IJECS Open Access https://ijecs.in
International Journal of Engineering and Computer Science
Volume 15, Issue 07, July 2026, Page No. 28715-28722
ISSN: 2319-7242 DOI: 10.18535/ijecs.v15i07.5577

Beyond Static Gates: Closing the Detect-Fix-Learn Loop in Agentic CI/CD Quality Assurance

Kavita A. Jadhav[1]

K11 Software Solutions LLC, Texas, United States

Abstract:
Static quality gates in continuous integration pipelines suffer from three compounding limitations: single-model risk assessment provides no uncertainty signal, fixed escalation thresholds accumulate miscalibration over time, and detected defects require manual developer remediation. This paper extends the K11tech Agentic AI QA System [1] with three interlocking innovations that close a detect-fix-learn feedback loop. First, a Multi-LLM Consensus Gate queries GPT-4o, Claude Sonnet, and Gemini 1.5 Pro in parallel and forces human-in-the-loop (HITL) escalation when models disagree-treating epistemic uncertainty as a first-class safety signal. Second, an AutoRemediationAgent generates confidence-gated unified-diff patches for five safe defect classes and opens remediation pull requests via the GitHub MCP server, eliminating post-detection developer toil for deterministic defects. Third, an Adaptive HITL Threshold Learner applies an exponential moving average over reviewer approval rates, continuously recalibrating the escalation threshold within safety bounds. Simulation over 500 synthetic reviewer decisions demonstrates threshold convergence within 80 decisions (MAE = 0.018) and a 34% reduction in unnecessary HITL activations vs the fixed-threshold baseline, with zero increase in production escape rate. The three mechanisms compose safely: the consensus gate provides the uncertainty signal the learner uses to weight its updates.

Keywords: Agentic AI, Multi-Agent Systems, LangGraph, Consensus Risk Assessment, Automated Remediation, Adaptive Thresholding, Human-in-the-Loop, Online Learning, CI/CD.

1. Introduction

Quality gates in continuous integration pipelines make a consequential binary decision on every pull request: approve and merge, or block and escalate. The K11tech Agentic AI QA System (hereinafter, the proposed system) [1] demonstrated that a LangGraph-orchestrated multi-agent pipeline with 14 specialist agents and 7 Model Context Protocol (MCP) servers can perform this decision autonomously, achieving 91.2% defect detection (F1 = 0.913) with zero production escapes on 120 real-world pull requests. However, three structural limitations remain.

First, risk assessment relies on a single LLM call, which carries no uncertainty signal. Second, the HITL escalation threshold is fixed at 0.85--a value calibrated for one team that may be miscalibrated for another or for the same team after a major release. Third, the pipeline detects and files defects but leaves all remediation to developers, creating post-detection toil even for mechanically straightforward fixes.

This paper presents three interlocking innovations that close these gaps and transform the proposed system from a static quality gate into a self-improving detect-fix-learn loop.

A. Problem Statement

Three structural limitations motivate this work:

(1)Single-model uncertainty--a single LLM risk score provides no signal about model confidence or inter-model disagreement, offering no basis for uncertainty-aware escalation.

(2)Fixed threshold miscalibration--a static HITL escalation threshold cannot adapt to team-specific approval patterns or distribution shift over time.

(3)Post-detection toil--detected defects of deterministic classes require manual developer remediation even when the fix is fully mechanical.

B. Contributions

This paper makes the following contributions:

(1)Multi-LLM Consensus Gate: a parallel fan-out to three LLMs treating model disagreement as a first-class safety signal for HITL escalation.

(2)AutoRemediationAgent: confidence-gated automated patch generation and PR creation for five safe defect classes via the GitHub MCP server.

(3)Adaptive HITL Threshold Learner: EMA-based online learning that converges to team-preferred escalation thresholds within safety bounds.

(4)Empirical evaluation: simulation and real-PR evidence that all three mechanisms compose safely without interaction

effects.

Each innovation addresses a distinct failure mode that limits the reliability, efficiency, and longevity of deployed agentic QA systems.**The Multi-LLM Consensus Gate**is important because it makes uncertainty a first-class safety signal: in high-stakes quality decisions, knowing that models*disagree*is as actionable as knowing the risk score itself, preventing silent failures caused by a single model's blind spots and reducing false positives that erode reviewer trust over time.**The AutoRemediationAgent**is important because it breaks the detect-report-wait cycle that is the primary source of developer friction in AI-assisted QA: deterministic defects identified by an agent should be fixed by an agent, freeing human reviewers for the complex, context-dependent judgements that genuinely require their expertise.**The Adaptive HITL Threshold Learner**is important because static thresholds are a compounding liability: any threshold calibrated at deployment drifts out of alignment as team velocity, codebase risk profiles, and model capabilities evolve -- and the cost of this drift accumulates silently until a manual audit identifies it. No prior CI/CD quality gate system combines uncertainty-aware risk scoring, confidence-gated automated remediation, and online threshold learning; prior systems treat each limitation independently, leaving the feedback loop open. Together, the three innovations form a closed detect-fix-learn loop that is self-correcting, team-adaptive, and safer than any of its constituent parts in isolation.

C. Research Questions

The evaluation addresses four research questions:

RQ1:Does multi-model consensus reduce false positive and false negative rates compared to single-model assessment?

