

CoEvolve: Feedback-Driven Policy-Harness Maintenance for Industrial LLM Auditing Systems

Yiyang Peng ✉ 

China Telecom Sichuan Branch, Chengdu, China

Yifeng Gou ✉

China Telecom Sichuan Branch, Chengdu, China

Bo Zhang ✉

China Telecom Sichuan Branch, Chengdu, China

Qing Xu ✉

China Telecom Sichuan Branch, Chengdu, China

Yu Shen ✉

China Telecom Sichuan Branch, Chengdu, China

Xinyao Xiao ✉

China Telecom Sichuan Branch, Chengdu, China

Gang Lv ✉

China Telecom Sichuan Branch, Chengdu, China

Qizhao Wang ✉

China Telecom Sichuan Branch, Chengdu, China

Ming Zhang ✉

China Telecom Sichuan Branch, Chengdu, China

Xiaosong Ding ✉

China Telecom Sichuan Branch, Chengdu, China

Yusang Xiong ✉

China Telecom Sichuan Branch, Chengdu, China

Xinyu Liu ✉

China Telecom Sichuan Branch, Chengdu, China

Abstract

Industrial LLM compliance-auditing systems are maintained through both expert policy and executable support chains. After deployment, changing standards, user appeals, expert reviews, and report variants expose failures in rule interpretation, parsing, evidence binding, context assembly, output contracts, and regression checks. We propose CoEvolve, a feedback-driven policy-harness maintenance workflow in which an LLM maintenance agent proposes coordinated updates, an independent verifier checks target repair, regression, stability, and semantic risks, and experts confirm releasable changes. We report a five-month industrial experience in a Chinese compliance-auditing service, covering 33 production feedback instances, 128 audited systems, 384 reports, and 24,192 checklist items. The evaluation compares four frozen workflow states formed by different maintenance organizations; it is not a controlled comparison of methods on identical maintenance inputs. On a 979-item monthly holdout benchmark, the CoEvolve replay state reached 93.5% accuracy, 4.1 percentage points above prompt/SOP maintenance at 89.4%. Manual co-maintenance reached 93.6% (916/979), while CoEvolve reached 915/979; their Wilson intervals overlap and the five-system effective sample is small. Because the replay input included visible historical manual artifacts, this result shows that CoEvolve reconstructed a comparable release state from those artifacts, not that it substituted for manual co-maintenance. Across replay tasks, CoEvolve externalized 4.97 candidate attempts per feedback instance on average. The experience supports governed policy-harness maintenance without retraining and highlights verifier-gate predicates and blocked-candidate records as transferable release-governance mechanisms.

2012 ACM Subject Classification Software and its engineering → Software creation and management; Computing methodologies → Natural language processing

Keywords and phrases LLM auditing, policy-harness maintenance, production feedback, verifier gates, software engineering in practice

Digital Object Identifier 10.4230/LIPIcs.ESEM.2026.80

Category Software Engineering in Practice Track Paper

1 Introduction

Industrial compliance auditing is adopting LLMs to extract evidence, interpret rules, and produce structured checklist-level judgments across multiple reports, standards, and checklist items. Unlike general question answering, the output must remain tied to inspectable



© Yiyang Peng, Gang Lv, Yifeng Gou, Qizhao Wang, Bo Zhang, Ming Zhang, Qing Xu, Xiaosong Ding, Yu Shen, Yusang Xiong, Xinyao Xiao, and Xinyu Liu;
licensed under Creative Commons License CC-BY 4.0

20th International Symposium on Empirical Software Engineering and Measurement (ESEM 2026).

Editors: Robert Feldt, Maria Paasivaara, Daniel Mendez, Stefan Wagner, and Marvin Muñoz Barón;

Article No. 80; pp. 80:1–80:18



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

evidence, applicable standards, and explicit audit items. After deployment, user appeals, expert findings, runtime failures, and changing standards expose boundary cases. Many feedback signals appear to concern prompts, while the actual repair surface may involve rule interpretation, report parsing, evidence binding, context extraction, output contracts, or verification.

Model-external failures cannot be repaired by changing model weights alone. Industrial teams therefore also maintain assets around a fixed model without retraining. Domain experts revise SOPs, rules, and prompt criteria. Developers maintain parsers, extractors, evidence routing, context schemas, and structured output contracts. This split helps initial delivery, but it also reflects a long-standing property of expert work: experts often know more than they can fully articulate at system cold start [17]. Production feedback turns some of these tacit boundaries into versioned rules, cases, and release records. A disputed judgment may require fixing field extraction before revising a rule boundary, while a local text repair may create regressions in previously confirmed behavior.

CoEvolve reframes the maintenance object as a *policy-harness bundle*. Policy denotes domain knowledge maintained by experts: standard interpretation, rule boundaries, audit criteria, SOPs, and prompt criteria. Harness denotes the executable support chain that makes policy usable: parser, extractor, reference retrieval, evidence binding, context assembly, and output contract. In CoEvolve, experts first adjudicate feedback and fix the expected audit semantics. An LLM maintenance agent proposes owner hypotheses and coordinated policy-harness candidates. The independent verifier derives target and regression cases from feedback records, candidate changes, and historically confirmed behavior, then checks target repair, regression, contract stability, and semantic-governance risks. Experts confirm release or scoped release after verification.

We study CoEvolve in a production Chinese safety/compliance auditing service. From December 2025 to April 2026, the service covered 128 audited target systems, 384 reports, and 24,192 checklist items, and accumulated 33 production feedback instances. One feedback instance may contain multiple disputed records or related issues; we count feedback by production maintenance triggers.

To observe the frozen states formed by different maintenance organizations, we score four completed workflow states on the same 979-item monthly holdout benchmark. This is a workflow-state comparison, not a controlled comparison of methods on identical maintenance inputs: the workflows share final scoring controls but differ in work organization, intermediate records, and human-effort profiles. The primary empirical contrast is CoEvolve at 93.5% versus prompt/SOP maintenance at 89.4%, a 4.1-percentage-point difference on the same holdout. Intensive manual co-maintenance reaches 93.6%, one checklist item above CoEvolve; the two estimates have overlapping Wilson intervals and only five system-level clusters. CoEvolve was introduced in April 2026 and evaluated retrospectively. Its replay input included visible historical manual artifacts, so the replay reconstructs a comparable release state rather than demonstrating substitution for manual co-maintenance. Section 4 details the replay boundary and scoring controls. The all-industry author team combines system development, release governance, and domain-expert adjudication roles.

This paper makes four contributions:

1. It frames post-deployment maintenance of industrial LLM auditing systems as policy-harness co-evolution.
2. It presents CoEvolve, a feedback-to-release workflow involving expert adjudication, owner hypotheses, policy-harness candidates, independent verification, expert confirmation, and release records.

