Testing

CS 485/698: AI-Assisted SE

Today’s Agenda

- Team meeting (~15 minutes)
  - Sprint retro/planning: how did P4 go? What will you be testing for P5?
  - Michael and I will be coming by to check in with your teams
- Testing basics lecture
- In-class activity: testing testing strategies with AI tinder

What is testing?

Definition: a ***test*** executes a given input on a program (the ***system under test*** or ***SUT***) and compares the SUT's output to a given oracle

What is testing?

Definition: a ***test*** executes a given input on a program (the ***system under test*** or ***SUT***) and compares the SUT's output to a given oracle

Aside: testing is the canonical example of a ***dynamic analysis***, which is program analysis that requires running the program

What is testing?

Definition: a ***test*** executes a given input on a program (the ***system under test*** or ***SUT***) and compares the SUT's output to a given oracle

```
./prog < input > output && diff output oracle
```

What is testing?

Definition: a ***test*** executes a given input on a program (the ***system under test*** or ***SUT***) and compares the SUT's output to a given oracle

```
./prog < input > output && diff output oracle
```

SUT

What is testing?

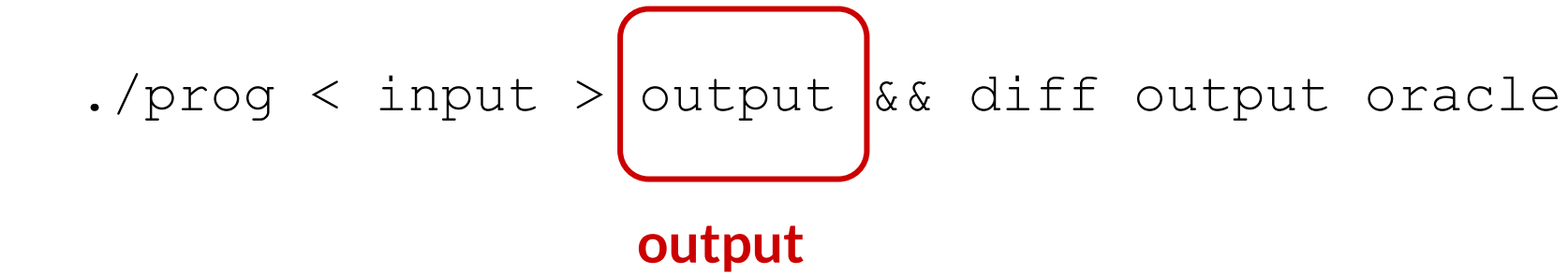
Definition: a ***test*** executes a **given input** on a program (the ***system under test*** or ***SUT***) and compares the SUT's output to a given oracle

```
./prog < input > output && diff output oracle
```

input

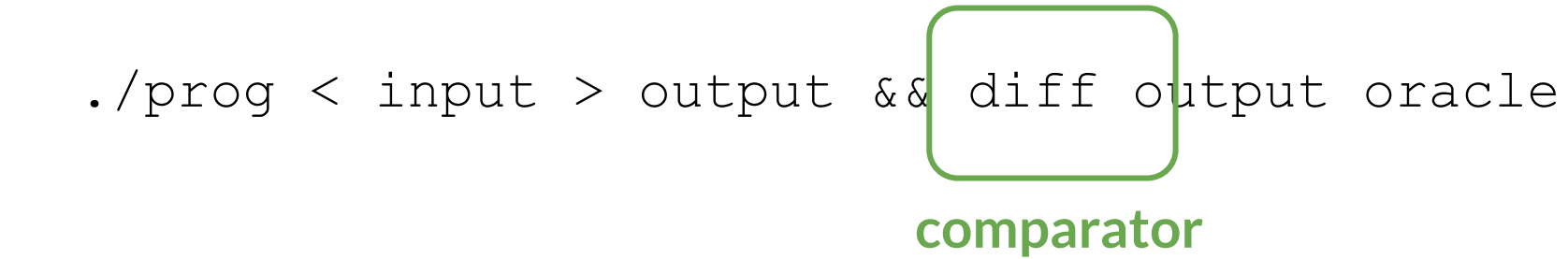
What is testing?

Definition: a ***test*** executes a **given input** on a program (the ***system under test*** or ***SUT***) and compares the SUT's **output** to a given oracle



What is testing?

Definition: a ***test*** executes a **given input** on a program (the ***system under test*** or ***SUT***) and **compares** the SUT's **output** to a given oracle



What is testing?

