ICSE '26 · RIO DE JANEIRO, BRAZIL

Change And Cover (ChaCo):

Last-Mile, Pull Request-Based Regression Test Augmentation

Zitong Zhou* UCLA
Matteo Paltenghi* University of Stuttgart
Miryung Kim UCLA
Michael Pradel CISPA

* Both authors contributed equally to this work

48th International Conference on Software Engineering
April 12–18, 2026

Motivation: The Last-Mile Regression Test Gap

Software evolves through pull requests (PRs).

Testing code changes is critical for maintaining software quality.

Key Insight: PRs provide a natural playground for adding tests—each PRs is a discrete unit of reviewable changes with rich context.

Motivating Example: SciPy PR

scipy/signal/zpk2tf function

```
def zpk2tf(z, p, k):
    """
    Return polynomial transfer function ...
    """
+   xp = array_namespace(z, p)
+   z, p, k = map(xp.asarray, (z, p, k))
+   ...
+   if z.ndim > 1:                    UNCOVERED!
+       temp = _pu.poly(z[0], xp=xp)
+       b = xp.empty((z.shape...),...)
+       if k.shape[0] == 1:
            k = [k[0]] * z.shape[0]
        for i in range(z.shape[0]):
+           b[i] = k[i] * _pu.poly(z[i], xp=xp)
    else:
+       b = k * _pu.poly(z, xp=xp)
```

ChaCo contributed a test that exposed a bug in the uncovered branch

Research Gap & Challenge

Prior work rarely leverage the rich context available in PRs , such as links to related PRs and issues.

- PR title and description
- Developer discussions
- CI messages and code reviews
- Linked issues and related PRs

How to contextualize test generation for PR?

Test generation approaches often ignore the structure and utilities of existing tests.

- Fixtures and setup methods
- Custom markers and decorators
- Data generators and mocks
- Various project-specific conventions

How to generate tests that seamlessly integrate with existing conventions?

Motivation: Generating high-quality regression tests is difficult.

Test generated by ChaCo, more likely to be accepted by developers

```
import ...
from scipy.linalg import mat_decomp
from ... import assert_matrix...
dtypes = ...


class TestMatrixOperations:
    def setup_method(self):
        ...

    @pt.mark.parametrize("shape", [(5, 5), (10, 5), (0,
0)])@pt.mark.parametrize("dtype", dtypes)
    def test_matrix_decompose(self, shape, dtype):
        matrix = self.rng.random(shape).astype(dtype)

        if matrix.size == 0:
            with suppress_warnings() as sup:
                sup.filter(UserWarning, "Empty Matrix")
                U, V = mat_decomp(matrix)
            assert U.size == 0 and V.size == 0
            return

        U, V = matrix_decompose(matrix)
        assert_matrix_almost_equal(U @ V, matrix)
```

Test by direct prompting, needs manual adjustments to be accepted

```
import ...
from scipy.linalg import mat_decomp
# MISSING dependencies
# MISSING dtypes definition

# MISSING proper proper test class
# --- class TestMatrixOperations ---
def test_matrix_decompose():
    # FORGOT to reuse random seed
    # HARDCODED matrix shape & type
    matrix = np.random.randn(5, 5)

    # OVERLOOK edge case: empty matrix
    # FORGOT to check if warning is emitted
    U, V = mat_decomp(matrix)

    # FORGOT to use custom assertion for matrices
    assert np.allclose(U @ V, matrix)
```

Motivation: Generating high-quality regression tests is difficult.

Test generated by ChaCo, more likely to be accepted by developers

```
import ...
from scipy.linalg import mat_decomp
from ... import assert_matrix...
dtypes = ...


class TestMatrixOperations:
    def setup_method(self):
        ...

    @pt.mark.parametrize("shape", [(5, 5), (10, 5), (0,
0)])@pt.mark.parametrize("dtype", dtypes)
    def test_matrix_decompose(self, shape, dtype):
        matrix = self.rng.random(shape).astype(dtype)

        if matrix.size == 0:
            with suppress_warnings() as sup:
                sup.filter(UserWarning, "Empty Matrix")
                U, V = mat_decomp(matrix)
            assert U.size == 0 and V.size == 0
            return

        U, V = matrix_decompose(matrix)
        assert_matrix_almost_equal(U @ V, matrix)
```

Test by direct prompting, needs manual adjustments to be accepted

```
import ...
from scipy.linalg import mat_decomp
# MISSING dependencies
# MISSING dtypes definition
# MISSING proper pytest class
# --- Class TestMatrixOperations ---
def test_
    # FORGOT to reuse random seed
    # HARDCODED matrix shape & type
    matrix = np.random.randn(5, 5)

    # OVERLOOK edge case: empty matrix
    # FORGOT to check if warning is emitted
    U, V = mat_decomp(matrix)

    # FORGOT to use custom assertion for matrices
    assert np.allclose(U @ V, matrix)
```

This feels carelessly AI-generated. I'd need to clean it up substantially before accepting.