3. It reports a five-month industrial trajectory and a 979-item monthly holdout comparison of frozen workflow states, leading with the 4.1-percentage-point contrast between CoEvolve and prompt/SOP maintenance and bounding the manual comparison as a historical reconstruction result.
4. It distills SEIP practice lessons: owner-before-patch, denominator-before-score, gate-before-release, and negative candidates as governance evidence.

2 Industrial Context and Problem

2.1 Runtime Audit Path

Figure 1 summarizes the runtime path. An audited target system typically provides three reports and related materials. These reports may include tables, screenshots, scanned pages, and cross-report references. Before the LLM auditor is called, the system parses multimodal reports, maps sections and checklist items, synchronizes cross-report references, checks system attributes and dates, selects applicable standard versions, binds evidence, assembles audit context and prompt criteria, and enforces a structured output contract.

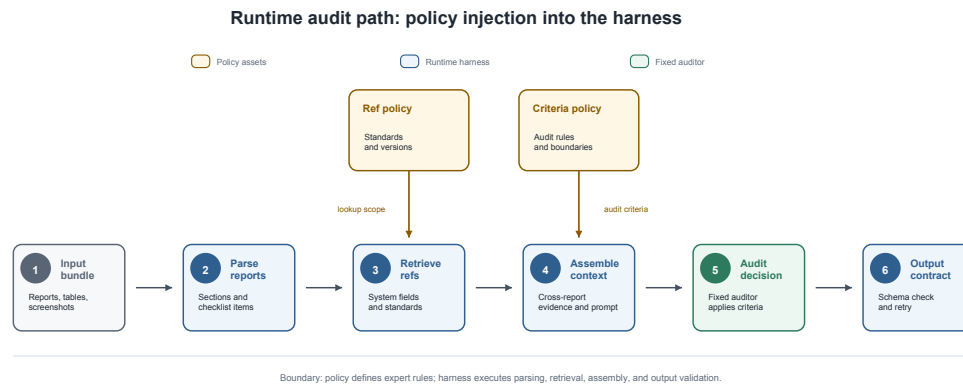


Figure 1 Runtime audit path and responsibility surfaces. Policy is expert-maintained domain knowledge; harness components make that knowledge executable and usable by the fixed auditor.

The key point of Figure 1 is responsibility separation. Policy includes expert-provided standards, SOPs, audit criteria, responsibility boundaries, and prompt criteria. Harness includes executable components for multimodal parsing, cross-file retrieval, external-standard selection, evidence binding, context assembly, output contracts, and replay tests. Release gates are executed by the independent verifier. In the historical workflow, experts mainly owned policy while developers owned harness. CoEvolve addresses feedback that forces these assets to evolve together.

2.2 Why Prompt-Only Maintenance Is Insufficient

Production feedback comes from user appeals, expert findings, runtime failures, regression failures, and release audits. Several recurring signals require harness maintenance beyond prompt/SOP text: report-format changes can make stable fields disappear from the parsed input; evidence may exist but bind to the wrong checklist item; report-level context may be missing even when the prompt asks for it; structured outputs may violate the required schema;

and a local prompt repair may break historically confirmed behavior. These failures explain the policy-harness framing. Prompt text remains important, while production reliability also depends on the executable path that delivers facts, standards, and evidence to the fixed auditor.

3 CoEvolve Method

3.1 Maintained Object

Figure 1 motivates the maintained object. A reliable released state depends on whether expert policy can be delivered, assembled, checked, and invoked through executable harness components. We model a releasable state as a maintained runtime bundle plus an independent verification layer:

$$B = \langle P, H_{in}, H_{ref}, H_{ctx}, H_{out} \rangle, \quad V = \langle V_{target}, V_{replay}, V_{static}, V_{release} \rangle.$$

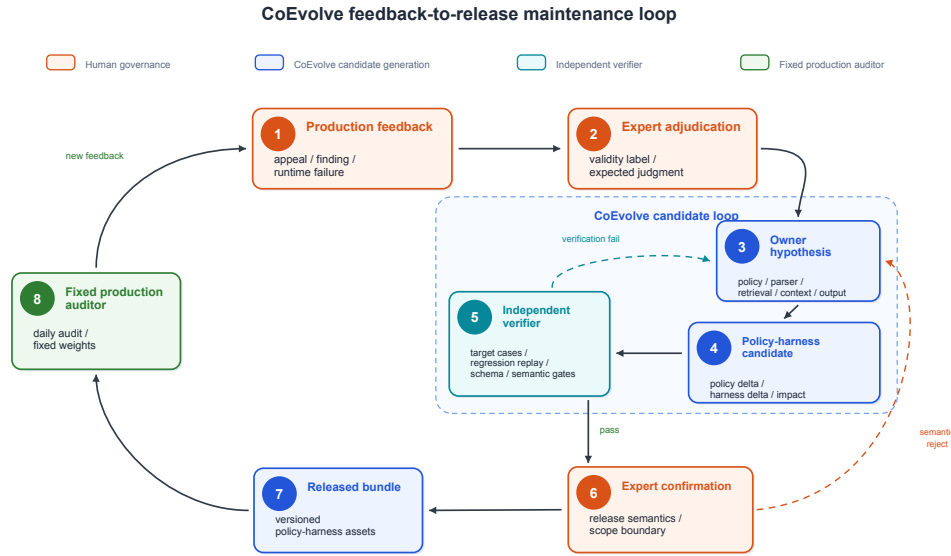
Here B is the runtime bundle maintained by CoEvolve. P represents expert policy; H_{in} , H_{ref} , H_{ctx} , and H_{out} represent harness slots that make policy executable. The maintenance agent can propose changes to B , together with an owner hypothesis, affected surfaces, and expected impact. V is held by the independent verification agent; in this paper, gates are the checks and blocking rules executed by that verifier. The verifier derives target and regression cases from feedback records, candidate changes, historically confirmed behavior, and sampled non-holdout items. Verification has two families. Hard checks cover target repair, historical replay, rerun stability, output-contract validity, parse-ok status, evidence/context validity, and release scope. Semantic-governance checks cover sample memorization, metric chasing, policy consistency, rule overreach, and denominator integrity. Validation rules, scoring manifests, target/regression denominators, and verifier gates remain in the verification layer.

■ **Table 1** Policy-harness bundle slots and their runtime roles.

