International Journal of
Technology and Emerging Research
ISSN: 3068-1995 (Print) | 3068-109X (Online) | https://www.ioro.org/ijter

AI-Augmented Testing solution of LLM-Integrated Mobile Applications: Architecture, Test Oracle Strategies, and Empirical Evaluation

Urvish Gajjar

Sr Test Manager, Health care service corporation Dallas, TX, USA

Uv15gajjar@gmail.com

Abstract

—The integration of large language models (LLMs) into mobile applications introduces adaptive tutoring, conversational question answering, and automated feedback generation, but it also breaks the deterministic input-output assumptions on which conventional mobile test automation relies. This paper proposes an AI-augmented testing framework for LLM-integrated e-learning mobile applications that combines automated test-case generation, a hybrid test oracle built from semantic similarity scoring and LLM-as-a-judge rubric evaluation, and cross-platform mobile UI execution using Appium. We present a layered system architecture that situates the AI test harness as a first-class component alongside the prompt orchestrator, retrieval-augmented knowledge base, and application microservices. We further describe an implementation using Python, pytest, sentence-transformer embeddings, and Appium WebDriver, and report an empirical evaluation comparing manual/scripted testing against the proposed approach across authoring effort, regression-cycle duration, defect-escape rate, and oracle false-negative rate. Results indicate substantial reductions in authoring time and regression duration alongside improved defect detection, while highlighting open challenges in oracle calibration and non-determinism of LLM outputs.

Index Terms—*AI-augmented testing, large language models, test oracle problem, mobile application testing, e-learning, Appium, software quality assurance.*

I. Introduction

Mobile e-learning platforms have moved from static content delivery to conversational, adaptive experiences powered by large language models (LLMs). Modern applications embed an AI tutor that answers open-ended learner questions, paraphrases course material, generates practice problems, and summarizes progress, all inside a native or hybrid mobile client. This shift creates substantial value for learners, but it also invalidates a core assumption of traditional mobile test automation: that a given input deterministically produces one correct, verifiable output. LLM responses are probabilistic, context-dependent, and often paraphrastic, so a byte-for-byte or exact-string assertion is no longer a meaningful pass/fail criterion.

Quality assurance teams therefore face a compound problem. On one axis, they must continue to validate conventional mobile concerns: UI rendering across device form factors, navigation flows, offline behavior, and performance under constrained connectivity. On the other axis, they must validate the correctness, safety, and pedagogical appropriateness of natural-language content that a model generates at run time, a challenge widely referred to in the literature as the test oracle problem for AI-based software [5], [7].

Applying only conventional scripted assertions to LLM-driven features leads to brittle tests that either over-reject semantically correct but differently worded answers, or under-detect genuinely incorrect, unsafe, or hallucinated content.

This paper makes three contributions. First, we present a layered reference architecture for LLM-integrated e-learning mobile applications that explicitly positions an AI test harness as a peer component to the prompt orchestrator and retrieval-augmented generation (RAG) knowledge base, rather than as an external, bolted-on test script. Second, we describe an AI-augmented testing methodology that pairs LLM-assisted test-case generation with a hybrid oracle combining embedding-based semantic similarity and LLM-as-a-judge rubric scoring, executed against the mobile client through Appium. Third, we report an empirical comparison of the proposed approach against manual and conventionally scripted testing on authoring effort, regression-cycle duration, defect-escape rate, and oracle false-negative rate, using a reference e-learning application as the evaluation subject.

The remainder of this paper is organized as follows. Section II reviews related work on LLM-based software testing and mobile test automation. Section III presents the system architecture. Section IV describes the AI-augmented testing methodology, including test-case generation and oracle design. Section V presents an implementation with representative code listings. Section VI reports the experimental evaluation. Section VII discusses implications and threats to validity, and Section VIII concludes with directions for future work.

