CAT: Coverage-Adaptive Testing — Structured Test Suite Generation for Coding Agent Handoff

Sloane Sambuco[1,2], Majid Abdul[2], Sireesha Penumadu[2], Alfred Spector[1], Anukul Goel[2]

[1]Massachussets Institute of Technology
[2]Amazon Web Services

sloane23@mit.edu, abdmaj@amazon.com, penumadu@amazon.com, alfreds@mit.edu, anukul@amazon.com

Abstract

Integration testing for ML systems faces unique challenges due to stochastic behavior and vast input spaces, while Continuous Integration/Continuous Delivery (CI/CD) pipelines have budgets and constraints that limit the number of tests that can be within an integration test suite. We present CAT (Coverage-Adaptive Testing), a framework that generates structured test inputs for coding-agent-based integration testing. CAT enables users to define coverage objectives through category taxonomies (e.g., adversarial attacks, compliance checks), then systematically generates high-coverage inputs through iterative refinement. A generator LLM produces candidate inputs targeting coverage gaps, while a judge LLM validates and classifies them. Greedy minimization selects compact test suites within user-specified budgets. In a case study on AWS Bedrock Guardrails, CAT achieves above 80% coverage across 27 adversarial categories for an integration test suite of 8 tests. CAT's structured outputs, natural language inputs with category labels and validation metadata, are designed to enable handoff to coding agents for test implementation. This architecture demonstrates a reusable human-AI collaboration pattern: humans define coverage objectives, CAT systematically explores the input space creating integration test inputs, and coding agents (future work) transform inputs into executable tests.

Introduction

Integration testing is critical for teams deploying ML models in production, whether through APIs, third-party services, or internal systems. However, practical constraints, particularly limited compute budgets in CI/CD pipelines, typically restrict integration test suites to approximately 8-10 selected examples that serve as checks for fundamental behaviors. Developing these test suites remains predominantly manual, relying on engineer intuition and experience.

While coding agents accelerate software development, without structured input they can lack domain knowledge needed to generate optimal test suites independently. Moreover, model consumers typically lack training data access, making representative test creation challenging. This gap motivates a collaborative approach: leveraging AI's systematic generation capabilities while preserving human expertise in coverage definition.

We propose CAT, a framework that generates integration testing inputs for coding-agent hand-off. Users specify their test budget constraint and relevant categories for their use case, and CAT generates high-coverage inputs that coding agents can transform into executable tests. We demonstrate CAT through a case study on Bedrock Guardrails, a safety mechanism for LLM-based systems.

LLM-Based Test Generation. Recent work demonstrates LLM effectiveness in automated test generation through feedback-driven iterative refinement. CoverUp identifies untested code segments, then employs multi-turn dialogue to prompt LLMs for targeted tests (Pizzorno and Berger 2024). SymPrompt guides LLMs to generate tests for specific program paths through multi-stage reasoning (Ryan et. al. 2024). Meta's TestGen-LLM uses coverage improvement as a filtering criterion to ensure generated tests provide measurable value (Alshawan et. al. 2024). While these approaches address code coverage optimization, they focus on unit testing scenarios where the goal is maximizing coverage of implementation logic.

Our contributions are as follows:

- We adapt coverage-guided test generation to domain-specific integration testing by redefining coverage over user-specified category taxonomies rather than code-level metrics, enabling systematic testing of black-box ML systems and API services.
- We introduce explicit budget constraints into the generation process, producing compact test suites for resource-constrained CI/CD pipelines while maintaining high category coverage.
- We validate CAT through a case study on Bedrock Guardrails, achieving above 80% category coverage for 8 integration tests inputs across 27 adversarial attack categories.
- We are designing a coding-agent handoff architecture where CAT generates natural language test inputs that agents transform into executable integration tests, separating coverage-driven input selection from environment-specific test implementation.

Broader Implications: Collaborative Workflows

CAT demonstrates a broader principle for human-AI collaboration in software engineering: decomposing complex tasks into specialized subtasks that leverage complementary AI and human strengths. The test-input generation vs. implementation separation exemplified by CAT's handoff to coding agents represents a reusable pattern applicable to other testing domains such as API testing, security testing, and performance testing, where coverage criteria vary but the need

for systematic exploration and resource-constrained selection remains constant. CAT's creates a Gen-AI enhanced development workflow and human-AI partnership that enables rapid prototyping and iterative refinement, where human engineers can review and adapt AI-generated solutions to fit specific system contexts and business requirements.

Methodology

Phase 1 - Initialization and Seed Validation. The process begins with a small, human-provided seed set of representative integration test inputs. Each seed (we start with two for the Bedrock Guardrails case study) is first validated through execution against the guardrail to confirm it elicits the expected blocked response, analogous to the filter in TestGen-LLM (Alshawan et. al. 2024). Validated seeds are then classified by a judge LLM into predefined adversarial categories using few-shot prompting. This classification establishes an initial coverage vector and identifies untested categories. The user also specifies the maximum number of tests (budget) for their integration suite.