Layer	Runtime role	Typical content
P	Domain policy and prompt criteria	Standard interpretation, rule boundaries, responsibility criteria
H_{in}	Input parsing	Multimodal parsing, chunking, section mapping, checklist-item location
H_{ref}	Reference retrieval	Official standard versioning, knowledge-base lookup, external field extraction
H_{ctx}	Context assembly	Cross-report evidence binding, checklist context, prompt assembly
H_{out}	Structured output	Output schema, parse-ok checks, retry contract
V	Independent verification	Target repair, historical replay, rerun stability, semantic governance, release record

3.2 Feedback-Driven Maintenance Loop

Figure 2 shows the maintenance loop. Production feedback first receives expert adjudication. The LLM maintenance agent then proposes failure-owner hypotheses and candidate policy-harness patches. Each candidate must declare the hypothesized owner, policy delta, harness delta, expected impact, and release risk. The independent verification agent derives target and regression cases from feedback records, candidate changes, and historical behavior, then



■ **Figure 2** CoEvolVe maintenance loop. The LLM maintenance agent expands candidate repairs; the independent verifier and experts select releasable updates.

produces a verification summary. Gate failures and expert semantic rejection return to owner-hypothesis search. Released or scoped updates become the bundle used by the fixed production auditor.

The fixed production auditor handles daily audit checks. It is distinct from the offline candidate-generation endpoint used to propose maintenance candidates. Model weights remain fixed during the study; learning happens through versioned policy-harness assets and release records.

3.3 Maintenance-Agent and Verification-Agent Contract

The maintenance agent can access redacted feedback summaries, expert adjudication, current policy-harness assets, historical patches, and verification summaries. It cannot access production credentials, public copies of raw customer materials, benchmark labels, verification rules, scoring manifests, or production-release authority. It outputs candidate patch bundles to the independent verifier and expert-confirmation process.

The independent verifier holds validation logic outside the writable scope of the maintenance agent. It produces verification summaries, block reasons, residual owners, and release recommendations. Gate implementation is mixed. Programmatic checks cover parsing, schema and output-contract validity, denominator integrity, target-case scoring, historical replay, and repeated-run comparisons; these checks execute the fixed auditor where an LLM judgment must be rerun. Semantic consistency, rule scope, ambiguous target judgments, and final release scope require expert review. The maintenance agent does not approve its own candidates. Its hard checks ask whether the target issue is repaired, historical regression cases remain stable, repeated runs are consistent, output contracts remain parseable, evidence/context bindings are valid, and a candidate needs scoped release. Its semantic-governance checks look for memorization, metric chasing, policy inconsistency, rule overreach, and denominator changes. This separation matters in practice: the maintenance agent can suggest repairs, while release validation remains governed by an independent gate and expert confirmation. Here, independence means separation of role, permission, and writable assets:

■ **Table 2** Maintenance-agent contract and verification boundary.

Boundary	Maintenance agent	Verifier / expert governance
Visible inputs	Redacted feedback, expert judgment, current bundle, historical patches, run summaries	Candidate bundle, feedback records, confirmed behavior, sampled non-holdout items
Writable records	Owner hypothesis, policy/harness deltas, expected impact, release risk	Verification summary, block reason, residual owner, release recommendation
Held outside	Raw customer materials, credentials, benchmark labels, scoring manifests, release authority	Gate rules, regression denominator, semantic-governance criteria
Decision role	Generates candidate repairs and revision options	Selects releasable updates through verifier gates and expert confirmation

the candidate generator does not also hold verifier-gate authority, and verification rules, semantic-governance criteria, and scoring manifests are not edited by the maintenance agent. We report check families, redacted traces, and record schemas, while the full production gate rules remain internal control assets.

Three verifier-gate predicates illustrate the mechanism. A parse-ok gate blocks a candidate when required fields are missing or the structured output is malformed. A rerun-stability gate requests retry or rollback when repeated runs change the target judgment, parse state, or key context binding. A semantic-governance gate blocks release when a candidate copies feedback-specific answers into SOP or prompt text, which indicates metric chasing.

Operationally, the fixed production auditor is used for daily audit decisions. The offline candidate-generation endpoint is organization-approved and used only for maintenance candidates. Its exact vendor and model version are redacted for procurement and security reasons, while the replay uses a fixed endpoint configuration. The verification agent is a separate role: it consumes candidates and gate inputs, then produces verification summaries; bundle edits stay with candidate generation. A candidate attempt is one owner hypothesis plus a proposed policy delta, harness delta, or both, submitted to gate or expert judgment. Retries count as new attempts when they re-enter verification with a changed owner, delta, or target/regression set.

4 Study Design

4.1 Research Questions

We ask four research questions. RQ1 asks to what extent production feedback crosses the policy-harness boundary. RQ2 examines how a production feedback instance moves through owner hypotheses, policy-harness deltas, verification summaries, and release decisions. RQ3 compares the frozen CoEvolve replay state with initial, prompt/SOP, and manual co-maintenance workflow states under shared offline scoring controls. RQ4 measures candidate exploration and verification workload.

4.2 Data and Labels

The study window is December 2025 to April 2026. The production trajectory covers 128 audited target systems, 384 reports, and 24,192 checklist items. It contains 33 production feedback instances from user appeals, expert findings, and runtime issues. Feedback instances are not deduplicated, because they are the maintenance triggers observed in production; one instance may contain multiple disputed records.

For counting, a *feedback instance* is a production maintenance trigger, such as a user appeal, expert finding, or runtime failure report. A *disputed record* is a checklist/report-level disagreement inside an instance. A *candidate attempt* is a concrete maintenance proposal that enters verification or expert judgment. These levels are reported separately because one feedback instance can create several disputed records and several candidate attempts.

The monthly holdout benchmark samples one audited target system per month from December 2025 to April 2026. Each target system contains three reports, yielding five target systems, fifteen reports, and 979 checklist items. Benchmark labels were produced through two-person independent labeling and the organization’s audit-adjudication process. The two annotators labeled the benchmark items without seeing each other’s labels; disagreements were discussed and final labels followed the organization’s adjudication rule. The expert group included four business experts, two with more than five years of relevant audit experience, and semantic disputes followed the senior-expert adjudication process. Benchmark labels were finalized before workflow scoring.

The holdout benchmark is within-setting: it shares the production domain, report families, and audit standards with the maintenance trajectory. The five holdout target systems and their fifteen reports were reserved as final scoring materials. Their raw report content, checklist outputs, labels, scoring manifest, and aggregate scores were not available during maintenance-state construction, and they did not enter replay as feedback, target, or regression material. The 33 feedback instances served as maintenance triggers, target cases checked whether feedback-triggered issues were repaired, and regression cases protected historically confirmed behavior. Final benchmark scoring was performed only after each workflow state was frozen. Thus, the holdout is deliberately close to the industrial stream in domain and standards; the isolation boundary is the system/report scoring materials, labels, outputs, manifest, and aggregate scores, rather than an external benchmark.