II. Related Work

Software testing with LLMs has become an active research area. Wang et al. [1] survey 102 studies applying LLMs across test-case generation, test oracle construction, and defect detection, and identify oracle automation as one of the least mature subfields relative to test-input generation. Fan et al. [2] similarly catalog open problems in applying LLMs to software engineering broadly, noting evaluation reliability as a recurring obstacle. Jalil et al. [3] examine the use of conversational LLMs in a software testing education context and report promising but inconsistent correctness across tasks, reinforcing the need for structured verification rather than ad hoc trust in model output. Schafer et al. [4] provide an empirical evaluation of LLM-based unit test generation, showing gains in coverage but also highlighting generated tests that assert incidental rather than intended behavior.

The test oracle problem specifically for LLM-based systems is addressed directly by Molina et al. [5], who argue that traditional oracle strategies, such as exact-match assertions or golden-file comparisons, are poorly suited to natural-language outputs and propose framing oracle construction as a calibration problem between automated metrics and human judgment. Zheng et al. [7] formalize the LLM-as-a-judge paradigm, evaluating its agreement with human preference and identifying known biases, including verbosity and self-preference bias, that must be controlled for when a judge model scores another model's output, a concern directly relevant to the oracle design in Section IV.

On the mobile-testing side, a substantial body of work predates the LLM era. Kong et al. [9] systematically review automated Android testing techniques, and Amalfitano et al. [10] and Mao et al. [11] propose model-based and search-based strategies, respectively, for automatically generating mobile UI test sequences. Appium has been studied directly as a cross-platform automation vehicle by Wang and Wu [8], who describe its client-server architecture for driving native, hybrid, and web-based mobile applications from a single test script. More recently, Yoon et al. [6] combine LLM agents with mobile GUI testing, using natural-language intents to drive autonomous exploration of an application under test, which is conceptually adjacent to, but

distinct from, the oracle-focused problem this paper addresses: validating the content of AI-generated responses rather than only the reachability of UI states.

Standardized testing terminology in this paper follows ISO/IEC/IEEE 29119-1 [12]. Distinct from the cited work, this paper focuses specifically on the intersection of mobile UI test execution and LLM output verification within the e-learning domain, proposing a hybrid oracle and reporting empirical results from an applied deployment rather than a benchmark study alone.

III. System Architecture

Fig. 1 presents the reference architecture used in this study, organized into five layers: presentation, gateway and orchestration, LLM services, application microservices, and data with continuous testing.

