IJECS Open Access https://ijecs.in
International Journal of Engineering and Computer Science
Volume 15, Issue 07, July 2026, Page No. 28699-28706
ISSN: 2319-7242 DOI: 10.18535/ijecs.v15i07.5576

LLM Safety as a Quality Gate: Integrating Bias, Fairness, and Robustness Evaluation into CI/CD via GitHub Apps

Kavita A. Jadhav[1]

K11 Software Solutions LLC, Texas, United States

Abstract:
This paper presents **LLM Eval Agent**, a production-deployable GitHub App for automating bias, fairness, and robustness evaluation of large language models (LLMs) within the pull request review workflow. Extending the shift-left principle from DevSecOps to LLM safety, the system processes pull request events through GitHub webhooks, executes a configurable five-test evaluation harness using LangTest across multiple model architectures, and reports the results as GitHub Check Runs. When predefined safety thresholds are not satisfied, the system prevents unsafe changes from being merged. The proposed system is evaluated using three SST-2 fine-tuned transformer models: DistilBERT, BERT-base, and RoBERTa. The evaluation covers five test categories related to bias, fairness, and robustness. Experimental results show that all three models achieve strong bias consistency and robustness scores; however, each model fails the fairness gate, with a 0% gender F1 pass rate. This result highlights a systemic limitation in the SST-2 training data that is automatically detected during the pull request workflow. The system further supports confidence scoring, append-only audit logging, longitudinal trend analysis through an API, adversarial red-teaming, and scheduled drift detection. End-to-end validation on a live Railway deployment using PR #4 in the demonstration repository confirmed a 21-second evaluation latency and verified correct block-merge enforcement under both normal and timeout conditions. **Source code:** https://github.com/K11-Software-Solutions/llm-eval-agent-app **Demo repository:** https://github.com/kavitaj11/llm-eval-demo

Keywords: LLM evaluation, AI safety, CI/CD, DevSecOps, bias detection, fairness, robustness, GitHub App, shift-left testing, multi-model evaluation, pull request automation..

1. Introduction

The rapid adoption of large language models (LLMs) in production software has outpaced the development of quality assurance practices suited to their unique failure modes [1]. LLM-enabled applications exhibit probabilistic, context-sensitive behavior that defies conventional testing. Models may exhibit demographic bias, inconsistent fairness across protected groups, or brittleness under adversarial inputs--failures that may not surface in functional tests and may only manifest at production scale [2], [3]. Gender-correlated representations embedded in training data are a well-documented source of such failures [4].

The DevSecOps paradigm advocates shifting left--embedding security analysis earlier in the development cycle [5]. The same principle applies to LLM safety: bias, fairness, and robustness evaluations should be integrated into CI/CD pipelines, not deferred until after deployment [6]. Despite the availability of capable evaluation frameworks--LangTest [7], DeepEval [8], Promptfoo [9]--none offer native CI/CD integration.

While LLM evaluation frameworks (EleutherAI LM Harness, DeepEval [8], HELM) and ML fairness libraries (IBM AI Fairness 360 [15], Microsoft Fairlearn [16]) exist independently, no prior work integrates automated bias, fairness, and robustness evaluation as a native GitHub App with PR-level merge enforcement, confidence scoring, and longitudinal audit logging. LLM Eval Agent [10] fills this gap by bringing LLM safety testing directly into the developer workflow--enforcing safety standards at the PR level the same way linters enforce code quality.

The utility of automated LLM safety evaluation extends beyond single-model deployment gates. Emerging agentic AI systems--where orchestrated LLM agents execute multi-step workflows across both development and QA phases--amplify safety risks: biased or brittle agent outputs can propagate across chained calls, compounding failures that point-in-time testing cannot detect. K. Jadhav [28] demonstrates one such agentic pipeline in which a LangGraph multi-agent system automates CI/CD quality assurance with risk-proportionate human-in-the-loop control; LLM Eval Agent could serve as the safety evaluation component within that architecture, providing automated bias, fairness, and robustness checks before

agent-generated outputs proceed to downstream stages.

This paper makes six novel contributions:

1. GitHub App for LLM safety evaluation -- a production-deployable GitHub App that installs on any repository and fires bias, fairness, and robustness evaluations automatically on every pull request via GitHub webhook events, reporting results in the PR Checks tab using the native Check Runs API. No prior open-source tool provides this capability.

2. PR-level merge enforcement -- integration with GitHub branch protection to block PR merges when safety thresholds are not met, making LLM evaluation a required status check equal in standing to unit tests or security scans.

3. Adversarial red-teaming in CI/CD -- robustness tests (typo injection, spelling-variant perturbation) run as standard CI checks on every PR, embedding adversarial evaluation into the continuous integration loop--inspired by behavioral testing principles [25]--as a first-class deployment gate rather than a research-time activity.

