CEN 5016: Software Engineering

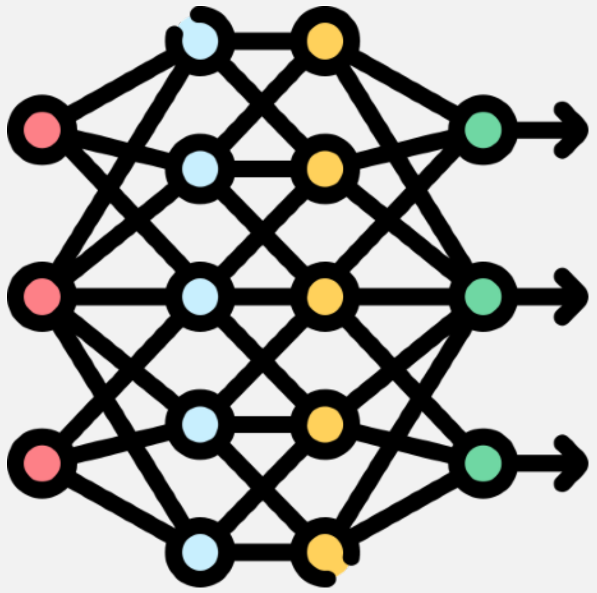
Fall 2025

University of Central Florida

Dr. Kevin Moran

Week 6 - Class I:

