
- 359 Wang, Jiong Xiao et al. 2025. “Reinforcement Learning for Self-Improving Agent with Skill
360 Library.” arXiv Preprint. <https://arxiv.org/abs/2512.17102>.
- 361 Wang, Xiaoxing et al. 2026. “AutoAgent: Evolving Cognition and Elastic Memory Orches-
362 tration for Adaptive Agents.” arXiv Preprint. <https://arxiv.org/abs/2603.09716>.
- 363 Wang, Zhenhailong et al. 2025. “Mobile-Agent-E: Self-Evolving Mobile Assistant for Com-
364 plex Tasks.” arXiv Preprint. <https://arxiv.org/abs/2501.11733>.
- 365 Wu, Rong et al. 2025. “EvolveR: Self-Evolving LLM Agents Through an Experience-Driven
366 Lifecycle.” arXiv Preprint. <https://arxiv.org/abs/2510.16079>.
- 367 Wu, Xiyang et al. 2026. “Co-Evolving LLM Decision and Skill Bank Agents for Long-
368 Horizon Tasks.” arXiv Preprint. <https://arxiv.org/abs/2604.20987>.
- 369 Wu, Zhiyong et al. 2024. “OS-Copilot: Towards Generalist Computer Agents with Self-
370 Improvement.” arXiv Preprint. <https://arxiv.org/abs/2402.07456>.
- 371 Xia, Peng et al. 2026. “SkillRL: Evolving Agents via Recursive Skill-Augmented Reinforce-
372 ment Learning.” arXiv Preprint. <https://arxiv.org/abs/2602.08234>.
- 373 Xu, Jinhang et al. 2026. “NanoResearch: Co-Evolving Skills, Memory, and Policy for
374 Personalized Research Automation.” arXiv Preprint. <https://arxiv.org/abs/2605.10813>.
- 375 Xu, Zhenlin et al. 2026. “From Storage to Steering: Memory Control Flow Attacks on LLM
376 Agents.” arXiv Preprint. <https://arxiv.org/abs/2603.15125>.

- 377 Yang, Yifan, Ziyang Gong, Weiquan Huang, et al. 2026. SkillOpt: Executive Strategy for
378 Self-Evolving Agent Skills. arXiv preprint arXiv:2605.23904. <https://microsoft.github.io/SkillOpt>.
379
- 380 Yao, Shunyu, Jeffrey Zhao, Dian Yu, et al. 2023. “ReAct: Synergizing Reasoning and
381 Acting in Language Models.” International Conference on Learning Representations
382 (ICLR). <https://arxiv.org/abs/2210.03629>.
- 383 Yuksekgonul, Mert, Federico Bianchi, Joseph Boen, et al. 2024. “TextGrad: Automatic
384 ‘Differentiation’ via Text.” arXiv Preprint. <https://arxiv.org/abs/2406.07496>.
- 385 Zehle, Tom. 2026. “CANTANTE: Optimizing Agentic Systems via Contrastive Credit
386 Attribution.” arXiv Preprint. <https://arxiv.org/abs/2605.13295>.
- 387 Zhang, Hanrong et al. 2026. “CoEvoSkills: Self-Evolving Agent Skills via Co-Evolutionary
388 Verification.” arXiv Preprint. <https://arxiv.org/abs/2604.01687>.
- 389 Zhang, Jenny et al. 2025. “Darwin Gödel Machine: Open-Ended Evolution of Self-
390 Improving Agents.” arXiv Preprint. <https://arxiv.org/abs/2505.22954>.
- 391 Zhang, Qizheng, Changran Hu, Shubhangi Upasani, et al. 2025. Agentic Context En-
392 gineering: Evolving Contexts for Self-Improving Language Models. arXiv preprint
393 arXiv:2510.04618. <https://arxiv.org/abs/2510.04618>.
- 394 Zhang, Xing, Yanwei Cui, Guanghui Wang, et al. 2026. “Ratchet: A Minimal Hygiene
395 Recipe for Self-Evolving LLM Agents.” arXiv Preprint. <https://arxiv.org/abs/2605.22148>.
396
- 397 Zhou, Yingli, Wang Shu, Yaodong Su, et al. 2026. “A Comprehensive Survey on Agent
398 Skills: Taxonomy, Techniques, and Applications.” arXiv Preprint. <https://arxiv.org/abs/2605.07358>.
399
- 400 Zou, Wei et al. 2024. “PoisonedRAG: Knowledge Corruption Attacks to Retrieval-
401 Augmented Generation of Large Language Models.” arXiv Preprint. <https://arxiv.org/abs/2402.07867>.
402

403 Appendix A. Formalization of artifact-layer descent

404 This appendix supplies the verification machinery for §3: the setup, a one-step descent
405 inequality under a noisy estimator, a convergence proposition with its assumptions, and the
406 optimisation-analogy correspondence.

407 Setup. Let the frontier model have fixed parameters θ_M and let the scaffold be a structured
408 object $\theta_s \in \Theta_s = \Theta_1 \times \dots \times \Theta_6$ over the six substrate coordinates; Θ_s is discrete, symbolic,
409 and versioned. For a deployment patch D , $L_D(\theta_s) = \mathbb{E}_{x \sim D}[\ell(\pi(\theta_M, \theta_s, x))]$. A candidate
410 edit $z_t \sim Q_t(z \mid H_t, \theta_s^t)$ is drawn from a proposal process conditioned on the accumulated
411 failure history H_t . Q_t is not uniform: for a mode m that recurs with frequency f_m and carries
412 per-occurrence loss s_m , its expected contribution is $f_m s_m$ — repeatability times severity —
413 so high-frequency or high-severity modes are proposed first. The finite directional difference
414 is $\Delta_z L_D(\theta_s) = L_D(\theta_s \oplus z) - L_D(\theta_s)$; the gate sees only a noisy estimate $\hat{\Delta}_z L_D$.