How to automatically generate high-quality regression tests that minimize developer's manual fix-up?

Enrich LLM test generation with PR Context + Test Context

ChaCo: Three-Stage Approach

1 Codebase Analysis

Patch Coverage + PR Context + Test Context

2 Test Generation & Refinement

Test Generation + Execution + Iterative Test Refinement

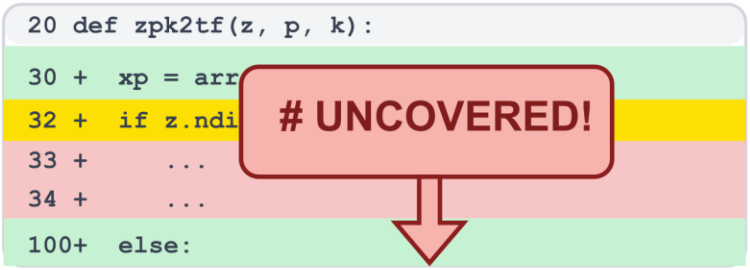
3 Test Integration & Report

Merge Test Case into Test Suite, + Select and generate report

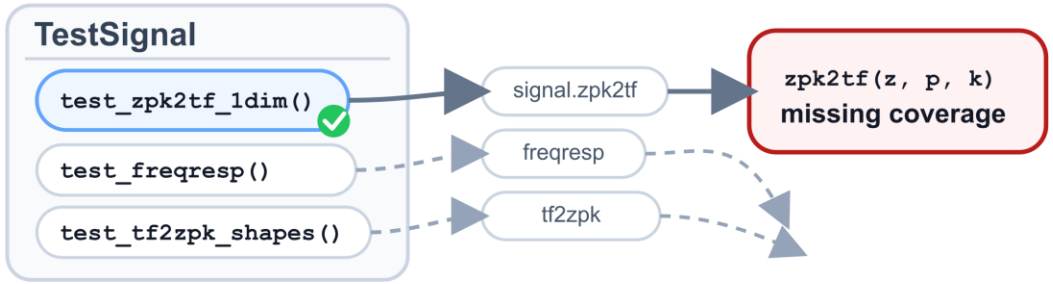
ChaCo is an end-to-end pipeline that analyzes PRs, generates test, and produces a summary. We envision it as part of Continuous Integration on new PRs, providing a low-friction way to improve test suites.