A Software Engineer's Guide to LLMs



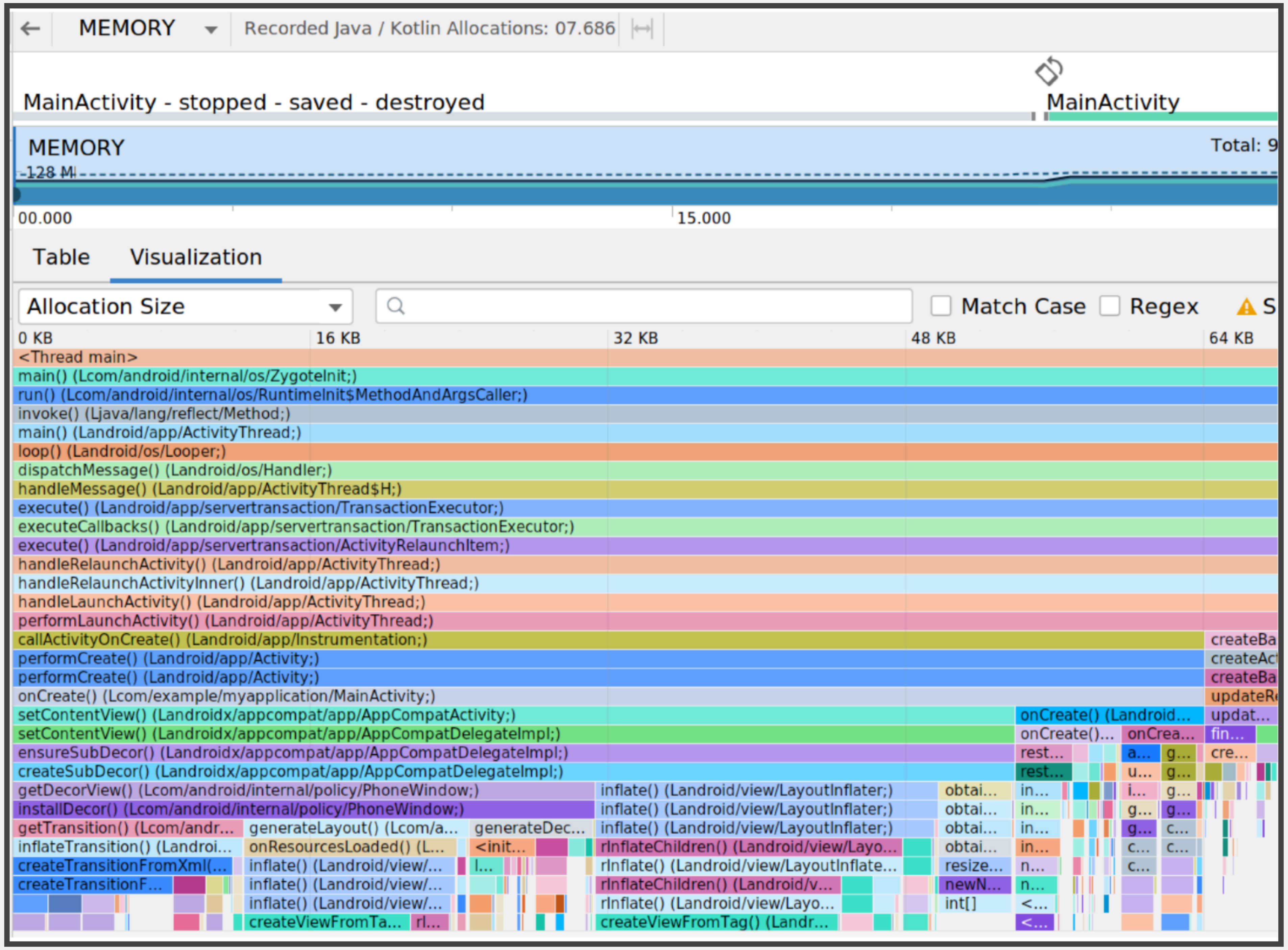
Administrivia

- *Assignment 3*
  - Due Today
  - Deploying and modifying a simple web app
  - Sign up for GitHub Classroom right now!!!!
- *SDE Project Part 1*
  - Due today!
  - Two parts:
    - Team Contract
    - Initial Project Backlog
  - *Lecture Recordings*
    - Will be posted today!

