

Non-testable Programs and Metamorphic Testing

CS453 Automated Software Testing

How many digits of Pi can you recall?

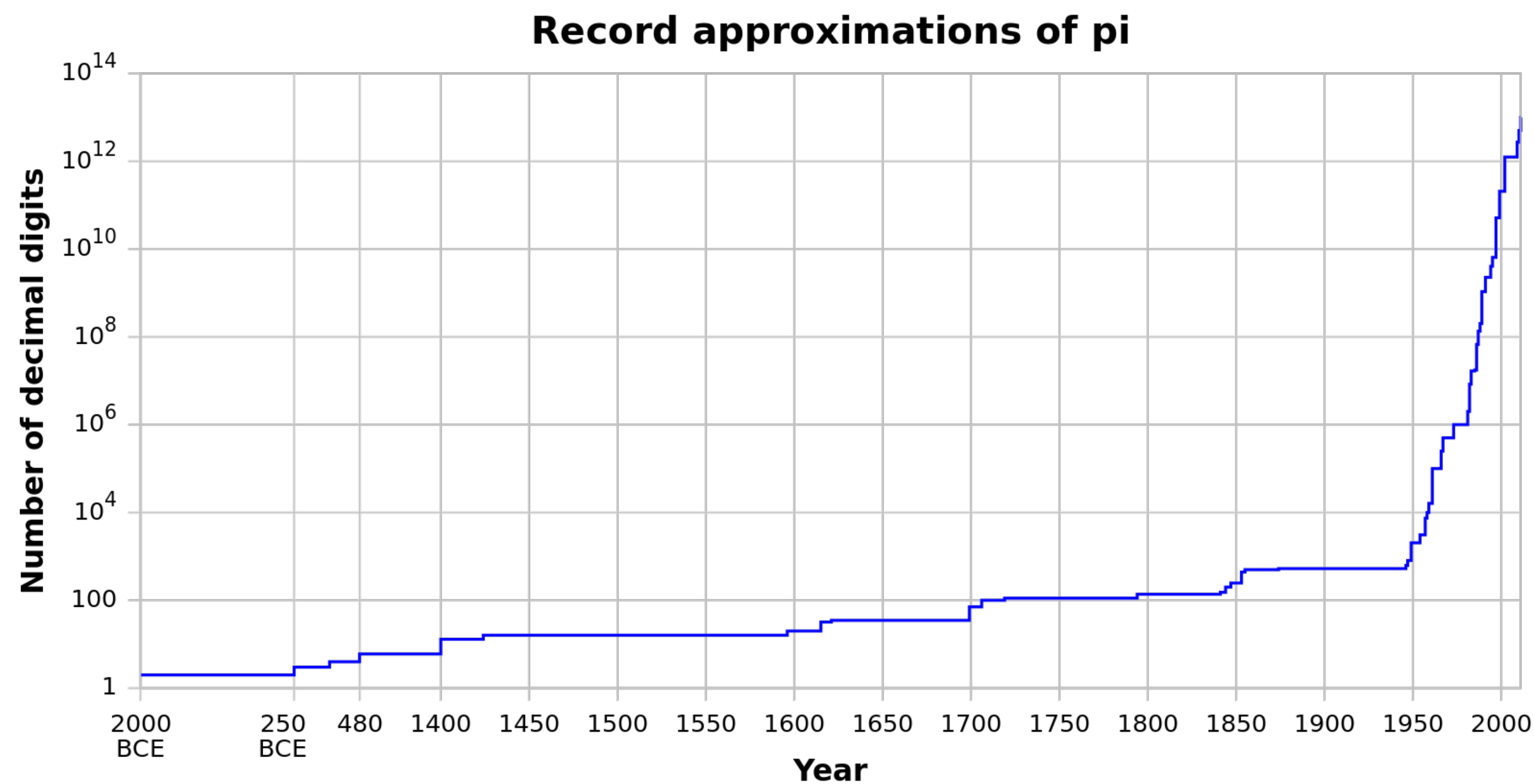
History of Pi

- BC 2000, Babylonia: $3 + 1/8 = 3.125$
- BC 250, Archimedes: $223/71 < \text{Pi} < 22/7$
- AD 5, Lu Xin: 3.1457 (method not known)
- AD 150, Ptolemy: $377/120 = 3.14166\ldots$
- AD 480, Zu Chongzhi: $355/113, 3.1415926 < \text{Pi} < 3.1415927$

History of Pi

- 1400, Madhava: power series expansion (aka Leibniz formula), 10 decimal places
- 1706, William Jones: the first use of Greek letter π
- 1775, Euler: used π in his book, assuring its popularity
- 1874, William Shanks: took 15 years to calculate 707 decimal places, but was only correct up to 527th (error only found in 1946)
- 1949, ENIAC: 2,037 decimal places

History of Pi



https://en.wikipedia.org/wiki/Chronology_of_computation_of_pi

Two Questions

- How do you write a program that computes the value of Pi?
- How do you test the program you just wrote?

Background

- It is impossible to test a program exhaustively: there are infinitely many inputs, we cannot extrapolate how program will react to all inputs.
- We test the program anyway (better than nothing): test oracles check whether programs behave correctly against sampled inputs
- Oracle: a mechanism that validates the correctness of a program under test.

Oracle Assumption

- The belief that Oracle can determine program correctness and, consequently, without an oracle, we cannot test a program.
- This assumption is violated when:
 - It is not possible to write an oracle, at all, or
 - Oracle may exist, but finding it requires impractical amount of effort
- We call these programs non-testable.