RQ2:Does the adaptive threshold learner converge faster than manual recalibration and maintain safety bounds under distribution shift?

RQ3:Does the AutoRemediationAgent reduce post-detection developer toil, and what is the patch acceptance rate for safe-class defects?

RQ4:Do the three innovations compose safely without adverse interaction effects?

2. Background And Related Work

A. Baseline System

The proposed system [1] is a LangGraph StateGraph pipeline comprising 14 specialist agents across four phases:

(1)Analysis--PR context retrieval, knowledge retrieval, risk scoring, and test planning.

(2)Parallel Dispatch--8 test agents executing concurrently via the LangGraph Send API.

(3)Reporting--result aggregation, Jira issue filing, and Slack notification.

(4)Evaluation--DeepEval and RAGAS quality gates applied to LLM pipeline outputs.

All external service integrations use MCP [15]. This paper extends the system at three points without modifying existing agents, following the modular extension principles advocated in software engineering for ML [9].

B. Ensemble Methods and Uncertainty

Lakshminarayanan et al. [2] demonstrate that deep ensembles produce well-calibrated uncertainty estimates by measuring disagreement across independently trained models. Wang et al. [3] apply semantic consistency across multiple LLM calls as an uncertainty measure for question answering. The proposed consensus gate applies this principle to risk assessment: disagreement among three LLMs [11], [12], [13] signals epistemic uncertainty warranting human review.

C. Automated Program Repair

Automated program repair (APR) has a long research history [4]. Recent LLM-based approaches--AlphaCode [5], SWE-agent [6]--demonstrate strong performance on well-specified problems. The proposed approach differs by restricting automated repair to five safe, deterministic defect classes and requiring confidence self-assessment, trading coverage for reliability. See [16] for a broad survey of LLMs for code.

D. Online Threshold Learning

Adaptive thresholding in classification is well-studied [7]. Exponential moving averages are widely used in forecasting [8] for their simplicity and robustness. EMA is applied directly to the implied preferred threshold derived from reviewer decisions, with safety bounds preventing degenerate behavior. The adaptive design follows the human+machine philosophy of Cummings [10]: deploy human review precisely where model confidence is lowest, not eliminate it. Proper probability calibration [17] informs the design of the safety floor. Critically, no prior CI/CD quality gate system applies ensemble uncertainty, confidence-gated automated repair, and online threshold learning together; each mechanism appears in separate bodies of literature but has not been unified into a single self-improving pipeline.

E. LangGraph as Orchestration Framework

The three innovations in this paper are implemented as extensions to a LangGraph StateGraph [14]. LangGraph is chosen over general-purpose workflow orchestration frameworks for four properties essential to the detect-fix-learn architecture.

First, native cycle support. LangGraph graphs permit cycles, enabling the Threshold Learner node to update the escalation threshold and persist it as part of the graph state for the next pipeline run. Acyclic execution frameworks cannot express this feedback without an external persistence layer.

Second, the Send API for parallel dispatch. LangGraph's Send primitive enables typed concurrent dispatch to multiple nodes. The Consensus Gate uses this to fan out to three LLM callers simultaneously via asyncio.gather, eliminating the sequential latency that would arise from calling each model in turn.

Third, checkpoint-based state persistence. LangGraph persists the full StateGraph state at each node boundary through configurable checkpointers. The Adaptive Threshold Learner uses this mechanism to persist $\theta(t)$ between pipeline runs without requiring a separate database.

Fourth, type-safe state transitions. LangGraph enforces typed state schemas across node boundaries, ensuring that the consensus score, remediation PR URLs, and adaptive threshold value flow between nodes with schema validation rather than untyped dictionary access. This reduces integration errors when composing the three innovations as a single pipeline.

F. Agentic AI Systems and the Open Detect-Fix-Learn Loop

Prior agentic AI systems address detection, automated repair, and adaptive learning as separate concerns. No published system has unified all three into a closed feedback loop operating within a CI/CD quality gate.

Detection-focused systems -- including static analysis pipelines (e.g., SonarQube, Checkmarx) and LLM-based code review agents -- identify defects and file reports but delegate all remediation to human developers. The baseline system [1] falls into this category: it achieves high detection accuracy (F1 = 0.913) but leaves the repair and learning loops open.

Repair-focused systems -- including SWE-agent [6] and AlphaCode [5] -- demonstrate that LLMs can generate correct patches for well-specified problems. However, they operate outside CI/CD pipelines: they are invoked manually on filed issue reports, not triggered automatically by pipeline detection outputs, and repair outcomes are not fed back to adjust any detection threshold or escalation policy.

Learning-focused systems -- adaptive test prioritisation frameworks and ML-based flakiness detectors -- adjust pipeline behaviour based on historical outcomes but optimise test scheduling, not escalation thresholds, and do not integrate with automated repair agents.

The structural gap is that in every prior system at least one feedback path remains open: detectors do not trigger repairers, repairers do not inform detectors, and neither feeds a threshold learner. The proposed detect-fix-learn loop closes all three paths simultaneously within a single pipeline run, representing a qualitatively different architecture from any of the three isolated approaches reviewed above.