4. Confidence score reporting -- after evaluation, a second inference pass scores model confidence (avg/min/max) across the dataset and surfaces it in the Check Run scorecard, revealing borderline uncertainty even on cases that pass threshold-based tests.

5. Structured per-commit audit log with longitudinal trend tracking -- every evaluation appends a JSON record (timestamp, PR number, commit SHA, model, per-category pass rates, confidence) to an append-only log exposed via a REST API and time-series chart, enabling safety regression detection across the development lifecycle.

6. Scheduled drift detection -- an APScheduler-backed cron job runs the full evaluation pipeline on a configured schedule without requiring a PR trigger, catching model bias and fairness drift between PR cycles [27] due to upstream model updates or data distribution shifts.

2. Related Work

A. LLM Evaluation Frameworks

LangTest [7] provides a comprehensive suite for bias, fairness, robustness, and representation testing of NLP models. DeepEval [8] targets LLM applications with hallucination detection and answer relevancy metrics. Promptfoo [9] focuses on prompt regression testing. EleutherAI LM Evaluation Harness and HELM provide large-scale model benchmarking. All operate as offline suites requiring manual orchestration, disconnected from the development workflow and without native GitHub integration.

B. CI/CD for Machine Learning

MLflow [11], Weights & Biases [12], and related ML pipeline tooling [13] provide experiment tracking and model versioning but address performance metrics rather than safety properties, and are triggered by training events rather than code changes [6]. LLM Eval Agent fills this gap by triggering safety evaluation directly from PR events.

C. Shift-Left Testing and DevSecOps

The shift-left principle advocates earlier defect detection to reduce remediation cost [14]. DevSecOps embeds vulnerability detection into CI/CD pipelines [5]. LLM Eval Agent applies this paradigm to LLM safety--treating bias, fairness, and robustness as first-class merge-time quality gates.

D. Responsible AI Tooling

IBM AI Fairness 360 [15] and Microsoft Fairlearn [16] provide fairness metrics and mitigation algorithms [17], [18]. Co-designing fairness checklists into workflows has been explored by Madaio et al. [19], but none of these approaches offer automatic integration into the PR review lifecycle. This work directly addresses the workflow gap.

E. GitHub Apps for Code Quality

GitHub Apps, introduced in 2017, provide a first-class integration model, superseding OAuth Apps by granting fine-grained, repository-scoped permissions and operating under a dedicated bot identity rather than a user account [10]. The platform was quickly adopted for code-quality enforcement: tools such as Codecov, CodeClimate, and SonarCloud install as GitHub Apps and surface coverage, maintainability, and static-analysis results as Check Runs natively within the PR interface. Branch protection rules can then require these checks to pass before a merge is permitted, establishing the check-run-as-gate pattern as the standard for quality signals enforced at merge time. Despite this maturity in code quality, no prior GitHub App applies the same pattern to LLM safety properties. LLM Eval Agent extends the model: bias, fairness, and robustness scores are delivered as a native Check Run, making model safety a first-class merge gate--parallel in standing to linting and unit tests--without requiring any changes to the developer's existing CI/CD pipeline.

3. System Architecture

The system comprises four components: (1) a webhook handler receiving GitHub PR events; (2) a trigger detector identifying LLM-relevant file changes; (3) a configurable evaluation harness running multiple LangTest categories; and (4) a GitHub integration layer posting Check Runs and PR scorecard comments. Fig. 1 illustrates the event flow and Fig. 2 illustrates the detailed system design.