Types of Non-testable Programs

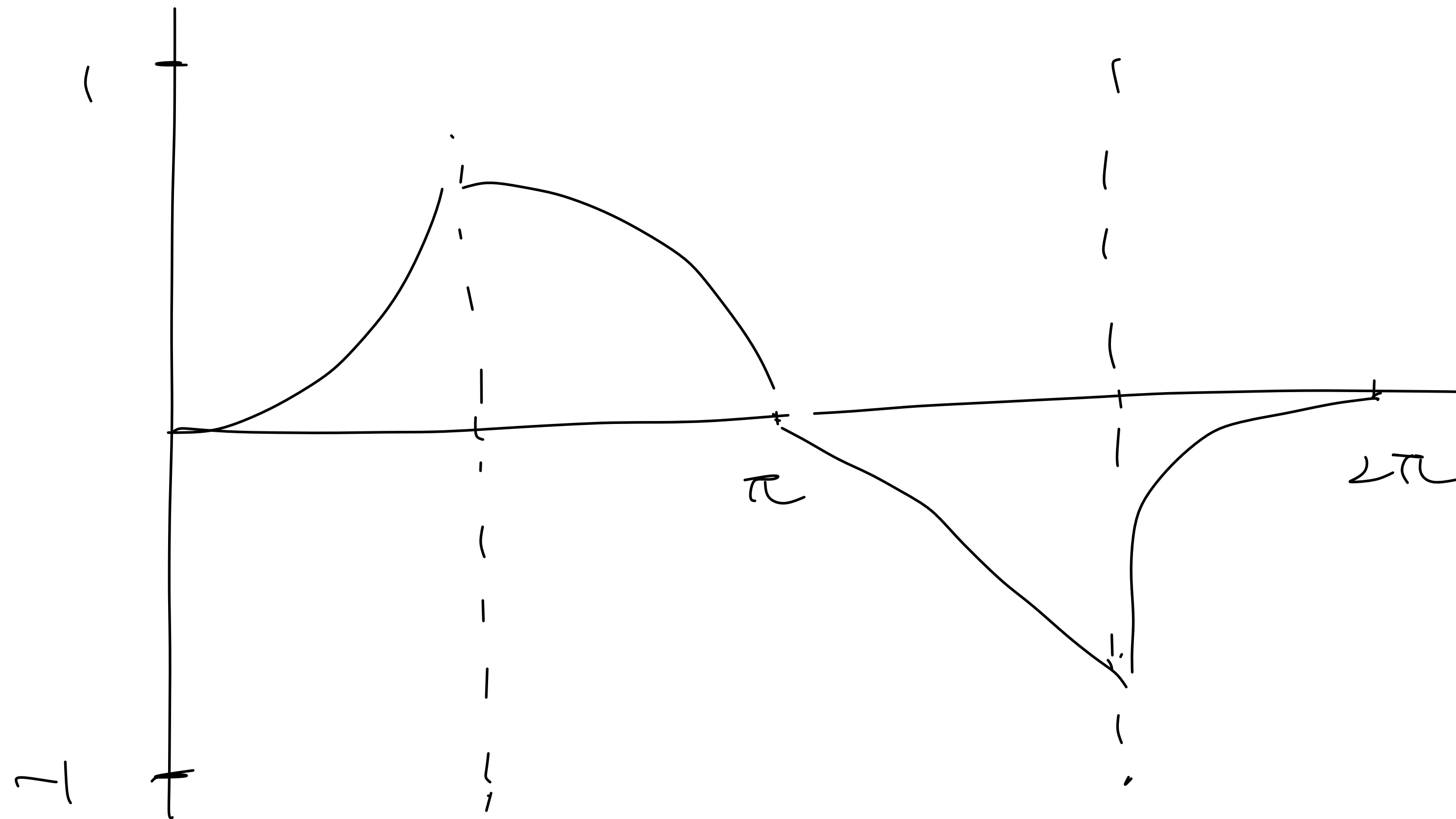
- Type 1: the program was written to determine the answer to a problem we haven't solved - if we knew the oracle, we would've solved the problem!
- Type 2: the program generates so much output that verifying all of them becomes too expensive
- Type 3: the program is misunderstood - testers act as human oracles, but make bad decisions
- From our point of view, type 1 is the most interesting, as it touches on something fundamental about testing.

Rising Number of Non-testable Programs

- Type 1: most scientific computation, certain branches of Artificial Intelligence/Machine Learning
 - What is the “correct” way to play a video game, if you are applying reinforcement learning? What is the correct value of π ? What is the “correct” way of clustering objects in an arbitrary domain?
- Type 2: some other branches of Artificial Intelligence/Machine Learning
 - Summarisation of biomedical literature using Natural Language Processing, Image classification

Suppose you are implementing: `double sin(double x)`

- What can you test, using only your existing knowledge of the function `sin`, without using a table of pre-calculated values?
 - `sin(0)` should be 0
 - `sin($\pi/2$)` should be 1, `sin(π)` should be 0
 - `sin($3\pi/2$)` should be -1, `sin(2π)` should be 0
 - if $0 < x < \pi/2$, `sin(x) < sin(x + ϵ)`, etc



??

What can we do more?

We can use a little bit more domain knowledge:

$$\sin(x) = \alpha \rightarrow \sin(\pi - x) = \alpha$$

Metamorphic Relationship

If the program p has metamorphic relationship f and g :

$$p(i) = r \rightarrow p(f(i)) = g(r)$$

For example, if p is \sin , $f(x) = \pi - x$, $g(x) = x$.