Stage 1: Codebase Analysis: Context Extraction

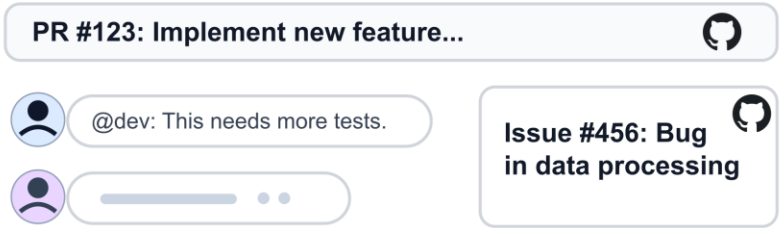
Patch Coverage Analysis



Test Context Analysis

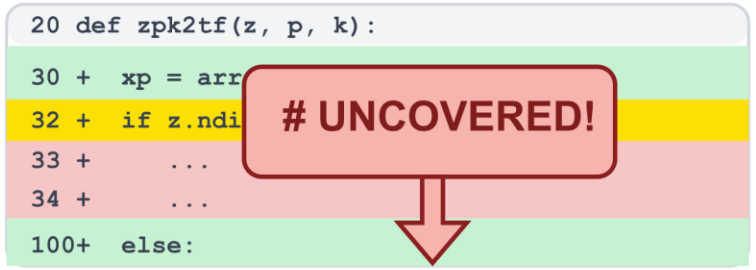


PR Context Analysis



Stage 1: Codebase Analysis: Context Extraction

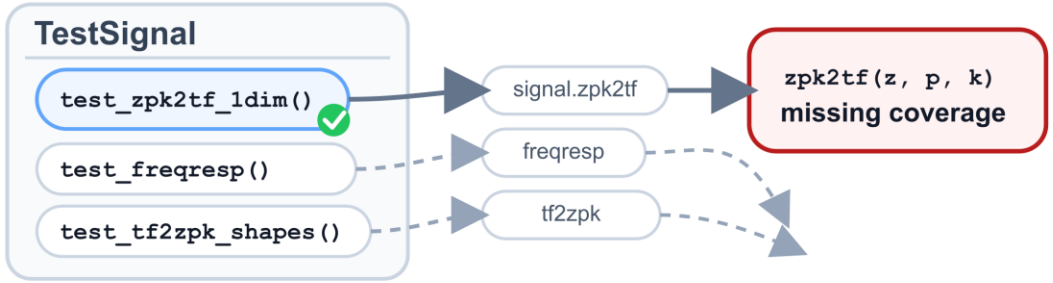
Patch Coverage Analysis



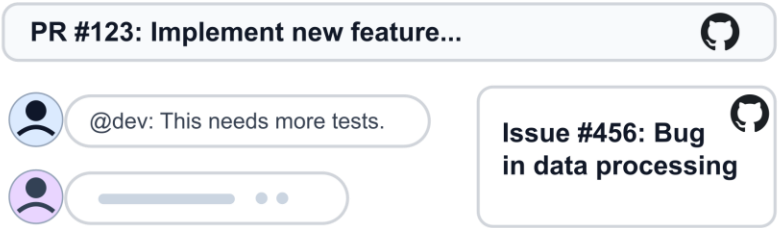
Stage 2

Function with missing coverage

Test Context Analysis



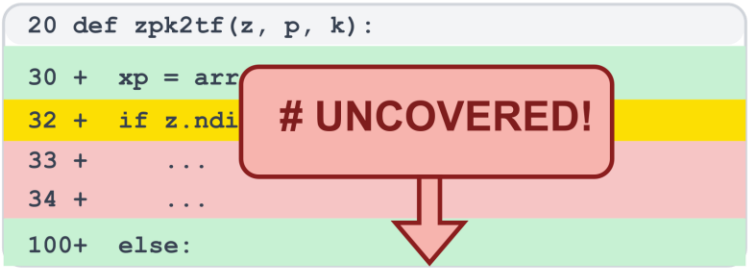
PR Context Analysis



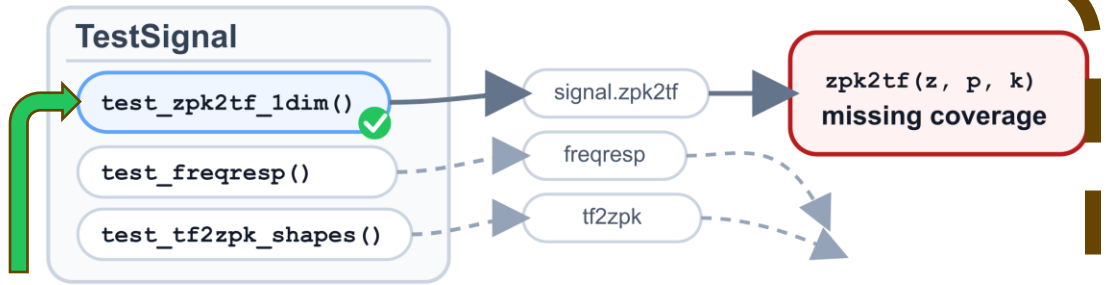
Stage 1: Codebase Analysis: Context Extraction

