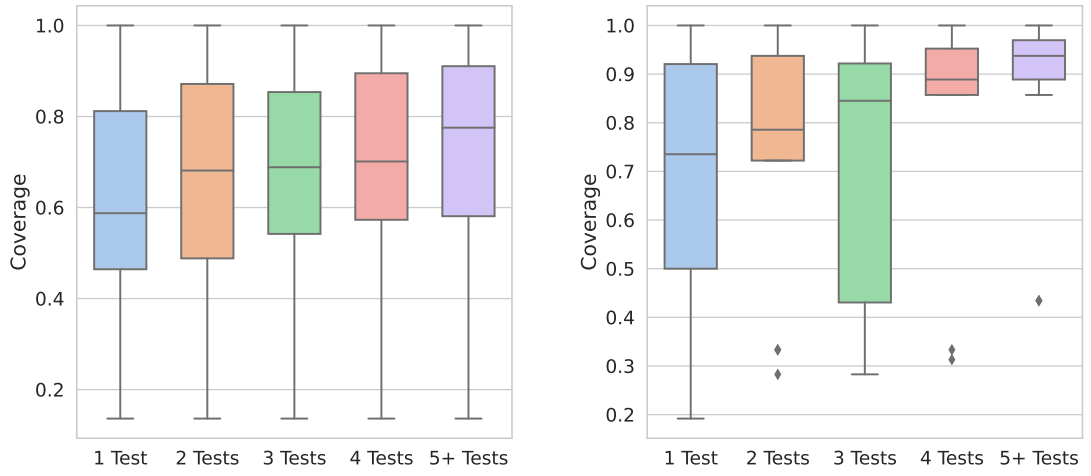
CAT-LM Training Language Models on Aligned Code And Tests

Supplementary Material

1 Additional Dataset Visualizations

Figure 1 presents the coverage distribution for tests in the test suite across Python and Java.



(a) Coverage distribution of tests for Python.

(b) Coverage distribution of tests for Java.

Figure 1: Coverage distribution of tests for different programming languages. We find that added coverage decreases for each new test as number of tests increases.

2 Defining the evaluation metrics

2.1 Complete list of Lexical metrics

We describe all lexical metrics we use to evaluate CAT-LM in greater detail.
Exact Match Accuracy is the percentage of cases where generated and gold test tokens are identical. This metric is the most strict, considering only exact syntactic and semantic equivalence. It does not account for cases where the generated test is semantically equivalent, but different lexically.
CodeBLEU is an enhanced version of BLEU score for code. In addition to n-gram matching, CodeBLEU also considers the BLEU score of reserved keywords, AST match and dataflow match.
ROUGE measures the longest overlapping sub sequence of tokens between the generated test and gold test, using F1 score. ROUGE has been used in prior test completion work as a metric of statement similarity.
BLEU measures the percentage of n-grams that exist in both the generated test and gold test. Specifically, we measure the overlap of 1-grams to 4-grams, using smoothing.

Edit Similarity is the single character similarity between the generated test and gold test. Specifically it is 1 - Levenstein edit distance or number of single character edits (addition, deletion or substitutions) needed to transform the generated test to the gold test, normalized by the total number of characters.

2.2 Complete list of Runtime metrics

We describe all runtime metrics we use to evaluate CAT-LM in greater detail.
Unique # is the number of generations that are not duplicates and contain a test method.
Assert # is the number of generations that contain at least one assert. For measuring compiling generations and passing generations, we filter out tests that do not contain any asserts, as usually these tests correspond with cases where the model invokes the method under test without actually testing it.
Compiling # is the number of generations that compile. For Python, we classify any syntax or import errors as compilation failures.
Passing # is the number of generations that pass the existing test suite. These tests could theoretically be added to a developer's test suite to test new functionality.
Coverage Improvement (Model) % measures the average improvement in code coverage that a passing test adds to an existing test suite. We also report the average of cases where the generated tests improve coverage by a nonzero amount.
Coverage Improvement (Human) % measures the average improvement in code coverage that the human gold test adds to an existing test suite. We also report the average human coverage improvement of cases where the model's generated tests improve by a nonzero amount.

3 Additional Results: Test Method Generation

We additionally obtain coverage with the original, human-written test files under the same conditions, keeping only the first or all tests as baselines for first and last test prediction respectively. Note that there is no baseline for the extra test generation task.

