LLM-as-a-Judge for Scalable Test Coverage Evaluation: Accuracy, Operational Reliability, and Cost

Donghao Huang[1,2], **Shila Chew**[3], **Anna Dutkiewicz**[2], **Zhaoxia Wang**[1]

[1]School of Computing and Information Systems, Singapore Management University, Singapore
[2]Research and Development, Mastercard, Arlington, VA, USA
[3]Research and Development, Mastercard, Singapore, Singapore
dh.huang.2023@smu.edu.sg, shila.chew@mastercard.com, anna.dutkiewicz@mastercard.com, zxwang@smu.edu.sg

Abstract

Assessing software test coverage at scale remains a bottleneck in QA pipelines. We present *LLM-as-a-Judge (LAJ)*, a production-ready, rubric-driven framework for evaluating Gherkin acceptance tests with structured JSON outputs. Across 20 model configurations (GPT-4, GPT-5 with varying reasoning effort, and open-weight models) on 100 expert-annotated scripts over 5 runs (500 evaluations), we provide the first comprehensive analysis spanning accuracy, operational reliability, and cost. We introduce the *Evaluation Completion Rate* (ECR@1) to quantify first-attempt success, revealing reliability from 85.4% to 100.0% with material cost implications via retries. Results show that smaller models can outperform larger ones: GPT-4o Mini attains the best accuracy (6.07 MAAE), high reliability (96.6% ECR@1), and low cost ($1.01 per 1K), yielding a 78× cost reduction vs. GPT-5 (high reasoning) while improving accuracy. Reasoning effort is model-family dependent: GPT-5 benefits from increased reasoning (with predictable accuracy–cost trade-offs), whereas open-weight models degrade across all dimensions as reasoning increases. Overall, cost spans 175× ($0.45–$78.96 per 1K). We release the dataset, framework, and code to support reproducibility and deployment.

Introduction

Automated assessment of software test coverage presents a persistent challenge in quality assurance: manual evaluation by domain experts is accurate but prohibitively expensive and slow at the scales required for modern continuous integration/continuous deployment (CI/CD) pipelines. Traditional static analysis tools provide quantitative metrics (e.g., line coverage, branch coverage) but cannot assess the *semantic completeness* of test scenarios—whether tests adequately capture business requirements, edge cases, and error conditions (Karpurapu et al. 2024).

Recent advances in large language models (LLMs) have demonstrated capabilities in code understanding, semantic analysis, and structured evaluation tasks (Alshahwan et al. 2024). This raises a compelling question: can LLMs serve as reliable, scalable, and cost-effective judges for evaluating test coverage quality? While the concept of "LLM-as-a-Judge" has gained traction in various evaluation contexts (Zheng et al. 2023), systematic investigation of its application to software testing—particularly with respect to accuracy, cost, and *operational reliability*—remains limited.

In this work, we introduce a comprehensive **LLM-as-a-Judge (LAJ)** framework specifically designed for evaluating Gherkin-style acceptance test coverage[1]. Our framework employs rubric-driven assessment aligned with industry best practices and HTTP method-specific testing guidelines. Beyond measuring assessment accuracy, we introduce novel metrics that capture operational reliability: the *Evaluation Completion Rate (ECR@1)*, which measures the percentage of evaluations that succeed on the first attempt, and *adjusted cost metrics* that account for retry overhead in production deployments.

Our main contributions are as follows:

- We introduce *LLM-as-a-Judge (LAJ)*, a production-ready framework for automated test coverage evaluation that uses rubric-driven assessment.
- We propose reliability-aware metrics—*Evaluation Completion Rate (ECR@1)* and reliability-adjusted cost—that capture deployment-critical robustness beyond accuracy alone.
- We conduct a comprehensive evaluation of 20 model configurations across 500 runs, systematically quantifying accuracy–reliability–cost trade-offs spanning a 175× range ($0.45–$78.96 per 1K evaluations).
- We provide evidence-based deployment recommendations, identifying GPT-4o Mini as production-optimal (6.07 MAAE, 96.6% ECR@1, $1.01/1K), and publicly release our dataset, evaluation framework, and implementation for reproducibility.[2]

Related Work

Traditional Test Coverage Tools. Established static analysis tools like JaCoCo (Artho et al. 2010), coverage.py (Batchelder 2024), and Istanbul (Istanbul Team 2024) provide quantitative metrics (line coverage, branch coverage) efficiently at scale. However, these tools cannot assess

[1]Gherkin is a domain-specific language for behavior-driven development (BDD) that uses Given-When-Then syntax to specify test scenarios in natural language.