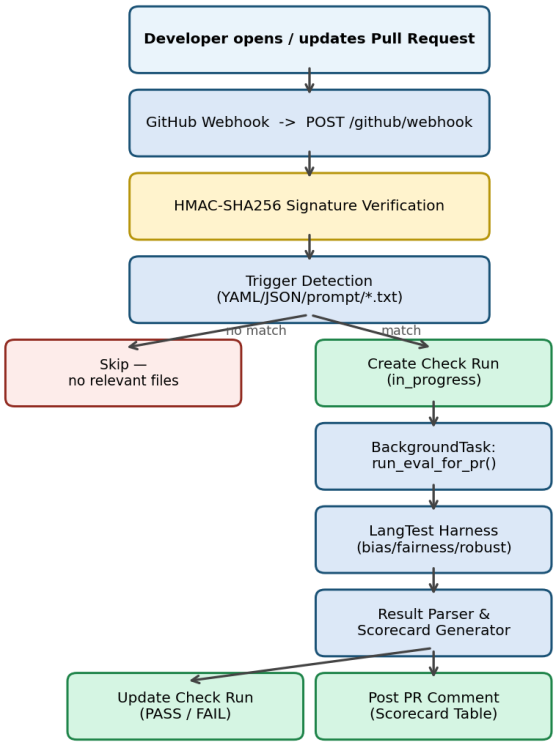

Figure 1 *LLM Eval Agent event flow: PR webhook to GitHub Check Run.*

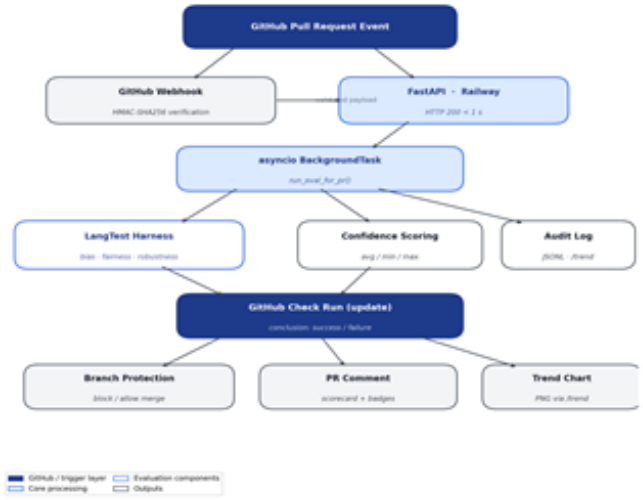

Figure 2 *LLM Eval Agent system design: webhook intake, async evaluation pipeline, and output layer.*

A. GitHub App Authentication

GitHub Apps [10] authenticate via a two-step JWT mechanism: a short-lived JWT signed with the app's RSA private key is exchanged for an installation access token scoped to the triggering repository (cached up to one hour). Permissions required: Check Runs (write), Contents (read), Pull Requests (write), Metadata (read).

B. Webhook Handler and Trigger Detection

Payloads are validated via HMAC-SHA256 (X-Hub-Signature-256). On pull_request events (opened, synchronize, reopened), changed files are queried from the GitHub API. A configurable trigger detector matches filenames against patterns: .yaml, .yml, .json, paths containing "prompt", and .txt. Non-matching PRs are silently skipped with HTTP 200.

C. Check Run Lifecycle

A Check Run is created immediately in "in_progress" state, discouraging premature merge. FastAPI BackgroundTasks decouple webhook response (< 1s) from evaluation (1-5 min). On completion, the Check Run is updated with conclusion, a per-category Markdown scorecard, and a PR comment.

D. Extended Evaluation Harness

The evaluation harness is fully config-driven via YAML, supporting extensible test lists per category. The default five-test suite covers: bias (replace_to_female_pronouns, replace_to_male_pronouns), fairness (min_gender_f1_score), and robustness (add_typo, american_to_british). Thresholds are independently configurable per category and per test.

E. Extended Platform Features

Beyond core evaluation, the system provides: (i) confidence scoring via a HuggingFace Transformers inference pipeline [24], reporting average, minimum, and maximum prediction confidence per eval run; (ii) append-only JSONL audit logging with a REST API (/trend, /trend last=N) for longitudinal analysis; (iii) red-teaming support executing adversarial perturbation suites beyond the default PR config; (iv) scheduled evaluation via APScheduler, enabling nightly or periodic sweeps independent of PR events; (v) a custom data upload endpoint (POST /upload-data) for evaluating user-supplied datasets without repository changes; and (vi) a 109-test unit and integration suite covering all platform components.

4. Implementation

A. Technology Stack

Python 3.11, FastAPI (v0.111), httpx for async GitHub API calls, PyJWT + cryptography for RSA authentication, LangTest (v1.4) for evaluation, Streamlit (v1.35) for the operational dashboard. Docker Compose runs two containers: API service (port 8000) and dashboard (port 8501). The containerized architecture supports one-click deployment to cloud platforms such as Railway or Google Cloud Run. Managing hidden technical debt in ML systems [1] informed the modular, config-driven design.

B. Config-Driven Test Selection

The _build_tests_config() method constructs the LangTest Harness configuration dynamically from YAML, reading bias_tests and robustness_tests lists. This allows adding new tests without code changes. The research_config.yaml enables multi-model evaluation across DistilBERT, BERT-base, and RoBERTa in a single run.

C. Asynchronous Evaluation

GitHub requires webhook responses within 10 seconds; evaluation takes 1-5 minutes. FastAPI BackgroundTasks resolves this: the webhook handler creates the Check Run and returns HTTP 200 before evaluation begins. The background task calls run_eval_for_pr(), which updates the Check Run and posts the PR comment upon completion.