Stage 2

Patch Coverage Analysis

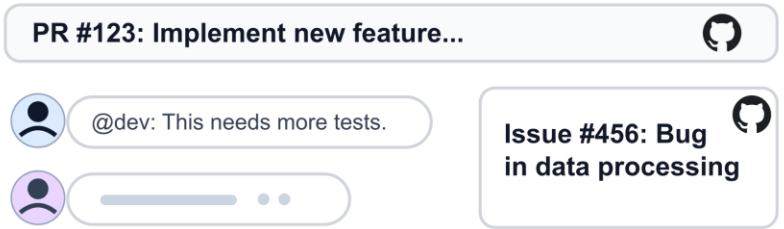


Test Context Analysis



Dynamic regression test selection finds the best suited tests

PR Context Analysis



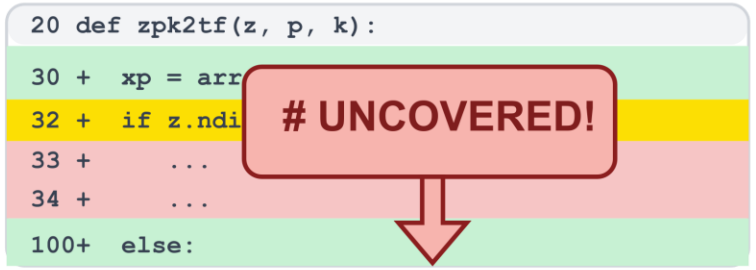
Function with missing coverage

Test Context: Existing test cases and dependencies that cover the func

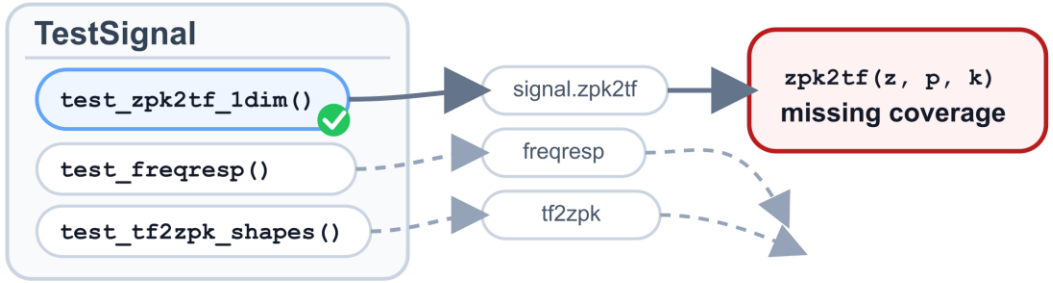
Helps the LLM write tests that look like it belong in the project.

Stage 1: Codebase Analysis: Context Extraction

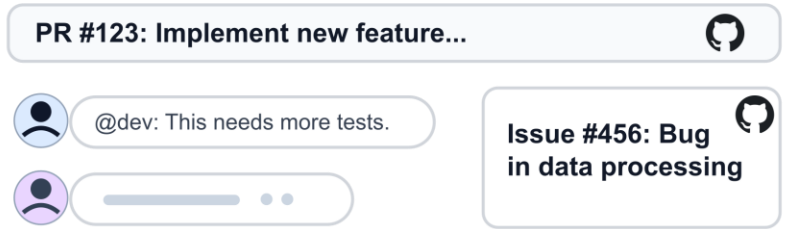
Patch Coverage Analysis



Test Context Analysis



PR Context Analysis



Stage 2

Function with missing coverage

Test Context: Existing test cases and dependencies

PR Summary

Test Context: structure, utilities, imports, conventions

With Test Context

```
import ...
from scipy.linalg import mat_decomp
from ... import assert_matrix...
dtypes = ...


class TestMatrixOperations:
    def setup_method(self):
        ...

    @pt.mark.parametrize("shape", [(5, 5), (10, 5), (0,
0)])@pt.mark.parametrize("dtype", dtypes)
    def test_matrix_decompose(self, shape, dtype):
        matrix = self.rng.random(shape).astype(dtype)

        if matrix.size == 0:
            with suppress_warnings() as sup:
                sup.filter(UserWarning, "Empty Matrix")
                U, V = mat_decomp(matrix)
            assert U.size == 0 and V.size == 0
            return

        U, V = matrix_decompose(matrix)
        assert_matrix_almost_equal(U @ V, matrix)
```

Without Test Context

```
import ...
from scipy.linalg import mat_decomp
# MISSING dependencies
# MISSING dtypes definition

# MISSING proper proper test class
# --- class TestMatrixOperations ---
def test_matrix_decompose():
    # FORGOT to reuse random seed
    # HARDCODED matrix shape & type
    matrix = np.random.randn(5, 5)

    # OVERLOOK edge case: empty matrix
    # FORGOT to check if warning is emitted
    U, V = mat_decomp(matrix)

    # FORGOT to use custom assertion for matrices
    assert np.allclose(U @ V, matrix)
```