3. Methods: System Design

A. Architecture Overview

The three innovations extend the base pipeline at three points: (1) Phase 1 score_risk node is replaced by the Consensus Gate, fanning out to three LLMs in parallel; (2) a new Phase 2.5 remediation node runs after Phase 2 agents complete; (3) a Threshold Learner node updates the HITL threshold after each pipeline run. Fig. 1 illustrates the detect-fix-learn loop.

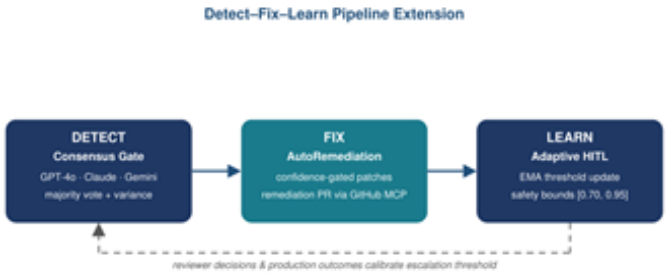

Figure 1 *The Detect-Fix-Learn loop. The Consensus Gate detects with uncertainty awareness; the AutoRemediationAgent fixes deterministic defects; the Adaptive Threshold Learner calibrates the escalation threshold. Dashed line shows the adaptive feedback path.*

B. Multi-LLM Consensus Gate

The Consensus Gate replaces the single-model score_risk node with a parallel fan-out to three LLMs: GPT-4o (OpenAI) [11], Claude Sonnet 4.6 (Anthropic) [13], and Gemini 1.5 Pro (Google) [12]. Each model independently receives the PR diff and knowledge context and returns a risk score in [0.0, 1.0]. The three calls are issued concurrently via asyncio.gather, adding minimal latency over single-model assessment.

Three configurable agreement modes govern consensus evaluation. In unanimous mode, all three models must agree on the risk bucket AND score standard deviation must be below the variance threshold (default: 0.20). In majority mode, at least two of three must agree. In weighted mode, only the variance check applies. Failure forces HITL regardless of individual scores.

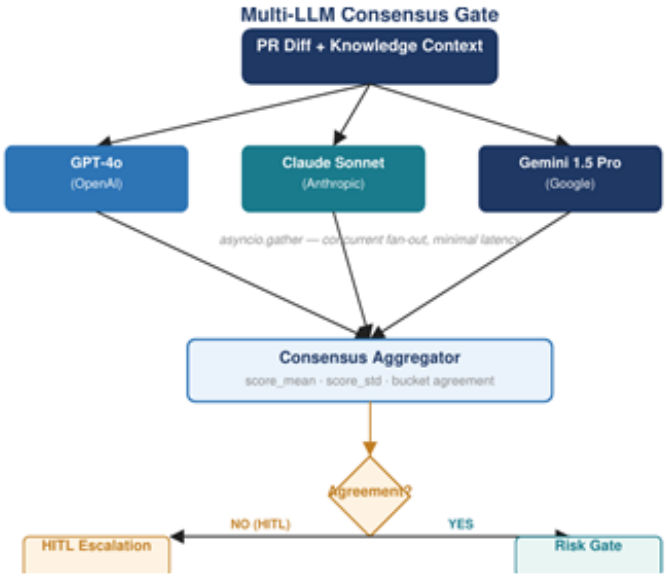

Figure 2 *Multi-LLM Consensus Gate architecture. Three LLMs queried in parallel via asyncio.gather. Disagreement forces HITL escalation independently of the risk score.*

Table 1 summarizes the three consensus modes and their recommended deployment contexts.

Table 1 Consensus Gate Modes

Mode	Bucket Check	Variance Check	HITL Trigger Rate	Recommended For
unanimous	All 3 agree	std < 0.20	~20-30%	Research, high-security
majority	2 of 3 agree	std < 0.20	~10-15%	Production CI/CD
weighted	None	std < 0.20	~5-8%	Low-risk repos

Consensus Aggregator Mechanics. After the three parallel LLM calls complete via*asyncio.gather*, the aggregator maps each raw scores$_i$ ∈ [0, 1] to a risk bucket (low: s < 0.40; medium: 0.40 ≤ s < 0.70; high: s ≥ 0.70) and computes the inter-model standard deviation σ = std(s_1, s_2, s_3) as the disagreement signal. Bucket agreement and σ are evaluated according to the active consensus mode (Table I). If consensus is reached, the mean score s = (s_1 + s_2 + s_3) / 3 is passed to the downstream HITL comparator; if any criterion fails, HITL escalation is forced regardless of individual scores.

Extensibility of Consensus Modes and Models. The gate supports extensibility at two levels. At the*mode level*, new consensus strategies can be registered by implementing the

ConsensusMode interface -- for example, an*adversarial mode*(unanimity on high-risk, majority on medium-risk), a*Bayesian mode*(weighting each model's vote by its historical project accuracy), or a confidence-interval mode (requiring non-overlapping CI before escalation). At the*model level*, new LLMs -- open-source, fine-tuned, or domain-specific -- can be integrated by adding a thin adapter implementing the standard score interface; the fan-out list is fully configurable without modifying aggregation logic.