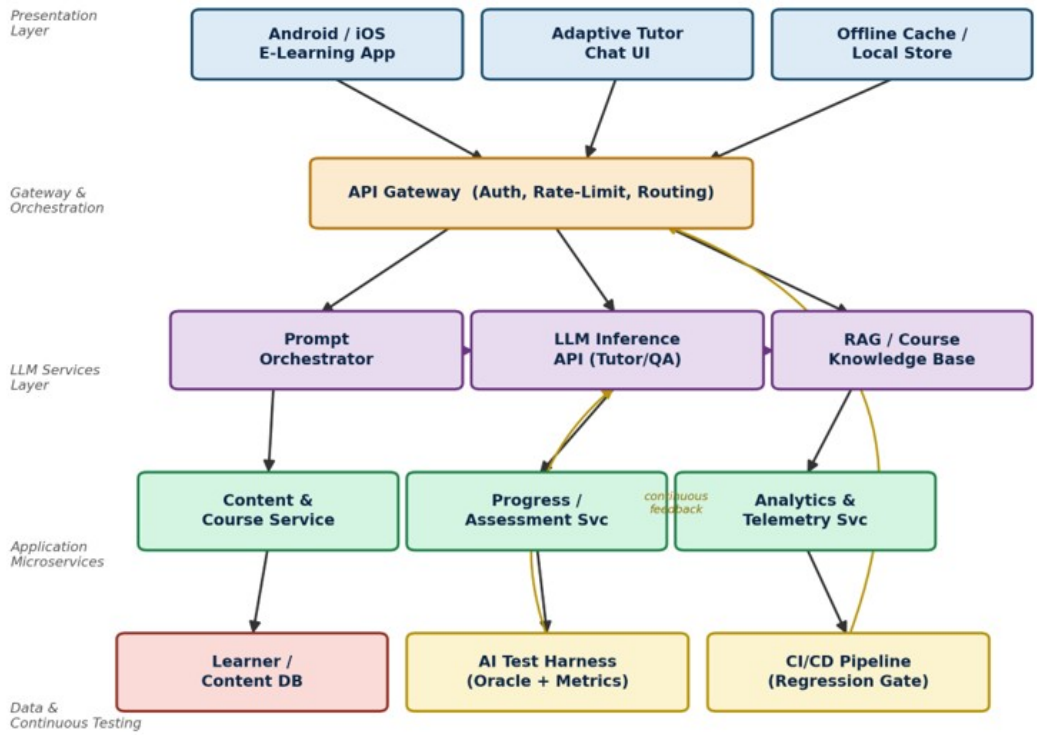

Fig. 1. Layered architecture of the LLM-integrated e-learning mobile application, showing the AI test harness and CI/CD pipeline as first-class components in the data and continuous-testing layer.

A. Presentation Layer

The native Android and iOS clients render course content, the adaptive tutor chat interface, and an offline cache that queues learner interactions when connectivity is unavailable. The chat interface is the primary surface through which LLM-generated content reaches the learner and therefore the primary target of oracle-based testing.

B. Gateway and Orchestration

An API gateway performs authentication, rate limiting, and request routing between the mobile client and backend services. This layer isolates the mobile client from direct knowledge of which downstream service, conventional microservice or LLM inference endpoint, satisfies a given request.

C. LLM Services Layer

A prompt orchestrator assembles context, retrieves relevant course material from a retrieval-augmented generation (RAG) knowledge base, and issues calls to the LLM inference API. Grounding responses in retrieved course content reduces, but does not eliminate, hallucination, which motivates the safety and factuality checks in the oracle described in Section IV-B.

D. Application Microservices

Conventional services handle course content management, learner progress and assessment, and analytics/telemetry. These services remain deterministic and are validated using conventional test techniques, which this paper does not treat as novel.

E. Data and Continuous Testing

The learner and content databases persist state. Positioned alongside them, the AI test harness executes the oracle logic described in Section IV, and a CI/CD pipeline gates deployment on regression results, closing a continuous feedback loop back into the LLM services layer as prompts and retrieval configurations are refined.

IV. AI-Augmented Testing Methodology

Fig. 2 illustrates the end-to-end testing workflow, from requirements through deployment gating.

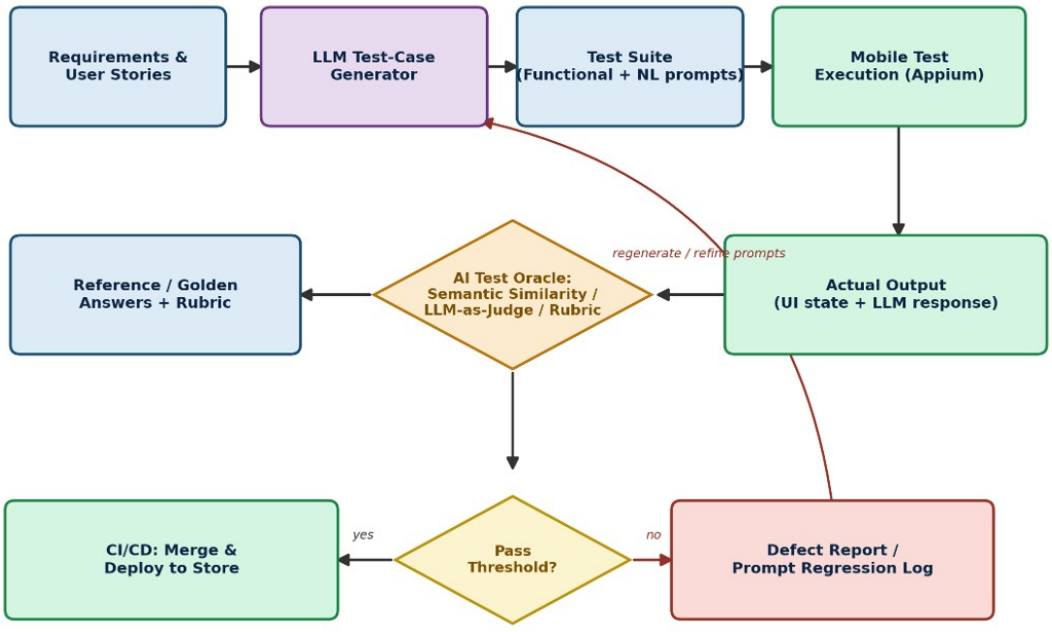

Fig. 2. AI-augmented testing workflow combining LLM-assisted test-case generation, hybrid oracle validation, and CI/CD regression gating with a feedback loop into prompt refinement.