4.3 Baseline Construction and Fairness Controls

The study compares frozen maintenance states constructed from the same ordered stream of audited target systems. From December 2025 to April 2026, production maintenance was mainly conducted through expert-developer co-maintenance. We reconstruct additional workflow states along that trajectory. Because the workflows differ in organization, intermediate signals, editable assets, and human-effort profiles, this design compares completed workflow states; it does not estimate method effects under identical maintenance inputs.

We align workflows by the ordered system-input stream rather than by identical feedback text. Each workflow follows the same chronological sequence of audited target systems and submitted reports, with the same available external materials, feedback-raising channel, and expert-adjudication group. The concrete feedback records entering each workflow are workflow-specific. Domain experts owned the prompt/SOP baseline and could revise only SOP, rule, and prompt text; they had access to the same adjudication group and stopped at the workflow’s normal release or no-change endpoint before the assets were frozen. Harness and regression-logic edits were not allowed in that state. Comparable total human-hours were not recorded for prompt/SOP maintenance, manual co-maintenance, and CoEvolve expert confirmation, so the study does not make an effort or cost comparison. Facing the same system, expert-developer co-maintenance may surface a parser or rule issue through human debugging, prompt/SOP maintenance may surface a text-maintenance problem or an escalation, and CoEvolve may surface a context-owner hypothesis, a verifier block, and a later candidate retry. These differences are part of the workflow behavior under study.

Except for the initial snapshot, each maintenance state completes one maintenance pass over the aligned system stream and freezes the resulting assets before offline scoring. All states then run on the same 979-item monthly holdout benchmark, using the same auditor interface, output schema, denominator, labels, and scoring manifest. The monthly holdout is not used as the target or regression denominator during maintenance. Target cases come from feedback-triggered disputed items, while regression cases come from historically confirmed behavior and sampled non-holdout items. Before final scoring, no item from the 979-item monthly holdout was used for candidate generation, target-repair checks, dynamic regression construction, prompt/SOP tuning, manual debugging, verifier-gate tuning, or release decisions. In the historical manual workflow, regression records were maintained by experts and developers; in CoEvolve replay, the verifier maintains and expands the regression denominator. Benchmark labels, benchmark outputs, scoring manifests, production credentials, and raw customer materials are outside maintenance-state construction.

■ **Table 3** Baseline construction, validation boundary, and scoring boundary.

State	Maintenance surface	Feedback signal	Validation boundary	Scoring boundary
Initial	Initial policy and harness	Initial requirements and delivery materials	No post-deployment feedback maintenance	Frozen first release; same holdout
Prompt/SOP tuning	Policy text only	Text-maintenance signals surfaced from the aligned stream	Experts confirm feedback resolution; no harness or regression-logic edits	Frozen text assets; same holdout
Manual co-maintenance	Policy and harness	Historical system stream and human debugging signals	Experts and developers maintain target, regression, and release records	Frozen human release; same holdout
CoEvolve replay state	Candidate policy-harness bundle	Replay feedback signals from the aligned stream	Verifier maintains target/regression cases and checks repair, regression, rerun, semantic risk	Expert-confirmed replay state; same holdout

The four states use different maintenance organizations, so the comparison is a workflow-state comparison. The scoring step is shared: after each state reaches its normal release or no-change endpoint, assets are frozen and evaluated offline on the same benchmark. The prompt/SOP state uses its workflow-specific feedback records and adjudication to update text assets only. Manual co-maintenance uses the historical expert-developer release record, including human-maintained policy, harness, and regression records. The CoEvolve replay is not a clean-room run from the initial release. Its input boundary includes the historically maintained asset state, the same system-input stream, redacted run summaries, verification summaries, and structured historical maintenance records. Those records include versioned manual release assets, repair records, and verification summaries. CoEvolve therefore reconstructs a comparable release state from visible historical manual artifacts through candidate search, independent verification, and expert confirmation. Raw expert communications, private root-cause rationales, and item-level holdout answers stay outside the replay input boundary. They do not contain benchmark outputs, labels, or scoring manifests. CoEvolve has no direct production-release authority.

The replay protocol has four steps. First, the target-system order, the three submitted reports, external materials, and the expert-adjudication entry point are fixed in production-time order. Second, each workflow forms its own feedback signals and maintenance records on that aligned input stream; CoEvolve generates owner hypotheses and candidate bundles at this step. Third, candidates enter independent verification, where the verifier actively maintains target/regression cases and records target repair, historical replay, rerun stability, parse/schema, semantic-governance, and release/block outcomes. Finally, after each workflow reaches a release, scoped release, block, or no-change endpoint, its assets are frozen and scored offline on the same monthly holdout benchmark.

4.4 Process Metrics

RQ4 measures the CoEvolve replay process directly. Active maintenance time is an internal replay-loop interval: it starts when the maintenance agent begins candidate processing and ends when the release decision record is formed. It includes candidate generation, verifier-gate execution, and the recorded confirmation step when present. It excludes user-waiting time, cross-team queueing, handoff delay, release windows, long-term gate maintenance, and organizational communication cost. Candidate count measures candidate patch / maintenance attempts that enter verifier or expert judgment. These metrics describe only replay-loop workload and are not an end-to-end efficiency comparison.

5 Results

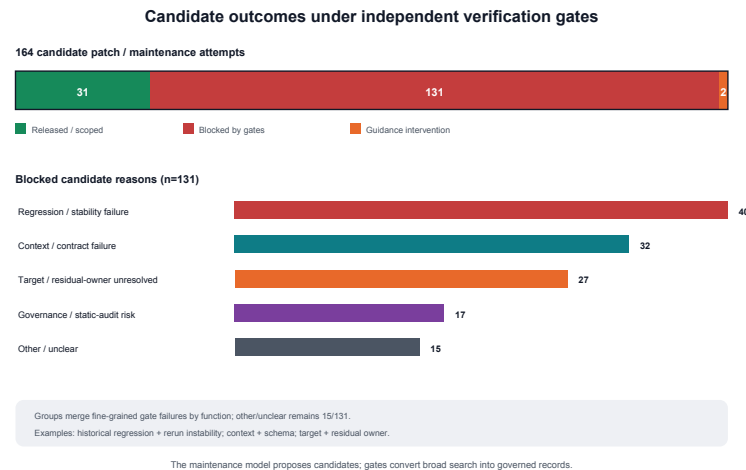
5.1 RQ1: Feedback Crosses Policy and Harness

In 33 feedback-owner audits, 18 instances were policy-only, 6 were harness-only, 7 involved both policy and harness, and 2 were unclear or review-needed. The counts represent maintenance signals rather than mutually exclusive defect classes, and they are not a prevalence estimate for all production defects. They show that among feedback instances that triggered maintenance in our setting, a subset required harness or policy-harness repair. Overall, 13/33 feedback instances touched harness surfaces, and 7/33 required policy-harness co-maintenance.