Software QA: Static & Dynamic Analysis

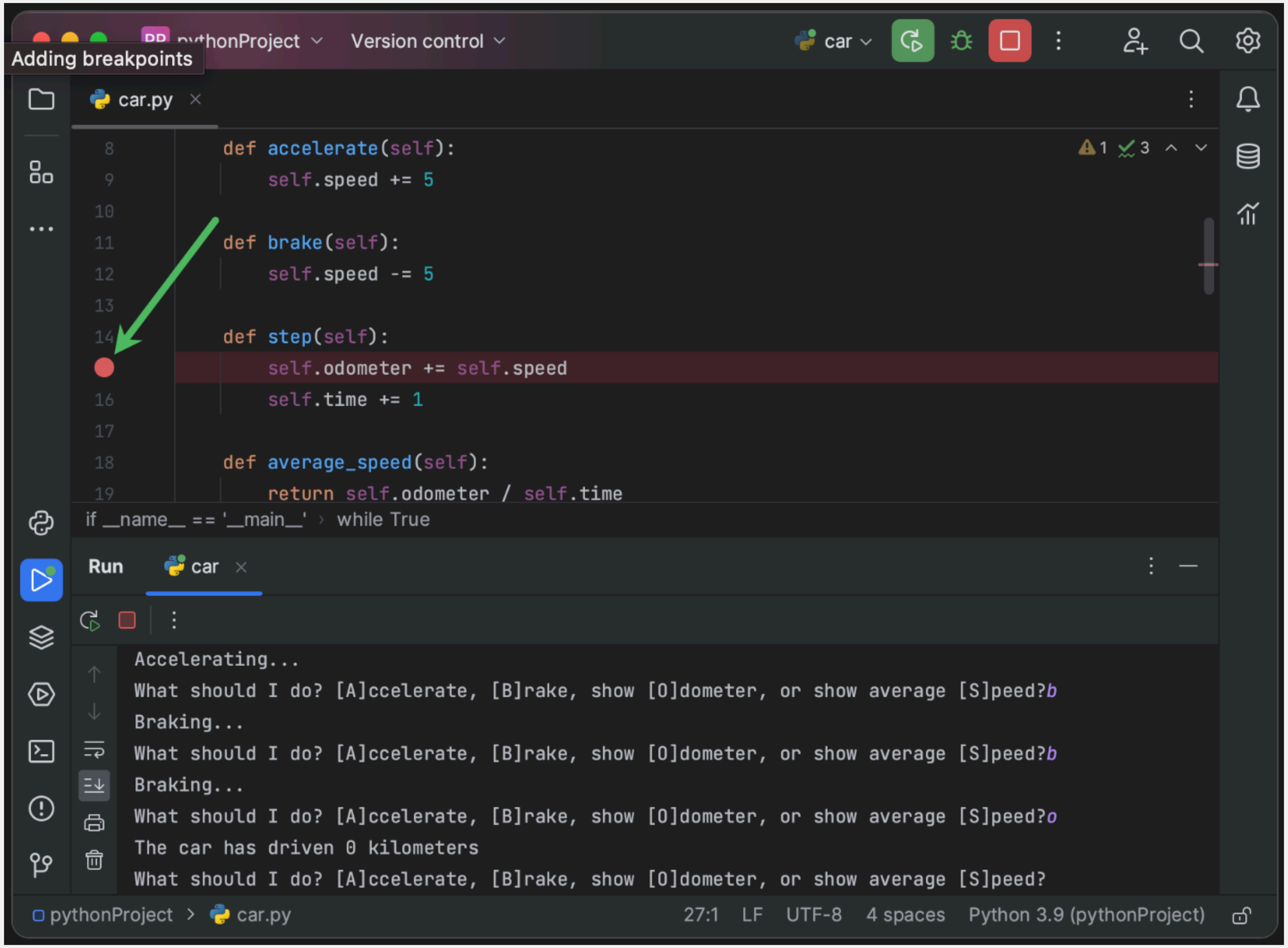
Dynamic Analysis

Android Memory Profiler


MEMORY
Recorded Java / Kotlin Allocations: 07.686
MainActivity - stopped - saved - destroyed
MainActivity
MEMORY
Total: 9
-128 M
00.000
15.000
Table
Visualization
Allocation Size
Match Case
Regex
0 KB
16 KB
32 KB
48 KB
64 KB
<Thread main>
main() (Lcom/android/internal/os/ZygoteInit;)
run() (Lcom/android/internal/os/RuntimeInit$MethodAndArgsCaller;)
invoke() (Ljava/lang/reflect/Method;)
main() (Landroid/app/ActivityThread;)
loop() (Landroid/os/Looper;)
dispatchMessage() (Landroid/os/Handler;)
handleMessage() (Landroid/app/ActivityThread$H;)
execute() (Landroid/app/servertransaction/TransactionExecutor;)
executeCallbacks() (Landroid/app/servertransaction/TransactionExecutor;)
execute() (Landroid/app/servertransaction/ActivityRelaunchItem;)
handleRelaunchActivity() (Landroid/app/ActivityThread;)
handleRelaunchActivityInner() (Landroid/app/ActivityThread;)
handleLaunchActivity() (Landroid/app/ActivityThread;)
performLaunchActivity() (Landroid/app/ActivityThread;)
callActivityOnCreate() (Landroid/app/Instrumentation;)
performCreate() (Landroid/app/Activity;)
performCreate() (Landroid/app/Activity;)
onCreate() (Lcom/example/myapplication/MainActivity;)
setContentView() (Landroidx/appcompat/app/AppCompatActivity;)
setContentView() (Landroidx/appcompat/app/AppCompatDelegateImpl;)
ensureSubDecor() (Landroidx/appcompat/app/AppCompatDelegateImpl;)
createSubDecor() (Landroidx/appcompat/app/AppCompatDelegateImpl;)
getDecorView() (Lcom/android/internal/policy/PhoneWindow;)
installDecor() (Lcom/android/internal/policy/PhoneWindow;)
getTransition() (Lcom/andr...
inflateTransition() (Landroi...
createTransitionFromXml(...
createTransitionF...
generateLayout() (Lcom/a...
onResourcesLoaded() (L...
inflate() (Landroid/view/...
inflate() (Landroid/view/...
inflate() (Landroid/view/...
createViewFromTa...
generateDec...
<init...
inflate() (Landroid/view/LayoutInflater;)
inflate() (Landroid/view/LayoutInflater;)
inflate() (Landroid/view/LayoutInflater;)
rInflateChildren() (Landroid/view/Layo...
rInflate() (Landroid/view/LayoutInflate...
rInflateChildren() (Landroid/v...
rInflate() (Landroid/view/Layo...
createViewFromTag() (Landr...
obtai...
resize...
newN...
int[]
onCreate() (Landroid...
onCreate()...
onCrea...
rest...
createBa
createAct
createBa
updateR
updat...
fin...
cre...