Test Context: structure, utilities, imports, conventions

With Test Context

```
import ...
from scipy.linalg import mat_decomp
from ... import assert_matrix...
dtypes = ...


class TestMatrixOperations:
    def setup_method(self):
        ...

    @pt.mark.parametrize("shape", [(5, 5), (10, 5), (0,
0)])@pt.mark.parametrize("dtype", dtypes)
    def test_matrix_decompose(self, shape, dtype):
        matrix = self.rng.random(shape).astype(dtype)

        if matrix.size == 0:
            with suppress_warnings() as sup:
                sup.filter(UserWarning, "Empty Matrix")
                U, V = mat_decomp(matrix)
            assert U.size == 0 and V.size == 0
            return

        U, V = matrix_decompose(matrix)
        assert_matrix_almost_equal(U @ V, matrix)
```

✓ Correct Placement
Import Dependencies

Without Test Context

```
import ...
from scipy.linalg import mat_decomp
# MISSING dependencies
# MISSING dtypes definition

# MISSING proper proper test class
# --- class TestMatrixOperations ---
def test_matrix_decompose():
    # FORGOT to reuse random seed
    # HARDCODED matrix shape & type
    matrix = np.random.randn(5, 5)

    # OVERLOOK edge case: empty matrix
    # FORGOT to check if warning is emitted
    U, V = mat_decomp(matrix)

    # FORGOT to use custom assertion for matrices
    assert np.allclose(U @ V, matrix)
```

✗ Wrong Placement
Missing Imports

Test Context: structure, utilities, imports, conventions

With Test Context

```
import ...
from scipy.linalg import mat_decomp
from ... import assert_matrix...
dtypes = ...
```

✓ Reuse Test Fixture

```
class TestMatrixOperations:
    def setup_method(self):
        ...

    @pt.mark.parametrize("shape", [(5, 5), (10, 5), (0,
0)])@pt.mark.parametrize("dtype", dtypes)
    def test_matrix_decompose(self, shape, dtype):
        matrix = self.rng.random(shape).astype(dtype)

        if matrix.size == 0:
            with suppress_warnings() as sup:
                sup.filter(UserWarning, "Empty Matrix")
                U, V = mat_decomp(matrix)
            assert U.size == 0 and V.size == 0
            return

        U, V = matrix_decompose(matrix)
        assert_matrix_almost_equal(U @ V, matrix)
```

Without Test Context

```
import ...
from scipy.linalg import mat_decomp
# MISSING dependencies
# MISSING dtypes definition
```

✗ No Fixture Reuse

```
# MISSING proper proper test class
# --- class TestMatrixOperations ---
def test_matrix_decompose():
    # FORGOT to reuse random seed
    # HARDCODED matrix shape & type
    matrix = np.random.randn(5, 5)

    # OVERLOOK edge case: empty matrix
    # FORGOT to check if warning is emitted
    U, V = mat_decomp(matrix)

    # FORGOT to use custom assertion for matrices
    assert np.allclose(U @ V, matrix)
```

Test Context: structure, utilities, imports, conventions

With Test Context

```
import ...
from scipy.linalg import mat_decomp
from ... import assert_matrix...
dtypes = ...
```

✓ **Parameterize Test Inputs**

```
class TestMatrixOperations:
    def setup_method(self):
        ...

    @pt.mark.parametrize("shape", [(5, 5), (10, 5), (0,
0)])@pt.mark.parametrize("dtype", dtypes)
    def test_matrix_decompose(self, shape, dtype):
        matrix = self.rng.random(shape).astype(dtype)

        if matrix.size == 0:
            with suppress_warnings() as sup:
                sup.filter(UserWarning, "Empty Matrix")
                U, V = mat_decomp(matrix)
            assert U.size == 0 and V.size == 0
            return

        U, V = matrix_decompose(matrix)
        assert_matrix_almost_equal(U @ V, matrix)
```

Without Test Context

```
import ...
from scipy.linalg import mat_decomp
# MISSING dependencies
# MISSING dtypes definition
```

✗ **Hardcoded Inputs**

```
# MISSING proper proper test class
# --- class TestMatrixOperations ---
def test_matrix_decompose():
    # FORGOT to reuse random seed
    # HARDCODED matrix shape & type
    matrix = np.random.randn(5, 5)

    # OVERLOOK edge case: empty matrix
    # FORGOT to check if warning is emitted
    U, V = mat_decomp(matrix)

    # FORGOT to use custom assertion for matrices
    assert np.allclose(U @ V, matrix)
```

Test Context: structure, utilities, imports, conventions

With Test Context

```
import ...
from scipy.linalg import mat_decomp
from ... import assert_matrix...
dtypes = ...
```