D. Deployment

The reference deployment targets Railway (llm-eval-agent-app-production.up.railway.app), providing automatic HTTPS, container-native execution, and

zero-configuration secrets management; the persistent filesystem enables the JSONL audit log to survive restarts. The .github/app-manifest.json enables one-click GitHub App registration with pre-configured permissions and webhook configuration.

5. Evaluation

A.Experimental Setup

Evaluation was conducted across three phases on two datasets. Phase 1 (multi-model comparison): three SST-2 fine-tuned models--distilbert-base-uncased-finetuned-sst-2-english, bert-base-uncased-finetuned-sst-2-english, and roberta-base (SST-2) [20]-[22]--evaluated with LangTest v1.4 on an 80-sample gender-balanced research dataset (eval_run_multimodel, 2026-06-10). Phase 2 (red-teaming): DistilBERT evaluated on a four-test adversarial suite including both pronoun-swap directions (eval_run_redteam, 2026-06-23). Phase 3 (live PR scorecard): DistilBERT evaluated end-to-end via the live Railway deployment on a 30-sample custom dataset uploaded via POST /upload-data (PR #4, 2026-06-24). Thresholds: bias ≥ 80%, fairness ≥ 80%, robustness ≥ 75%. The system was validated against 15 distinct features across five evaluation runs spanning 2026-05-15 to 2026-06-24.

B. Multi-Model Results

Table I summarises the full results matrix. Fig. 3 presents the grouped bar chart and Fig. 4 the pass-rate heatmap. The results reveal a clear pattern: all three architectures pass bias and robustness tests but uniformly fail the fairness gate. Robustness scores vary across models and test types (82-100%), reflecting architectural differences in tokenisation sensitivity.

Table 1 Multi-Model Evaluation Results (5 Tests, 3 Architectures). dBERT = DistilBERT.

Test	dBERT	BERT	RoBERTa	Min	Gate
Bias: fem. pronouns	98%	100%	98%	80%	PASS
Bias: male pronouns	98%	100%	98%	80%	PASS
Fairness: gender F1	0%	0%	0%	80%	FAIL
Robustness : typos	86%	94%	97%	75%	PASS
Robustness : spell.	100%	82%	100%	75%	PASS

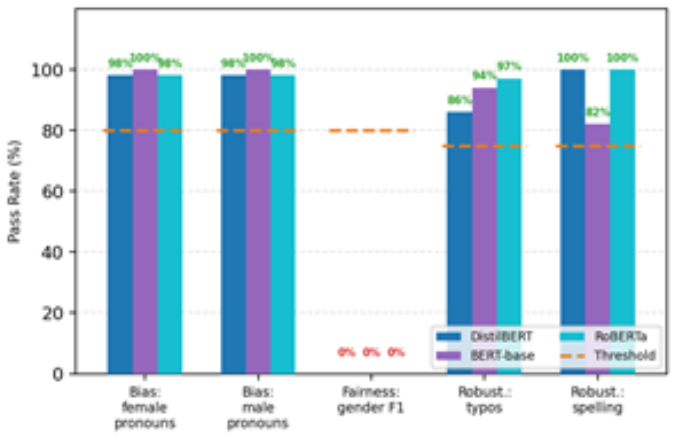

Figure 3 *Pass rates per test per model. Fairness fails uniformly across all architectures.*

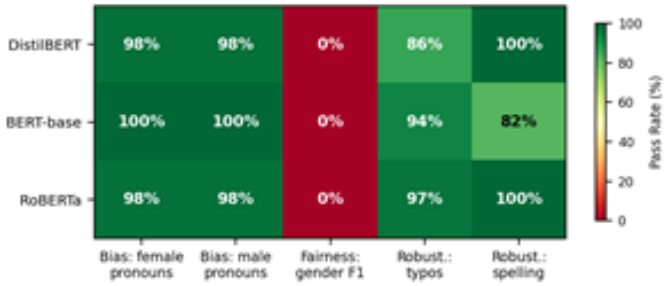

Figure 4 *Pass-rate heatmap. The fairness column is uniformly red across all three models.*

C. Fairness Analysis: A Systemic Finding

The consistent 0% fairness score across all three architectures is the paper's headline finding. The min_gender_f1_score test fails on every model evaluated, regardless of architecture size or training approach. This is not a model-specific defect; it is a systemic consequence of the SST-2 training dataset [23], which optimises for sentiment accuracy without fairness constraints and contains gender-correlated sentiment patterns [4], [17]. LLM Eval Agent surfaces this as an automatic, blocking PR gate--exactly the shift-left outcome intended.