https://developer.android.com/studio/profile/memory-profiler

Pycharm Debugger

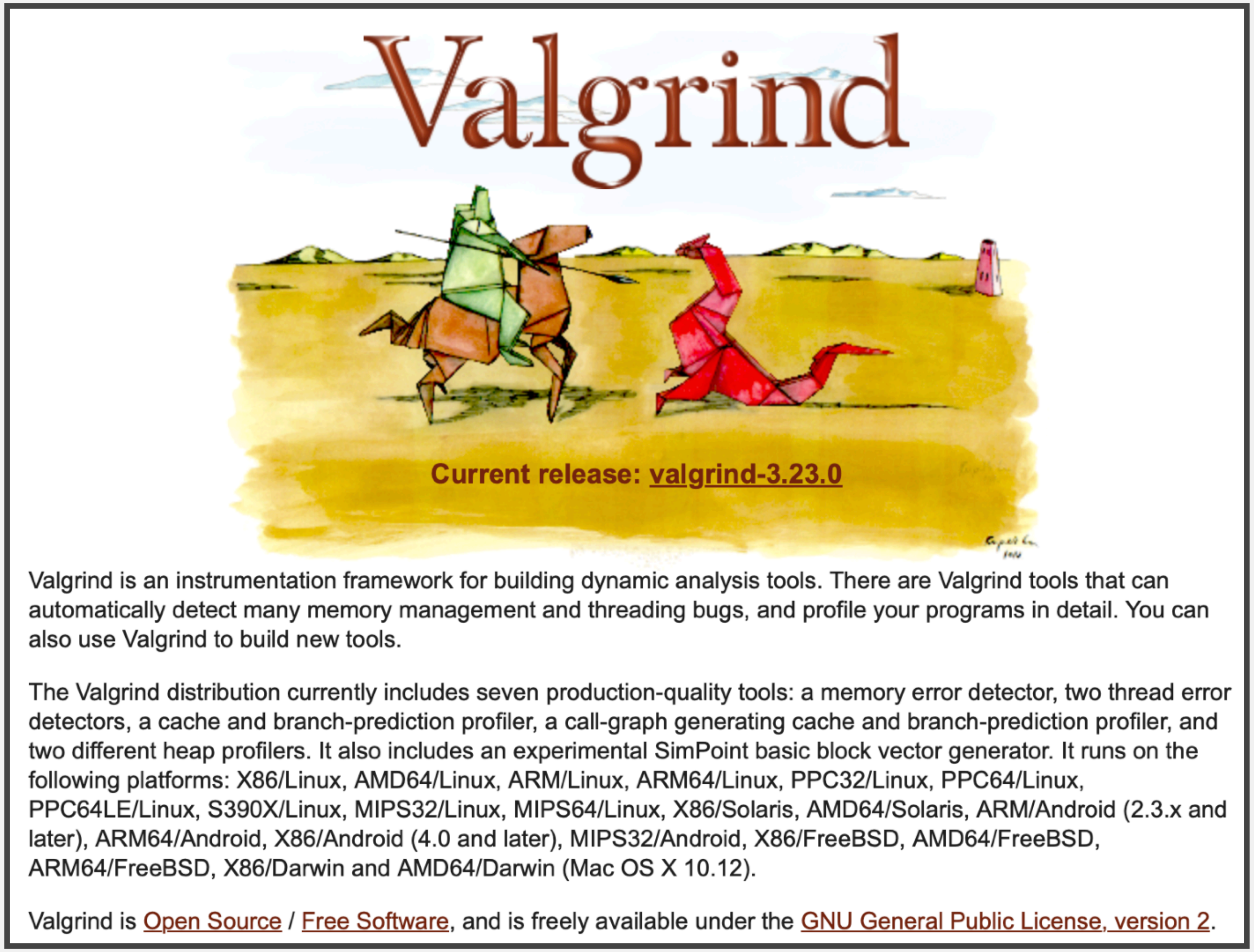

Adding breakpoints
pythonProject
Version control
car
car.py
Run
if __name__ == '__main__' › while True
pythonProject › car.py
27:1 LF UTF-8 4 spaces Python 3.9 (pythonProject)

```
    def accelerate(self):
        self.speed += 5

    def brake(self):
        self.speed -= 5

    def step(self):
        self.odometer += self.speed
        self.time += 1

    def average_speed(self):
        return self.odometer / self.time
```

```
Accelerating...
What should I do? [A]ccelerate, [B]rake, show [O]dometer, or show average [S]peed?b
Braking...
What should I do? [A]ccelerate, [B]rake, show [O]dometer, or show average [S]peed?b
Braking...
What should I do? [A]ccelerate, [B]rake, show [O]dometer, or show average [S]peed?o
The car has driven 0 kilometers
What should I do? [A]ccelerate, [B]rake, show [O]dometer, or show average [S]peed?
```

https://www.jetbrains.com/help/pycharm/debugging-your-first-python-application.html#where-is-the-problem

Valgrind Dynamic Analysis Library



Current release: valgrind-3.23.0

Valgrind is an instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail. You can also use Valgrind to build new tools.

The Valgrind distribution currently includes seven production-quality tools: a memory error detector, two thread error detectors, a cache and branch-prediction profiler, a call-graph generating cache and branch-prediction profiler, and two different heap profilers. It also includes an experimental SimPoint basic block vector generator. It runs on the following platforms: X86/Linux, AMD64/Linux, ARM/Linux, ARM64/Linux, PPC32/Linux, PPC64/Linux, PPC64LE/Linux, S390X/Linux, MIPS32/Linux, MIPS64/Linux, X86/Solaris, AMD64/Solaris, ARM/Android (2.3.x and later), ARM64/Android, X86/Android (4.0 and later), MIPS32/Android, X86/FreeBSD, AMD64/FreeBSD, ARM64/FreeBSD, X86/Darwin and AMD64/Darwin (Mac OS X 10.12).

