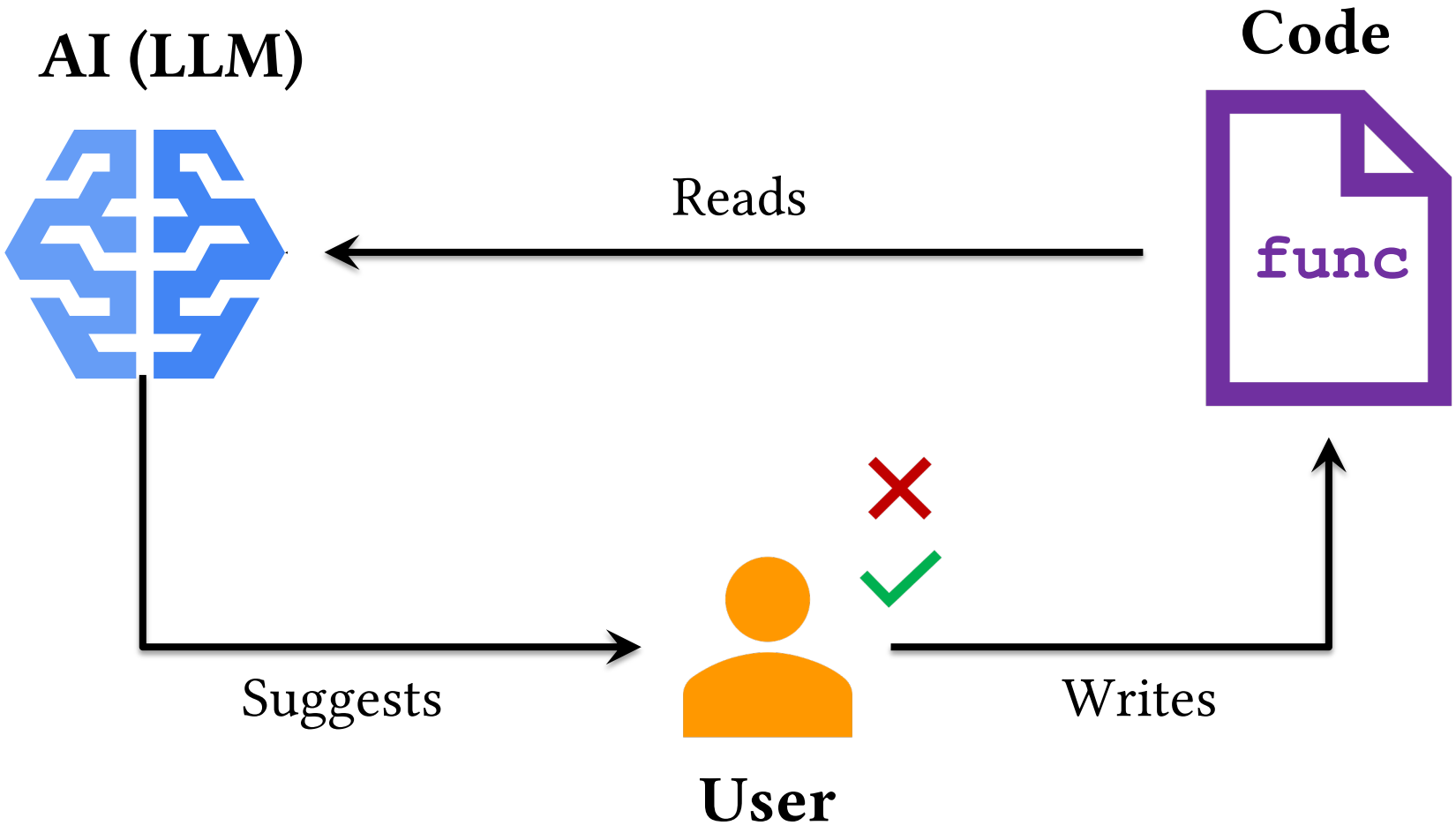
Security and Coding LLMs

Umar Farooq

Writing code



```c
#include <stdio.h>
int main () {
    char username[8];
    int allow = 0;
    printf("Username: ");
    gets(username);
    if (grantAccess(username)) {
        allow = 1;
    }
    if (allow != 0) {
        privilegedAction();
    }
    return 0;
}
```

Coding

```c
#include <stdio.h>
int main () {
    char username[8];
    int allow = 0;
    printf("Username: ");
    gets(username);
    if (grantAccess(username)) {
        allow = 1;
    }
    if (allow != 0) {
        privilegedAction();
    }
    return 0;
}
```

Testing

```c
#include <stdio.h>
int main () {
    char username[8];
    int allow = 0;
    printf("Username: ");
    gets(username);
    if (grantAccess(username)) {
        allow = 1;
    }
    if (allow != 0) {
        privilegedAction();
    }
    return 0;
}
```

Debugging

```c
#include <stdio.h>
int main () {
    char username[8];
    int allow = 0;
    printf("Username: ");
    - gets(username);
    if (grantAccess(username)) {
        allow = 1;
    }
    if (allow != 0) {
        privilegedAction();
    }
    return 0;
}
```

