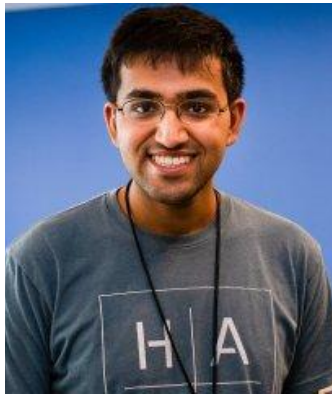
TestGenEval: A Real World Unit Test Generation and Test Completion Benchmark



Kush Jain

Baptiste Roziere

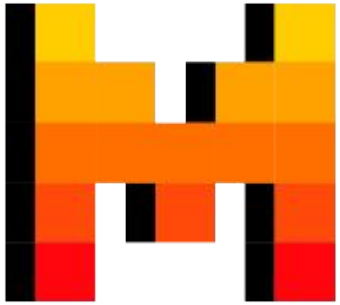
Gabriel Synnaeve

S3D Software and Societal Systems Department

Meta

kdjain@andrew.cmu.edu

Problem: many models exist for test generation, but no large scale testing benchmark exists



Existing work benchmarks test generation on small, toy problems

```python
# Solution
def twoSum(self, nums: List[int], target: int) -> List[int]:
    n = len(nums)
    for i in range(n - 1):
        for j in range(i + 1, n):
            if nums[i] + nums[j] == target:
                return [i, j]
    return []  # No solution found
```

```python
# Test 1
assert twoSum([3,2,4], 6) == [0, 1]
# Test 2
assert twoSum([3,3], 6) == [0, 1]
```

TestEval: Generating tests for LeetCode problems

```python
# Solution
def truncate(number: float):
    return number % 1.0
```

```python
# Test 1
assert truncate(3.5) == 0.5
# Test 2
assert truncate(1.33) = 0.33
```

HumanEvalFix: tests for basic python programming problems

Existing work benchmarks test generation on small, toy problems

Neither resembles real world software testing 😢

Benchmarks are saturated: state of the art achieves nearly 100% coverage

TestEval: Generating tests for LeetCode problems

HumanEvalFix: tests for basic python programming problems

Despite saturation on existing benchmarks, on real code even the best models struggle to generate correct tests

```
def test_exact_lookup():
    lhs = F('field')
    lookup = Exact(lhs, 'value')
    self.assertIn('%s', ...)
```

GPT-4o hallucinated class (F doesn't exist)

```
from django.db import connection, models

def test_query_set_init(self):
    model = models.Model()
    qs = QuerySet(model=model)
    self.assertEqual(qs.model, model)
```

Llama 3.1 405B hallucinated import of django.db

Despite saturation on existing benchmarks, on real code even the best models struggle to generate correct tests

```
def test_exact_lookup():
    lhs = F('field')
    lookup = Exact(lhs, 'value')
    self.assertIn('%s', ...)
```

GPT-4o hallucinated class (F doesn't exist)

```
from django.db import connection, models

def test_query_set_init(self):
    model = models.Model()
    qs = QuerySet(model=model)
    self.assertEqual(qs.model, model)
```

Llama 3.1 405B hallucinated import of django.db

Issue: need to benchmark on real projects: large scale, complex classes and dependencies

Part of the saturation issue: existing metrics are also flawed, typically pass@k or coverage

Test 1

Test 2

Test 3

pass@k - whether the test passes under the code under test

```java
@SuppressWarnings("resource")
public static InputStream unpack(final InputStream input)
        throws IOException {
    final ByteArrayOutputStream buffer = new ByteArrayOutputStream();
    final JarOutputStream jar = new JarOutputStream(buffer);
    try {
        final Object unpacker = Class.forName("java.util.jar.Pack200")
                .getMethod("newUnpacker").invoke(null);
        Class.forName("java.util.jar.Pack200$Unpacker")
                .getMethod("unpack", InputStream.class,
                        JarOutputStream.class)
                .invoke(unpacker, new NoCloseInput(input), jar);
    } catch (ClassNotFoundException e) {
        throw newIOException(e);
    } catch (NoSuchMethodException e) {
        throw newIOException(e);
    } catch (IllegalAccessException e) {
        throw newIOException(e);
    } catch (InvocationTargetException e) {
        throw newIOException(e.getCause());
    }
    jar.finish();
    return new ByteArrayInputStream(buffer.toByteArray());
}
```