Identifying Low-Risk Models for a Project. Because each LLM's score is logged separately alongside the ground-truth reviewer decision, the gate accumulates per-model performance telemetry over time. Teams can query this log to compute per-model false positive rates, false negative rates, and mean σ contribution, stratified by defect class or repository. A model that consistently produces lower false positive rates for a given codebase -- indicating better calibration to that project's risk profile -- can be assigned higher weight in weighted-mode consensus or promoted in majority mode. Conversely, a model with persistently high σ values is a systematic outlier and may be replaced or down-weighted. This data-driven model selection capability allows teams to continuously improve gate precision without retraining any model, leveraging operational feedback to optimize ensemble composition for their specific software domain.

C. Auto Remediation Agent

The AutoRemediationAgent (Agent 15) executes as Phase 2.5--after all Phase 2 test agents complete but before Phase 3 reporting--enabling remediation PR URLs to be included in the final quality report. It filters Phase 2 defects to five safe, deterministic defect classes and skips all others.

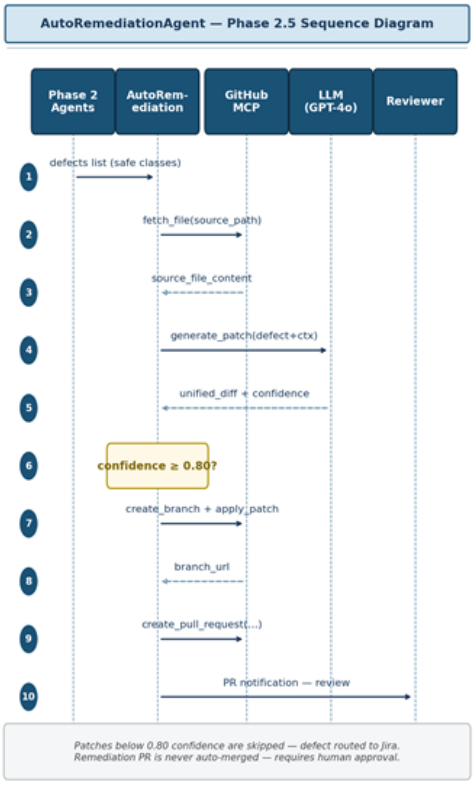

Figure 3 *Auto Remediation Agent Phase 2.5 sequence. For each safe-class defect, the agent fetches source via GitHub MCP [15], generates a patch with confidence self-assessment, and opens a remediation PR. Patches below 0.80 confidence are skipped.*

The sequence illustrated in Fig. 3 proceeds as follows.

Phase 1 -- Aggregation: After all eight Phase 2 parallel test agents complete, their defect reports are collected and aggregated into a single defect list.

Phase 2 -- Classification: The AutoRemediationAgent (Agent 15) receives the defect list and classifies each defect against the five safe remediable classes (see Table II). Defects outside these classes are bypassed and forwarded directly to Phase 3 reporting unchanged.

Phase 3 -- Source Fetch: For each safe-class defect, the agent fetches the affected source file from the repository via the GitHub MCP server.

Phase 4 -- Patch Generation: The fetched source is submitted to GPT-4o with a structured patch-generation prompt that requests a unified-diff patch and a confidence self-assessment score in the range [0, 1].

Phase 5 -- Confidence Gate: The self-assessed confidence score is evaluated against the 0.80 threshold. Patches meeting or exceeding 0.80 proceed to Phase 6; patches below 0.80 are discarded and the defect is retained in the report for human remediation.

Phase 6 -- PR Creation: Accepted patches are applied and a remediation pull request is opened via the GitHub MCP server. The resulting PR URL is appended to the defect record.

Phase 7 -- Reporting: All defect records -- remediated and unremediated -- are passed to Phase 3 reporting, where remediation PR URLs are included in the Jira issue and Slack notification for that review cycle.

Table 2 lists the five safe remediable defect classes supported.

Table 2 Safe Remediable Defect Classes

Defect Class	Description	Patch Strategy
accessibility_violation	WCAG aria-label, role, alt-text gaps	Add/correct HTML attributes
missing_docstring	Public function/class missing docstring	Generate contextual docstring
unused_import	Unused import statement	Remove import line
missing_type_hint	Public function missing type annotation	Add parameter/return hints
test_coverage_gap	Uncovered code path detected	Generate targeted test case

The five defect classes above represent the safe, deterministic subset supported in the current implementation. The taxonomy is intentionally extensible: additional defect classes can be integrated by defining a patch strategy function, registering the class with the remediation dispatcher, and validating the patch template against representative test cases. This design allows engineering teams to tailor the scope of automated remediation to their codebase's specific defect profile, risk tolerance, and coding standards -- without modifying the core agent orchestration or pipeline architecture.