✓ Handle Edge Case

```
class TestMatrixOperations:
    def setup_method(self):
        ...

    @pt.mark.parametrize("shape", [(5, 5), (10, 5), (0,
0)])@pt.mark.parametrize("dtype", dtypes)
    def test_matrix_decompose(self, shape, dtype):
        matrix = self.rng.random(shape).astype(dtype)

        if matrix.size == 0:
            with suppress_warnings() as sup:
                sup.filter(UserWarning, "Empty Matrix")
                U, V = mat_decomp(matrix)
            assert U.size == 0 and V.size == 0
            return

        U, V = matrix_decompose(matrix)
        assert_matrix_almost_equal(U @ V, matrix)
```

Without Test Context

```
import ...
from scipy.linalg import mat_decomp
# MISSING dependencies
# MISSING dtypes definition
```

✗ No Edge Case

```
# MISSING proper proper test class
# --- class TestMatrixOperations ---
def test_matrix_decompose():
    # FORGOT to reuse random seed
    # HARDCODED matrix shape & type
    matrix = np.random.randn(5, 5)

    # OVERLOOK edge case: empty matrix
    # FORGOT to check if warning is emitted
    U, V = mat_decomp(matrix)

    # FORGOT to use custom assertion for matrices
    assert np.allclose(U @ V, matrix)
```

Test Context: structure, utilities, imports, conventions

With Test Context

```
import ...
from scipy.linalg import mat_decomp
from ... import assert_matrix...
dtypes = ...


class TestMatrixOperations:
    def setup_method(self):
        ...

    @pt.mark.parametrize("shape", [(5, 5), (10, 5), (0,
0)])@pt.mark.parametrize("dtype", dtypes)
    def test_matrix_decompose(self, shape, dtype):
        matrix = self.rng.random(shape).astype(dtype)

        if matrix.size == 0:
            with suppress_warnings() as sup:
                sup.filter(UserWarning, "Empty Matrix")
                U, V = mat_decomp(matrix)
            assert U.size == 0 and V.size == 0
            return

        U, V = matrix_decompose(matrix)
        assert_matrix_almost_equal(U @ V, matrix)
```

Custom Assertion

Without Test Context

```
import ...
from scipy.linalg import mat_decomp
# MISSING dependencies
# MISSING dtypes definition

# MISSING proper proper test class
# --- class TestMatrixOperations ---
def test_matrix_decompose():
    # FORGOT to reuse random seed
    # HARDCODED matrix shape & type
    matrix = np.random.randn(5, 5)

    # OVERLOOK edge case: empty matrix
    # FORGOT to check if warning is emitted
    U, V = mat_decomp(matrix)

    # FORGOT to use custom assertion for matrices
    assert np.allclose(U @ V, matrix)
```

Generic Assertion

Stage 2: Test Generation with Iterative Refinement

Iterative Test Generation Loop

1. **Initial Prompt:** PR diff, PR Summary, focal function, Test Context
2. **LLM Generate Test + Execute**
3. **Measure Coverage & Refine** : 🔄
   Check if test passes and adds new coverage, enter iterative loop