[2]GitHub repository: https://github.com/inflaton/LAJ-Gherkin.

semantic completeness—whether tests adequately capture business requirements, realistic edge cases, and meaningful error conditions. Our LAJ framework complements rather than replaces these tools: static tools measure *what code is executed*, while LAJ assesses *whether executed tests address specified requirements*.

LLM Evaluation and Judge Models. The use of LLMs as evaluators has emerged across multiple domains, with notable work in natural language generation (Zheng et al. 2023), creative writing assessment, and open-ended question answering. However, these applications focus primarily on subjective quality judgments where ground truth is inherently ambiguous. Software testing presents distinct challenges: assessments must align with concrete technical requirements, industry standards, and structured rubrics. Recent work has explored LLM-based code evaluation for correctness (Chen et al. 2021), but coverage assessment—determining whether tests adequately address requirements, edge cases, and error conditions—remains largely unaddressed.

Automated Test Generation. LLM-based test generation has advanced rapidly, with industrial deployments like Meta's TestGen-LLM achieving 75% build success rates (Alshahwan et al. 2024). However, recent surveys identify persistent gaps: "there is still no research on the use of LLMs in integration testing and acceptance testing" (Wang et al. 2024). More critically, the challenge of *evaluating* generated tests at scale remains unaddressed. Our LAJ framework bridges this gap by enabling automated, rubric-aligned assessment that maintains strong agreement with expert judgment while operating at speeds incompatible with manual review.

Cost and Reliability in LLM Deployment. While substantial research addresses LLM inference optimization (Huang and Wang 2025), studies typically focus on throughput and latency. The reliability dimension—completion rates, retry handling, and their impact on operational costs—has received limited attention. For production deployment, reliability failures translate directly to increased costs and degraded user experience. Our introduction of ECR@1 and adjusted cost metrics addresses this critical but underexplored aspect of LLM deployment.

Methodology

Problem Formulation

Given a software requirement specification (e.g., a Jira ticket) R and a corresponding Gherkin acceptance test script T, the goal is to automatically assess the *test coverage completeness*—the degree to which the test script adequately addresses the specified requirements. An LLM-as-a-Judge model M produces an assessment $A_M(R, T) \in [0, 100]$ representing estimated coverage percentage. We evaluate M against expert-provided ground truth $A^*(R, T)$ to measure assessment quality.

Benchmark Dataset Construction

Our evaluation dataset was constructed through a rigorous three-stage process by domain experts, targeting the Kill Bill subscription billing platform[3]—a production-grade system with complex business logic typical of enterprise subscription billing applications.

Stage 1: Jira Ticket Creation. Experienced product owners hand-crafted 100 Jira tickets, each carefully designed to reflect realistic API development scenarios. Each ticket follows a consistent structure:

- **Descriptive Title**: Clear, concise identification of API functionality
- **Comprehensive Description**: Detailed endpoint specification with expected behavior and business context
- **Acceptance Criteria**: Structured requirements defining scope and boundaries
- **Success Scenarios**: Explicit positive test cases and expected outcomes
- **Error Scenarios**: Comprehensive failure modes, edge cases, and exception handling requirements

The tickets maintain realistic distribution across HTTP methods: GET (50 tickets, 50%), POST (21 tickets, 21%), DELETE (15 tickets, 15%), PUT (14 tickets, 14%).

Stage 2: Gherkin Script Development. A team consisting of one software developer and one quality engineer collaborated to develop a Python-based automation pipeline leveraging `GPT-4.1` for Gherkin script generation. For each of the 100 Jira tickets, the system automatically produced behavior-driven development (BDD) acceptance test scripts written in Gherkin syntax.