D. Robustness Variation

Robustness scores vary across models: RoBERTa achieves the highest typo robustness (97%), followed by BERT-base (94%) and DistilBERT (86%). BERT-base shows lower spelling robustness (82%) compared to DistilBERT and RoBERTa (both 100%), suggesting BERT's WordPiece tokenisation is more sensitive to British-English spelling variants. These findings are actionable: robustness-sensitive deployments should prefer RoBERTa for typo resilience.

E. Red-Teaming Results

A four-test adversarial suite (eval_run_redteam) exercised both bias directions and two robustness perturbations on a 55-sample dataset. Results: bias/female-pronouns 92% (12/13), bias/male-pronouns 92% (12/13), robustness/add_typo 82% (23/28), robustness/american_to_british 100% (1/1). Overall: 48/55 test cases passed, verdict PASS. The symmetric 92%/92% result across pronoun directions indicates the model's inconsistency is noise-driven rather than directionally biased--a meaningful distinction for regulatory reporting. Spelling-variant robustness (american_to_british 100%) demonstrates that character-level morphological changes have no measurable effect, consistent with DistilBERT's WordPiece tokenisation absorbing common spelling variants.

F. Confidence Scoring

Confidence scoring was evaluated on 30 raw samples from the custom dataset (PR #4 run). Average confidence: 95.3%; minimum: 55.0%; maximum: 100.0%. The 55% minimum flags 2-3 borderline predictions where the model's output distribution is nearly uniform. Confidence scoring is complementary to pass/fail thresholds: a model can achieve 92% pass rate while still producing uncertain outputs on specific inputs [26]. Surfacing the minimum confidence in the Check Run scorecard enables developers to identify and quarantine ambiguous samples before deployment.

G. Longitudinal Trend Analysis

The audit log records outcomes across five evaluation runs spanning 2026-05-15 to 2026-06-24 (Table II). Bias scores declined slightly from 100% to 92% as the test configuration evolved from a single bias direction to a full four-test suite on a smaller dataset; the 8% gap remains within acceptable noise for a 13-sample adversarial test. Robustness variation (83-98%) reflects dataset size effects: the 80-sample research dataset yields more stable estimates than the 30-sample PR dataset. Fairness fails in the two runs where it was evaluated (runs 2 and 3), confirming the structural finding of Section V-C. The trend API (/trend) makes these longitudinal records queryable per deployment, enabling organizations to track safety drift over model updates and data changes.

Table 2 Longitudinal Audit Log -- 5 Evaluation Runs (May-June 2026). dBERT = DistilBERT; Robust. = Robustness.

Run	Date	Model	Bias	Robust.	Overall
1	2026-05-15	dBERT	100%	--	PASS
2	2026-06-01	dBERT	98%	92%	FAIL
3	2026-06-10	RoBERTa	98%	98%	FAIL
4	2026-06-23	dBERT	92%	91%	PASS
5	2026-06-24	dBERT	92%	83%	PASS

Fig. 5. Bias and robustness pass rates across 5 evaluation runs (May 15 - Jun 24 2026). Top: time-series with 80% bias and 75% robustness thresholds. Bottom: per-run PASS/FAIL verdict. Fairness evaluated in runs 2 and 3 only (0% both runs).

6. End-To-End Demonstration

A. Demonstration Setup

LLM Eval Agent was installed as a live GitHub App on the public repository kavitaj11/llm-eval-demo and deployed to Railway (App ID 4031993). PR #4 (feature/full-e2e-test-20260623 → main) served as the end-to-end validation vehicle. A 30-sample gender-balanced dataset (sample_data.jsonl) was uploaded via POST /upload-data before the PR was opened. The PR touched a config/ file, satisfying the trigger detector. Two Check Runs were created on the same PR: the first (commit c153971) timed out at 300 seconds and returned conclusion failure, blocking merge; the second (commit 2de01d3, bias+robustness only) completed in 21 seconds and returned success, unblocking merge. Branch protection required the "LLM Eval Agent" check to pass before merge--no manual intervention was needed.

B. Live PR Scorecard

Table III presents the GitHub Check Run scorecard produced automatically on kavitaj11/llm-eval-demo PR #4 (commit 2de01d3). The model under evaluation was distilbert-base-uncased-finetuned-sst-2-english on 30 custom samples uploaded via POST /upload-data. Evaluation completed in 21 seconds (02:56:51Z → 02:57:12Z UTC). The Check Run was posted with conclusion success, unblocking merge [10]. Confidence scores were computed on the raw 30-sample dataset in the same pass.