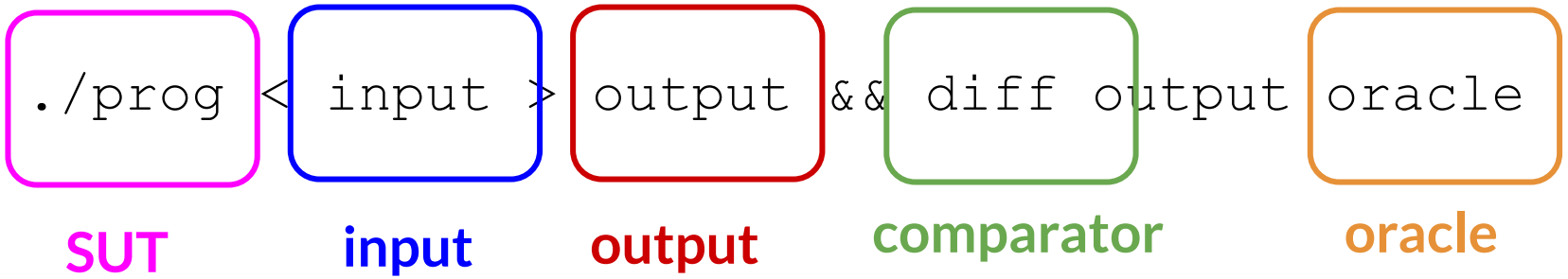
Definition: a ***test*** executes a **given input** on a program (the ***system under test*** or ***SUT***) and **compares** the SUT's **output** to a given **oracle**

```
./prog < input > output && diff output oracle
```

oracle

What is testing?

Definition: a ***test*** executes a **given input** on a program (the ***system under test*** or ***SUT***) and **compares** the SUT's **output** to a given **oracle**



Building a test case

- You usually know the SUT

Building a test case

- You usually know the SUT
- **You choose** inputs (**how?**)

Building a test case

- You usually know the SUT
- **You choose** inputs (**how?**)
- Run the SUT on the chosen inputs to produce the output

Building a test case

- You usually know the SUT
- **You choose** inputs (**how?**)
- Run the SUT on the chosen inputs to produce the output
- **You choose** the comparator (**how?**)

Building a test case

- You usually know the SUT
- **You choose** inputs (**how?**)
- Run the SUT on the chosen inputs to produce the output
- **You choose** the comparator (**how?**)
- **You choose** the oracle (**how?**)

Building a test case

- You usually know the SUT
- **You choose** inputs (**how?**)
- Run the SUT on the chosen inputs to produce the output
- **You choose** the comparator (**how?**)
- **You choose** the oracle (**how?**)

Ideal situation: you can test every input (“**exhaustive testing**”)

Building a test case

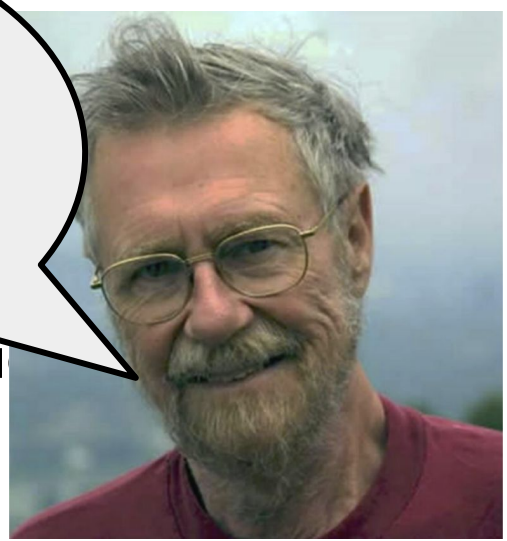
- You usually know the SUT
- **You choose** inputs (**how?**)
- Run the SUT on the chosen inputs to produce the output
- **You choose** the comparator (**how?**)
- **You choose** the oracle (**how?**)

Ideal situation: you can test every input ("**exhaustive testing**")

- in practice, rarely possible: **input space is too large**

Building a test cas

- You usually know th
- **You choose** inputs (**ho**
- Run the SUT on the chosen inp produ
- **You choose** the comparator (**how?**)
- **You choose** the oracle (**how?**)



Ideal situation: you can test every input (“**exhaustive testing**”)

- in practice, rarely possible: **input space is too large**

Choosing a comparator

- Most common: **exact match** (often a good choice!)