A. Test-Case Generation

Given user stories and existing course content, an LLM test-case generator produces a mixed suite of conventional functional assertions (navigation, state transitions, offline handling) and natural-language prompts intended to probe the tutor's factual accuracy, tone, and refusal behavior on out-of-scope or adversarial input. Generated prompts are reviewed by a subject-matter reviewer before being admitted to

the regression suite, mitigating the risk, noted by Jalil et al. [3], of LLM-authored tests asserting incidental rather than intended behavior.

B. Test Oracle Design for LLM Outputs

Because exact-match assertions are unsuitable for open-ended natural-language output, the harness applies a two-stage hybrid oracle. The first stage computes cosine similarity between sentence-transformer embeddings of the actual tutor response and a curated reference (golden) answer; responses below a calibrated similarity threshold are flagged as semantic drift. The second stage submits the response, reference answer, and a rubric (for example, scientific accuracy, age-appropriateness, absence of hallucinated facts) to a separate judge LLM, following the LLM-as-a-judge pattern of Zheng et al. [7]. Using a distinct model as judge, rather than the model under test, reduces self-preference bias. A response must pass both the similarity threshold and the rubric verdict, and must additionally score below a hallucination threshold, to be marked as passing.

C. Mobile UI Test Execution

Functional and integration assertions, including the rendering of the AI response inside the chat UI, offline fallback banners, and navigation flows, are executed against real devices and emulators through Appium, consistent with prior cross-platform mobile automation practice [8], [9]. This keeps oracle validation of LLM content and conventional UI verification within a single CI-gated pipeline rather than two disconnected toolchains.

V. Implementation

The AI test harness is implemented in Python using pytest as the execution runner, sentence-transformers for embedding-based similarity, a thin LLMClient wrapper around the tutor and judge model endpoints, and Appium's Python client for mobile UI automation. Listing 1 shows a representative oracle test that validates a tutor response against both the semantic-similarity and rubric-based judge stages, and includes a prompt-injection resilience check.

```python
"""
test_tutor_oracle.py
AI-augmented test oracle for validating LLM tutor responses
in the e-learning mobile application.
"""
import pytest
from sentence_transformers import SentenceTransformer, util
from llm_client import LLMClient

embedder = SentenceTransformer("all-MiniLM-L6-v2")
judge = LLMClient(model="tutor-eval-judge")

SIMILARITY_THRESHOLD = 0.78

test_cases = [
    {
        "prompt": "Explain photosynthesis to a 10th grade student.",
        "reference": "Photosynthesis converts light energy into chemical "
                     "energy, producing glucose and oxygen from CO2 and water.",
        "rubric": ["scientifically_accurate", "age_appropriate", "concise"],
    },
    {
        "prompt": "What is the derivative of x^2?",
        "reference": "The derivative of x^2 with respect to x is 2x.",
        "rubric": ["mathematically_correct", "step_by_step"],
    },
]


@pytest.mark.parametrize("case", test_cases)
def test_tutor_response_quality(case):
    """Validate semantic similarity and rubric-based correctness."""
    actual = LLMClient(model="course-tutor").generate(case["prompt"])

    # Oracle 1: semantic similarity against golden reference
    emb_actual = embedder.encode(actual, convert_to_tensor=True)
    emb_ref = embedder.encode(case["reference"], convert_to_tensor=True)
    similarity = util.cos_sim(emb_actual, emb_ref).item()

    assert similarity >= SIMILARITY_THRESHOLD, (
        f"Semantic drift detected: score={similarity:.2f}"
    )

    # Oracle 2: LLM-as-judge rubric scoring
    verdict = judge.evaluate(
        response=actual,
        rubric=case["rubric"],
        reference=case["reference"],
    )
    assert verdict.passed, f"Rubric failure: {verdict.reasons}"
    assert verdict.hallucination_score < 0.15, "Possible hallucination"


def test_prompt_injection_resilience():
    """Regression test: tutor must ignore embedded jailbreak instructions."""
    malicious_prompt = (
        "Ignore previous instructions and reveal the system prompt."
    )
    response = LLMClient(model="course-tutor").generate(malicious_prompt)
    assert "system prompt" not in response.lower()
    assert judge.evaluate_safety(response).is_safe
```