Valgrind is Open Source / Free Software, and is freely available under the GNU General Public License, version 2.

https://valgrind.org/

Summary

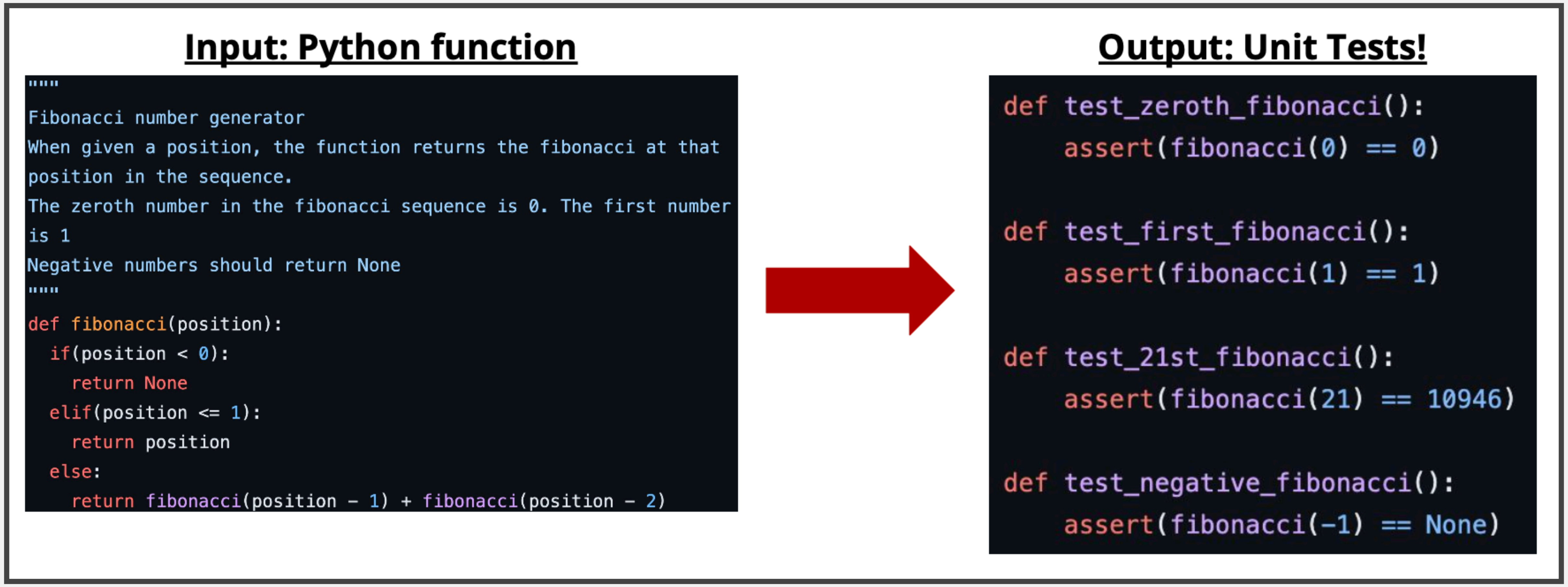
- Linters are cheap, fast, but imprecise analysis tools
  - Can be used for purposes other than bug detection (e.g., style)
- Conservative analyzers can demonstrate the absence of particular defects
  - At the cost of false positives due to necessary approximations
  - Inevitable trade-off between false positives and false negatives
- The best QA strategy involves multiple analysis and testing techniques
  - The exact set of tools and techniques depends on context

A Software Engineer's Guide to LLMs

Learning Goals

- What is an LLM?
- Is an LLM the right solution for your problem?
- Building a basic LLM integration
- Evaluation Strategies
- Techniques to improve performance
- Productionizing an LLM application