Choosing a comparator

- Most common: **exact match** (often a good choice!)
- Also common:
  - **over-approximation** (“is the output one of these good values”, or, more commonly, “is there any output at all”)
  - **under-approximation** (“does the output contain this expected value”)

Choosing a comparator

- Most common: **exact match** (often a good choice!)
- Also common:
  - **over-approximation** (“is the output one of these good values”, or, more commonly, “is there any output at all”)
  - **under-approximation** (“does the output contain this expected value”)
- But, could be an **arbitrarily-complex boolean** function
  - must be boolean, because test needs to either **pass** or **fail**

Choosing a comparator

- Most common: **exact match** (often a good choice!)
- Also common:
  - **over-approximation** (“is the outp
    values”, or, more commonly, “is th
  - **under-approximation** (“does the
    expected value”)
- But, could be an **arbitrarily-complex boolean** function
  - must be boolean, because test needs to either **pass** or **fail**

Can we use an LLM as the comparator? What might go wrong if we did?

Aside: flaky tests

Definition: a ***flaky*** test fails non-deterministically: that is, they sometimes pass and sometimes fail

Aside: flaky tests

Definition: a ***flaky*** test fails non-deterministically: that is, they sometimes pass and sometimes fail

- sometimes caused by brittleness (e.g., relying on the network)

Aside: flaky tests

Definition: a ***flaky*** test fails non-deterministically: that is, they sometimes pass and sometimes fail

- sometimes caused by brittleness (e.g., relying on the network)
- sometimes caused by **non-determinism** in the program itself or in the test infrastructure
    - e.g., relying on randomness, ***LLM output***, iteration order, etc.

Aside: flaky tests

Definition: a ***flaky*** test fails non-deterministically: that is, they sometimes pass and sometimes fail

- sometimes caused by brittleness (e.g., relying on the network)
- sometimes caused by **non-determinism** in the program itself or in the test infrastructure
  - e.g., relying on randomness, ***LLM output***, iteration order, etc.
- are a **major problem in practice**
  - difficult to debug, so waste a lot of developer time
  - detecting them is an active research area

Choosing an oracle

Choosing an oracle

- As a human, you get this from the **specification**
  - ask yourself: “what should the program do with this input?”

Choosing an oracle

- As a human, you get this from the **specification**
  - ask yourself: “what should the program do with this input?”
- Traditionally, choosing oracles automatically is considered **implausibly difficult**
  - big promise of LLMs in testing: the LLM can pick the oracle

Choosing an oracle

- As a human, you get this from the **specification**
  - ask yourself: “what should the program do with this input?”
- Traditionally, choosing oracles automatically is considered **implausibly difficult**
  - big promise of LLMs in testing: the LLM can pick the oracle
- However, the ***intent*** of the human operator still needs to be conveyed somehow
  - *“The act of describing a program in unambiguous detail and the act of programming are one and the same”* (Kevlin Henney)

Choosing inputs

- When writing tests by hand, this is often the hardest part

Choosing inputs

- When writing tests by hand, this is often the hardest part
- Strategies:
  - choose at random (avoid when writing by hand)
  - edge cases
  - partition testing
  - white-box testing
  - black-box testing

Choosing “at random”

- Computers can pick inputs **very fast** (given some policy)
  - What’s an appropriate policy?

Choosing “at random”

- Computers can pick inputs **very fast** (given some policy)
  - What’s an appropriate policy?
    - pick “at random” (***fuzzing***)

Aside: fuzzing and random testing

Definition: ***fuzzing*** (or ***fuzz testing***) is an automated testing technique that involves providing random or semi-random inputs to a program and monitoring for violations of an implicit oracle.

Aside: fuzzing and random testing

Definition: ***fuzzing*** (or ***fuzz testing***) is an automated testing technique that involves providing random or semi-random inputs to a program and monitoring for violations of an implicit oracle.

- typical oracle: **crashes**

Aside: fuzzing and random testing

Definition: ***fuzzing*** (or ***fuzz testing***) is an automated testing technique that involves providing random or semi-random inputs to a program and monitoring for violations of an implicit oracle.

- typical oracle: **crashes**
- modern fuzzers can deal with **structured input**, given seeds

Aside: fuzzing and random testing

Definition: ***fuzzing*** (or ***fuzz testing***) is an automated testing technique that involves providing random or semi-random inputs to a program and monitoring for violations of an implicit oracle.