Metamorphic Testing

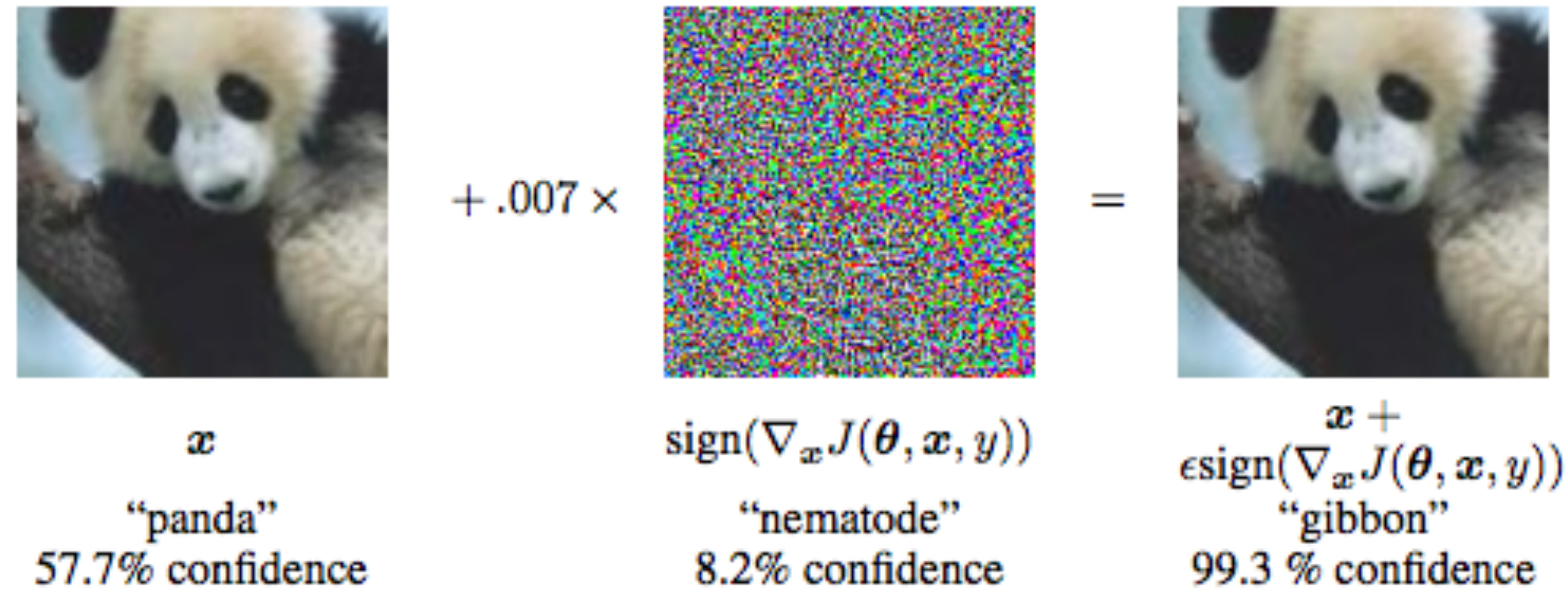
- Use metamorphic relationship to form a pseudo Oracle: existing input-output pairs allow you to predict the input-output pairs using the metamorphic relationship.
- While MT is about input/output relationship, we can also consider this as an “invariant” in a way. For example:
 - Differential testing: if X is implementing spec A, then another implementation Y should agree with X on the same input

Metamorphic Testing: Challenges

- Learning metamorphic relationships: manual identification is too costly - can we automate it?
 - So far, automated MR identification is mostly based on templates. For example: J. Zhang, J. Chen, D. Hao, Y. Xiong, B. Xie, L. Zhang, and H. Mei. Search-based inference of polynomial metamorphic relations, ASE 2014. For example, quadratic:
$$c_1 O_1^2 + c_2 O_1 O_2 + c_3 O_2^2 + d_1 O_1 + d_2 O_2 + e = 0$$
- Non-numeric metamorphic relationships: what can we apply to input of non-numerical types?
 - So far, non-numerical MRs are mostly combinatorial. For example: C. Murphy, K. Shen, and G. Kaiser. Automatic system testing of programs without test oracles. ISSTA 2009 - changes training sets of ML classifiers by: permutating datapoint order, add/multiply constants to datapoint, reverse the order, etc.

**...and then an unexpected
comeback!**

Adversarial Examples



The diagram illustrates the process of creating an adversarial example. It shows three images in a row, separated by mathematical operators. The first image is a clear photo of a panda, labeled x with a confidence of 57.7%. The second image is a square of random noise, labeled $\text{sign}(\nabla_x J(\theta, x, y))$ with a confidence of 8.2%. The third image is the result of adding the noise to the first image, labeled $x + \epsilon \text{sign}(\nabla_x J(\theta, x, y))$ with a confidence of 99.3% for the class "gibbon".

$$\begin{array}{ccc} \text{Image 1} & + .007 \times & \text{Image 2} & = & \text{Image 3} \\ \begin{array}{c} x \\ \text{"panda"} \\ 57.7\% \text{ confidence} \end{array} & & \begin{array}{c} \text{sign}(\nabla_x J(\theta, x, y)) \\ \text{"nematode"} \\ 8.2\% \text{ confidence} \end{array} & & \begin{array}{c} x + \\ \epsilon \text{sign}(\nabla_x J(\theta, x, y)) \\ \text{"gibbon"} \\ 99.3\% \text{ confidence} \end{array} \end{array}$$

Explaining and harnessing adversarial examples, Goodfellow et al., (<https://arxiv.org/abs/1412.6572>)

More importantly, it turns out that you can be much more systematic than adding noise and hoping to get an adversarial example.