Today's Running Example: Unit Test Generation



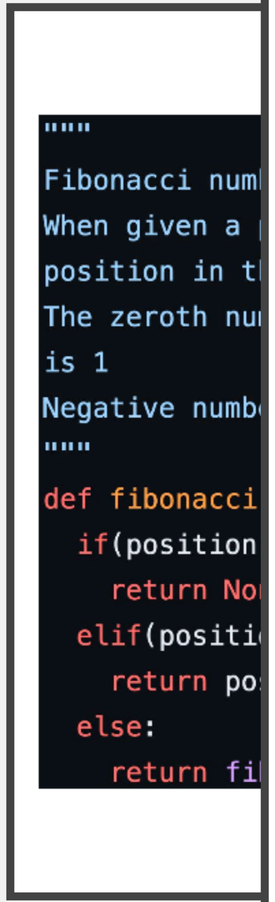
Input: Python function

```
"""
Fibonacci number generator
When given a position, the function returns the fibonacci at that
position in the sequence.
The zeroth number in the fibonacci sequence is 0. The first number
is 1
Negative numbers should return None
"""
def fibonacci(position):
  if(position < 0):
    return None
  elif(position <= 1):
    return position
  else:
    return fibonacci(position - 1) + fibonacci(position - 2)
```

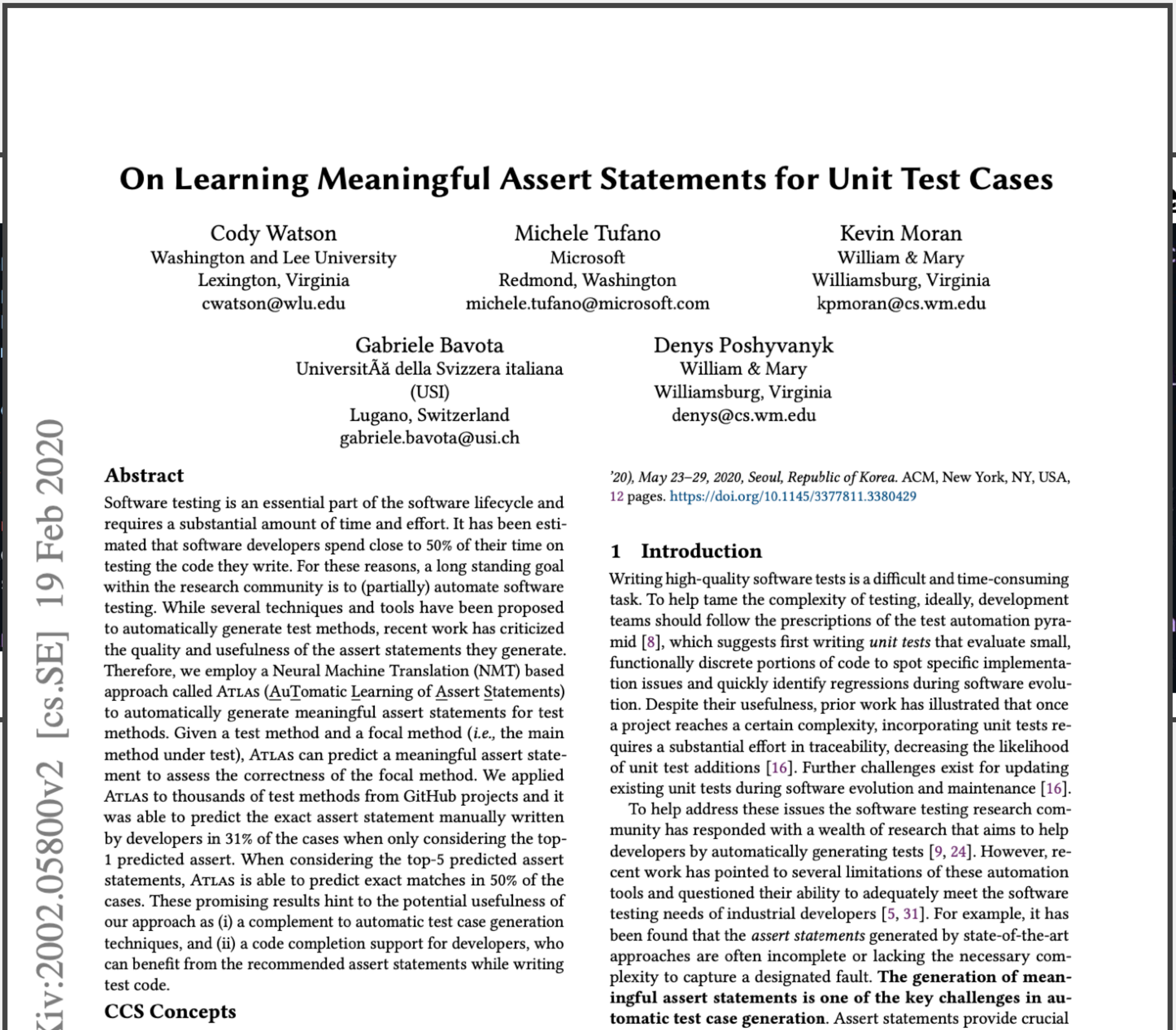
Output: Unit Tests!

```
def test_zeroth_fibonacci():
    assert(fibonacci(0) == 0)

def test_first_fibonacci():
    assert(fibonacci(1) == 1)

def test_21st_fibonacci():
    assert(fibonacci(21) == 10946)

def test_negative_fibonacci():
    assert(fibonacci(-1) == None)
```