- typical oracle: **crashes**
- modern fuzzers can deal with **structured input**, given seeds
- fuzzing is **common in industry**
  - AFL (most famous coverage-guided fuzzer) was built at Google
  - oss-fuzz project fuzzes many important open-source projects constantly using industry resources

Choosing “at random”

- Computers can pick inputs **very fast** (given some policy)
  - What’s an appropriate policy?
    - pick “at random” (***fuzzing***)
    - pick using logic (***symbolic execution***)

Choosing “at random”

- Computers can pick inputs **very fast** (given some policy)
  - What’s an appropriate policy?
    - pick “at random” (***fuzzing***)
    - pick using logic (***symbolic execution***)
- Can also pick using an LLM; LLMs generally use **same strategies as humans**
  - so let’s review human strategies so that you can use them in your prompting

Choosing inputs

- When writing tests by hand, this is often the hardest part
- Strategies:
  - choose at random (avoid when writing by hand)
  - **edge cases**
  - partition testing
  - white-box testing
  - black-box testing

Edge case examples:

- 0, 1, -1
- null
- empty list
- empty file
- etc.

Choosing inputs

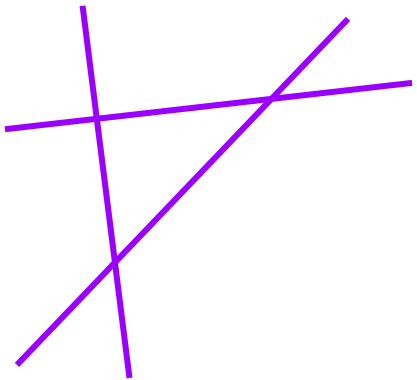
- When writing tests by hand, this is often the hardest part
- Strategies:
  - choose at random (avoid when writing by hand)
  - edge cases
  - **partition testing**
  - white-box testing
  - black-box testing

Partition testing

Key idea: split up the input space into redundant “regions”

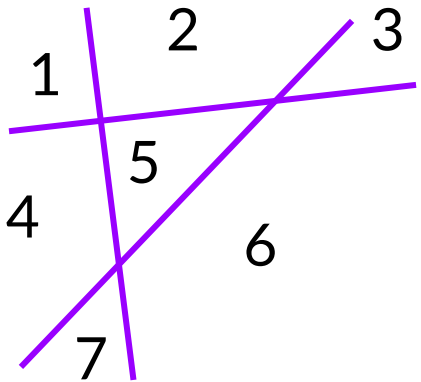
Partition testing

Key idea: split up the input space into redundant “regions”



Partition testing

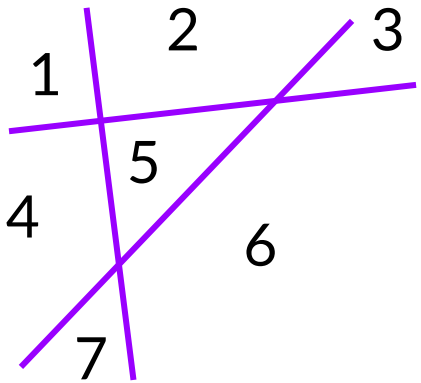
Key idea: split up the input space into redundant “regions”



- write one test **for each region**

Partition testing

Key idea: split up the input space into redundant “regions”

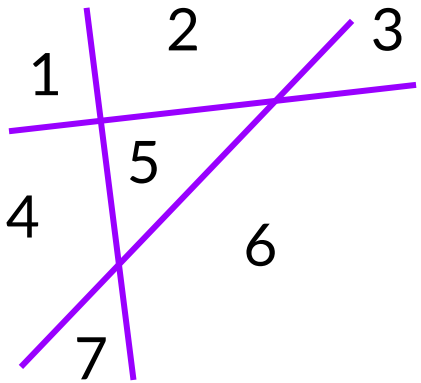


- write one test **for each region**
- possible ways to split up the input:
  - parity (even, odd)
  - positive, negative, zero
  - jpg files vs png files
  - correctly-formatted input vs incorrectly-formatted input

Partition testing

Key idea: split up the input space into

Common technique: split up input space k ways, write 2^k tests



- write one test **for each region**
- possible ways to split up the input:
  - parity (even, odd)
  - positive, negative, zero
  - jpg files vs png files
  - correctly-formatted input vs incorrectly-formatted input