Coverage - proportion of lines executed by code under test

Part of the saturation issue: existing metrics are also flawed, typically pass@k or coverage

Test 1

Test 2

Test 3

pass@k - whether the test passes under the code under test

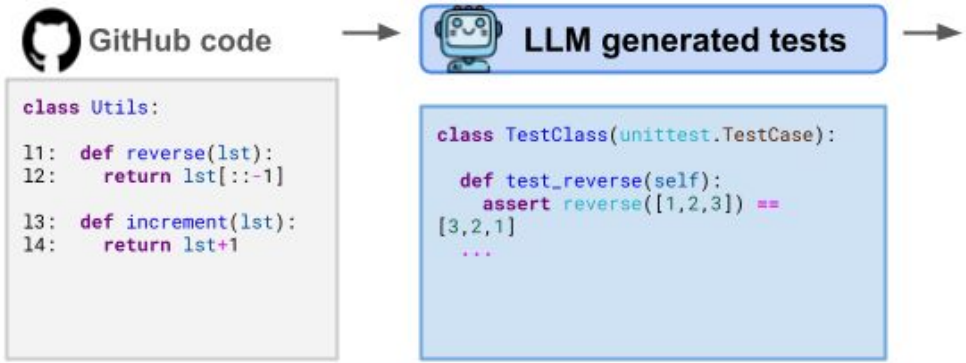
```java
@SuppressWarnings("resource")
public static InputStream unpack(final InputStream input)
        throws IOException {
    final ByteArrayOutputStream buffer = new ByteArrayOutputStream();
    final JarOutputStream jar = new JarOutputStream(buffer);
    try {
        final Object unpacker = Class.forName("java.util.jar.Pack200")
                .getMethod("newUnpacker").invoke(null);
        Class.forName("java.util.jar.Pack200$Unpacker")
                .getMethod("unpack", InputStream.class,
                        JarOutputStream.class)
                .invoke(unpacker, new NoCloseInput(input), jar);
    } catch (ClassNotFoundException e) {
        throw newIOException(e);
    } catch (NoSuchMethodException e) {
        throw newIOException(e);
    } catch (IllegalAccessException e) {
        throw newIOException(e);
    } catch (InvocationTargetException e) {
        throw newIOException(e.getCause());
    }
    jar.finish();
    return new ByteArrayInputStream(buffer.toByteArray());
}
```

Coverage - proportion of lines executed by code under test

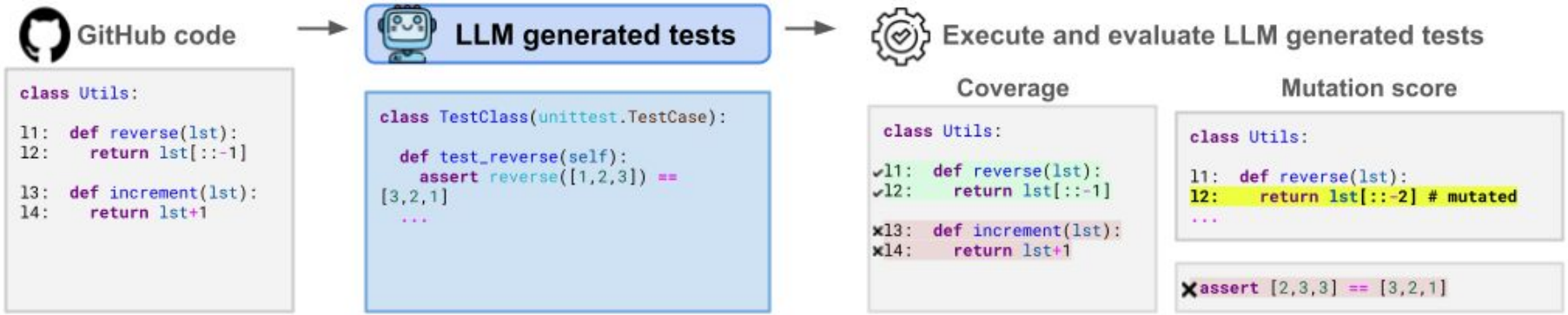
Both easy to game :(, just call code under test and assert true!!