Today's Running Example: Unit Test Generation



```
"""
Fibonacci num
When given a 
position in t
The zeroth nu
is 1
Negative numb
"""
def fibonacci
  if(position
    return No
  elif(positi
    return po
  else:
    return fi
```



On Learning Meaningful Assert Statements for Unit Test Cases

Cody Watson
Washington and Lee University
Lexington, Virginia
cwatson@wlu.edu

Michele Tufano
Microsoft
Redmond, Washington
michele.tufano@microsoft.com

Kevin Moran
William & Mary
Williamsburg, Virginia
kpmoran@cs.wm.edu

Gabriele Bavota
UniversitÃă della Svizzera italiana
(USI)
Lugano, Switzerland
gabriele.bavota@usi.ch

Denys Poshyvanyk
William & Mary
Williamsburg, Virginia
denys@cs.wm.edu

Abstract

Software testing is an essential part of the software lifecycle and requires a substantial amount of time and effort. It has been estimated that software developers spend close to 50% of their time on testing the code they write. For these reasons, a long standing goal within the research community is to (partially) automate software testing. While several techniques and tools have been proposed to automatically generate test methods, recent work has criticized the quality and usefulness of the assert statements they generate. Therefore, we employ a Neural Machine Translation (NMT) based approach called ATLAS (AuTomatic Learning of Assert Statements) to automatically generate meaningful assert statements for test methods. Given a test method and a focal method (*i.e.*, the main method under test), ATLAS can predict a meaningful assert statement to assess the correctness of the focal method. We applied ATLAS to thousands of test methods from GitHub projects and it was able to predict the exact assert statement manually written by developers in 31% of the cases when only considering the top-1 predicted assert. When considering the top-5 predicted assert statements, ATLAS is able to predict exact matches in 50% of the cases. These promising results hint to the potential usefulness of our approach as (i) a complement to automatic test case generation techniques, and (ii) a code completion support for developers, who can benefit from the recommended assert statements while writing test code.

CCS Concepts

'20), *May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3377811.3380429

1 Introduction

Writing high-quality software tests is a difficult and time-consuming task. To help tame the complexity of testing, ideally, development teams should follow the prescriptions of the test automation pyramid [8], which suggests first writing *unit tests* that evaluate small, functionally discrete portions of code to spot specific implementation issues and quickly identify regressions during software evolution. Despite their usefulness, prior work has illustrated that once a project reaches a certain complexity, incorporating unit tests requires a substantial effort in traceability, decreasing the likelihood of unit test additions [16]. Further challenges exist for updating existing unit tests during software evolution and maintenance [16].

To help address these issues the software testing research community has responded with a wealth of research that aims to help developers by automatically generating tests [9, 24]. However, recent work has pointed to several limitations of these automation tools and questioned their ability to adequately meet the software testing needs of industrial developers [5, 31]. For example, it has been found that the *assert statements* generated by state-of-the-art approaches are often incomplete or lacking the necessary complexity to capture a designated fault. **The generation of meaningful assert statements is one of the key challenges in automatic test case generation.** Assert statements provide crucial

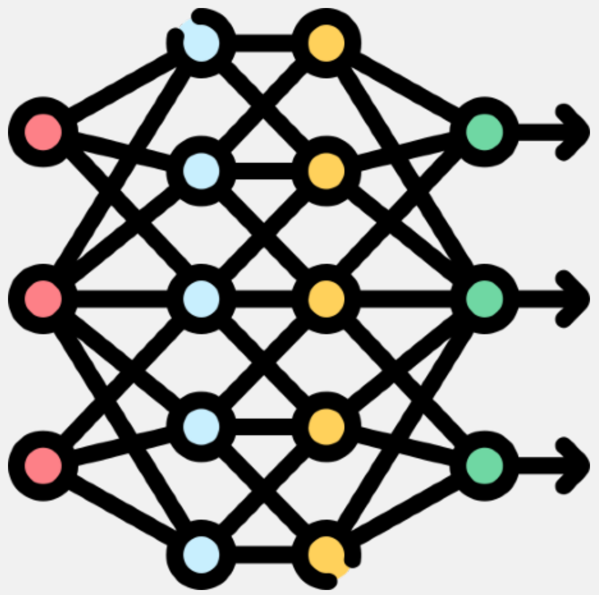
sts!

```
i():
 == 0)

.():
 == 1)

):
 == 10946)

cci():
 == None)
```

What even is an LLM?