Stage 3: Expert Annotation (Ground Truth). A separate group of senior quality assurance engineers with extensive domain expertise performed comprehensive manual reviews of all 100 Gherkin scripts. The annotation panel consisted of 3 senior QA engineers, each with at least 8 years of professional experience in API testing and prior familiarity with the Kill Bill platform. Expert annotators assessed test coverage quality using a four-dimensional weighted rubric: (1) *Scenario completeness* (40%): coverage of happy path, error conditions, edge cases; (2) *Acceptance criteria alignment* (30%): explicit validation of specified requirements; (3) *HTTP method-specific concerns* (20%): appropriate handling of idempotency, caching, state changes; and (4) *Assertion quality* (10%): depth and specificity of validation steps. Scores were aggregated via weighted sum on a 0–10 scale, then normalized to 0–100 percentage scale to ensure consistency with the LAJ output format. This rubric was embedded in the LAJ prompt to ensure alignment between expert and model assessments, with normalized scores serving as ground truth for evaluating LAJ model performance.

The resulting dataset—comprising 100 Jira tickets, 100 corresponding Gherkin test scripts, and expert-validated ground truth annotations—provides a valuable benchmark for systematic evaluation of LLM-as-a-Judge capabilities in test coverage assessment.

LAJ Framework Design

Design Principles The LAJ framework emphasizes three core principles: (1) **agreement with humans** via rubric-grounded scoring that aligns with expert judgment, (2)

[3]https://github.com/killbill

scalability through batched evaluation enabling high-throughput assessment, and (3) **traceability** via structured outputs with concise rationales for auditability.

Evaluation Process

Inputs For each evaluation, the judge model receives:
- Ticket requirements and acceptance criteria
- The Gherkin test script
- A comprehensive rubric specifying coverage expectations

Assessment Process The judge model:
1. Analyzes alignment between script and requirements
2. Checks breadth and depth of scenarios (happy path, errors, edges, state variations)
3. Applies the rubric to produce a scalar coverage score with justification

Outputs The model returns structured JSON containing:
- Coverage percentage (0–100)
- Coverage analysis: scenarios covered, gaps identified, recommendations
- Rubric-aligned flags for downstream analysis

Prompt Engineering The LAJ framework employs a two-part prompt design:

System Prompt Establishes role and expertise context, incorporating the four-dimensional weighted rubric (Stage 3 above) to ensure alignment between human and model assessments:

> *"You are a senior QA engineer specializing in behavior-driven development and test coverage analysis. Your task is to analyze how well a set of Gherkin-style acceptance tests cover the requirements of a given Jira story, based on a defined set of testing guidelines. Provide a coverage percentage, highlight what's covered, identify gaps or missing scenarios, and recommend improvements if needed. Use the following rubric for your assessment: {four_dimensional_weighted_rubric}"*

User Prompt Template The user prompt provides a structured input comprising: (1) Jira story details (ID, title, description), (2) corresponding Gherkin test cases, (3) standard testing guidelines, and (4) required JSON output specification. The template structure is as follows.

```
Below is a Jira story, a set of Gherkin acceptance
↪  tests,
and standard testing guidelines. Analyze how well the
Gherkin tests cover the story based on the guidelines.

Jira Story:
  ID: "{jira_id}"
  Title: "{jira_title}"
  Description: "{jira_description}"

Gherkin Test Cases:
  {gherkin_tests}

Standard Guidelines:
  {guidelines}

Output format (strict JSON):
  {example_output}
```

The testing guidelines specify coverage expectations per HTTP method. An abbreviated example:

```
GET: Valid requests, empty responses, pagination,
     query parameters, authorization (401/403),
     rate limiting, caching headers
POST: Valid/invalid payloads, duplicates, validation,
      large payloads, error handling (500)
PUT: Valid updates, partial updates, non-existent
     resources, concurrency
DELETE: Valid deletion, non-existent resources,
        soft deletes, concurrency
```

Models and Evaluation Protocol

We evaluate twenty model configurations spanning three model families: **GPT-4** (GPT-4o, GPT-4o Mini, GPT-4.1, GPT-4.1 Mini, GPT-4.1 Nano), **GPT-5** (high-, medium-, and low-reasoning-effort variants of GPT-5, GPT-5 Mini, and GPT-5 Nano), and **open-weight** models (GPT-OSS 20B and 120B, each evaluated under high-, medium-, and low-reasoning-effort settings). Proprietary models are accessed through the official OpenAI APIs, while open-weight models are accessed via OpenRouter.