Adversarial Examples as Verification Counterexamples



Fig. 1. Automobile
wrong

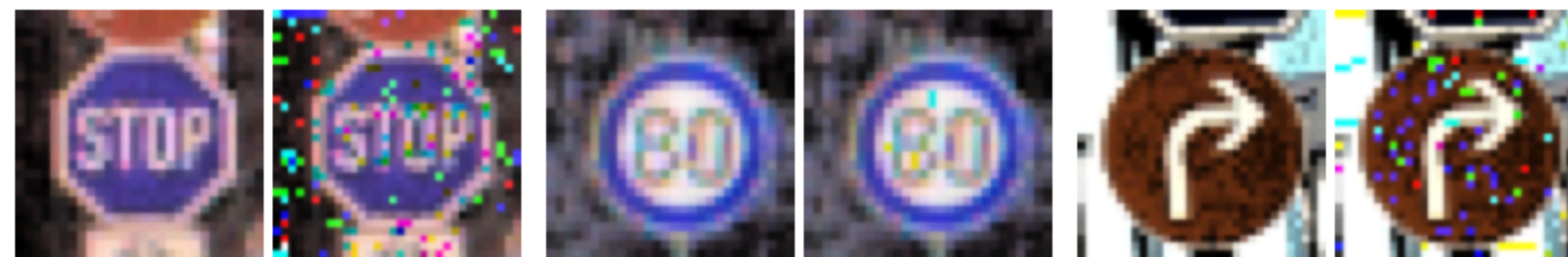
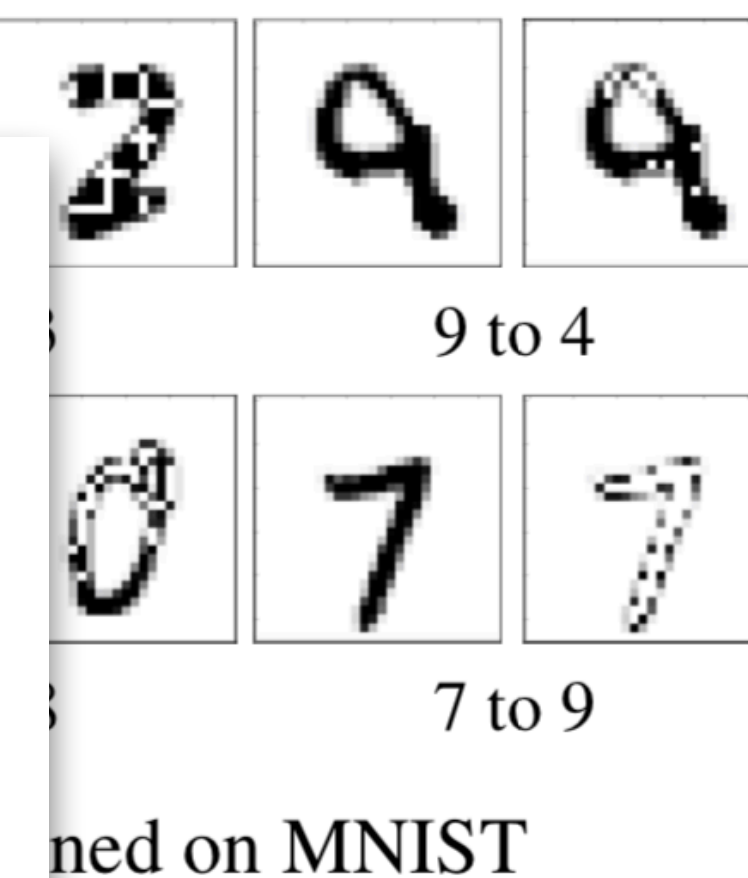


Fig. 11. Adversarial examples for the network trained on the GTSRB dataset by multi-path search



**This is extremely hard to test for.
To begin with, what is the oracle?**

Metamorphic Oracles

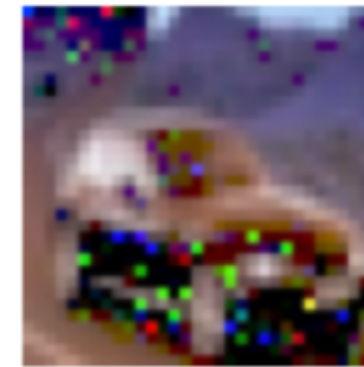
Metamorphic testing is a surprisingly effective conceptual tool for testing DNNs (at least so far).

Given that DNN(



) produces the output “car”,

MT suggests that DNN(



) should also produce the output “car”.

Input MR: images are perceptively identical to human eyes.

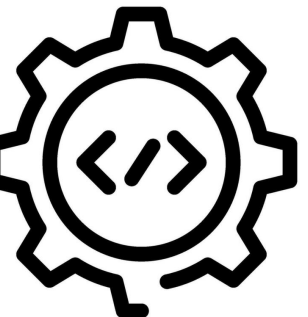
Output MR: class labels should be identical.