✅ Pass + Coverage ↑
Test is good → proceed to Stage 3

❌ Fail + No Coverage
Prompt LLM with error message to repair test

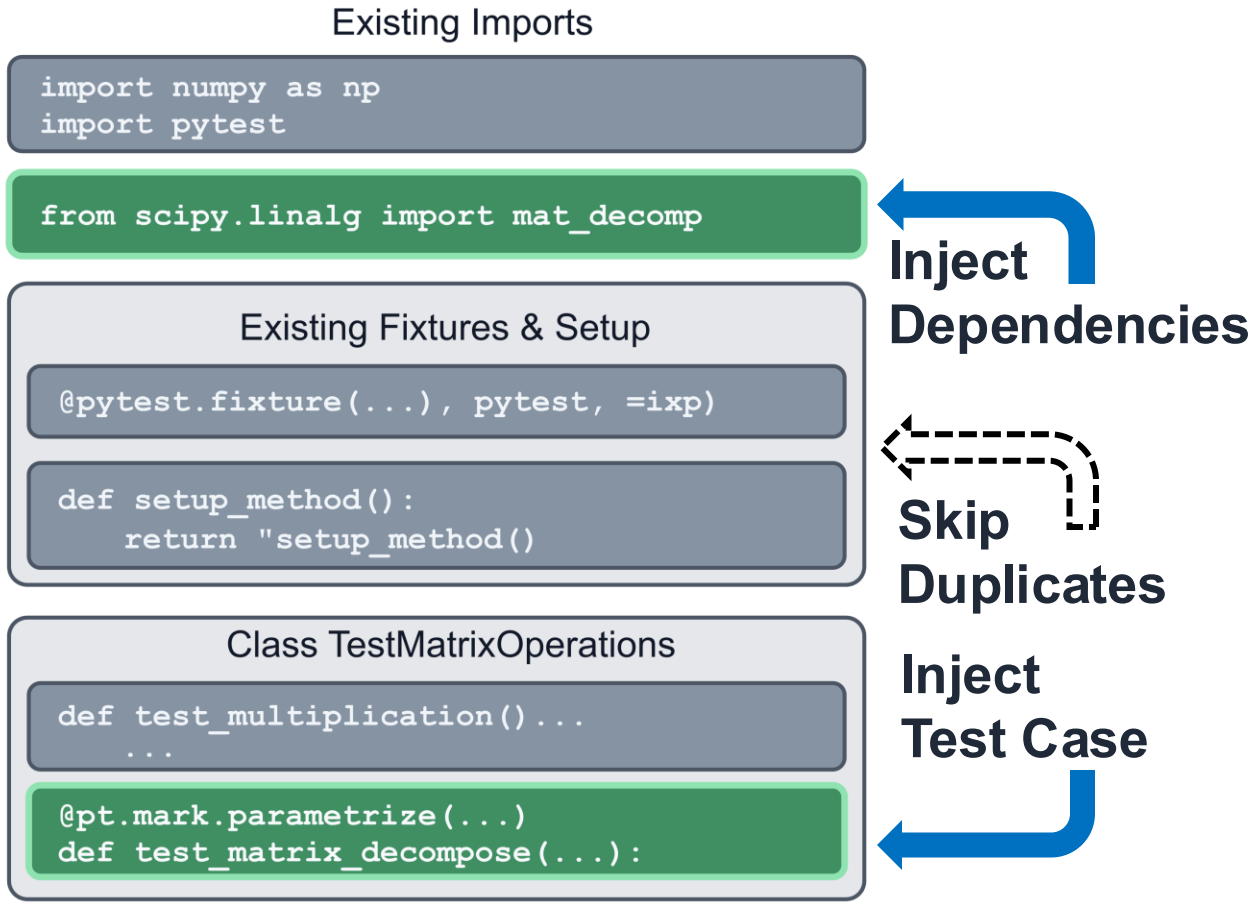
⚠️ Fail + Coverage ↑
Test is correct except oracle → repair test & keep coverage

ℹ️ Pass + No Coverage
Wrong focal function/Not satisfying certain branches→ repair test

Insight: Human developers need "trial and error"—LLMs benefit from iterative refinement. Ablation shows feedback 5.6× coverage improvement.

Stage 3: Test Integration & Report

AST-based Test Merging



Test Report attached to PR

ChaCo-CI commented on Jun 30, 2025 — Contributor

This test addition ensure that `signal.zpk2tf` is correct when given multidimensional inputs. PR #22896 updated `zpk2tf` to support array API, but some test coverage was still missing.
This new test adds coverage to the following modified lines:

```
# scipy/signal/_filter_design.py
def zpk2tf(z, p, k):
    """
    Return polynomial transfer function representation from ...
    """
    xp = array_namespace(z, p)
    # ... OMITTED
    if z.ndim > 1 :
        temp = _pu.poly(z[0], xp=xp) # ✅ NOW COVERED
        b = xp.empty((z.shape[0], z.shape[1] + 1),
            dtype=temp.dtype)   # ✅ NOW COVERED
        if k.shape[0] == 1 :   # ✅ NOW COVERED
    # ... OMITTED
```

Note: Parts of this test have been automatically generated by a an academic research project aiming at improving test coverage using LLMs ...