The evaluation follows a systematic protocol:
1. **Model evaluation**: Each of 20 model configurations evaluates all 100 scripts with identical prompts across 5 independent runs
2. **Reliability tracking**: Record completion status, retry attempts, and token usage for each evaluation
3. **Metric computation**: Calculate accuracy, reliability, and cost metrics for all models with statistical aggregation across runs
4. **Statistical analysis**: Compare model performance across all dimensions using mean and standard deviation

For each model configuration, we compute mean and standard deviation across the 5 runs to assess both performance and variance. This multi-run approach enables robust statistical analysis and confidence in our findings.

Performance Metrics

Assessment Accuracy Metrics

Mean Absolute Assessment Error (MAAE) Measures average deviation between model assessments and ground truth in percentage points:

$$\text{MAAE} = \frac{1}{N}\sum_{i=1}^{N} |A_M(R_i, T_i) - A^*(R_i, T_i)| \qquad (1)$$

where $N = 100$ in our benchmark. Unlike standard MAE, MAAE is bounded to [0, 100] percentage points.

Assessment Performance Score (APS) Provides intuitive interpretation of accuracy:

$$\text{APS} = 100\% - \text{MAAE} \qquad (2)$$

Higher values indicate better alignment with expert judgment.

Perfect Match Rate (PMR) Percentage of predictions that exactly match ground truth (zero error):

$$\text{PMR} = \frac{\text{\# exact matches}}{N} \times 100\% \tag{3}$$

Close Match Rate (CMR) Percentage of predictions within ± 5 percentage points of ground truth:

$$\text{CMR} = \frac{\text{\# predictions with } |A_M - A^*| \leq 5}{N} \times 100\% \tag{4}$$

Operational Reliability Metrics

Evaluation Completion Rate (ECR@1) Measures the percentage of evaluations that produce valid, parseable output on the first attempt (reliability failures include API timeouts, malformed JSON, or schema violations):

$$\text{ECR@1} = \frac{\text{\# successful first attempts}}{N} \times 100\% \tag{5}$$

Mean Attempts per Evaluation Average number of API calls required to obtain N valid evaluations:

$$\text{Mean Attempts} = \frac{\text{Total API calls}}{\text{\# valid evaluations obtained}} \tag{6}$$

This directly impacts operational costs in production deployment.

Cost-Effectiveness Metrics

Cost per 1,000 Evaluations (C_M^{1K}) LLM API costs comprise prompt (input) and completion (output) token charges. For model M with pricing rates r_M^{prompt} and r_M^{compl} (USD per million tokens) and token counts t_i^{prompt} and t_i^{compl} for evaluation case i, the mean evaluation cost is computed as:

$$C_M = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{t_i^{\text{prompt}}}{10^6} \cdot r_M^{\text{prompt}} + \frac{t_i^{\text{compl}}}{10^6} \cdot r_M^{\text{compl}} \right) \tag{7}$$

The cost per 1,000 evaluations is then given by:

$$C_M^{1K} = C_M \times 1000 \tag{8}$$

Adjusted Cost per 1,000 Evaluations ($C_M^{1K,\textbf{adj}}$) To account for retry overhead during deployment, we define an adjusted cost metric:

$$C_M^{1K,\text{adj}} = C_M^{1K} \times \frac{100}{\text{ECR@1}_M} \tag{9}$$

This measure captures the effective deployment cost when retries are required due to incomplete or failed evaluations.

Results

Table 1 presents comprehensive results across all 20 model configurations, showing accuracy, reliability, and cost metrics.

Assessment Accuracy Performance

Elite Performance Tier (MAAE $<$ 7.0%): GPT-4o Mini achieves the best overall accuracy with MAAE of 6.07±0.08, corresponding to an APS of 93.93%. It also demonstrates the highest perfect match rate (32.6%), indicating strong agreement with expert assessments. The GPT-5 family at high reasoning effort also performs in this elite tier, with GPT-5 (high) achieving 6.16±0.26 MAAE (93.84% APS) and GPT-5 (medium) at 6.64±0.33 MAAE.

Strong Performance Tier (MAAE 7.0–9.0%): Several models achieve strong performance including GPT-4.1 Mini (6.92±0.10), GPT-5 Mini variants, GPT-4.1, GPT-4o, and GPT-4.1 Nano. These models maintain APS scores above 90%. GPT-4.1 Nano achieves the highest close match rate (66.8%), demonstrating strong performance within ±5 percentage points.