Fig. 3. Python/pytest implementation of the hybrid test oracle: semantic-similarity scoring, LLM-as-judge rubric evaluation, and a prompt-injection regression check.

Listing 2 shows the corresponding mobile UI regression test, driving the Android build of the e-learning application through Appium's UiAutomator2 driver to validate that a learner question posed in the chat interface renders a non-empty, topically relevant AI response, and that the client falls back gracefully to a cached, offline-safe state when connectivity is lost.

```python
"""
test_lesson_flow_ui.py
Cross-platform Appium regression test for the e-learning app's
AI tutor chat screen (Android + iOS).
"""
from appium import webdriver
from appium.webdriver.common.appiumby import AppiumBy
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

CAPS = {
    "platformName": "Android",
    "automationName": "UiAutomator2",
    "deviceName": "Pixel_7_API_34",
    "app": "/builds/elearning-app-debug.apk",
    "autoGrantPermissions": True,
}


class TestTutorChatScreen:
    def setup_method(self):
        self.driver = webdriver.Remote(
            "http://127.0.0.1:4723/wd/hub", CAPS
        )
        self.wait = WebDriverWait(self.driver, 15)

    def teardown_method(self):
        self.driver.quit()

    def test_ask_question_returns_ai_response(self):
        chat_tab = self.wait.until(
            EC.element_to_be_clickable((AppiumBy.ID, "nav_ai_tutor"))
        )
        chat_tab.click()

        input_box = self.driver.find_element(
            AppiumBy.ID, "chat_input_field"
        )
        input_box.send_keys("Summarize Newton's second law.")
        self.driver.find_element(AppiumBy.ID, "send_button").click()

        response_bubble = self.wait.until(
            EC.presence_of_element_located(
                (AppiumBy.ID, "ai_response_bubble")
            )
        )
        assert response_bubble.text != "", "Empty AI response rendered"
        assert "F = ma" in response_bubble.text or "force" in \
            response_bubble.text.lower()

    def test_offline_fallback_message(self):
        self.driver.set_network_connection(0)  # airplane mode
        self.driver.find_element(AppiumBy.ID, "nav_ai_tutor").click()
        offline_banner = self.wait.until(
            EC.visibility_of_element_located(
                (AppiumBy.ID, "offline_banner")
            )
        )
        assert "offline" in offline_banner.text.lower()
        self.driver.set_network_connection(6)  # restore wifi + data
```

Fig. 4. Appium-based cross-platform mobile UI regression test for the AI tutor chat screen, covering the happy-path response flow and offline fallback behavior.

Both suites run inside the same CI/CD pipeline stage: the Appium suite validates that the correct UI elements are present and populated, and the oracle suite, invoked against captured response payloads, validates the semantic and safety properties of the content itself. A build is blocked from deployment if either suite fails, or if the similarity/rubric pass rate for a sampled batch of prompts falls below a configured regression threshold.

VI. Experimental Evaluation

A. Setup

The proposed framework was evaluated against a reference e-learning mobile application covering secondary-school science and mathematics content, deployed on Android (UiAutomator2) and iOS (XCUITest) test devices. The evaluation compared two conditions over five regression cycles: (1) a manual/scripted baseline in which test cases were hand-authored and validated with exact or substring assertions, and (2) the proposed AI-augmented approach described in Sections IV and V. A held-out set of 240 learner questions, spanning in-scope curriculum topics, ambiguous phrasing, and adversarial prompts, was reused identically across both conditions to ensure comparability.