Code-Documentation Inconsistency

```
/**
 * Gets the package name from a String.
 * The string passed in is assumed to be a class name - it
 * is not checked.
 * If the class is unpackaged, return an empty string.
 * @param className the className to get the package name
 * for, may be null
 * @return the package name or an empty string
 */
```

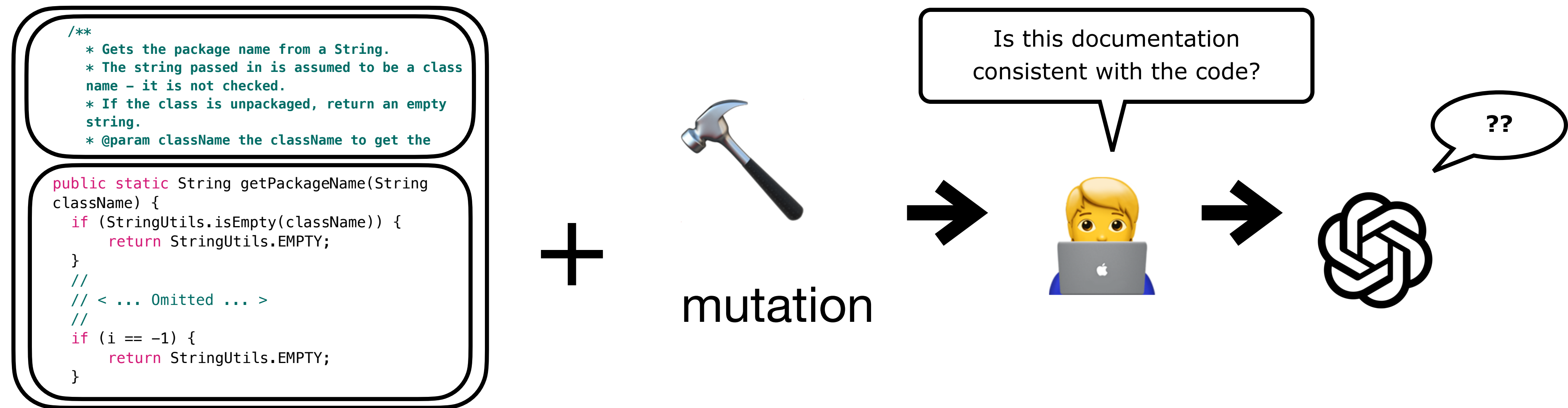


```
public static String getPackageName(String className) {
    if (StringUtils.isEmpty(className)) {
        return StringUtils.EMPTY;
    }
    //
    // < ... Omitted ... >
    //
    if (i == -1) {
        return StringUtils.EMPTY;
    }
    // ⚠ Bug! (should be substring(0, i);)
    return className.substring(1, i);
}
```



- Prone to get out of sync, which in turn harm maintainability of the code
- Requires semantic understanding of code to resolve —> HARD!
- Can LLMs help with this? 🧐

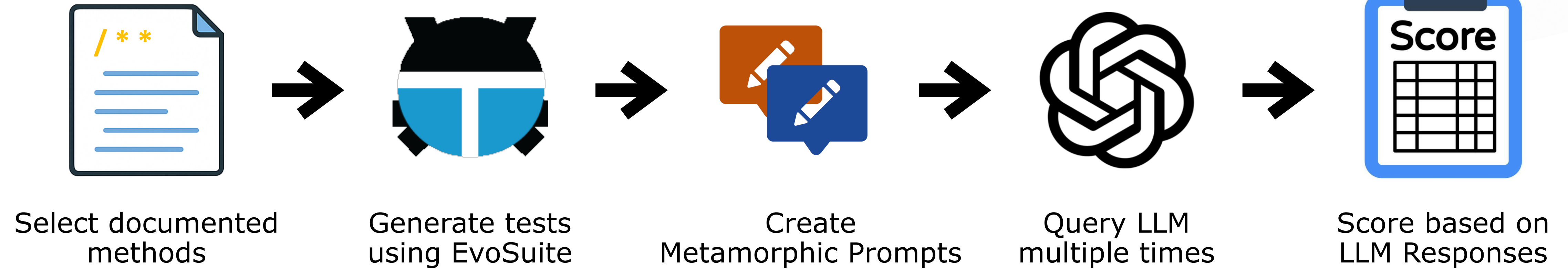
Existing Approach



- Straightforward prompting + mutation to evaluate the capabilities of LLMs to understand the consistency between code and comments
- Good starting point, but can still hallucinate + cannot capture detailed behaviour

MetaMon

Contributions



- We capture dynamic behaviour of target methods using regression test cases generated using EvoSuite.
- We propose metamorphic LLM queries, with metamorphic relations for generated regression test assertions.

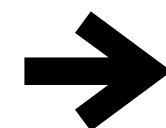
Comparing Assertions to Documentation

...might be easier...?

```
public static String getPackageName(String
className) {
    if (StringUtils.isEmpty(className)) {
        return StringUtils.EMPTY;
    }
    //
    // < ... Omitted ... >
    //
    if (i == -1) {
        return StringUtils.EMPTY;
    }
    // ⚠ Bug! (should be substring(0, i);)
    return className.substring(1, i);
}
```

```
/**
 * Gets the package name from a String.
 * The string passed in is assumed to be a class name - it
 * is not checked.
 * If the class is unpackaged, return an empty string.
 * @param className the className to get the package name
 * for, may be null
 * @return the package name or an empty string
 */
```

Does this code match the documentation?

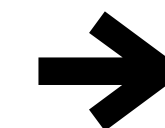


?

```
public void test() throws Throwable {
    String string0 =
        ClassUtils.getPackageName("line.separator");
    assertEquals("ine", string0);
}
```

```
/**
 * Gets the package name from a String.
 * The string passed in is assumed to be a class name - it
 * is not checked.
 * If the class is unpackaged, return an empty string.
 * @param className the className to get the package name
 * for, may be null
 * @return the package name or an empty string
 */
```

Does this assertion match the documentation?



NO!