Economy Tier (MAAE $>$ 9.0%): Open-weight models (GPT-OSS 20B and GPT-OSS 120B families) along with GPT-5 Nano variants demonstrate significantly higher error rates (14.00–18.78 MAAE), with APS scores ranging from 81.22% to 86.00%. While these models offer substantially lower API costs, the accuracy trade-off may be significant for applications requiring high-fidelity assessments.

Operational Reliability Analysis

Perfect Reliability: Three models achieve 100% ECR@1: GPT-4o, GPT-4.1, and GPT-5 (low). These models never require retries, ensuring predictable costs and latencies in production deployment.

High Reliability (ECR@1 $>$ 95%): GPT-4o Mini (96.6±2.2%), GPT-5 family variants, GPT-4.1 Mini, and GPT-4.1 Nano demonstrate near-perfect reliability, requiring minimal retries (mean attempts: 1.00–1.04), resulting in negligible cost increases from reliability overhead. GPT-4o Mini's combination of elite accuracy and high reliability positions it as an optimal production choice.

Moderate Reliability (90% $<$ ECR@1 $<$ 95%): Open-weight models primarily occupy this tier, with ECR@1 ranging from 91.0% to 94.6%. These models exhibit higher variance in completion rates and may require 1.07–1.19 average attempts per evaluation. The reliability degradation manifests primarily as JSON parsing errors and occasional schema violations.

Low Reliability (ECR@1 $<$ 90%): GPT-OSS 20B (high) demonstrates the lowest reliability at 85.4±5.7% ECR@1, requiring an average of 1.17 attempts per evaluation. The high variance (±5.7%) indicates inconsistent behavior across runs, problematic for production deployment requiring predictable performance.

Failure modes across models include: (1) JSON parsing errors due to malformed output, (2) missing required fields, (3) rare API timeouts. Higher-capacity models demonstrate better instruction-following for structured output.

Cost-Performance Trade-offs

The adjusted cost metrics accounting for retry overhead reveal the true deployment economics. Models range from $0.45/1K (GPT-OSS 20B low) to $78.96/1K (GPT-5

Table 1: Complete Model Performance Across All Dimensions (Mean ± Std, 5 runs)

Family	Model	MAAE (%)	APS (%)	PMR (%)	CMR (%)	ECR@1 (%)	Attempts	Cost ($/1K)
GPT-4	**GPT-4o Mini**	**6.07±0.08**	**93.93±0.08**	32.6±1.4	56.8±0.7	96.6±2.2	1.04±0.03	1.01±0.02
	GPT-4o	8.34±0.15	91.66±0.15	9.6±2.1	61.4±0.5	**100.0±0.0**	1.00±0.00	16.76±0.02
	GPT-4.1 Nano	8.61±0.14	91.39±0.14	0.4±0.5	**66.8±1.3**	99.8±0.4	1.00±0.00	0.76±0.00
	GPT-4.1 Mini	6.92±0.10	93.08±0.10	**35.0±1.4**	52.4±0.5	99.8±0.4	1.00±0.00	3.08±0.01
	GPT-4.1	8.14±0.06	91.86±0.06	3.4±1.0	56.2±1.5	**100.0±0.0**	1.00±0.00	15.23±0.02
GPT-5	GPT-5 Nano (low)	9.70±0.26	90.30±0.26	4.6±2.3	51.6±3.7	94.0±2.8	1.07±0.03	0.74±0.02
	GPT-5 Nano (med)	14.41±0.45	85.59±0.45	3.0±0.6	35.0±2.9	97.8±1.5	1.02±0.01	2.11±0.04
	GPT-5 Nano (high)	17.01±0.48	82.99±0.48	4.8±2.0	30.0±1.3	91.0±16.0	1.19±0.36	4.66±0.87
	GPT-5 Mini (low)	7.26±0.19	92.74±0.19	4.0±0.6	51.8±2.6	96.8±1.2	1.03±0.01	3.81±0.05
	GPT-5 Mini (med)	8.50±0.72	91.50±0.72	6.2±1.6	46.4±3.8	97.8±1.3	1.02±0.01	5.81±0.11
	GPT-5 Mini (high)	7.17±0.44	92.83±0.44	7.6±2.1	53.6±5.7	98.6±1.0	1.01±0.01	15.26±0.19
	GPT-5 (low)	7.69±0.26	92.31±0.26	4.4±0.5	38.2±2.2	**100.0±0.0**	1.00±0.00	23.57±0.25
	GPT-5 (med)	6.64±0.33	93.36±0.33	6.2±3.1	44.6±1.4	99.8±0.4	1.00±0.00	46.62±0.37
	GPT-5 (high)	6.16±0.26	93.84±0.26	5.0±1.5	45.6±3.8	99.8±0.4	1.00±0.00	78.96±0.89
GPT-OSS	GPT-OSS 20B (low)	16.83±0.32	83.17±0.32	8.8±0.7	24.2±2.7	93.8±3.1	1.07±0.03	**0.45±0.03**
	GPT-OSS 20B (med)	18.66±0.42	81.34±0.42	5.6±1.4	19.4±4.6	92.2±4.4	1.08±0.05	0.51±0.03
	GPT-OSS 20B (high)	18.78±0.65	81.22±0.65	4.6±2.4	20.4±2.7	85.4±5.7	1.17±0.08	0.63±0.09
	GPT-OSS 120B (low)	14.00±0.57	86.00±0.57	2.4±2.3	39.2±1.6	96.6±2.9	1.04±0.03	0.75±0.02
	GPT-OSS 120B (med)	15.31±0.97	84.69±0.97	2.8±2.5	35.0±4.1	94.6±3.2	1.06±0.03	0.84±0.03
	GPT-OSS 120B (high)	15.84±0.43	84.16±0.43	2.2±1.5	29.2±2.9	93.4±1.5	1.07±0.01	1.06±0.04