Contribution: We release TestGenEval, the first large scale unit test generation benchmark

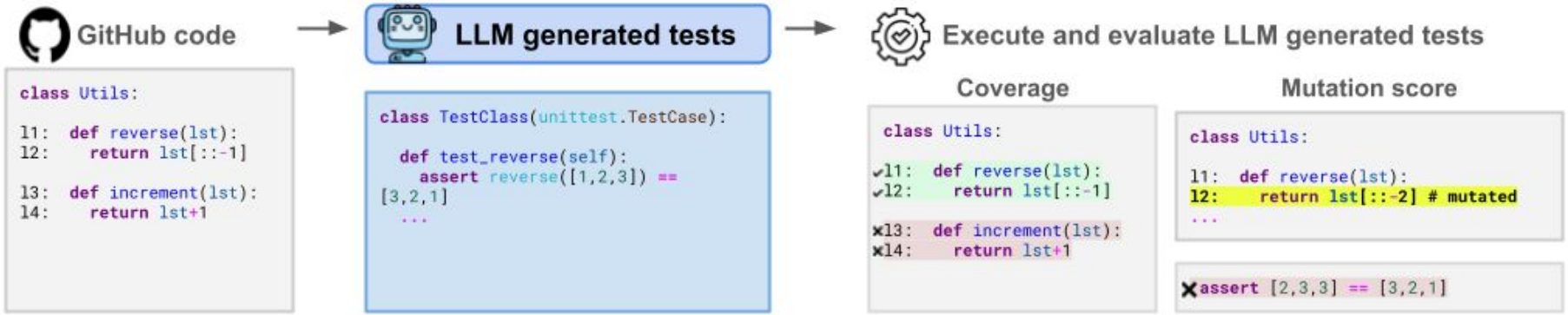
Contribution: We release TestGenEval, the first large scale unit test generation benchmark



Contribution: We release TestGenEval, the first large scale unit test generation benchmark



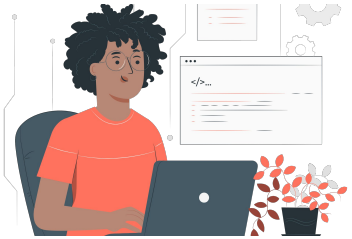
Contribution: We release TestGenEval, the first large scale unit test generation benchmark



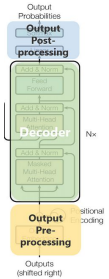
+ Mutation score (much harder to game)

+ Large scale real world repos

We model the tasks in TestGenEval after real world development



Developer

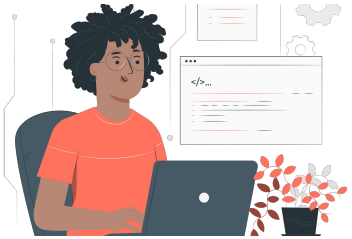


LLM

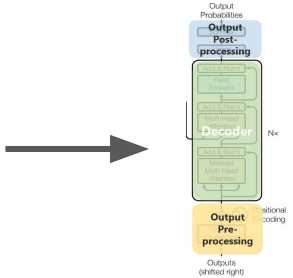
Full test suite

Full test suite generation

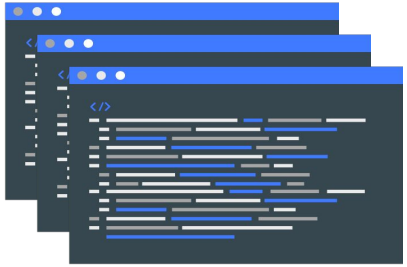
We model the tasks in TestGenEval after real world development