Table 1 and Table 2 show the full set of lexical metrics (described in Section 2.1) for our Python and Java test set. Similar to our paper, we find that CAT-LM outperforms existing CodeGen baselines across all modes.

Table 3 and Table 4 show the full set of runtime metrics (described in Section 2.2) for our Python and Java test set. CAT-LM also outperforms CodeGen baselines here, generating more compiling and passing tests.

4 Additional Results: Test Completion

Table 5 shows the full set of lexical metrics (described in Section 2.1) for TeCo vs CAT-LM. Similar to our conclusions in the paper, we find that CAT-LM outperforms TeCo across this expanded set of lexical metrics.

5 Additional Examples: Qualitative Study

Figure 2 and Figure 3 present additional examples of last test completion and extra test completion by CAT-LM, GPT-4 and EvoSuite. Our conclusions in the paper also extend to these examples. In general, GPT-4 tends to generate the highest quality tests of all three models, with GPT-4 generations both being readable and testing new functionality. CAT-LM also generates high quality tests, however as our extra test example shows, it sometimes fails to test new functionality in this mode. EvoSuite consistently struggles with readability across all three examples, with the last test and extra test examples showing cases where it is unclear that the EvoSuite generation is even correct.

Table 1: Lexical metrics performance comparison of our model vs CodeGen baselines on our held out test set for Java. We report lexical metrics for CAT-LM both with and without context.

Model	CodeBLEU	BLEU	XMatch	EditSim	Rouge-F	Rouge-P	Rouge-R
First Test							
CAT-LM w Context	21.0%	9.6%	0.3%	38.3%	39.4%	43.7%	42.5%
CAT-LM w/o Context	17.7%	7.4%	0.4%	31.6%	30.2%	37.5%	31.5%
Codegen-2B-multi	16.2%	6.4%	0.0%	26.7%	29.0%	33.3%	32.9%
Codegen-16B-multi	18.1%	8.8%	0.0%	32.6%	31.6%	38.1%	33.6%
Last Test							
CAT-LM w Context	38.3%	29.3%	4.8%	53.1%	54.9%	60.0%	57.6%
CAT-LM w/o Context	33.2%	24.3%	1.4%	49.2%	51.9%	57.0%	54.7%
Codegen-2B-multi	34.8%	26.8%	3.9%	47.6%	48.9%	53.9%	51.2%
Codegen-16B-multi	34.8%	26.9%	4.6%	49.7%	49.7%	56.2%	51.1%

Table 2: Lexical metrics performance comparison of our model vs CodeGen baselines on our held out test set for Python. We report lexical metrics for CAT-LM both with and without context.

Model	CodeBLEU	BLEU	XMatch	EditSim	Rouge-F	Rouge-P	Rouge-R
First Test							
CAT-LM w Context	41.4%	31.7%	15.4%	53.9%	60.9%	65.3%	60.7%
CAT-LM w/o Context	37.5%	27.0%	15.4%	51.7%	56.5%	62.7%	55.5%
Codegen-2B-multi	35.8%	26.6%	8.0%	50.7%	57.2%	61.3%	58.2%
Codegen-16B-multi	41.8%	32.1%	8.0%	56.6%	60.5%	64.7%	61.2%
Codegen-2B-mono	18.7%	8.1%	0.0%	32.7%	31.0%	35.4%	33.9%
Codegen-16B-mono	20.1%	9.0%	0.3%	34.6%	33.6%	36.5%	39.6%
Last Test							
CAT-LM w Context	55.4%	48.0%	20.8%	68.1%	70.8%	74.5%	70.0%
CAT-LM w/o Context	53.6%	45.8%	20.8%	67.2%	68.9%	73.4%	67.8%
Codegen-2B-multi	53.8%	46.3%	14.3%	67.6%	68.7%	71.7%	69.2%
Codegen-16B-multi	56.5%	48.6%	15.5%	68.9%	70.2%	73.9%	69.8%
Codegen-2B-mono	35.3%	26.6%	2.4%	50.8%	52.6%	57.3%	54.8%
Codegen-16B-mono	36.4%	28.4%	3.2%	51.5%	51.8%	56.2%	55.8%

Table 3: Comparison of runtime and coverage metrics for our model vs. CodeGen baselines for Python test set (total, unique, assert-containing, compiling, and passing completions; baseline coverage; human and model-generated test coverage improvements).