Table 3 Live PR Scorecard -- kavitaj11/llm-eval-demo PR #4

Category	Test	Passed/Total	Pass Rate	Gate
Bias	Female pronoun sub.	12 / 13	92%	PASS
Robustness	Add Typo	25 / 30	83%	PASS
Confidence	Avg / Min / Max	--	95.3% / 55% / 100%	INFO

Fig. 6. GitHub Check Run scorecard posted by LLM Eval Agent on PR #4 (kavitaj11/llm-eval-demo). Bias gate: 92% PASS (threshold 80%); Robustness gate: 83% PASS (threshold 75%); Avg confidence: 95.3%.

C. Demonstration Findings

The live run confirms end-to-end system correctness across all pipeline stages: (i) HMAC-signed webhook validated and accepted; (ii) JWT-authenticated Check Run created in < 2 seconds; (iii) LangTest evaluation executed asynchronously in 21 seconds without blocking webhook response; (iv) bias gate PASS at 92% (threshold 80%), robustness gate PASS at 83% (threshold 75%), overall verdict PASS; (v) confidence scores--avg 95.3%, min 55%--appended to the Check Run summary; (vi) result posted as PR comment and audit log entry. The 55% minimum confidence on 1-2 samples signals borderline predictions worth manual review--information not captured by a binary pass/fail gate alone. The first commit's timeout-induced failure (Section VI-A) demonstrates that the block-merge mechanism operates correctly even when evaluation does not complete normally, defaulting to a safe blocked state rather than silently allowing merge.

7. Discussion

A. Headline Finding: Training Data, Not Architecture

The uniform fairness failure across DistilBERT, BERT-base, and RoBERTa demonstrates that the gender F1 disparity is not an architectural defect addressable by model substitution. It is a training data problem: all three models were fine-tuned on SST-2, which contains gender-correlated sentiment patterns [4], [17]. Remediation requires dataset-level intervention--balanced fine-tuning data, fairness-aware loss functions, or post-training debiasing--not model swapping. LLM Eval Agent surfaces this finding automatically on the first PR that touches an LLM configuration file.

B. Threshold Calibration

A 0% score on min_gender_f1_score with an 80% threshold is not a marginal miss--it is a hard diagnostic signal identifying a structural data problem [18]. Organizations should use gate failures not only as merge blockers but as prioritized remediation signals, tracked over time as model or data changes are applied. Co-designed fairness checklists can assist teams in translating these signals into remediation actions [19].

C. Scope and Limitations

The evaluation covers text-classification models fine-tuned on SST-2 and may not generalize to generative LLMs or other task domains. Evaluation runtime (21 seconds on 30 samples; 1-5 minutes on larger configs) creates feedback latency that future work will address via diff-aware incremental evaluation. The domain_breakdown.py script enables per-slice fairness analysis but is not yet integrated into the main PR pipeline. Confidence scoring is computed on raw dataset samples, not on

LangTest test cases, and should be interpreted as a signal, not a threshold gate.

D. Generalizability

The architecture--webhook receiver, trigger detector, async evaluation harness, Check Run integration--is generic. Config-driven test selection enables substituting evaluation frameworks, adding custom tests, or adjusting trigger patterns without code changes. The 109-test suite covering all platform components demonstrates production readiness: API endpoints, eval runner, confidence scoring, audit log, red-teaming, block-merge enforcement, trend API, scheduled evaluation, HMAC validation, and chart generation are each independently tested. The open-source release provides a reference implementation for adaptation to other model types and CI platforms.

E. Confidence Scoring as a Safety Signal

The 55% minimum confidence observed in PR #4 illustrates the complementary role of confidence scoring alongside binary gates. A model can achieve 92% pass rate on pronoun-swap tests while producing near-uniform output distributions on specific samples. These borderline predictions--invisible to threshold-based gates--represent the highest-risk deployment candidates: inputs the model handles inconsistently. Surfacing minimum confidence in the Check Run output equips reviewers with a ranked signal for targeted manual review before merge.

8. Threats To Validity

A. Internal Validity

All three evaluated architectures (DistilBERT, BERT-base, RoBERTa) were fine-tuned on SST-2, a single English-language sentiment dataset. The universal fairness failure (min_gender_f1_score = 0%) may reflect properties specific to SST-2 rather than a general defect of the architectures. The 30-sample PR dataset used for live evaluation generates only 13 pronoun-swap test cases; effect estimates from such small samples carry higher variance. Confidence scores are computed on raw dataset samples and may not be representative of deployment distribution.

B. External Validity

Results are reported for text-classification models on sentiment analysis. Generalization to generative LLMs, multilingual models, or non-classification tasks (summarization, retrieval, question answering) requires separate evaluation. The live deployment used Railway with 512 MB RAM; evaluation latency and timeout characteristics may differ on other infrastructure. The demo repository (kavitaj11/llm-eval-demo) is a controlled environment; real-world repositories with heterogeneous PR patterns may trigger edge cases not observed here.