Developer

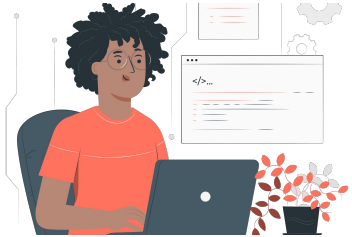


LLM



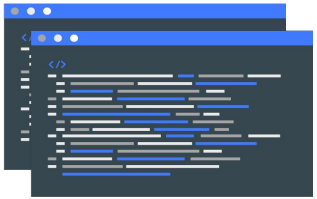
Full test suite

Full test suite generation

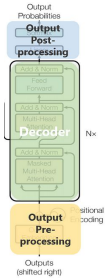


Developer

+



Partial test suite

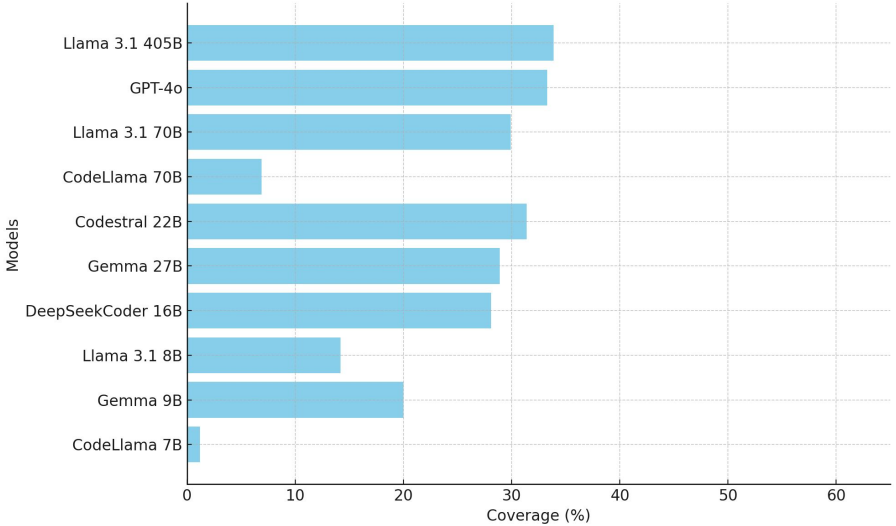


LLM

Test completion

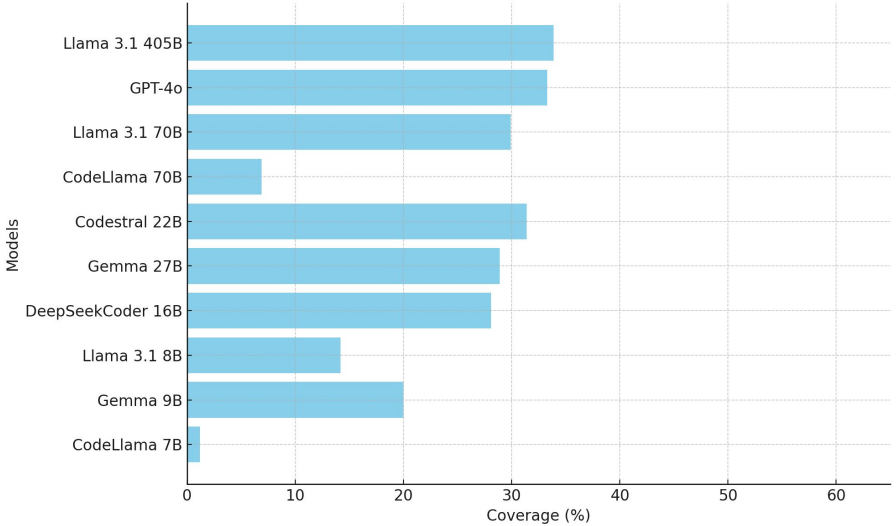
Test completion

Generally models perform relatively poorly on test generation, slightly better at test completion