MAAE = Mean Absolute Assessment Error; APS = Assessment Performance Score; PMR = Perfect Match Rate; CMR = Close Match Rate; ECR@1 = Evaluation Completion Rate (first attempt); Cost = adjusted cost per 1K evaluations (accounting for retries). Yellow highlighting indicates optimal production model; red indicates poorest reliability; bold indicates best performance per metric.

high)—a 175× span. Models with lower ECR@1 experience cost increases: GPT-OSS 20B (high) reaches $0.63/1K (up from $0.53 nominal, +18.1%) due to 85.4% ECR@1. In contrast, models with perfect reliability (GPT-4o, GPT-4.1, GPT-5 low) have identical nominal and adjusted costs.

The evaluation reveals distinct cost-performance tiers. At the ultra-low-cost end ($0.45–$0.84/1K), open-weight models offer budget-friendly options but with significant accuracy penalties (14.00–18.78 MAAE). The mid-tier range ($1.01–$5.81/1K) includes GPT-4o Mini, GPT-4.1 Nano, GPT-4.1 Mini, and GPT-5 Mini variants, offering strong accuracy-cost balance. The premium tier ($15–$79/1K) comprises GPT-4o, GPT-4.1, and GPT-5 variants, delivering elite accuracy with perfect reliability at substantially higher costs.

For a deployment scenario requiring 100,000 evaluations per month: GPT-4o Mini costs $101; GPT-4.1 costs $1,523; GPT-5 (high) costs $7,896; GPT-OSS 20B (low) costs $45. GPT-4o Mini provides a 78× cost reduction compared to GPT-5 (high) while simultaneously achieving superior accuracy (6.07 MAAE vs 6.16 MAAE, a 0.09pp improvement). This combination of best-in-class accuracy and exceptional cost-effectiveness makes GPT-4o Mini the optimal production choice.

Impact of Reasoning Effort

We analyze how reasoning effort settings affect performance across GPT-5 and GPT-OSS model families, revealing distinct patterns.

GPT-5 Family: Accuracy-Cost Trade-off. Within the GPT-5 family, higher reasoning effort consistently improves accuracy at the cost of increased inference expenditure. For the base GPT-5 model, high reasoning achieves 6.16±0.26 MAAE (93.84% APS) at $78.96/1K, medium reasoning yields 6.64±0.33 MAAE (93.36% APS) at $46.62/1K (41% cost reduction), and low reasoning produces 7.69±0.26 MAAE (92.31% APS) at $23.57/1K (70% cost reduction). This represents a 1.53pp accuracy degradation for a 70% cost savings when moving from high to low reasoning. GPT-5 Mini exhibits a similar pattern: high reasoning (7.17 MAAE, $15.26/1K), medium reasoning (8.50 MAAE, $5.81/1K, 62% cheaper), and low reasoning (7.26 MAAE, $3.81/1K, 75% cheaper). Interestingly, GPT-5 Mini (low) outperforms medium reasoning despite lower cost, suggesting non-monotonic optimization. GPT-5 Nano shows the most dramatic variance: high reasoning (17.01 MAAE, $4.66/1K), medium reasoning (14.41 MAAE, $2.11/1K), and low reasoning (9.70 MAAE, $0.74/1K). Remarkably, low reasoning achieves 7.31pp better accuracy at 84% lower cost than high reasoning, indicating severe overparameterization at higher reasoning levels for this capacity tier.