Phase 2 - Iterative Test Generation. CAT conducts iterative generation rounds in a feedback-driven loop that systematically targets coverage gaps. Each round begins by constructing a structured prompt for the generator LLM that explicitly communicates: the seed input as a template, the full category taxonomy, the current coverage state (which categories remain untested), and guided prompt instructions to generate variants targeting these gaps. To support exploration, inspired by TAP's tree-based branching approach (Mehrotra et. al. 2024), the generator produces N=3 candidate tests, which CAT validates by executing them against the target system. Successful variants, which are those triggering the expected system behavior, are then passed to the judge LLM to confirm category classification. This classification updates the coverage vector, which informs the next generation round, creating a tight feedback loop. When coverage growth stalls despite multiple rounds, CAT shifts strategy: rather than targeting single categories, it prompts the generator to compose multi-category variants that deliberately combine multiple attack strategies. This approach draws inspiration from TestGen-LLM's filtering technique (Alshawan et. al. 2024) and CoverUp's coverage-guided dialogue (Pizzorno and Berger 2024), but replaces code-level metrics with domain-specific category coverage.

Phase 3 - Greedy Test Suite Minimization. Once generation reaches saturation, CAT applies a greedy set-cover algorithm to select a minimal subset of validated tests within the budget constraint. At each iteration, the algorithm selects the test that covers the largest number of remaining uncovered categories. The greedy algorithm was selected because of its simplicity and effectiveness (Putra and Legowo 2025). The selected test is added to the final suite, coverage is updated, and the process repeats until the budget is exhausted or no remaining tests add new coverage.

Implementation Details. CAT is implemented using AWS Bedrock foundation models. Mistral Large 2 serves as the generator LLM for adversarial variant production, while Claude 3.5 Haiku acts as the judge model for classification and coverage tracking. The generator uses a temperature setting of T = 0.7 to encourage diverse generation, whereas the judge uses T = 0.1 for deterministic categorization.

Preliminary Results

We evaluated CAT on a taxonomy of 27 adversarial attack categories using AWS Bedrock Guardrails as the system under test. Starting with 2 human-provided seed examples and a candidate branching level of N=3, CAT achieved 80% category coverage on average across 20 independent runs, achieving the minimized test suite goal of 8 tests. Given the ability of LLMs to synthesize adversarial techniques into one utterance, across iterations almost all test outputs included combined categories. This demonstrates that coverage-adaptive prompting can enable LLMs to autonomously discover multi-category inputs that human authors would likely miss. A one-shot baseline that prompted the LLM to generate an 8-test integration suite covering as many categories as possible (without iteration, branching, or filtering) achieved 40% category coverage on average across 20 independent runs.

Ongoing Work

While CAT demonstrates effective coverage-driven input generation, finalizing the handoff of integration test inputs to coding agents for integration test suite implementation is ongoing research. CAT currently produces natural language test inputs with metadata (categories, expected behavior, validation status). Key research directions include finding optimal specification granularity, balancing explicit implementation guidance vs. agent autonomy, and investigating interfaces for agents to provide feedback. Ongoing work also includes testing other use-cases, benchmarking against existing methods, and potentially integrating the final methodology into AWS software development workflows.

In addition, while the initial objective is to find a suite of integration tests that satisfy the constraints, the search itself currently suffers from combinatorial explosion. The number of ways C categories can be covered in t test cases is $(2^t\text{-}1)^C$. This vast search space requires efficient optimization techniques beyond the greedy approach. The iterative generation and validation process can be computationally intensive, especially as the category taxonomy grows. Future work will explore ways to make the generation more sample-efficient, such as leveraging active learning or Bayesian optimization to intelligently direct the search. The goal of these future directions is to continue improving the scalability, efficiency, and seamless integration of the CAT framework within software development lifecycles that leverage both human and machine intelligence.

Acknowledgements

Thank you to Mohammad Fazel-Zarandi (Massachusetts Institute of Technology) for his support and review of this research.

References

Alshahwan, N.; Chheda, J.; Finegenova, 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. In Proceedings of the ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2024). New York: Association for Computing Machinery.

Altmayer Pizzorno, J.; and Berger, E. 2025. CoverUp: Coverage-Guided LLM-Based Test Generation. In Proceedings of the ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2025). New York: Association for Computing Machinery.

Mehrotra, A.; Zampetakis, M.; Kassianik, P.; Nelson, B.; Anderson, H.; Singer, Y.; and Karbasi, A. 2024. Tree of Attacks: Jailbreaking Black-Box LLMs Automatically. In Advances in Neural Information Processing Systems 37 (NeurIPS 2024). Red Hook, NY: Curran Associates, Inc.

Putra, A. W.; and Legowo, N. 2025. Greedy Algorithm Implementation for Test Case Prioritization in the Regression Testing Phase. *Journal of Computer Science* 21 (2): 290–303.

Ryan, G.; Jain, S.; Shang, M.; Wang, S.; Ma, X.; Ramanathan, M. K.; and Ray, B. 2024. Code-Aware Prompting: A Study of Coverage-Guided Test Generation in Regression Setting Using LLM. In Proceedings of the ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2024). New York: Association for Computing Machinery.