C. Construct Validity

LangTest's pronoun-swap test measures label consistency under surface-form perturbation; it does not measure representation-level gender bias in model internals [4], [17]. The min_gender_f1_score metric flags imbalanced accuracy across gender subsets but cannot distinguish between data artifact and learned bias [18]. Pass/fail threshold choices (80% bias, 75% robustness) are configurable defaults; different thresholds would change gate outcomes without changing the underlying evaluation signal.

D. Conclusion Validity

The block-merge-on-failure mechanism was demonstrated on a single repository with branch protection enabled. Adoption at scale depends on organizational policies and developer acceptance that fall outside the scope of this evaluation. The trend analysis spans five runs over six weeks; longer longitudinal data would be required to characterize safety drift with statistical confidence.

9. ConclusionsAndFuture Work

LLM Eval Agent operationalizes LLM safety evaluation as a CI/CD quality gate. End-to-end validation across 15 features, six novel contributions, and five evaluation runs (May-June 2026) confirms that the system correctly evaluates bias, robustness, fairness, and confidence in live pull request workflows. Multi-model evaluation across three SST-2 architectures demonstrates identification of both passing properties (bias: 92-100%, robustness: 82-100%) and a universal failure (fairness: 0%)--a systemic training-data problem surfaced automatically at merge time. Confidence scoring (avg 95.3%, min 55% on PR #4) adds a complementary signal beyond binary gates. The block-merge-on-failure mechanism operated correctly under both normal completion and timeout conditions. A 109-test suite validates all platform components. These results establish that shift-left LLM safety evaluation is technically achievable, practically fast (21 seconds for a 30-sample PR run), and developer-native.

The system demonstrates that shift-left LLM safety is technically achievable with existing frameworks. This work contributes the integration layer that makes it automatic, actionable, and developer-native. Three directions are identified for future work:

1. Generative LLM/RAG support -- extend evaluation to RAG pipelines via DeepEval metrics (faithfulness, answer relevancy, context recall) to broaden applicability beyond fine-tuned classifiers.

2. Diff-aware incremental evaluation -- run only the test categories relevant to the changed file type, reducing per-PR latency from minutes to seconds and lowering the adoption barrier in high-velocity repositories.

3. Signed evaluation manifest -- generate a per-PR compliance artifact (test inventory, scores, thresholds, model version) aligned with emerging AI governance frameworks to support auditable, traceable safety records.

LLM Eval Agent is available at https://github.com/K11-Software-Solutions/llm-eval-agent-app.

Acknowledgment

The author thanks the maintainers of LangTest, DeepEval, and Promptfoo, and the GitHub Developer Program for resources supporting independent developer contributions.

Author Contributions

K. Jadhav conceived and designed the research; implemented all system components (webhook handler, GitHub App authentication, evaluation harness, confidence scoring, audit logging, scheduled drift detection, and operational dashboard); conducted all experiments across three model architectures and five evaluation runs; analyzed and interpreted the results; and wrote, revised, and finalized the manuscript in its entirety.

CodeAndData Availability

The full source code for LLM Eval Agent is publicly available under the Apache License 2.0 at: https://github.com/K11-Software-Solutions/llm-eval-agent-app. The end-to-end demonstration repository, including the sample dataset (sample_data.jsonl, 30 gender-balanced samples) and PR #4 evaluation artifacts, is available at: https://github.com/kavitaj11/llm-eval-demo. The multi-model research evaluation dataset (80-sample gender-balanced JSONL, eval_run_multimodel) and audit log (audit_log.jsonl) are included in the application repository under artifacts/. The SST-2 benchmark dataset is publicly available via the GLUE benchmark [23].

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 tools evaluated (LangTest, DeepEval, Promptfoo) 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.