415 Descent inequality. Assume (A1) positive descent bias on accepted edits — $\mathbb{E}[\hat{\Delta}_{z_t} L_D \mid$
416 $G_t=1] \leq -\gamma_t - \tau_t$ for some $\gamma_t > 0$ — and (A2) bounded estimator bias — $|\mathbb{E}[\hat{\Delta}_{z_t} L_D -$
417 $\hat{\Delta}_{z_t} L_D \mid G_t=1]| \leq \varepsilon_t$. With $p_t = \Pr(G_t=1 \mid \theta_s^t)$ and the gated update of §3,

$$\mathbb{E}[L_D(\theta_s^{t+1}) \mid \theta_s^t] \leq L_D(\theta_s^t) - p_t(\gamma_t + \tau_t) + p_t \varepsilon_t.$$

Per-step expected improvement is the gate’s margin minus its estimator bias, weighted by acceptance probability.

Convergence (informal). Add (A3) proposal coverage (at every non-gate-stable scaffold, Q_t places probability $\geq q > 0$ on an admissible edit with $\mathbb{E}[\Delta_z L_D] \leq -\delta$), (A4) effective compactness of the reachable orbit, and (A5) persistent acceptance with summable bias ($\sum_t p_t(\gamma_t + \tau_t) = \infty$, $\sum_t p_t \varepsilon_t < \infty$), with $0 \leq L_D \leq B$. Then $L_D(\theta_s^t)$ is a non-negative supermartingale up to summable noise, converges almost surely, and the limit set is gate-stable: no admissible edit clears the gate with negative expected directional loss in the limit. This is convergence of the loss and stability of the gate decision, not convergence to a global optimum — a globally suboptimal gate-stable scaffold is consistent with the result, exactly as for local stochastic optimization under noisy estimators. In non-stationary patches ($D = D_t$) the same mechanism tracks a moving optimum rather than converging once, which is why recency weighting, pruning, expiry, and rollback are the scaffold-level analogues of continual-learning machinery rather than optional hygiene.

Optimization concept	Scaffold analogue
Parameter vector	Instructions, skills, memories, tools, routing graphs, eval gates, governance rules
Loss	Failure, user correction, eval regression, retrieval miss, tool error, schema violation
Gradient estimate	Credit assignment identifying the coordinate responsible for a failure
SGD step	Accepted scaffold edit
Learning rate	Edit magnitude under $\oplus$
Update radius	Promotion scope: which loss is descended on (USER, PROJECT, STACK, TENANT, CORE)
Validation loss	Held-out patch eval before merge
Regularization	Complexity penalty $\Omega(z)$, bloat control, cross-patch regression checks
Overfitting	Scaffold improves on this patch and degrades elsewhere
Rollback	Reverting a harmful accepted update

Credit assignment here is attribution, not backpropagation: the descent argument needs only that a failure is traced to the responsible coordinate with non-zero probability, not that the trace is computed by chain rule.

Appendix B. Survey detail

A representative spot-check of the corpus appears below, covering all six substrates, both research and production, and all four clusters. PP is the integrated patch-local-adaptation score (0–5); Evidence abbreviates the strongest source tier. The full corpus, the contrast set, the gate-family mapping, and the per-system SkillOpt walkthrough are deferred to an extended version.

System	Year	Substrate	Loop 1	Loop 2	PP	Evidence
NanoResearch	2026	Integrated	Auto	Auto	5	T2
SkillOpt	2026	Integrated (S2-led)	Auto	Auto	5	T2
SkillRL	2026	Integrated	Auto	Auto	5	T2
AlphaEvolve	2025	Integrated	Auto	Auto	4	T2/T3
DGM	2025	Integrated	Auto	Auto	4	T2
SkillForge	2026	Integrated	Auto	Human	4	T2

System	Year	Substrate	Loop 1	Loop 2	PP	Evidence
CASCADE	2025	Integrated	Auto	Human	4.5	T2
PACE	2026	Integrated (small-model)	Auto	Auto	4	T2
CORAL	2026	Integrated (S5+S3 async)	Auto	Auto	4	T2
Meta-Harness	2026	S5 Orches- tration	Auto	n/a	4	T2
Voyager	2023	Integrated	Auto	Auto	4	T2
Claude	2025	S1	n/a	n/a	4	T3/T4
Code		Instruc- tions				
CLAUDE.md		S3	Auto	Auto	5	T2
Cuadros	2026	Memory				
et al. mem- ory						
MCP	2024	S4 Tools	n/a	n/a	5	T3
Braintrust	2024	S6 Gover- nance	n/a	Auto	5	T3
Self- Refine*	2023	(contrast)	n/a	n/a	1	T2
Mem0*	2024	(contrast)	n/a	n/a	2	T3/T4

Asterisks mark contrast systems that fail the patch-local discriminator, included for defini-
tional clarity.

The composite-two-loop systems sit in a 3×3 matrix indexed by gate type on each loop. [Auto $\times$ Auto] holds the research full-auto systems with machine-evaluated criteria on both loops. [Auto $\times$ Human] is the production-viable cell: Loop 1 fires at machine speed, Loop 2 requires human approval before propagation. [None $\times$ Auto] holds the governance-first prompts-as-code systems with eval-gated CI but no failure-triggered candidate generation. [Human $\times$ Auto] is empty by engineering logic (manual authorship offers little leverage for automated promotion); [Human $\times$ Human] was empty in the surveyed window; [Human $\times$ None] collapses to “engineer edits a config” and is not patch-local in the sense of §3. The empty cross-tenant governed-optimization corner of §6 is the gap, not any of these.