Setting	Runtime Metrics				Coverage				
	Unq.	**Asrt.**	**Cmp.**	**Pass**	**Baseline**	**Imp (M)**	**Imp (H)**	**Imp (M) - 0%**	**Imp (H) - 0%**
First test (Total: Java = 270, Python = 1120)									
CAT-LM w Context	780	575	384	44	0.0%	51.8%	56.8%	51.8%	56.8%
CAT-LM w/o Context	831	447	236	31	0.0%	50.6%	56.3%	50.6%	56.3%
Codegen-2B-multi	881	541	262	20	0.0%	55.2%	46.0%	55.2%	46.0%
Codegen-16B-multi	842	568	331	25	0.0%	56.7%	54.4%	56.7%	54.4%
CodeGen-2B-mono	851	501	259	35	0.0%	53.3%	57.5%	53.3%	57.5%
CodeGen-16B-mono	871	637	354	31	0.0%	57.5%	56.9%	57.5%	56.9%
Last test									
CAT-LM w Context	499	454	335	77	74.5%	2.9%	6.9%	10.5%	11.5%
CAT-LM w/o Context	549	475	350	79	71.3%	1.2%	8.0%	11.9%	22.5%
Codegen-2B-multi	556	483	353	59	63.0%	1.3%	13.3%	7.9%	20.6%
Codegen-16B-multi	503	430	286	57	61.9%	3.0%	13.5%	19.2%	36.2%
CodeGen-2B-mono	496	438	286	48	56.9%	2.3%	15.0%	21.9%	43.8%
CodeGen-16B-mono	544	461	317	46	69.8%	2.0%	11.6%	18.2%	26.7%
Extra test									
CAT-LM w Context	623	540	380	98	80.0%	0.2%	0.0%	2.6%	0.0%
CAT-LM w/o Context	621	565	425	104	82.6%	0.0%	0.0%	0.5%	0.0%
Codegen-2B-multi	587	460	326	45	79.2%	0.0%	0.0%	1.0%	0.0%
Codegen-16B-multi	612	487	321	50	85.8%	0.0%	0.0%	N/A	N/A
CodeGen-2B-mono	662	530	339	72	79.3%	0.2%	0.0%	1.6%	0.0%
CodeGen-16B-mono	635	519	359	60	83.6%	0.0%	0.0%	2.9%	0.0%

Table 4: Comparison of runtime and coverage metrics for our model vs. CodeGen baselines for Java test set (total, unique, assert-containing, compiling, and passing completions; baseline coverage; human and model-generated test coverage improvements).

Setting	Runtime Metrics				Coverage				
	Unq.	**Asrt.**	**Cmp.**	**Pass**	**Baseline**	**Imp (M)**	**Imp (H)**	**Imp (M) - 0%**	**Imp (H) - 0%**
First test (Total: Java = 270, Python = 1120)									
CAT-LM w Context	182	138	50	22	0.0%	44.1%	35.2%	57.0%	40.1%
CAT-LM w/o Context	176	102	9	9	0.0%	8.6%	19.0%	38.6%	21.1%
Codegen-2B-multi	195	159	22	13	0.0%	6.1%	3.2%	39.6%	21.1%
Codegen-16B-multi	175	150	29	7	0.0%	24.4%	7.4%	56.9%	17.2%
Last test									
CAT-LM w Context	81	75	54	17	69.6%	4.4%	4.4%	25.0%	25.0%
CAT-LM w/o Context	68	61	33	14	56.4%	0.0%	0.0%	N/A	N/A
Codegen-2B-multi	53	51	33	12	50.2%	0.0%	0.0%	N/A	N/A
Codegen-16B-multi	38	35	17	6	65.3%	0.0%	2.1%	N/A	N/A
Extra test									
CAT-LM w Context	62	52	41	17	86.3%	0.0%	0.0%	N/A	N/A
CAT-LM w/o Context	72	60	29	20	85.4%	0.0%	0.0%	N/A	N/A
Codegen-2B-multi	20	18	14	4	64.3%	0.0%	0.0%	N/A	N/A
Codegen-16B-multi	26	26	10	3	46.8%	11.1%	0.0%	16.7%	0.0%

Table 5: Comparison of TeCo vs CAT-LM both with and without context across all lexical metrics.