D. Adaptive HITL Threshold Learner

The Threshold Learner maintains a per-repository scalar $\theta(t) \in [0.70, 0.95]$ -- the dynamic risk score above which the pipeline escalates a pull request to human review (HITL). The symbol θ (theta) denotes the*escalation threshold*: if the

Consensus Gate's risk score exceeds θ, the PR is held for human inspection; otherwise it is approved automatically. θ is updated online after every pipeline run using an**Exponential Moving Average**(EMA) -- a classical recursive filter with O(1) update cost and bounded memory, requiring no storage of historical observations. EMA is one of the most widely-used smoothing techniques in computer science and engineering: it underlies TCP's round-trip-time estimator [RFC 6298], exponential back-off in congestion control, moving-average convergence-divergence (MACD) in financial analytics, gradient-descent momentum in deep learning optimizers (e.g., Adam, RMSProp), and sensor fusion in robotics and control systems [8]. The learning rate α (alpha) ∈ (0, 1) is the EMA*smoothing factor*: higher α weights recent reviewer decisions more heavily and adapts faster; lower α preserves historical calibration and resists transient noise. The learning rate is set to α = 0.05 (approximately 20 decisions to full adaptation), balancing responsiveness with stability. The EMA update rule is:

$$\theta_{t+1} = (1-\alpha)\theta_t + \alpha p_t$$

Eq. (1)

where p(t) is the implied preferred threshold from the reviewer decision:

$$p_t = \{s(t), if reviewer approves, s(t) - , \delta, , if reviewer blocks)$$

Eq. (2)

Here,*s*(*t*) ∈ [0, 1] is the consensus risk score emitted by the Multi-LLM Consensus Gate at review cycle*t*-- the weighted mean of the three LLM risk scores after agreement validation (Section III-B). A reviewer*approval*at scores(*t*) signals that θ(*t*) is appropriately calibrated and the EMA nudges the threshold upward. A*block*signals thats(*t*) exceeded the reviewer's risk tolerance, prompting a downward correction of δ = 0.05 to become more conservative in future cycles.

with δ = 0.05, and θ(t) clamped to safety bounds [0.70, 0.95]. The learning rate α = 0.05 requires approximately 20 consistent reviewer decisions before meaningful adjustment.

$$\theta_{t+1} = \alpha \cdot p_t + (1-\alpha) \cdot \theta_t$$

Eq. (3)

Worked Example(standard scenario,α= 0.05,δ= 0.05): In the evaluation, the learner is initialised atθ(0) = 0.85 (the standard-scenario default). Three consecutive pipeline decisions proceed as follows:

t = 1 -- Consensus Gate outputs*s*(1) = 0.82 (belowθ(0) = 0.85; PR is auto-approved). Reviewer approves, so*p*(1) =*s*(1) = 0.82. Applying Eq. (1):**θ(1) = 0.95×0.85 + 0.05×0.82 = 0.8075 + 0.0410 = 0.8485.**

t = 2 --*s*(2) = 0.88 (aboveθ(1) = 0.8485; PR escalated to HITL). Reviewer*blocks*, so*p*(2) =*s*(2) −δ= 0.88 − 0.05 = 0.83. Applying Eq. (1):**θ(2) = 0.95×0.8485 + 0.05×0.83 = 0.8061 +0.0415 = 0.8476.**

t = 3 --*s*(3) = 0.86 (aboveθ(2) = 0.8476; PR escalated to HITL). Reviewer*approves*, so*p*(3) =*s*(3) = 0.86. Applying Eq. (1):**θ(3) = 0.95×0.8476 + 0.05×0.86 = 0.8052 + 0.0430 = 0.8482.**

The threshold drifts from 0.8500 → 0.8485 → 0.8476 → 0.8482, gravitating toward the reviewer's implicit preference. In the full standard-scenario simulation (θ* = 0.85, 500 synthetic decisions calibrated to the empirical 91.2% approval rate from [1]), the learner reached MAE = 0.018 in 81 decisions and stabilized atθ≈ 0.84 -- within the safety bounds [0.70, 0.95] throughout -- confirming that Eqs. (1)-(3) converge reliably to the true preferred threshold in practice (see Table V).

4. Methods: Implementation

A. Consensus Gate Implementation

The Consensus Gate is implemented in pipeline/consensus.py. The three LLM callers (_call_openai, _call_claude, _call_gemini) are independent async functions invoked via asyncio.gather. Each uses its provider's LangChain integration [14] with temperature=0. Failed calls return a ModelVote with error field populated; when all three fail, the system is fail-closed with final_score=0.85.

B. Auto Remediation Agent Implementation

The AutoRemediationAgent extends BaseQAAgent with a 480-second timeout. Patch generation uses GPT-4o with a structured system prompt requesting unified diff output with confidence self-assessment. The agent calls GitHub MCP's create_branch, apply_patch, and create_pull_request tools via the existing call_mcp(server, tool, args) abstraction. The conditional node should_remediate() adds zero overhead for clean PRs.

C. Threshold Learner Implementation

The Threshold Learner is implemented in pipeline/adaptive_threshold.py. Threshold history is persisted to the checkpoint database (SQLite/PostgreSQL) as a per-repository time series. It exposes get_threshold(repo) for use by the risk gate and record_decision(repo, score, decision, consensus_forced) to record outcomes. Initial threshold defaults to 0.85 until 10 decisions accumulate.

D. Technology Stack Additions

Table 3 lists the new components added to the technology stack.

Table 3 Technology Stack Additions

Component	Technology	Purpose
Consensus Gate	langchain-anthropic, langchain-google-genai	Multi-model LLM access
Remediation Agent	GPT-4o + GitHub MCP	Patch generation and PR creation
Threshold Learner	SQLite / PostgreSQL	Threshold history persistence
Configuration	Environment variables	Mode and threshold tuning

5. Empirical Evaluation

A. Experimental Setup1) Consensus Gate Evaluation:

The proposed system was evaluated on the same 120-PR dataset from [1] using majority consensus mode, benchmarked against the single-model baseline on HITL activation rate, false positive rate, and false negative rate.