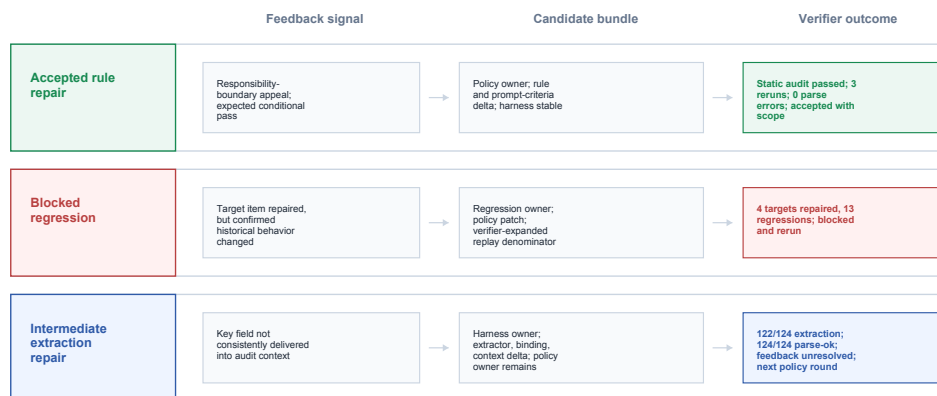
At the candidate level, the same 33 feedback instances generated 164 candidate attempts: 31 were accepted, released, or scoped; 131 were blocked by verifier gates; and 2 required maintenance-guidance intervention. We report candidate outcomes separately from feedback-owner counts because one feedback instance can trigger several owner hypotheses and candidate revisions. Owner categories were consolidated from feedback adjudication, candidate owner hypotheses, verifier residual owners, and release records. We keep the coarse production categories to show whether feedback touched policy, harness, or both; we do not expand them into a mutually exclusive defect taxonomy.

5.2 RQ2: From Feedback to Candidate, Verification, and Release

The evidence-extraction trace in Figure 4 shows how CoEvolve advances one feedback instance through the loop. The feedback signal showed that the system did not consistently use a key field. The first candidate treated this as an evidence-supply and harness failure, then updated extractor, binding, and context logic. The result stabilized the evidence interface: raw `report_selection` reached 117/124, gated extraction reached 122/124, and downstream parse-ok reached 124/124. Downstream accuracy remained 0.8387, indicating residual judgment failure.



■ **Figure 3** Candidate attempts and blocking reasons. Verifier gates blocked 131 of 164 candidate attempts. Fine-grained failures are grouped by verifier-gate function; 15/131 blocked candidates remained other/unclear after coding.



■ **Figure 4** Redacted end-to-end release traces used as in-paper governance evidence.

The next candidate shifted the remaining problem to a judgment-policy hypothesis, added a policy/prompt delta, passed runtime gates, and entered scoped release. The trace illustrates the loop: a local harness repair can be a necessary intermediate step; subsequent owner search handles remaining judgment errors.

Across replay, each CoEvolve candidate declared an owner hypothesis, policy/harness delta, expected impact, and release risk; target and regression cases were maintained by the verifier from feedback records, historically confirmed behavior, and sampled non-holdout items. Figures 3 and 4 show how candidate search is converted into governance evidence: release decisions, block records, residual owners, and scoped repairs.

Block categories were coded from verifier summaries and release records. Each candidate retains its original fine-grained block reason; Figure 3 merges related verifier-gate functions only for reporting, so recurring release risks do not appear as many small buckets.

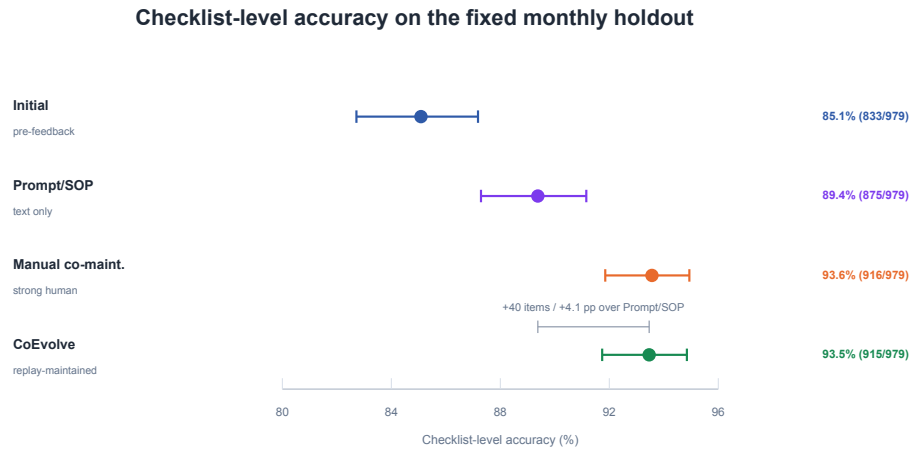
Before release decisions, the verifier checks two evidence families. Hard checks cover target repair, historical replay, rerun stability, output contracts, parse-ok status, evidence/context validity, and release scope. Semantic-governance checks cover sample memorization, metric chasing, policy consistency, rule overreach, and denominator integrity. Failures return a residual owner, block the candidate, narrow release scope, or escalate to expert confirmation.

The traces cover three release outcomes: scoped acceptance, regression block, and intermediate harness repair. The first shows how a policy-oriented repair enters scoped release; the second shows target repair blocked by historical regression; the third is intentionally unfinished: the harness repair stabilizes evidence delivery, but the original feedback remains unresolved and moves to a later policy round. They are included to make the release logic inspectable from the PDF: a candidate is useful when it clarifies an owner and produces stable evidence, even when it is not itself the final accuracy-improving patch. The fields also instantiate the minimum maintenance record: feedback signal, disputed record, owner hypothesis, policy delta, harness delta, target/regression evidence, verifier output, and release decision. Practitioners can reuse these trace fields as a template for candidate bundles, verifier summaries, and release decisions while replacing confidential materials with local identifiers.

5.3 RQ3: Audit Accuracy

On the same 979 checklist items, the CoEvolve replay state reached 93.5% accuracy (915/979), 4.1 percentage points above prompt/SOP maintenance at 89.4%. This is the primary quantitative contrast: both states follow the same ordered system-input stream and offline scoring controls, while the maintained surface expands from text policy to a verifiable policy-harness bundle. The initial state reached 85.1%. Manual co-maintenance reached 93.6% (916/979), one item more than CoEvolve. Given the overlapping Wilson intervals and only five system-level clusters, the manual and CoEvolve results are statistically indistinguishable and within sampling noise. Moreover, because visible manual artifacts entered replay, this similarity reflects reconstruction of a comparable release state and must not be interpreted as a method effect or evidence of substitution.