Model	**CodeBLEU**	**BLEU**	**XMatch**	**EditSim**	**Rouge-F**	**Rouge-P**	**Rouge-R**
CAT-LM w Context	67.1%	69.1%	50.4%	82.3%	82.8%	82.9%	84.2% %
CAT-LM w/o Context	65.9%	67.9%	48.9%	81.6%	82.2%	82.5%	83.5% %
TeCo	26.7%	31.6%	13.8%	58.0%	60.2%	59.0%	64.6% %

```java
public void testTlsSender() throws JoranException, InterruptedException {
    ...
    Logger logger = context.getLogger("test-tls");
    logger.info("test message over tls");

    context.stop();
    Thread.sleep(100);

    final String serverData = serverStream.toString();
    assertTrue("Server received: " + serverData, serverData.contains("test message over
        tls"));
}
```

Listing 1: CAT-LM last test generation

```java
public void testTlsSender() throws JoranException, InterruptedException {
    ...
    Logger logger = context.getLogger("test-tls");
    logger.info("test message over tls");

    context.stop();
    Thread.sleep(100);

    final String serverData = serverStream.toString();
    assertTrue("Server received: " + serverData, serverData.contains("test message over
        tls"));
}
```

Listing 2: GPT-4 last test generation

```java
@Test(timeout = 4000)
public void test19()  throws Throwable  {
  Syslog4jAppender<MultipleSyslog> syslog4jAppender0 = new
      Syslog4jAppender<MultipleSyslog>();
  // Undeclared exception!
  try {
    syslog4jAppender0.start();
    fail("Expecting exception: NullPointerException");

  } catch(NullPointerException e) {
     //
     // no message in exception (getMessage() returned null)
     //
     verifyException(
        "com.papertrailapp.logback
         .Syslog4jAppender", e);
  }
}
```

Listing 3: EvoSuite last test generation

Figure 2: Example last test generations by CAT-LM, GPT-4 and EvoSuite. Both CAT-LM and GPT-4 generate a well named test for sending a TLS message. The EvoSuite generation tests an exception is thrown, however it is unclear what is being tested or whether the tested behavior is correct.

```java
@Test
public void mapBuilder_addEntries_
    forNonEmptyMap_mustBuildMapWithAllEntries() {
    MapBuilder<String, String> mapBuilder = new MapBuilder<>();
    HashMap<String, String> entries = new HashMap<>();
    entries.put("foo", "fooValue");
    entries.put("bar", "barValue");
    mapBuilder.addEntries(entries);
    Map<String, String> res = mapBuilder
            .addEntries(new HashMap<>())
            .build();
    assertThat(res, is(aMapWithSize(2)));
    assertThat(res.get("foo"), is("fooValue"));
    assertThat(res.get("bar"), is("barValue"));
}
```

Listing 4: CAT-LM extra test generation

```java
@Test
public void mapBuilder_addEntries_
    toMapWithExistingEntries_mustCombineMaps() {
    MapBuilder<String, String> mapBuilder = new MapBuilder<>();
    mapBuilder.addEntry("key1", "value1");

    HashMap<String, String> additionalEntries = new HashMap<>();
    additionalEntries.put("key2", "value2");
    additionalEntries.put("key3", "value3");

    Map<String, String> res = mapBuilder
            .addEntries(additionalEntries)
            .build();

    assertThat(res, is(aMapWithSize(3)));
    assertThat(res.get("key1"), is("value1"));
    assertThat(res.get("key2"), is("value2"));
    assertThat(res.get("key3"), is("value3"));
}
```

Listing 5: GPT-4 extra test generation

```java
@Test(timeout = 4000)
public void test3()  throws Throwable  {
  MapBuilder<Object, Integer> mapBuilder0 = new MapBuilder<Object, Integer>();
  // Undeclared exception!
  try {
    mapBuilder0.addEntries((Map<Object, Integer>) null);
  } catch(NullPointerException e) {
     //
     // no message in exception (getMessage() returned null)
     //
     verifyException("java.util.HashMap", e);
  }
}
```

Listing 6: EvoSuite last test generation

Figure 3: Example extra test generations by CAT-LM, GPT-4 and last test generation by EvoSuite (EvoSuite has no concept of adding a test to an already complete test suite). CAT-LM copies the last test in the test suite, and just changes the name, not testing new functionality, while GPT-4 generates a new test that checks for new functionality. Similar to other examples, EvoSuite fails to generate a meaningful test, with poor readability and unclear correctness.