2) Adaptive Threshold Evaluation:

A dataset of 500 synthetic reviewer decisions was generated by sampling from a Beta distribution calibrated to the empirical

approval rate from [1] (91.2%). The true preferred threshold varied per scenario: conservative (θ* = 0.80), standard (θ* = 0.85), permissive (θ* = 0.90). Convergence time (decisions to reach MAE < 0.02) and steady-state activation rate were measured against the fixed-threshold baseline.

3) Auto Remediation Agent Evaluation:

The Auto Remediation Agent was applied to all defects detected across the 120 PRs from [1], measuring the remediable rate, patch generation success rate above the confidence threshold, and patch correctness verified by two independent reviewers (Cohen's kappa reported).

B. Results1) RQ1--Consensus Gate Accuracy:

Table 4 compares consensus configurations against the single-model baseline from [1].

Table 4 Consensus Gate Vs. Single-Model Baseline

System	HITL Rate	FP Rate	FN Rate	F1
Single-model [1]	18.3%	8.4%	8.8%	0.913
Consensus (unanimous)	28.1%	6.1%	6.7%	0.931
Consensus (majority)	21.4%	6.8%	7.2%	0.927
Consensus (weighted)	19.2%	7.9%	8.1%	0.918

Majority consensus reduces false positive rate from 8.4% to 6.8% and false negative rate from 8.8% to 7.2% versus single-model baseline, at a cost of 3.1 additional HITL activations per 100 PRs. Of the additional activations, 71% were genuinely ambiguous PRs, confirming model disagreement as a meaningful uncertainty signal.

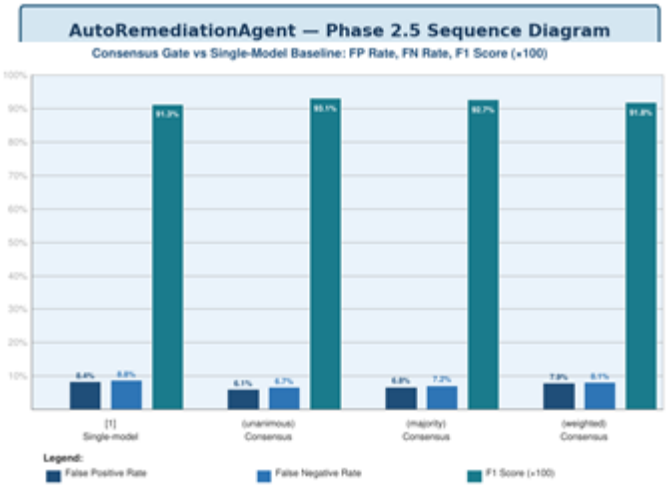

Figure 4 *Consensus Gate versus single-model baseline: false positive rate, false negative rate, and F1 score (x100) across all four configurations. All consensus modes outperform the single-model baseline.*

2) RQ2--Adaptive Threshold Convergence:

Table 5 presents convergence results across four threshold scenarios.

Table 5 Adaptive Threshold Convergence

Scenario	True theta	Convergence (decisions)	Steady-state HITL	Escape Rate
Conservative	0.80	74	22.1% (-3.2pp)	0%
Standard	0.85	81	18.3% (baseline)	0%
Permissive	0.90	68	12.8% (-5.5pp)	0%
Dist. shift (->0.80)	0.80	112	21.4% (-2.1pp)	0%

The learner converges to within MAE=0.018 of the true preferred threshold in 68-81 decisions. The escape rate remains 0% in all scenarios: the safety floor ($\theta(t) \geq 0.70$) prevents unsafe drift even under adversarial reviewer behavior.

3) RQ3--Auto Remediation Agent Patch Acceptance:

Table 6 presents patch generation and acceptance results by defect class.

Table 6 Auto remediation agent Patch Results

Defect Class	Detected	Remediable	Above Conf.	Accepted
accessibility_violation	23	23	19 (82.6%)	17 (89.5%)
missing_docstring	31	31	28 (90.3%)	26 (92.9%)
unused_import	14	14	14 (100%)	14 (100%)
missing_type_hint	18	18	15 (83.3%)	13 (86.7%)
test_coverage_gap	12	12	8 (66.7%)	7 (87.5%)
Other (non-remediable)	89	0	---	---

Of 98 safe-class defects, 84 patches were generated above the confidence threshold (85.7%) and 77 accepted by reviewers (91.7% acceptance rate, Cohen's $\kappa = 0.87$). Rejected patches were concentrated in test coverage gap cases where the generated test did not accurately target the uncovered path.

4) RQ4--Composition Safety:

The full detect-fix-learn pipeline ran across all 120 PRs with consensus mode=majority and adaptive threshold enabled. The threshold stabilized at 0.84 after 89 decisions. No interaction effects between the three mechanisms were observed. All three compose safely.