Metamorphic LLM Query

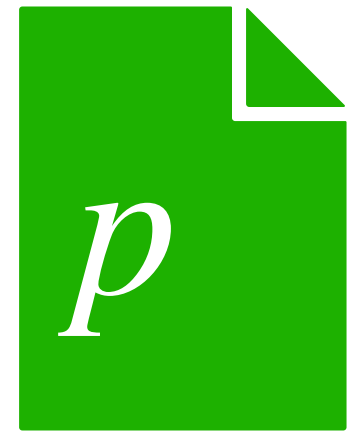
- A generated regression test case assertion is either correct or incorrect 🤖
- If $LLM(p) \rightarrow b$ and $p' = \neg p$, then $LLM(p') \rightarrow \neg b$, where p is a regression test assertion predicate, and b is whether the assertion is consistent with the documentation or not.
- For this work, we only target assertions whose semantics are boolean and thus can be easily flipped.

TABLE I: Oracle transformations based on MR

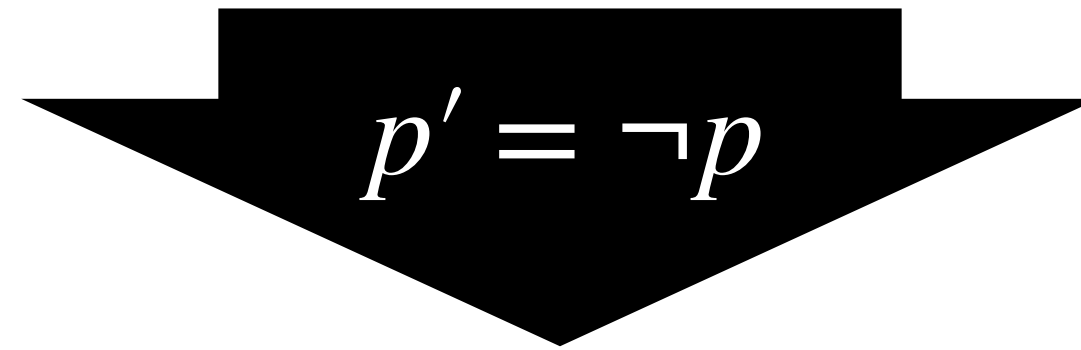
Transformation	Description
MR_T2F	Replacing assertTrue to assertFalse
MR_F2T	Replacing assertFalse to assertTrue
MR_N2NN	Replacing assertNull to assertNotNull
MR_NN2N	Replacing assertNotNull to assertNull
MR_E2NE	Replacing assertEquals to assertNotEquals
MR_NE2E	Replacing assertNotEquals to assertEquals
MR_S2NS	Replacing assertSame to assertNotSame
MR_NS2S	Replacing assertNotSame to assertSame

Metamorphic LLM Query

Original
Query



$$\rightarrow b = LLM(p), s_{orig}(b) = \begin{cases} +1 & \text{if } b = \text{correct} \\ 0 & \text{if } b = \text{don't know} \\ -1 & \text{if } b = \text{incorrect} \end{cases}$$



Metamorphic
Query



$$\rightarrow b = LLM(p'), s_{meta}(b) = \begin{cases} +1 & \text{if } b = \text{incorrect} \\ 0 & \text{if } b = \text{don't know} \\ -1 & \text{if } b = \text{correct} \end{cases}$$

$$score = \sum_{p \in P_{orig}} score_{orig}(p) + \sum_{p' \in P_{meta}} score_{meta}(p'),$$

Experimental Settings

- Dataset & Setup
 - Defects4j v2.0.1: 5 open source Java projects (Chart, Closure, Lang, Math, Time)
 - 9,482 method-documentation pairs
 - Test generation Tool: EvoSuite v1.0.7
 - LLM model: GPT-3.5-Turbo-0613
 - Mutant generation Tool: Major v1.3.4

Projects	# Mutants	# Test		
		w/ incorrect oracle	w/ correct oracle	Total
Chart	11,589	2,684	2,684	5,368
Closure	343	93	93	186
Lang	4,723	594	594	1,188
Math	11,168	740	740	1,480
Time	1,983	630	630	1,260
Total	29,806	4,741	4,741	9,482

Prompt consisted of <Test w/ **incorrect** oracle, Doc>
→ Program behaviour & doc is **inconsistent**

Prompt consisted of <Test w/ **correct** oracle, Doc>
→ Program behaviour & doc is **consistent**