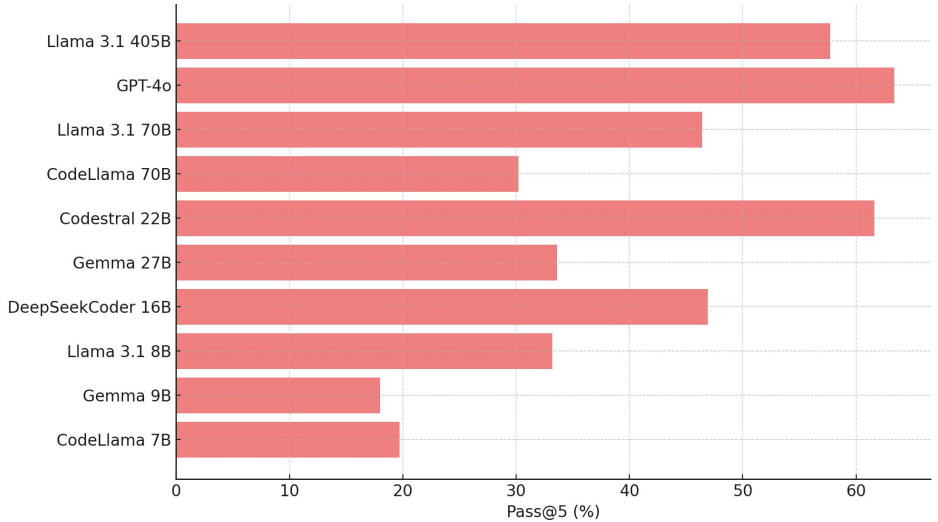


Coverage for full test suite generation - mutation even lower (all models below 35%)

Generally models perform relatively poorly on test generation, slightly better at test completion

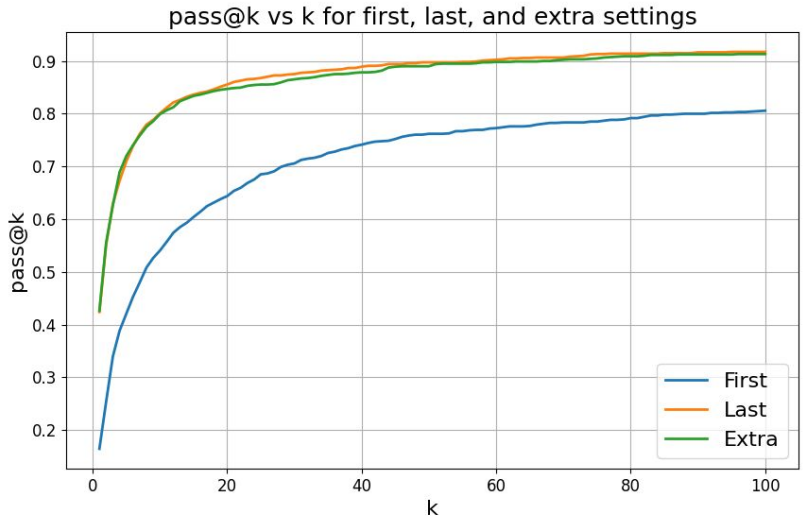


Coverage for full test suite generation - mutation even lower (all models below 35%)

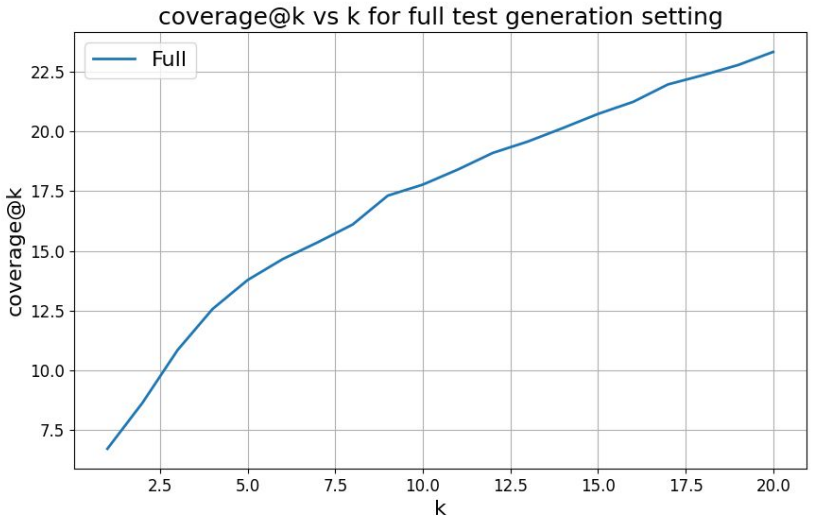


Pass@5 for test completion (reasonably high at around 60% for best models)