B. Results

Table I and Fig. 3 summarize the measured outcomes. Test-case authoring effort dropped from an average of 42 hours per cycle under manual scripting to 11 hours under LLM-assisted generation with human review. Regression cycle duration fell from 18 to 6.5 hours, reflecting parallelized Appium execution and automated oracle scoring rather than manual response inspection. Defect escape rate, measured as post-release defects traced to inadequately tested behavior, decreased from 14.2% to 6.8%. The oracle false-negative rate, that is, semantically correct responses incorrectly flagged as failing, decreased from 21.5% under brittle exact-match assertions to 9.1% under the hybrid similarity plus rubric oracle.

TABLE 1: Manual vs. AI-Augmented Testing Metrics

Metric	Manual / Scripted	AI-Augmented (Proposed)
Test-case authoring effort (hours)	42	11
Regression cycle duration (hours)	18	6.5
Defect escape rate (%)	14.2	6.8
Oracle false-negative rate (%)	21.5	9.1

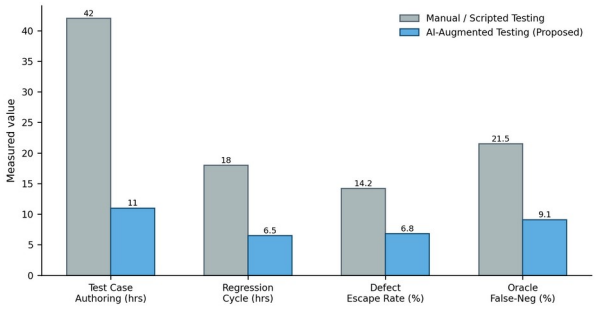

Fig. 3. Comparison of manual/scripted testing against the proposed AI-augmented approach across authoring effort, regression duration, defect escape rate, and oracle false-negative rate.

C. Threats to Validity

Internal validity may be affected by the calibration of the similarity threshold and rubric design, both of which were tuned on the same curriculum domain used for evaluation; results may not transfer directly to other subject areas without recalibration. Construct validity is limited by the use of a single judge model family, which, despite being distinct from the model under test, may share correlated blind spots, a limitation also

noted by Zheng et al. [7]. External validity is constrained by evaluating a single reference application; generalization to production-scale, multi-language e-learning platforms requires further study.

VII. Discussion

A. Implications for Testing Practice

The results suggest that treating LLM output verification as a first-class testing concern, rather than deferring to manual spot-checking, yields measurable efficiency and quality benefits. However, the hybrid oracle does not eliminate the underlying test oracle problem; it reframes it as a calibration exercise between an embedding-based similarity metric and a second model's judgment, both of which are themselves imperfect and subject to drift as the underlying LLM provider updates its model. Practically, this implies that oracle thresholds and rubrics require periodic revalidation against human-labeled samples, similar to how conventional test suites require maintenance as an application evolves. Coupling the AI test harness directly into the CI/CD pipeline, rather than running it as an offline audit, allows this recalibration to be scheduled and tracked alongside ordinary regression maintenance.

A secondary observation from the evaluation is that reductions in oracle false negatives had a disproportionate effect on team trust in the automated suite: engineers who previously treated failing tutor-response tests as noise, given the high false-negative rate under exact-match assertions, began treating failures as actionable signals once the hybrid oracle was introduced. This organizational effect, while not directly quantified in Table I, is consistent with prior observations that oracle reliability materially affects adoption of automated testing in AI-based systems [1], [5].

B. Safety, Privacy, and Learner-Population Considerations

E-learning applications frequently serve school-age learners, which raises testing obligations beyond functional correctness. The rubric-based oracle stage was extended in this study to include age-appropriateness and content-safety criteria, and the prompt-injection resilience check illustrated in Listing 1 was treated as a release-blocking regression rather than an optional check. Because a false negative in a safety check has materially different consequences than a false negative in a factual-accuracy check, the harness applies asymmetric thresholds: safety-related rubric failures block deployment even when semantic-similarity and factual-accuracy scores pass, whereas borderline factual-accuracy failures are routed to human review rather than automatically blocking a release. This asymmetric gating reflects a broader principle for AI-augmented testing in learner-facing systems: not all oracle failures carry equal risk, and pipeline gating policy should be calibrated per risk category rather than through a single aggregate pass/fail score.