Item-level Wilson 95% intervals are 82.7–87.2% for the initial state, 87.3–91.2% for prompt/SOP tuning, 91.9–94.9% for manual co-maintenance, and 91.7–94.8% for CoEvolve. Across the five audited target systems, CoEvolve remains between 93.2% and 93.7% and is higher than prompt/SOP maintenance on all five; its system-level gap from manual co-maintenance is at most about 0.5 percentage points. With five target systems and fifteen reports, these estimates remain descriptive and do not support a cluster-robust ranking of CoEvolve and manual maintenance.



■ **Figure 5** Accuracy on the same 979-item monthly holdout benchmark. Error bars show item-level Wilson 95% intervals. The CoEvolve and manual states differ by one item and have overlapping intervals; the primary contrast is CoEvolve versus prompt/SOP maintenance (+4.1 percentage points).

■ **Table 4** Accuracy by audited target system.

System/month	Items	Initial	Prompt/SOP	Manual co-maint.	CoEvolve
A / 2025-12	201	94.1%	90.1%	93.6%	93.4%
B / 2026-01	190	89.5%	89.7%	93.5%	93.7%
C / 2026-02	204	87.0%	88.7%	93.7%	93.2%
D / 2026-03	191	80.5%	89.1%	93.6%	93.5%
E / 2026-04	193	77.9%	89.6%	93.6%	93.6%
Total	979	85.1%	89.4%	93.6%	93.5%

The monthly holdout also shows a descriptive decline in the initial state, from 94.1% in December 2025 to 77.9% in April 2026, while the maintained states are flatter across the five samples. With one system per month, the study cannot distinguish distribution shift, sampling variation, and maintenance tracking of drift; the pattern should not be read as a causal drift result.

■ **Table 5** Decision-error directions on the 979-item benchmark.

State	False pass	False fail	Other wrong	Wrong total
Initial system	66	52	28	146
Prompt/SOP tuning	42	43	19	104
Manual co-maintenance	28	31	4	63
CoEvolve replay-maintained state	27	32	5	64

False pass means the expert label is fail while the system passes the item, a high-risk direction in compliance auditing. False fail means the expert label is pass while the system fails the item, increasing review and appeal workload. Other wrong covers evidence, rationale, category, structured-field, or output-parsing errors. Using the 979 checklist items as denominator, CoEvolve’s false-pass item share is 2.8% (27/979), and its false-fail item share is 3.3% (32/979); prompt/SOP tuning is 4.3% and 4.4%, respectively. These are descriptive risk-direction metrics. We do not report business-loss weights, severity strata, or conditional error rates by true pass/fail base.

5.4 RQ4: Candidate Exploration and Verification Workload

Historical expert-developer end-to-end turnaround includes waiting, queueing, cross-role handoff, and release windows. We therefore report CoEvolve’s directly measured replay-loop workload and do not interpret it as end-to-end speedup over manual co-maintenance. Across 33 matched replay tasks, active maintenance time averaged 0.37 hours, with a median of 0.34 hours, an interquartile range of 0.25–0.44 hours, and a range of 0.16–1.30 hours. This time includes candidate generation, verification execution, and the recorded expert-confirmation step when present; it excludes waiting, queueing, release windows, long-term gate maintenance, and organizational communication cost. Each task generated 4.97 candidate attempts on average, with a median of 5, an interquartile range of 4–6, and a range of 1–11. These figures describe the exploration and verification load inside the maintenance loop.

These metrics show that CoEvolve operates as candidate search under gate verification. A single feedback task usually explores several owner hypotheses and policy-harness candidates. Every candidate entering gate or expert judgment creates a verification summary, block record, or release record.

6 Release Governance and Negative Evidence

Production release is governed by more than target-case repair. Candidates must also pass historical replay, parse-ok checks, context validity, static audit, semantic audit, rerun stability, and expert release decisions. In the 164 candidate attempts, most candidates did not ship directly. They became block records, residual-owner labels, scoped-release evidence, or revised candidates. Two attempts required maintenance-guidance interventions: restructuring a long scattered SOP into a tree form and stopping metric chasing in which feedback samples were being memorized into SOP text.

Blocked candidates included target repairs that broke historical cases, high-scoring candidates that were unstable across reruns, parse/schema/context failures, shortcut-risk traces, and harness repairs that needed later policy/prompt updates. Negative evidence is therefore part of the contribution: it shows how LLM-generated candidates are converted into governed maintenance records. CoEvolve extends release engineering by using the maintenance agent to externalize multiple owner hypotheses and policy-harness repair options, then relying on verification and expert governance to select releasable updates. The 131 blocked candidates, residual-owner labels, and scoped repairs show that candidate generation expands the inspectable maintenance space, while release governance decides which candidates enter released assets. The familiar software-engineering principles become a concrete feedback-to-release record flow for LLM auditing systems.

7 Related Work

Expert systems and knowledge engineering provide the conceptual base for this work: professional systems depend on maintained domain knowledge, expert explanation, and case boundaries [17, 7, 6, 24]. CoEvolve revisits that maintenance problem in LLM-based auditing, where part of the expert system is textual policy and part is executable harness.

Prompt engineering and automatic prompt optimization improve fixed-model systems through textual feedback, prompt search, or program-like prompt structures [20, 1, 13, 27]. DSPy and related programming frameworks compile LM pipelines against demos and metrics [10]. These lines of work usually optimize model calls with examples, metrics, or textual feedback. CoEvolve treats the release object as a production asset: policy, parser, retrieval, binding, context, output contract, regression denominator, and release record evolve together, while benchmark labels and scoring manifests remain outside maintenance.

Meta-Harness is the closest technical neighbor: it studies how to search better harnesses around model calls [12]. CoEvolve studies a complementary production-maintenance problem: how feedback is organized into releasable, blockable, and traceable policy-harness records under independent verification and expert release authority. The distinction is not only the maintained surface, but also optimizer visibility and release authority: the maintenance agent does not see benchmark labels or scoring manifests, and release is controlled by the verifier and expert confirmation. Its output is a governed maintenance record rather than a single optimized harness. Reflective adaptation work also explores feedback loops without retraining [2, 25].