RQ1. Effectiveness

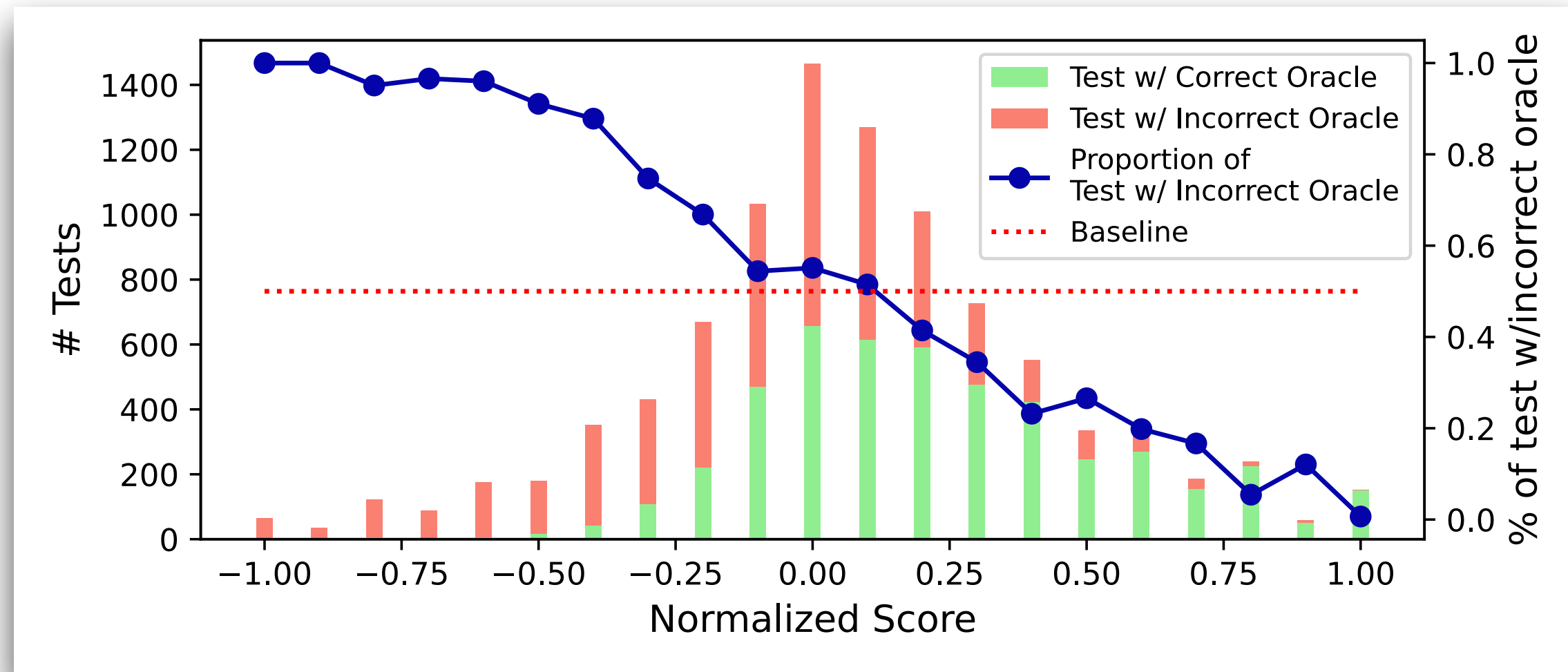
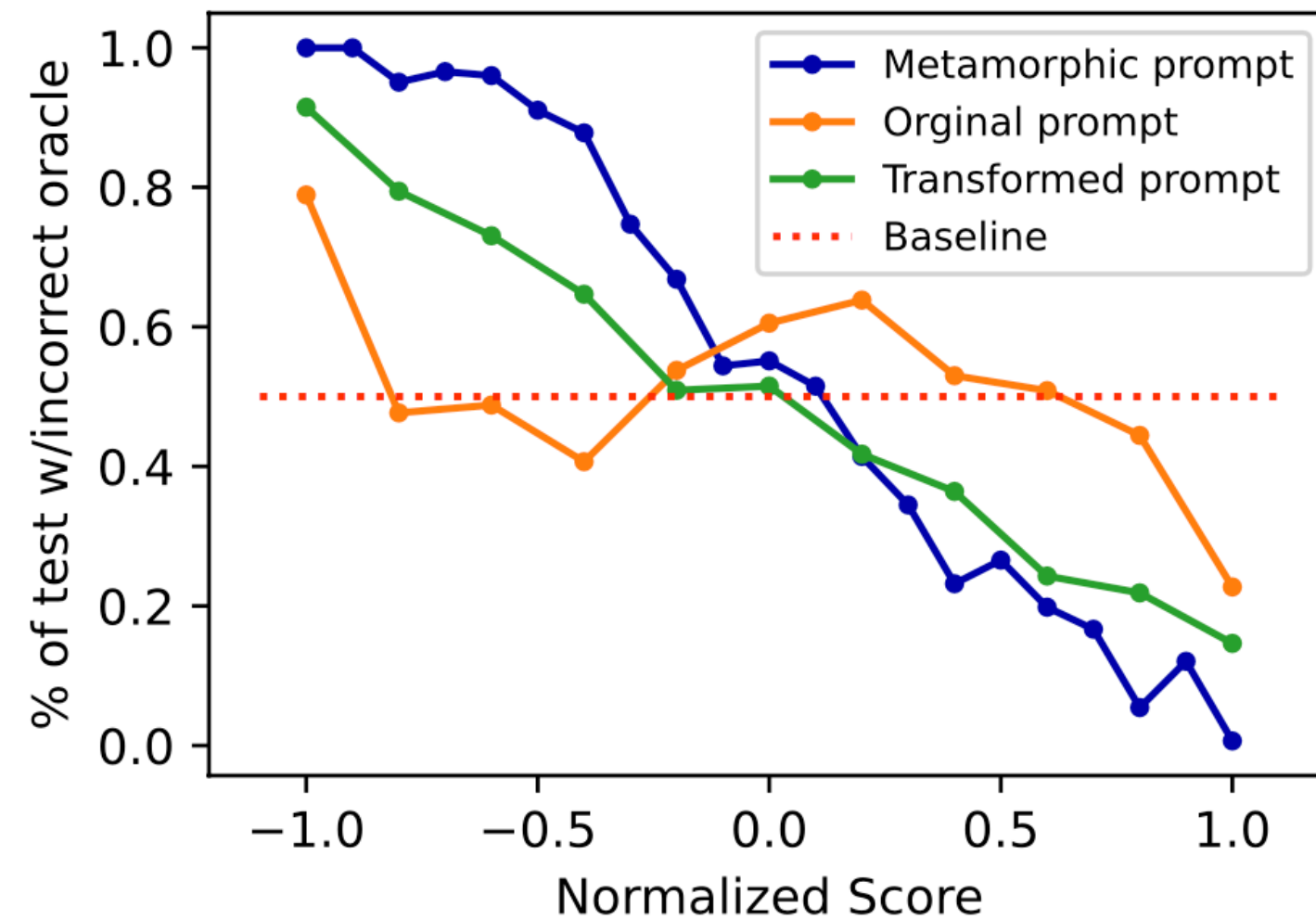