Reliability Implications. Reasoning effort also impacts operational reliability. GPT-5 (low) achieves perfect reliability (100% ECR@1) while GPT-5 (high/medium) maintain 99.8% ECR@1. This suggests lower reasoning modes may produce more consistent structured outputs. GPT-5 Nano exhibits the inverse pattern: high reasoning shows degraded reliability (91.0±16.0% ECR@1, high variance), while low

reasoning achieves 94.0±2.8% ECR@1. The 16% standard deviation at high reasoning indicates unstable behavior across runs.

GPT-OSS Models: Reasoning Overhead Without Benefit. Open-weight models demonstrate a contrasting pattern where increased reasoning effort degrades both accuracy and reliability while increasing costs. For GPT-OSS 120B, low reasoning achieves the best accuracy (14.00 MAAE, \$0.75/1K, 96.6% ECR@1), medium reasoning degrades to 15.31 MAAE (\$0.84/1K, 94.6% ECR@1), and high reasoning further deteriorates to 15.84 MAAE (\$1.06/1K, 93.4% ECR@1)—representing a 1.84pp accuracy loss, 41% cost increase, and 3.2pp reliability drop. GPT-OSS 20B exhibits an even more severe pattern: low reasoning (16.83 MAAE, \$0.45/1K, 93.8% ECR@1), medium reasoning (18.66 MAAE, \$0.51/1K, 92.2% ECR@1), and high reasoning (18.78 MAAE, \$0.63/1K, 85.4% ECR@1). The high reasoning configuration degrades accuracy by 1.95pp, increases costs by 40%, and suffers an 8.4pp reliability drop with high variance (ECR@1: 85.4±5.7%), making it unsuitable for production deployment.

Conclusion

We presented a comprehensive investigation of LLM-as-a-Judge for scalable test coverage evaluation, introducing a production-ready framework with novel reliability-aware metrics. Through systematic evaluation of 20 model configurations spanning GPT-4, GPT-5, and open-weight alternatives across 500 evaluation runs (100 expert-annotated Gherkin scripts × 5 runs), we provided multi-dimensional analysis encompassing accuracy, operational reliability, and cost-effectiveness for LLM-based test assessment.

Key Findings. Our investigation revealed several critical insights: (1) **Accuracy and reliability are independent dimensions**—GPT-5 (low) achieves 100% ECR@1 but moderate accuracy (7.69 MAAE), while GPT-4o Mini excels in both (6.07 MAAE, 96.6% ECR@1), demonstrating that multi-dimensional evaluation is essential for production deployment. (2) **Reliability failures have material cost impact**—low ECR@1 models incur retry overhead that substantially increases operational costs. GPT-OSS 20B (high) reaches \$0.63/1K adjusted cost due to 85.4% ECR@1; at 1M evaluations/month, reliability issues cost \$1,200 annually plus latency variance. (3) **Smaller models can outperform larger ones**—GPT-4o Mini (6.07 MAAE) surpasses GPT-4o (8.34 MAAE) and GPT-4.1 (8.14 MAAE), indicating that model optimization and training strategies matter more than parameter count alone. (4) **Reasoning effort optimization is model-family dependent**—GPT-5 models benefit from higher reasoning effort with predictable accuracy-cost trade-offs (70% cost reduction for 1.53pp accuracy loss when reducing from high to low), while open-weight models degrade across all dimensions with increased reasoning (1.95pp accuracy loss, 40% cost increase, 8.4pp reliability drop for GPT-OSS 20B from low to high). (5) **Cost spans 175×**—from \$0.45/1K (GPT-OSS 20B low) to \$78.96/1K (GPT-5 high), enabling informed model selection based on deployment constraints.