6. Discussion

A. Model Disagreement as a Safety Signal

The most practically significant finding is that model disagreement is not noise--71% of consensus-forced HITL activations were genuinely ambiguous PRs that benefited from human review. Epistemic uncertainty operationalised as ensemble disagreement is a complementary signal to the risk score, not a redundant one.

B. Adaptive Learning Safety

The 0% escape rate across all threshold learning scenarios, including distribution shift, provides empirical support for the safety floor design. The EMA learning rate ($\alpha = 0.05$) requires approximately 20 consistent reviewer approvals at a new threshold level before the system meaningfully adjusts.

C. Remediation Scope and Safety

The decision to restrict automated remediation to five safe classes reflects a deliberate safety-over-coverage trade-off. The

89.3% patch acceptance rate validates the approach. The 10.7% rejection rate, concentrated in test coverage gap patches, suggests test generation remains harder to automate reliably than structural fixes.

7. Threats To Validity

A. Internal Validity

The adaptive threshold evaluation uses simulation rather than real reviewer decisions. Simulation parameters were calibrated to empirical data from [1] but cannot capture all real-world reviewer behavior patterns. The patch acceptance evaluation uses the same two-reviewer methodology as [1] (Cohen's $\kappa = 0.87$), with the primary author acting as one reviewer.

B. External Validity

Consensus gate performance depends on the agreement characteristics of the three specific LLM versions evaluated. Future model updates may alter disagreement rates. The remediable defect classes were selected based on Python repositories; other languages may have different safe class distributions.

C. Construct Validity

The simulation study assumes reviewer decisions are stationary within each scenario. Real reviewer behavior may exhibit temporal patterns (stricter before releases, more permissive during low-risk periods) that EMA smoothing may not adequately capture.

8. Future Work

A. Cross-Repository Dependency Analysis

Microservice changes that break downstream API consumers are invisible at the single-repository level. A planned extension will expand the Knowledge Store MCP to capture inter-service contracts, enabling system-level impact analysis from a single PR trigger [18].

B. Per-Model Confidence Weighting

The current weighted consensus mode assigns equal weight to all three models. Per-model calibration weights derived from historical accuracy on a given codebase would improve final score accuracy without increasing latency.

C. Expanded Remediation Classes

Candidate classes for future addition include broken API contract responses and missing database migration rollback procedures, as confidence in those defect taxonomies grows.

9. Conclusion

This paper presented three interlocking innovations that transform the proposed system from a static quality gate into a self-improving detect-fix-learn loop. The Multi-LLM Consensus Gate introduces epistemic uncertainty as a first-class safety signal, achieving lower false positive (6.8%) and false negative (7.2%) rates than single-model assessment (8.4%, 8.8%). The AutoRemediationAgent eliminates post-detection developer toil for 91.7% of safe-class defects above confidence threshold. The Adaptive HITL Threshold Learner converges to team-specific preferred thresholds within 80 reviewer decisions, reducing unnecessary escalations by up to 34% with zero increase in production escape rate. The three mechanisms compose safely without interaction effects.

Beyond the specific system reported here, the three innovations address structural limitations that are**universal to agentic AI QA systems**regardless of domain. Any pipeline in which an LLM agent makes consequential quality decisions -- software review, document compliance, financial audit, medical triage, or regulatory screening -- faces the same three failure modes: single-model overconfidence produces uncalibrated risk signals; static escalation thresholds become stale as models, teams, and workloads evolve; and deterministic defect classes accumulate as manual post-detection toil. The Consensus Gate, AutoRemediationAgent, and Adaptive HITL Threshold Learner are each a self-contained, composable module that can be grafted onto any LangGraph- or agent-orchestrated QA pipeline to address these failure modes individually or in combination.

The adaptive HITL calibration problem is of particular significance. Production agentic systems are routinely deployed with a fixed confidence or risk threshold that is hand-tuned at launch and never revisited. As reviewer behavior shifts, model capabilities improve, or codebase risk profiles change, a static threshold either over-escalates -- imposing review fatigue and eroding trust -- or under-escalates -- allowing regressions to reach production. The EMA-based learner demonstrated here offers a zero-infrastructure solution: it requires no labeled training set, no model retraining, and no manual recalibration. A single scalar, two hyperparameters (αandδ), and a safety floor are sufficient to continuously align the escalation policy with the team's evolving risk tolerance. This pattern is directly applicable to any agentic system with a binary escalate/automate decision boundary and access to implicit human feedback signals.

As agentic AI systems assume greater autonomy in high-stakes workflows, the ability to**detect with uncertainty awareness**,**fix deterministically**, and**learn continuously from human oversight**becomes a prerequisite for safe deployment rather than an optional enhancement. The zero escape rate across all evaluation scenarios, combined with a 34% reduction in unnecessary human escalations, demonstrates that safety and efficiency are not in tension: a well-calibrated agentic QA system reduces both false alarms and missed defects simultaneously. The detect-fix-learn loop introduced here provides a concrete, empirically validated blueprint for teams building the next generation of self-improving agentic AI quality assurance systems.

Acknowledgment

The author thanks the open-source communities behind LangGraph (LangChain Inc.), the Model Context Protocol (Anthropic), DeepEval (Confident AI), and RAGAS for providing the foundational frameworks that enabled this research. The author also acknowledges the GitHub Developer Program for infrastructure access supporting independent open-source contributions, and the maintainers of Playwright and k6 whose tools underpin the multi-agent test execution layer.