Sampling more makes a difference, plateaus at around 20 samples (completion), generally increases (generation)

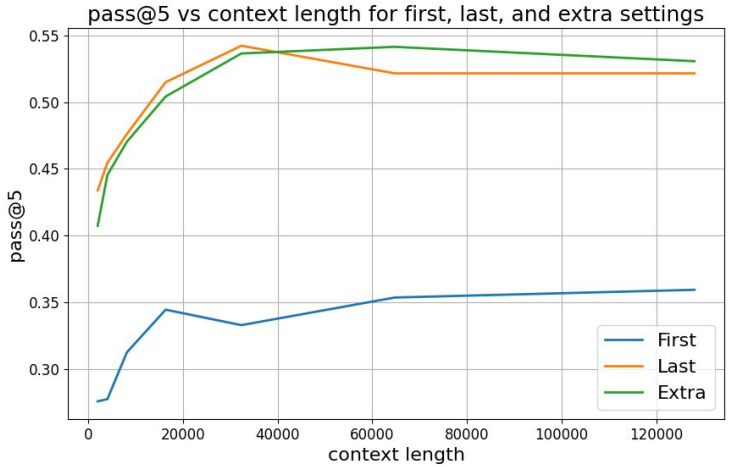


Test completion with more samples

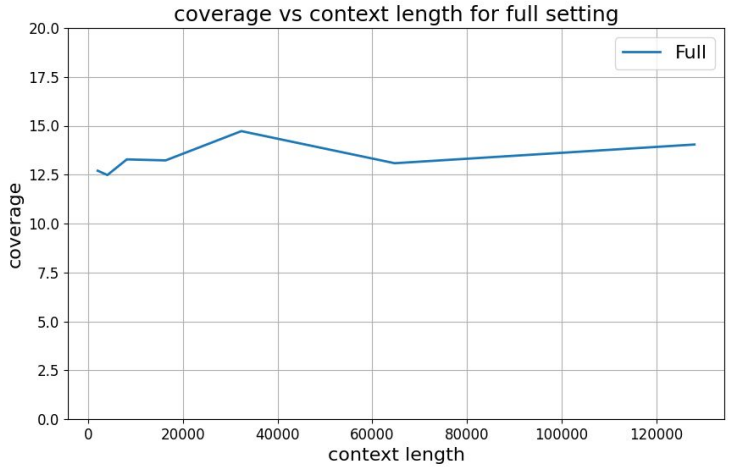


Coverage for full test generation with more samples

Increasing context helps test completion more than test generation

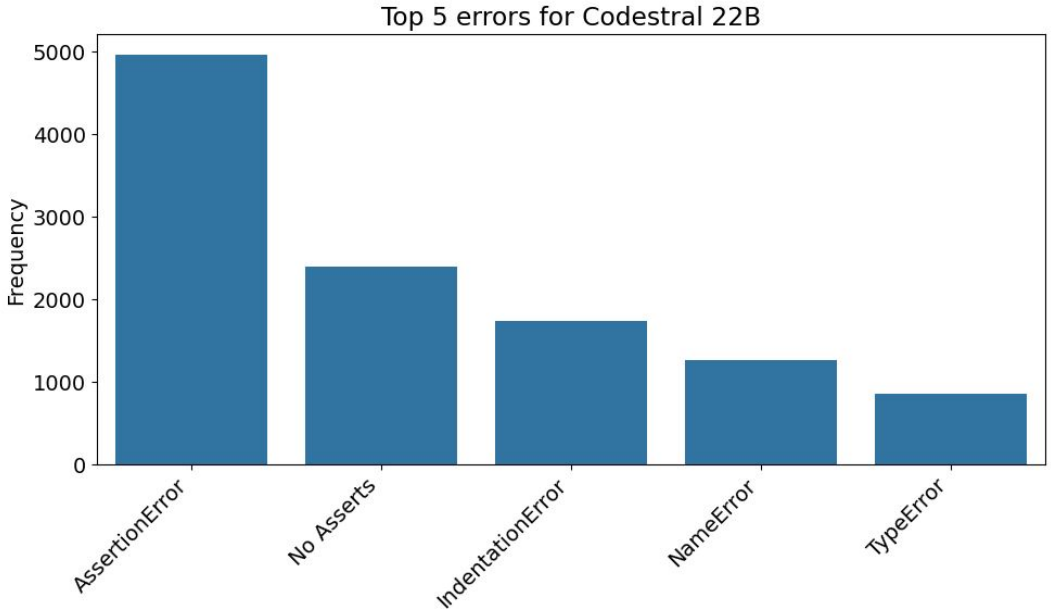


Pass@5 as context length changes (test completion settings)



Coverage as context length changes (full test suite generation)

Top models most commonly struggle to reason about execution, hallucination also a problem



Mostly errors related to either execution, formatting or hallucination

Codestral errors (assertion and hallucination)

```
def test_min_maxima_ratio():
    ...
    clust1 = OPTICS(min_samples=class="syntax-number">9,
min_maxima_ratio=class="syntax-number">0.001).fit(X)
    clust2 = OPTICS(min_samples=class="syntax-number">9,
min_maxima_ratio=class="syntax-number">0.01).fit(X)

    assert not np.array_equal(clust1.labels_, clust2.labels_)
```

Assertion error!

Labels does not include min_maxima_ratio