Production Guidance. Based on comprehensive empirical evidence, GPT-4o Mini emerges as the production-optimal choice: achieving best-in-class accuracy (6.07 MAAE, 93.93% APS), high reliability (96.6% ECR@1), and exceptional cost-effectiveness (\$1.01/1K)—delivering 78× cost reduction versus GPT-5 (high reasoning) while maintaining superior accuracy. For deployment scenarios requiring 100,000 evaluations monthly, this translates to \$101 versus \$7,896, enabling practical adoption at scale.

Deployment Recommendations. Practitioners should: (1) always measure ECR@1 alongside accuracy to capture operational reliability; (2) implement retry logic with 5–15% overhead budget; (3) monitor ECR@1, latency, and cost in production; (4) use adjusted cost metrics for total cost of ownership; and (5) start with GPT-4o Mini for most use cases.

Limitations. Our study has several limitations that suggest directions for future work: (1) **Domain-specificity**: evaluation limited to Gherkin/RESTful APIs may not generalize to other testing paradigms (unit tests, UI tests, security tests); replication across multiple platforms beyond Kill Bill would strengthen generalizability claims. (2) **Task complexity**: we treat all 100 test scripts as equally difficult; stratification by complexity could reveal whether premium models justify costs on harder scenarios. (3) **Systematic bias**: while we report mean absolute error, analysis of over/under-estimation patterns across different test types could guide model selection for specific applications. (4) **Temporal stability**: as APIs and model capabilities evolve, longitudinal studies are needed to assess performance stability. These limitations provide valuable directions for extending this work.

Future Directions. Several promising research directions warrant investigation: (1) **expanded test coverage** extending LAJ to unit, integration, UI, performance, and security tests; (2) **multi-domain validation** evaluating LAJ generalization across healthcare, finance, IoT, and mobile domains; (3) **task complexity stratification** analyzing performance variation across test difficulty levels to guide model selection; (4) **systematic bias mitigation** developing calibration techniques to reduce over/under-estimation patterns in specific model families.

This work establishes LLM-as-a-Judge as a viable approach for production test coverage evaluation, providing practitioners with evidence-based guidance for model selection, reliability-aware cost modeling, and reasoning effort optimization. By introducing reliability metrics alongside traditional accuracy measures, we enable informed deployment decisions that account for the full operational reality of LLM-based systems.

References

Alshahwan, N.; Chheda, J.; Finogenova, A.; Gokkaya, B.; Harman, M.; Harper, I.; Marginean, A.; Sengupta, S.; and Wang, E. 2024. Automated Unit Test Improvement using Large Language Models at Meta. *arXiv preprint arXiv:2402.09171*.

Artho, C.; Suzaki, K.; Di Nitto, E.; Tanabe, Y.; and Hagiya, M. 2010. Crap4j: Change Risk Anti-Patterns Detection for Java. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE '10, 499–500. New York, NY, USA: ACM.

Batchelder, N. 2024. Coverage.py: Code Coverage Measurement for Python. https://coverage.readthedocs.io. Accessed: 2024.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H. P. d. O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. 2021. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*.

Huang, D.; and Wang, Z. 2025. LLMs at the Edge: Performance and Efficiency Evaluation with Ollama on Diverse Hardware. In *Proceedings of the 2025 International Joint Conference on Neural Networks (IJCNN), Rome, Italy*, 1–8.

Istanbul Team. 2024. Istanbul: JavaScript Test Coverage Tool. https://istanbul.js.org. Accessed: 2024.

Karpurapu, S.; Myneni, S.; Nettur, U.; Gajja, L. S.; Burke, D.; Stiehm, T.; and Payne, J. 2024. Comprehensive evaluation and insights into the use of large language models in the automation of behavior-driven development acceptance test formulation. *IEEE Access*, 12: 58715–58721.

Wang, J.; Huang, Y.; Chen, C.; Liu, Z.; Wang, S.; and Wang, Q. 2024. Towards Understanding the Effectiveness of Large Language Models on Directed Test Input Generation. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2024.

Zheng, L.; Chiang, W.-L.; Sheng, Y.; Zhuang, S.; Wu, Z.; Zhuang, Y.; Lin, Z.; Li, Z.; Li, D.; Xing, E.; et al. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. *Advances in Neural Information Processing Systems*, 36.