Production ML and LLM systems require testing, monitoring, reproducibility controls, and release governance [21, 5, 19, 11, 18, 15, 23]. Recent SE work on benchmarks, regression tests, integration defects, and user-feedback issue detection motivates the use of fixed denominators, replay, and feedback-derived cases as maintained assets [8, 4, 14, 22, 9]. Automated compliance checking and document intelligence further show why policy and harness evolve together: professional audit often requires extracting obligations, fields, layouts, and evidence before a decision can be made [28, 3, 26, 16].

Here, “LLM auditing systems” refers to systems that use LLMs to perform compliance auditing; the audited objects are compliance materials processed by an LLM-enabled workflow. Foundation-model auditing is adjacent literature. Table 6 summarizes the nearest-neighbor boundary.

■ **Table 6** Positioning against nearest neighboring work.

Line of work	Maintained object	Feedback / metric visibility	Release authority
Prompt optimization	Prompt or instruction text	Task examples, scores, textual feedback	Optimizer or developer applies prompt
DSPy-style	LM modules and	Demos and metrics	Developer-controlled
pipelines	pipeline parameters	used for compilation	pipeline release
Meta-Harness-style	Harness around model	Harness-level metrics	Search result selected
search	calls	for search	by developer/evaluator
Production ML / LLMops	Models, data, tests, monitors, deployment process	Monitoring, test suites, operational signals	Release engineering and governance process
CoEvolve	Policy-harness bundle plus verification record	Redacted production feedback; benchmark labels and scoring manifest withheld from maintenance	Independent verifier and expert release confirmation

8 Discussion

8.1 From Prompt Tuning to Expert-Gated Maintenance

Prompt and SOP maintenance matter: prompt/SOP maintenance reaches 89.4% accuracy. The 4.1-percentage-point gap to the CoEvolve replay state is consistent with the practical value of allowing parsing, extraction, binding, context, output, and regression maintenance in addition to text edits. Because the workflow states differ in organization and intermediate inputs, the gap is not an isolated method effect. The evidence-extraction trace in Figure 4 illustrates the broader repair surface: unstable field extraction and binding prevented the auditor from reliably using the right evidence, and only after that interface was stabilized could remaining policy judgment be addressed.

8.2 Transfer Hypotheses

Table 7 separates the elements we expect to transfer from those that require local re-instantiation. These are design hypotheses derived from the observed workflow, not cross-domain validation claims.

■ **Table 7** Transfer hypotheses from the single-site experience.

Design element	Hypothesized portable core	Setting-specific instantiation
Maintenance record	Feedback, owner, deltas, checks, decision	Local identifiers, work-item system, confidentiality rules
Verifier gates	Target repair, replay, stability, contract checks	Audit semantics, risk thresholds, regression denominator
Policy-harness split	Separate domain meaning from executable delivery	Standards, report families, parsers, context fields
Expert release authority	Independent semantic and scope confirmation	Expert roles, escalation rules, release windows

8.3 Practitioner Playbook

Minimum adoption protocol. Record the feedback signal and affected item before repair; fix the expected audit judgment and rationale through domain-expert adjudication; assign an owner hypothesis before selecting a patch surface; version the candidate bundle with policy delta, harness delta, expected impact, and release risk; let an independent verifier maintain target cases, regression cases, and sampled non-holdout checks; run target repair, replay, parse-ok, schema/context, stability, and semantic-governance gates; record release, block, rerun, or scoped-release decisions together with residual owners.

Two operating rules were important. Accepted-with-scope is useful when target repair is valid, historical replay shows no high-risk regression, and residual owners are explicitly recorded. A high-risk false pass, a new report family, a weak regression set, or a new audit standard moves the workflow back to expert-led initialization before routine CoEvolve maintenance resumes. The minimum adoption protocol is to record feedback and expected judgment, form a candidate bundle, run independent verifier gates, preserve the verifier summary, obtain expert confirmation of release semantics, and record release/block/scope decisions in a work-item or versioning system. A team adopting the pattern also needs at least a stable output schema, a maintainable regression denominator, domain experts who can adjudicate audit semantics, and a verifier role separated from candidate generation.

8.4 Inference Cost and Deployment

CoEvolve runs outside the online audit path. The fixed production auditor handles production requests. The offline candidate-generation endpoint runs only when feedback, expert findings, runtime failures, or release audits trigger maintenance. This adds maintenance-time inference and regression-validation cost, with limited effect on user-facing audit latency.

9 Threats to Validity

Setting and evidence scope. This is a single-site industrial study in Chinese safety/compliance auditing. Language, report templates, standards, and organizational workflow may shape the result. Raw reports, feedback text, expert communications, credentials, and private deployment configuration remain inside the industrial project.

Benchmark, labels, and attribution. The 979-item benchmark covers five target systems and fifteen reports from the same production sample pool. It supports within-setting workflow-state behavior. Checklist labels were produced through two-person independent labeling and the organization’s audit-adjudication path. The two annotators worked without seeing each other’s labels; disagreements were discussed and final labels followed the organization’s adjudication rule before workflow scoring. The 33 feedback instances are not deduplicated, so owner audit counts maintenance signals rather than defect classes.

Workflow-state interpretation. The comparison uses completed maintenance states with different work organizations, intermediate signals, and human-effort profiles. Manual co-maintenance is the strongest historical human state in the deployment. CoEvolve is a retrospective replay state constructed from a matched system-input stream and visible historical manual artifacts, including versioned release assets, repair records, and verification summaries. It therefore shows reconstruction of a comparable release state, not independent reproduction of manual quality, substitution for manual co-maintenance, or a causal method effect. Comparable total human-hours were not collected. A prospective window in which no parallel manual release artifacts are visible is required to evaluate independent effectiveness and end-to-end maintenance cost.

Agent contribution. CoEvolve combines candidate generation, independent verification, historical replay, and expert confirmation. The study measures the governed workflow as a whole. The candidate counts and block records show how the offline candidate-generation endpoint externalized repair options that were then filtered by verification and expert governance. We do not isolate the maintenance agent’s net causal contribution; the better reading is that the LLM agent externalizes the maintenance space, while the verifier and experts decide which candidates enter released assets. Aggregate error totals also do not reveal which checklist items the manual and CoEvolve states disagree on; item-level disagreement analysis remains future work.

10 Conclusion

We presented CoEvolve, a feedback-driven policy-harness maintenance workflow for industrial LLM auditing systems operating without retraining. Across five months, the CoEvolve replay state reached 93.5%, 4.1 percentage points above prompt/SOP maintenance on the monthly holdout. Its 915/979 result is statistically indistinguishable from manual co-maintenance at 916/979; because replay used visible historical manual artifacts, the evidence supports reconstruction of a comparable release state, not substitution or a method effect. The workflow externalized multiple candidates per feedback task and used independent verification and expert confirmation to select releasable updates. The transferable lesson is to co-maintain policy and harness, keep denominator and regression as release conditions, and treat verifier blocks as governance evidence.