TABLE III: Precision, Recall, and F1 at different thresholds

Score	Pre.	Rec.	F1	Score	Pre.	Rec.	F1
≤ -0.1	0.722	0.480	0.576	≤ -0.6	0.967	0.099	0.180
≤ -0.2	0.808	0.361	0.499	≤ -0.7	0.971	0.064	0.120
≤ -0.3	0.873	0.267	0.409	≤ -0.8	0.973	0.046	0.087
≤ -0.4	0.926	0.199	0.328	≤ -0.9	1.000	0.021	0.042
≤ -0.5	0.952	0.134	0.235	≤ -1.0	1.000	0.014	0.027

METAMON is effective at detecting inconsistency between code and documentations at extreme ends of scores, with precision of 0.722 for $score \leq -0.1$ and 1.0 for $score \leq 0.9$.

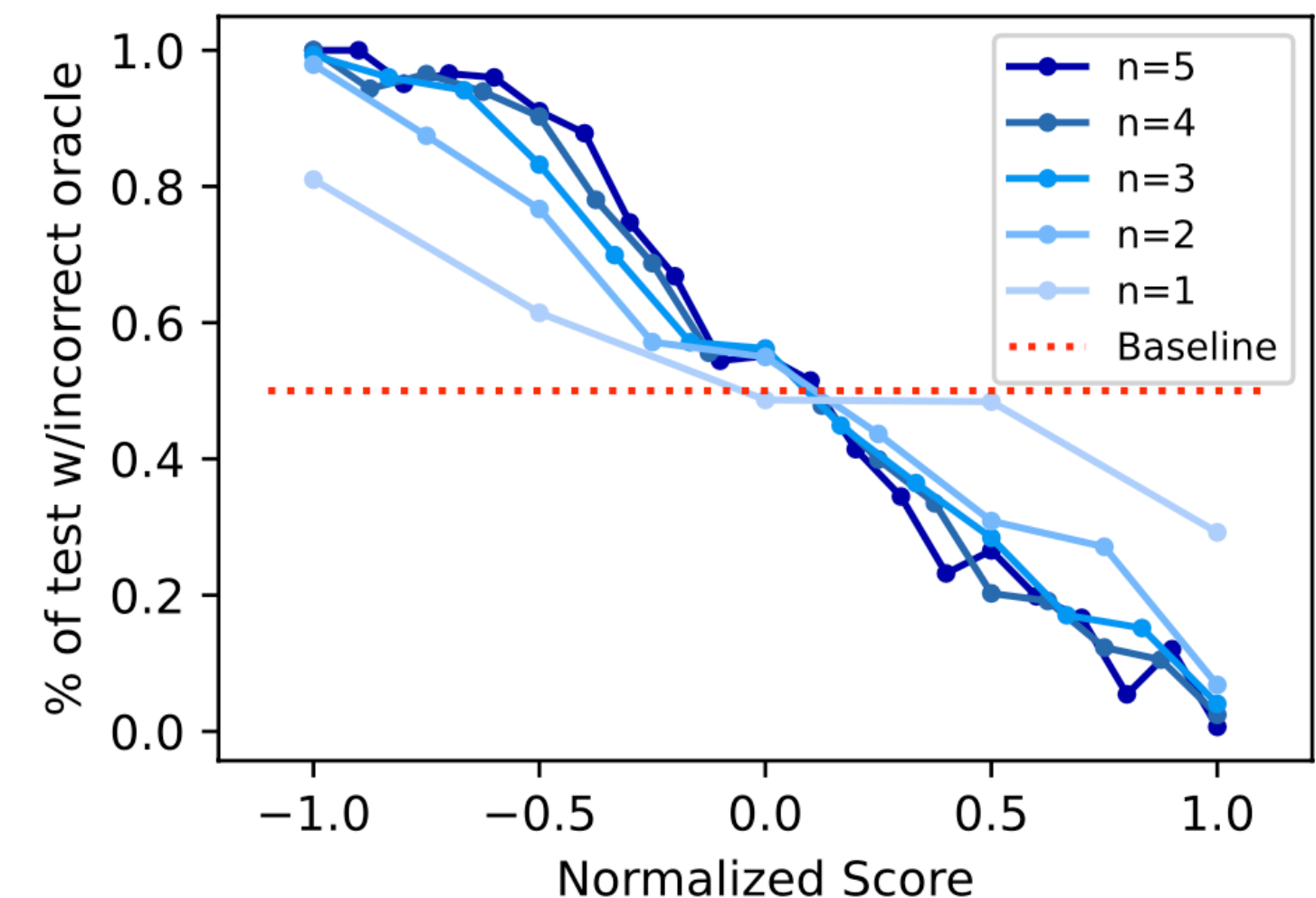
RQ2. Ablation



(a) metamorphic

Increasing the number of LLM queries improves performance:
→ More queries lead to **stronger distinction** between correct and incorrect oracles.

Metamorphic prompting enables METAMON to reliably associate low scores with incorrect oracles.
→ Without it, the results become **nearly indistinguishable** from random judgements.



(b) self-consistency

Summary

- Non-testable programs are the programs for which it is very difficult to generate oracles. In some cases, they are the programs written to solve problems, whose correct answers we do not know.
- Metamorphic relationship allows you to formulate a semi-oracle for non-testable programs.
- Incidentally, most of the learning tasks are non-testable: *this is the bleeding edge*.