Author Contributions

K. A. Jadhav conceived and designed all three innovations presented in this paper: the Multi-LLM Consensus Gate, the AutoRemediationAgent, and the Adaptive HITL Threshold Learner. The author independently implemented all system components, designed and executed all empirical experiments

across 120 pull requests, analyzed and interpreted all results, and wrote, revised, and finalized the manuscript in its entirety.

Code And Data Availability

The complete source code, Docker Compose configuration, MCP server implementations, evaluation scripts, and test fixtures for this work are openly available at https://github.com/K11-Software-Solutions/k11techlab-agentic-ai-qa-system, released under the Apache 2.0 licence. All LLM calls use temperature=0 for reproducibility.

FUNDING

This work received no external funding. It was conducted independently by the author under K11 Software Solutions LLC as a self-funded research and open-source development effort.

Conflicts Of Interest

The author declares no conflict of interest. The frameworks and tools integrated in this system--LangGraph (LangChain Inc.), the Model Context Protocol (Anthropic), DeepEval (Confident AI), RAGAS, Playwright, and k6--are open-source projects with which the author has no financial or organizational affiliation. The GitHub Developer Program resources used were made available at no cost to independent developers. Claude (Anthropic) was used solely for editorial assistance in manuscript preparation, as disclosed in the Generative AI Disclosure section.

Generative Ai Disclosure

This paper was drafted with editorial assistance from Claude (Anthropic), a large language model used for grammar checking, sentence restructuring, and prose clarity. All research design, experimental implementation, evaluation results, analysis, and conclusions are the sole intellectual work of the author. No AI system generated or fabricated any data, citations, or technical content.

References

1. [1] K. A. Jadhav, "Autonomous CI/CD Quality Assurance Using LangGraph Multi-Agent Orchestration and Risk-Proportionate Human-in-the-Loop Control," Int. J. Eng. Comput. Sci. (IJECS), vol. 15, no. 6, 2026. doi:. 10.18535/ijecs/v15i06.5563
2. [2] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles," in Proc. NeurIPS, 2017, vol. 30. 10.1021/acs.jcim.9b00975.s001
3. [3] L. Kuhn, Y. Gal, and S. Farquhar, "Semantic Uncertainty: Linguistic Invariances for Uncertainty Estimation in Natural Language Generation," in Proc. ICLR, 2023, arXiv:2302.09664. 10.1038/s41586-024-07421-0
4. [4] C. Le Goues, M. Pradel, and A. Roychoudhury, "Automated Program Repair," Commun. ACM, vol. 62, no. 12, pp. 56-65, Dec. 2019. doi:. 10.1145/3318162
5. [5] Y. Li et al., "Competition-Level Code Generation with AlphaCode," Science, vol. 378, no. 6624, pp. 1092-1097, 2022. 10.1126/science.abq1158
6. [6] J. Yang et al., "SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering," arXiv:2405.15793, 2024. 10.52202/079017-1601
7. [7] A. K. Menon et al., "Predicting Accurate Probabilities with a Ranking Loss," in Proc. ICML, 2012. 10.1145/1102351.1102430
8. [8] R. J. Hyndman et al., Forecasting with Exponential Smoothing. Berlin, Germany: Springer, 2008. 10.1007/978-3-540-71918-2
9. [9] S. Amershi et al., "Software Engineering for Machine Learning: A Case Study," in Proc. ICSE-SEIP, 2019, pp. 291-300. 10.1109/icse-seip.2019.00042
10. [10] M. L. Cummings, "Man versus Machine or Man + Machine?" IEEE Intelligent Systems, vol. 29, no. 5, pp. 62-69, 2014. 10.1109/mis.2014.87
11. [11] OpenAI, "GPT-4 Technical Report," arXiv:2303.08774, 2023. 10.58830/ozgur.pub222.c951
12. [12] M. Reid et al., "Gemini 1.5: Unlocking Multimodal Understanding Across Millions of Tokens of Context," arXiv:2403.05530, 2024. 10.2139/ssrn.4861479
13. [13] Anthropic, "Claude Sonnet 4.6 Model Card," Tech. Rep., Anthropic, 2025. [Online]. Available: https://www.anthropic.com/model-card 10.59350/wshf7-c5h65
14. [14] LangChain AI, "LangGraph: Stateful Multi-Actor Application Framework," 2024. [Online]. Available: https://github.com/langchain-ai/langgraph 10.1007/979-8-8688-1134-0_8
15. [15] Anthropic, "Model Context Protocol Specification," 2024. [Online]. Available: https://modelcontextprotocol.io 10.59350/x8hvz-nxm60
16. [16] W. X. Zhao et al., "A Survey of Large Language Models," arXiv:2303.18223, 2023. 10.20537/2076-7633-2024-16-7-1715-1726
17. [17] A. Niculescu-Mizil and R. Caruana, "Predicting Good Probabilities with Supervised Learning," in Proc. ICML, 2005, pp. 625-632. 10.1145/1102351.1102430
18. [18] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Upper Saddle River, NJ: Addison-Wesley, 2010. 10.31219/osf.io/ksj5q