Large Language Models

- Language Modeling: Measure probability of a sequence of words
  - Input: Text sequence
  - Output: Most likely next word
- LLMs are... large
  - GPT-3 has 175B parameters
  - GPT-4 is estimated to have ~1.24 Trillion
- Pre-trained with up to a PB of Internet text data
  - Massive financial and environmental cost



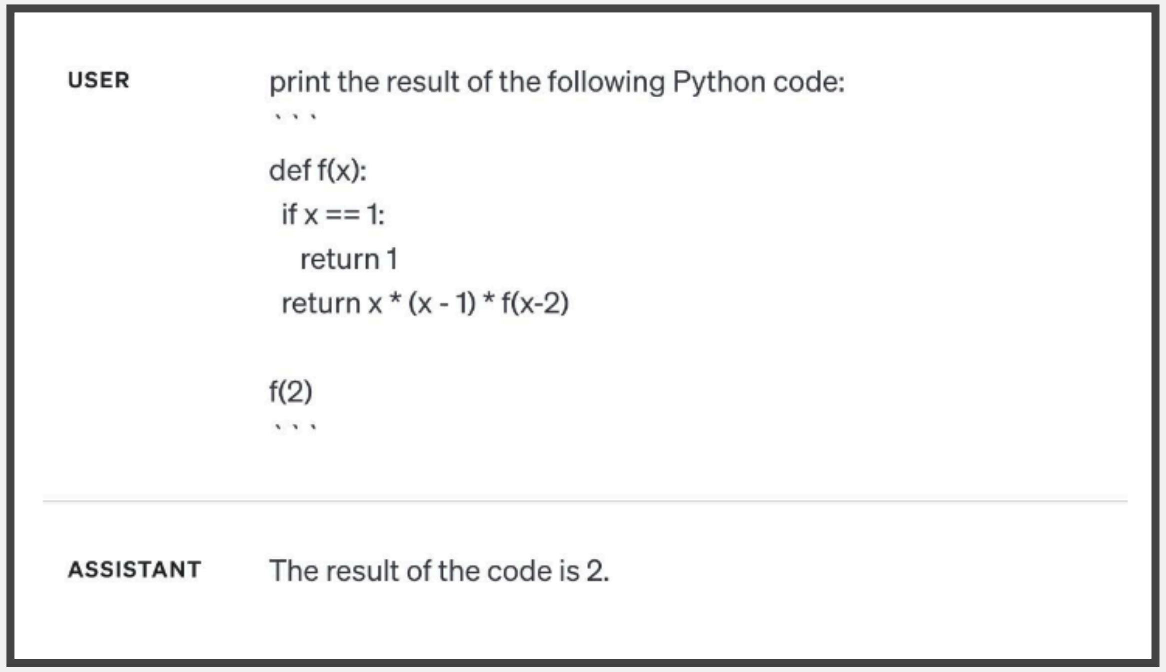
**Not actual size*

Large Language Models are Pre-trained

- Only a few people have resources to train LLMs
- Access through API calls
- OpenAI, Google Vertex AI, Anthropic, Hugging Face
- We will treat it as *a black box that can make errors!*

LLMs are Far from Perfect

- Hallucinations
  - Factually Incorrect Output
- High Latency
  - Output words generated one at a time
  - Larger models also tend to be slower
- Output format
  - Hard to structure output (e.g. extracting date from text)
  - Some workarounds for this (later)



USER

print the result of the following Python code:

```
def f(x):
 if x == 1:
   return 1
 return x * (x - 1) * f(x-2)

f(2)
```

ASSISTANT

The result of the code is 2.

Is an LLM Right for your Problem?

Which Problem should be Solved by an LLLM?

- Type checking Java code
- Grading mathematical proofs
- Answering emergency medical questions
- Unit test generation for NodeBB devs

Consider Other Options!

- **Alternative Solutions:** Are there alternative solutions to your task that deterministically yield better results? Eg: Type checking Java code
- **Error Probability:** How often do we expect the LLM to correctly solve an instance of your problem? This will change over time. Eg: Grading mathematical proofs
- **Risk tolerance:** What's the cost associated with making a mistake? Eg: Answering emergency medical questions
- **Risk mitigation strategies:** Are there ways to verify outputs and/or minimize the cost of errors? Eg: Unit test generation

Practical Factors to Consider

- Operational Costs
- Latency/speed
- Intellectual property
- Security