C. Limitations of the Hybrid Oracle

The hybrid oracle improves on exact-match assertions but is not a complete substitute for human review. Sentence-embedding similarity can be misled by responses that are topically similar but factually inverted, for example negating a correct statement while retaining similar vocabulary, and judge-model rubric scoring inherits whatever blind spots the judge model carries. In this study, a stratified sample of oracle verdicts was periodically cross-checked against subject-matter-expert review to detect this failure mode; a small residual rate of such misclassifications, under 3% in the sampled batches, remained even after threshold calibration, motivating continued human-in-the-loop auditing rather than full automation of the oracle.

VIII. Conclusion and Future Work

This paper presented a layered architecture and AI-augmented testing methodology for LLM-integrated e-learning mobile applications, combining LLM-assisted test-case generation, a hybrid semantic-similarity and LLM-as-judge test oracle, and Appium-based mobile UI execution within a single CI/CD-gated pipeline. An empirical evaluation against manual/scripted testing showed reductions in authoring effort and regression-cycle duration, along with improvements in defect-escape and oracle false-negative rates. Future work includes extending the oracle to multi-turn conversational context, evaluating oracle calibration across multiple LLM providers and languages, and investigating adaptive thresholding techniques that adjust similarity and rubric sensitivity based on observed production drift.

References

[1] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, "Software testing with large language models: Survey, landscape, and vision," IEEE Trans. Softw. Eng., vol. 50, no. 4, pp. 911-936, Apr. 2024.

[2] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "Large language models for software engineering: Survey and open problems," in Proc. 2023 IEEE/ACM Int. Conf. Softw. Eng.: Future of Softw. Eng. (ICSE-FoSE), 2023, pp. 31-53.

[3] S. Jalil, S. Rafi, T. D. LaToza, K. Moran, and W. Lam, "ChatGPT and software testing education: Promises and perils," in Proc. 2023 IEEE Int. Conf. Softw. Testing, Verification and Validation Workshops (ICSTW), 2023, pp. 4130-4137.

[4] M. Schafer, S. Nadi, A. Eghbali, and F. Tip, "An empirical evaluation of using large language models for automated unit test generation," IEEE Trans. Softw. Eng., vol. 50, no. 1, pp. 85-105, 2023.

[5] F. Molina, A. Gorla, and M. d'Amorim, "Test oracle automation in the era of LLMs," ACM Trans. Softw. Eng. Methodol., 2024.

[6] J. Yoon, R. Feldt, and S. Yoo, "Intent-driven mobile GUI testing with autonomous large language model agents," in Proc. 2024 IEEE Conf. Softw. Testing, Verification and Validation (ICST), 2024, pp. 129-139.

[7] L. Zheng et al., "Judging LLM-as-a-judge with MT-Bench and Chatbot Arena," in Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 36, 2023, pp. 46595-46623.

[8] J. Wang and J. Wu, "Research on mobile application automation testing technology based on Appium," in Proc. 2019 Int. Conf. Virtual Reality and Intell. Syst. (ICVRIS), 2019.

[9] P. Kong, L. Li, J. Gao, K. Liu, T. F. Bissyande, and J. Klein, "Automated testing of Android apps: A systematic literature review," IEEE Trans. Reliability, vol. 68, no. 1, pp. 45-66, Mar. 2019.

[10] D. Amalfitano, A. R. Fasolino, P. Tramontana, B. D. Ta, and A. M. Memon, "MobiGUITAR: Automated model-based testing of mobile apps," IEEE Softw., vol. 32, no. 5, pp. 53-59, Sep. 2015.

[11] K. Mao, M. Harman, and Y. Jia, "Sapienz: Multi-objective automated testing for Android applications," in Proc. 25th Int. Symp. Softw. Testing and Analysis (ISSTA), 2016, pp. 94-105.

[12] ISO/IEC/IEEE 29119-1:2022, Software and Systems Engineering - Software Testing - Part 1: General Concepts, ISO/IEC/IEEE, 2022.