Data Availability Statement

Raw industrial materials cannot be released. To support review, the paper provides the runtime and loop mechanisms, agent contract, feedback and candidate counts, verifier-gate outcomes, redacted release traces, accuracy and risk-error results, replay workload, and practitioner record fields. The label recheck and verifier-gate rubric are reported

at aggregate and schema level. Unavailable materials include raw reports and feedback, expert communications and internal labels, production configurations and credentials, and scoring manifests or gate rules used as operational control assets. Their review substitutes are aggregate counts, redacted traces, the label/adjudication protocol, role-level model boundaries, gate-rubric fields, and maintenance-record templates; these substitutes contain no customer reports, internal labels, or production configuration.

Generative AI Declaration

Generative-AI tools assisted the camera-ready revision of the manuscript and the redrawing of figures. The authors directed and verified all revisions. No experimental data, labels, or new empirical evidence were generated using these tools, and the authors take full responsibility for the final content.

References

- 1 Eshaan Agarwal, Joykirat Singh, Vivek Dani, Raghav Magazine, Tanuja Ganu, and Akshay Nambi. Promptwizard: Task-aware prompt optimization framework. *arXiv preprint*, 2024. URL: <https://arxiv.org/abs/2405.18369>, arXiv:2405.18369.
- 2 Lakshya A. Agrawal, Shangyin Tan, Dilara Soyly, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint*, 2025. doi:10.48550/arXiv.2507.19457.
- 3 Orlando Amaral Cejas, Muhammad Ilyas Azeem, Sallam Abualhaija, and Lionel C. Briand. NLP-based automated compliance checking of data processing agreements against GDPR. *IEEE Transactions on Software Engineering*, 49(9):4282–4303, 2023. doi:10.1109/TSE.2023.3288901.
- 4 Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. Swe-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. *arXiv preprint*, 2025. doi:10.48550/arXiv.2505.20411.
- 5 Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D. Sculley. The ml test score: A rubric for ml production readiness and technical debt reduction. In *2017 IEEE International Conference on Big Data*, pages 1123–1132, 2017. doi:10.1109/BigData.2017.8258038.
- 6 William J. Clancey. Heuristic classification. *Artificial Intelligence*, 27(3):289–350, 1985. doi:10.1016/0004-3702(85)90016-5.
- 7 Edward A. Feigenbaum. The art of artificial intelligence: I. themes and case studies of knowledge engineering. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 1014–1029, 1977. URL: <https://www.ijcai.org/Proceedings/77-2/Papers/092.pdf>.
- 8 Xing Hu, Feifei Niu, Junkai Chen, Xin Zhou, Junwei Zhang, Junda He, Xin Xia, and David Lo. Assessing and advancing benchmarks for evaluating large language models in software engineering tasks. *arXiv preprint*, 2025. doi:10.48550/arXiv.2505.08903.
- 9 Shuyao Jiang, Jiazhen Gu, Wujie Zheng, Yangfan Zhou, and Michael R. Lyu. Can user feedback help issue detection? an empirical study on a one-billion-user online service system, 2025. doi:10.48550/arXiv.2508.00593.
- 10 Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint*, 2023. doi:10.48550/arXiv.2310.03714.
- 11 Dominik Kreuzberger, Niklas Kuehl, and Sebastian Hirschl. Machine learning operations (mlops): Overview, definition, and architecture. *IEEE Access*, 11:31866–31879, 2023. doi:10.1109/ACCESS.2023.3262138.

- 12 Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint*, 2026. doi:10.48550/arXiv.2603.28052.
- 13 Wenwu Li, Xiangfeng Wang, Wenhao Li, and Bo Jin. A survey of automatic prompt engineering: An optimization perspective. *arXiv preprint*, 2025. doi:10.48550/arXiv.2502.11560.
- 14 Jing Liu, Seongmin Lee, Eleonora Losiouk, and Marcel Böhme. Can llm generate regression tests for software commits? *arXiv preprint*, 2025. doi:10.48550/arXiv.2501.11086.
- 15 Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. Large language model-based agents for software engineering: A survey. *arXiv preprint*, 2024. doi:10.48550/arXiv.2409.02977.
- 16 Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, et al. A survey of context engineering for large language models. *arXiv preprint*, 2025. doi:10.48550/arXiv.2507.13334.
- 17 Michael Polanyi. *The Tacit Dimension*. Doubleday, Garden City, NY, 1966.
- 18 Hongzhou Rao, Yanjie Zhao, Xinyi Hou, Shenao Wang, and Haoyu Wang. Software engineering for large language models: Research status, challenges and the road ahead. *arXiv preprint*, 2025. doi:10.48550/arXiv.2506.23762.
- 19 Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of nlp models with checklist. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4902–4912, 2020. doi:10.18653/v1/2020.acl-main.442.
- 20 Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, Hyojung Han, Sevien Schulhoff, et al. The prompt report: A systematic survey of prompt engineering techniques. *arXiv preprint*, 2024. arXiv:2406.06608.
- 21 D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems*, volume 28, pages 2503–2511, 2015. URL: <https://proceedings.neurips.cc/paper/2015/hash/86df7dcfd896fc2674f757a2463eba-Abstract.html>.
- 22 Yuchen Shao, Yuheng Huang, Jiawei Shen, Lei Ma, Ting Su, and Chengcheng Wan. Are llms correctly integrated into software systems?, 2024. arXiv:2407.05138.
- 23 Mohammed Latif Siddiq, Arvin Islam-Gomes, Natalie Sekerak, and Joanna C. S. Santos. Large language models for software engineering: A reproducibility crisis. *arXiv preprint*, 2025. doi:10.48550/arXiv.2512.00651.
- 24 Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, 25(1–2):161–197, 1998. doi:10.1016/S0169-023X(97)00056-6.
- 25 Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. A survey on self-evolution of large language models. *arXiv preprint*, 2024. doi:10.48550/arXiv.2404.14387.
- 26 Weishi Wang, Hengchang Hu, Zhijie Zhang, Zhaochen Li, Hongxin Shao, and Daniel Dahlmeier. Document intelligence in the era of large language models: A survey. *arXiv preprint*, 2025. doi:10.48550/arXiv.2510.13366.
- 27 Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic “differentiation” via text. *arXiv preprint*, 2024. doi:10.48550/arXiv.2406.07496.
- 28 Jiansong Zhang and Nora M. El-Gohary. Semantic nlp-based information extraction from construction regulatory documents for automated compliance checking. *Journal of Computing in Civil Engineering*, 30(2):04015014, 2016. doi:10.1061/(ASCE)CP.1943-5487.0000346.