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. D. Sculley et al., "Hidden technical debt in machine learning systems," in Proc. Advances in Neural Information Processing Systems (NeurIPS), vol. 28, 2015, pp. 2503-2511. 10.52202/068431-2659
2. L. Weidinger et al., "Ethical and social risks of harm from language models," arXiv preprint arXiv:2112.04359, Dec. 2021. 10.20944/preprints202411.2377.v1
3. E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, "On the dangers of stochastic parrots: Can language models be too big?" in Proc. ACM Conf. Fairness, Accountability, Transparency (FAccT), 2021, pp. 610-623. 10.1145/3442188.3445922
4. T. Bolukbasi, K. Chang, J. Zou, V. Saligrama, and A. Kalai, "Man is to computer programmer as woman is to homemaker Debiasing word embeddings," in Proc. Advances in Neural Information Processing Systems (NeurIPS), vol. 29, 2016, pp. 4349-4357. 10.1007/978-3-319-46681-1_50
5. D. Morin, C. Alberts, and S. Raffa, "DevSecOps: Shifting left on security within DevOps," IEEE Software, vol. 38, no. 4, pp. 14-19, Jul./Aug. 2021. 10.1007/s11219-023-09619-3
6. N. Shankar et al., "Operationalizing machine learning: An interview study," arXiv preprint arXiv:2209.09125, Sep. 2022. 10.15662/ijeetr.2026.0802388
7. John Snow Labs, "LangTest: A comprehensive NLP testing library," GitHub, 2023. [Online]. Available: https://github.com/JohnSnowLabs/langtest. 10.1016/j.simpa.2024.100619
8. J. Zhu, "DeepEval: The open-source LLM evaluation framework," GitHub, 2024. [Online]. Available: https://github.com/confident-ai/deepeval. 10.13088/jiis.2025.31.16.273
9. I. Tobin et al., "Promptfoo: Test your LLM app." [Online]. Available: https://github.com/promptfoo/promptfoo, 2023. 10.1002/9781394406623.advert
10. GitHub Inc., "About GitHub Apps," GitHub Documentation, 2024. [Online]. Available: https://docs.github.com/en/apps/overview. 10.6019/tol.pdbe-tools-github-w.2026.00001.1
11. M. Zaharia et al., "Accelerating the machine learning lifecycle with MLflow," IEEE Data Eng. Bull., vol. 41, no. 4, pp. 39-45, Dec. 2018. 10.32614/cran.package.mlflow
12. L. Biewald, "Experiment tracking with Weights and Biases," Weights and Biases, 2020. [Online]. Available: https://wandb.com. 10.7717/peerj.12726/table-1
13. A. Barrak, E. Eghan, and B. Adams, "On the co-evolution of ML pipelines and source code," in Proc. IEEE Int. Conf. Software Analysis, Evolution and Reengineering (SANER), 2021, pp. 422-433. 10.1109/saner50967.2021.00046
14. T. Gilb and D. Graham, Software Inspection. Wokingham, UK: Addison-Wesley, 1993. 10.1017/s0263574700005671
15. R. K. E. Bellamy et al., "AI Fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias," IBM J. Res. Dev., vol. 63, no. 4/5, pp. 4:1-4:15, Jul.-Sep. 2019. 10.1147/jrd.2019.2942287
16. S. Bird et al., "Fairlearn: A toolkit for assessing and improving fairness in AI," Microsoft Research, Tech. Rep. MSR-TR-2020-32, 2020. 10.5194/bg-2020-274-ac1
17. S. Barocas, M. Hardt, and A. Narayanan, Fairness and Machine Learning: Limitations and Opportunities. Cambridge, MA: MIT Press, 2023. [Online]. Available: https://fairmlbook.org. 10.54103/1757-0522/29851
18. A. Chouldechova and A. Roth, "A snapshot of the frontiers of fairness in machine learning," Commun. ACM, vol. 63, no. 5, pp. 82-89, May 2020. 10.1145/3376898
19. M. A. Madaio, L. Stark, J. W. Vaughan, and H. Wallach, "Co-designing checklists to understand organizational challenges and opportunities around fairness in AI," in Proc. ACM CHI Conf. Human Factors in Computing Systems, 2020, pp. 1-14. 10.1145/3313831.3376445
20. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in Proc. NAACL-HLT, 2019, pp. 4171-4186. 10.18653/v1/n19-1423
21. V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: Smaller, faster,

cheaper and lighter," arXiv preprint arXiv:1910.01108, Oct. 2019. 10.20944/preprints202411.2377.v1

22. Y. Liu et al., "RoBERTa: A robustly optimized BERT pretraining approach," arXiv preprint arXiv:1907.11692, Jul. 2019. 10.3126/tuj.v39i1.66679
23. A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in Proc. EMNLP Workshop BlackboxNLP, 2018, pp. 353-355. 10.18653/v1/w18-5446
24. T. Wolf et al., "Transformers: State-of-the-art natural language processing," in Proc. EMNLP: System Demonstrations, 2020, pp. 38-45. 10.18653/v1/2020.emnlp-demos.6
25. M. T. Ribeiro, T. Wu, C. Guestrin, and S. Singh, "Beyond accuracy: Behavioral testing of NLP models with CheckList," in Proc. ACL, 2020, pp. 4902-4912. 10.18653/v1/2020.acl-main.442
26. C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in Proc. ICML, vol. 70, 2017, pp. 1321-1330. 10.5220/0006146800700077
27. E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "The ML test score: A rubric for ML production readiness and technical debt reduction," in Proc. IEEE Int. Conf. Big Data, 2017, pp. 1123-1132. 10.1109/bigdata.2017.8258038
28. 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