Codestral errors (assertion and hallucination)

```
def test_min_maxima_ratio():
    ...
    clust1 = OPTICS(min_samples=class="syntax-number">9,
min_maxima_ratio=class="syntax-number">0.001).fit(X)
    clust2 = OPTICS(min_samples=class="syntax-number">9,
min_maxima_ratio=class="syntax-number">0.01).fit(X)

    assert not np.array_equal(clust1.labels_, clust2.labels_)
```

Assertion error!

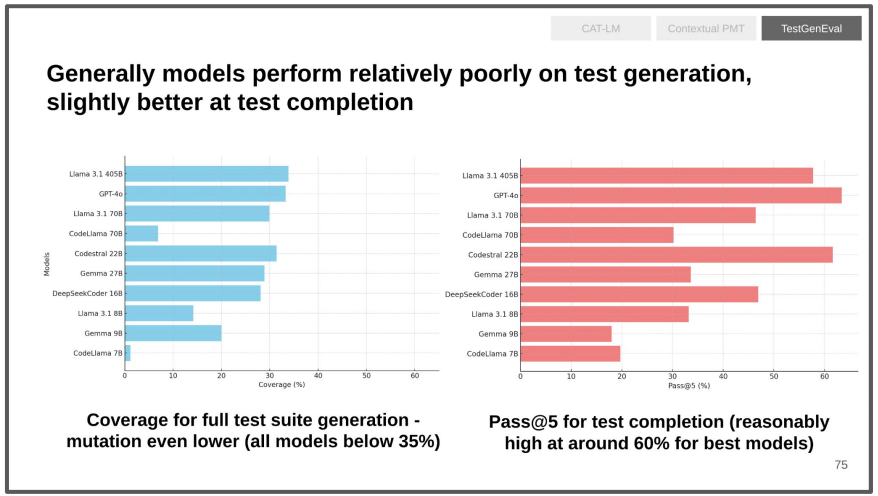
Labels does not include min_maxima_ratio

```
def test_get_paths(self):
    view = JavaScriptCatalog()
    paths = view.get_paths(['django.conf'])
    self.assertEqual(paths, [path.join(settings.BASE_DIR,
'django/conf/locale')])
    with self.assertRaises(ValueError):
        view.get_paths(['nonexistent_package'])
```

LLM hallucination!

django.conf is a hallucinated path

Summary: TestGenEval scales existing small scale benchmarks to large repositories



K. Jain, B. Rozière, and G. Synnaeve, TestGenEval: A Real World Unit Test Generation and Test Completion Benchmark
International Conference on Learning Representations (under submission ICLR 2025)