Merging is frictionless due to test context

Evaluation Setup

Research Questions

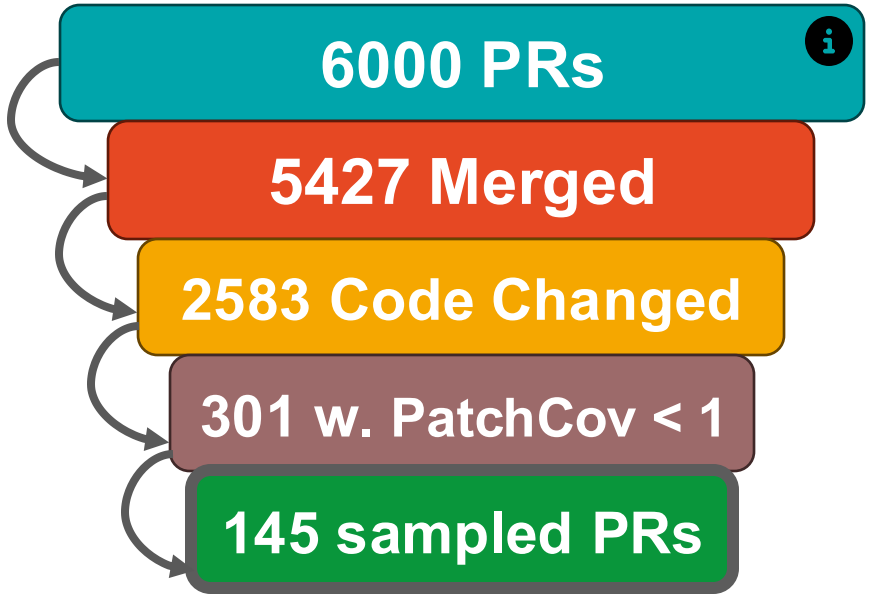
RQ1: Effectiveness: Can ChaCo increase patch coverage?

RQ2: Acceptability: Are ChaCo tests accepted by developers in real-world?

RQ3: Ablation: How do Test Context and Test Refinement components contribute to ChaCo's effectiveness?

Subject Projects and PRs

SciPy Qiskit pandas



2k most recent PRs per project collected in June 2025

RQ1. Effectiveness: ChaCo is effective at fixing coverage gaps

Table 1: Patch Coverage Increment

Project	#PR	#PR Cov Added	#PR 100% Cov	LnCov ↑	Cost ($)
SciPy	50	27	15	112	0.17
Qiskit	50	21	13	57	0.08
Pandas	45	17	14	20	0.08

45% PRs with coverage added

30% PRs achieved 100% patch coverage

Average cost: $0.11 per PR

Key Findings: ChaCo generated test cases add a total of 189 lines of coverage, fully covering 30% of PRs. Its PR-aware LLM-based test generation affordable at $0.11 per PR, making CI integration practical at scale.

RQ2: Acceptability: ChaCo tests are generally well-received by developers

RQ2.1: Qualitative Analysis

Two reviewers assessed test quality

Worthiness: 4.53/5.0

Test's potential to detect regressions or bugs

- "nice to cover an extra branch"

Integration: 4.20/5.0

Conformity to test suite's styling and structure

- "Good location", "right class"

Relevance: 4.70/5.0

Alignment with PR intent

- Confirms PR context design decision

RQ2.2: Test Submission

12 ChaCo tests to upstream projects

8	2
Merged	Open
2	**2**
Rejected	Bugs Found

Developer Feedback & Concerns:

+ Appreciated increased coverage
+ Supportive of AI-assisted testing
− Suggestions for better test utilities

Key Findings: ChaCo's tests are well-received. Developers appreciate increased test coverage but sometimes suggest using better test utilities could be used.

RQ3. Ablation Study: Dynamic Test Context and Iterative Test Refinement

Table 2: Performance of ChaCo and Ablated Variants (30 PRs)

Method	Pass Rate	LnCov ↑	Pass Rate	LnCov ↑	Pass Rate	LnCov ↑
ChaCo (Full)	28.8%	30	38.7%	18	19.2%	8
ChaCo (LLM TC)	29.5%	19	28.8%	2	18.8%	7
ChaCo (No TC)	31.5%	21	31.3%	2	27.7%	5
ChaCo (No FB)	7.6%	5	12.4%	2	3.5%	3

↑ **Dynamic Test Context Impact:** **2× coverage improvement** vs. LLM-only or no context

↑ **Iterative Feedback Impact:** provides **5.6× coverage improvement** and **312%** better test validity rate

Key Findings: Using Dynamic Test Context and Iterative Feedback substantially improves ChaCo's effectiveness.

Conclusion

Key Achievements

Effectiveness: 30% of PRs achieve full patch coverage at an affordable cost of $0.11 per PR.

Developer Reviewed: High qualitative ratings (>4.2/5)

Real-World Impact: 8/12 tests merged, 2 bugs discovered and fixed in production code

Vision: CI Workflow Integration

We Envision ChaCo as a GitHub Action running on new PRs, automating the last mile of regression test augmentation

Open-source:

github.com/UCLA-SEAL/Change-Cover