Choosing inputs

- When writing tests by hand, this is often the hardest part
- Strategies:
  - choose at random (avoid when writing by hand)
  - edge cases
  - partition testing
  - **white-box testing**
  - black-box testing

White-box testing

Key idea: choose test inputs based on the program's source code

White-box testing

Key idea: choose test inputs based on the program's source code

- for example, if there is a condition like `if (x > 10) { … }`, consider choosing 9 and 11 for `x`

White-box testing

Key idea: choose test inputs based on the program's source code

- for example, if there is a condition like `if (x > 10) { … }`, consider choosing 9 and 11 for `x`

Advantages:

White-box testing

Key idea: choose test inputs based on the program's source code

- for example, if there is a condition like `if (x > 10) { … }`, consider choosing 9 and 11 for `x`

Advantages:

- relies primarily on your programming skill
- lets us achieve high coverage

White-box testing

Key idea: choose test inputs based on the program's source code

- for example, if there is a condition like `if (x > 10) { … }`, consider choosing 9 and 11 for `x`

Advantages:

- relies primarily on your programming skill
- lets us achieve high coverage

Disadvantages:

White-box testing

Key idea: choose test inputs based on the program's source code

- for example, if there is a condition like `if (x > 10) { … }`, consider choosing 9 and 11 for `x`

Advantages:

- relies primarily on your programming skill
- lets us achieve high coverage

Disadvantages:

- you have to actually read the code
- easy to accidentally "bias" yourself towards what the code already does

Choosing inputs

- When writing tests by hand, this is often the hardest part
- Strategies:
  - choose at random (avoid when writing by hand)
  - edge cases
  - partition testing
  - white-box testing
  - **black-box testing**

Black-box testing

Key idea: don't look at what the code does at all! Instead, only consider the code's **interface**: the inputs and outputs.

Black-box testing

Key idea: don't look at what the code does at all! Instead, only consider the code's **interface**: the inputs and outputs.

- key advantage over white-box testing: you aren't biased by the implementation

Black-box testing

Key idea: don't look at what the code does at all! Instead, only consider the code's **interface**: the inputs and outputs.

- key advantage over white-box testing: you aren't biased by the implementation
- key disadvantage vs white-box testing: you can't tell how well you're covering the implementation, only the specification
    - hard to choose edge cases, get high coverage, etc.

Black-box vs white-box with LLMs

- **Question:** How do you control whether you are doing “white-box” or “black-box” testing when you’re using an LLM to generate test cases?

Black-box vs white-box with LLMs

- **Question:** How do you control whether you are doing “white-box” or “black-box” testing when you’re using an LLM to generate test cases?
  - **Answer:** ***CONTEXT***. Is the code in the prompt or not?

Black-box vs white-box with LLMs

- **Question:** How do you control whether you are doing “white-box” or “black-box” testing when you’re using an LLM to generate test cases?
  - **Answer:** ***CONTEXT***. Is the code in the prompt or not?
- LLMs can suffer from the same biases as humans when conducting white-box testing
  - In fact, early evidence suggests that LLMs may actually be *more* impacted: “my AI-generated tests always pass”

In-Class Activity: Testing AI Tinder

- I gave you a slip of paper with a PR number and an instruction
  - e.g., if your paper says “PR #19”, you will test https://github.com/kelloggm/ai-tinder-fork/pull/19
  - join the group for your PR, help each other configure test infrastructure

In-Class Activity: Testing AI Tinder

- I gave you a slip of paper with a PR number and an instruction
  - e.g., if your paper says “PR #19”, you will test https://github.com/kelloggm/ai-tinder-fork/pull/19
  - join the group for your PR, help each other configure test infrastructure
- Prompt an LLM to generate tests based on your instruction
  - e.g., if your instruction says “PARTITION”, try to get the LLM to do partition testing
  - feel free to discuss with others in your group/help each other

In-Class Activity: Discussion

- While we discuss the in-class activity, **open a PR** with whatever tests you've created during class today
  - target branch should be the one in your PR
  - completion grade for opening a PR by the end of class

In-Class Activity: Discussion

- While we discuss the in-class activity, **open a PR** with whatever tests you've created during class today
  - target branch should be the one in your PR
  - completion grade for opening a PR by the end of class
- **Discussion questions:**
  - Which strategies worked surprisingly well or poorly?
  - Did the LLM-generated tests match your intent?
  - How much did the quality of the tests generated by different members of your group differ?
    - What caused the differences? Prompts? Models?

Wrapup and Reminders

- Sign up for A5