Repairing

```c
#include <stdio.h>

/***** This program is for checking the user-
provided username and performing privileged
action if the username is correct. It will
not allow an incorrect username. *****/

int main () {
    char username[8];
    int allow = 0;
    printf("Username: ");
    gets(username);
    if (grantAccess(username)) {
        allow = 1;
    }
    if (allow != 0) {
        privilegedAction();
    }
    return 0;
}
```

Documentation

A Typical Program Repair Scenario

Template

```
      ...
-     return super.equals(object);
+     return this == object;
      ...
```

Search with the deleted pattern

```
public Model copy(Model instance){
    ...
    instance.notify();
    if (super.equals(instance) && !instance.isEmpty()){
        return instance.clone();
    }
    ...
}
```

Replace with the added pattern

```
public Model copy(Model instance){
    ...
    instance.notify();
    if (this == instance && !instance.isEmpty()){
        return instance.clone();
    }
    ...
}
```

A Typical Program Repair Scenario

Template

```
    ...
-   return super.equals(object);
+   return this == object;
    ...
```

Search with the deleted pattern

Low tolerance for Deviation

Too Many Different Patterns

```
public Model copy(Model instance){
    ...
    instance.notify();
    if (super.nequals(instance) && !instance.isEmpty()){
        return instance.clone();
    }
    ...
}
```

Replace with the added pattern

```
public Model copy(Model instance){
    ...
    instance.notify();
    if (this != instance && !instance.isEmpty()){
        return instance.clone();
    }
    ...
}
```

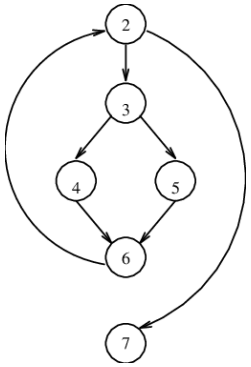
Challenges of Rule-Driven Approaches

Human Expert

Hand-Curate →

Rules and Heuristics

How-to →



Represent a Program

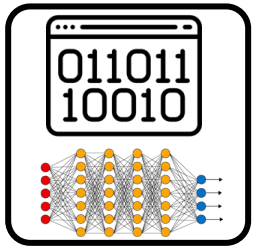
Input: $kill(B)$ and $gen(B)$ for every basic bloc B.

Output: $in(B)$ and $out(B)$ for every basic bloc B.

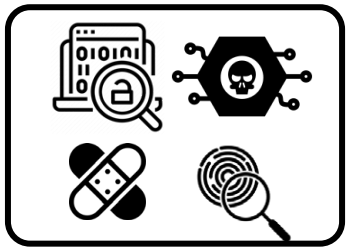
for each B **repeat**
  $out(B) := gen(B)$
while changes to any $out(B)$ occur **repeat**
  $in(B) := \bigcup_{B' \in \mathsf{pred}(B)} out(B')$
  $out(B) := gen(B) \bigcup (in(B) \setminus kill(B))$

Design Analysis rules

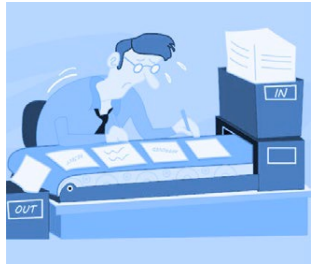
Heterogeneous Software



Various Analysis Tasks



Tune ←



- Significant Manual Effort
- Thus, Not Scalable

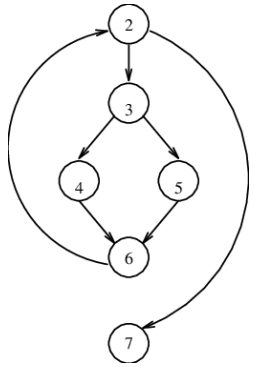
Challenges of Rule-Driven Approaches

Human Expert

Hand-Curate →

Rules and Heuristics

How-to →



Represent a Program

Input: $kill(B)$ and $gen(B)$ for every basic bloc B.

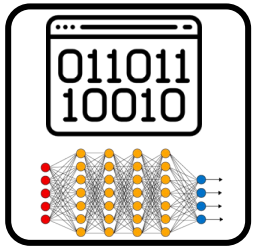
Output: $in(B)$ and $out(B)$ for every basic bloc B.

for each B **repeat**
  $out(B) := gen(B)$
while changes to any $out(B)$ occur **repeat**
  $in(B) := \bigcup_{B' \in \mathsf{pred}(B)} out(B')$
  $out(B) := gen(B) \bigcup (in(B) \setminus kill(B))$

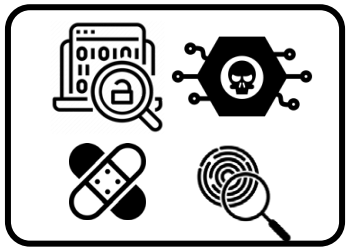
Design Analysis rules

Can we have a general-purpose program representation that can scale across heterogeneous software and different analysis tasks?

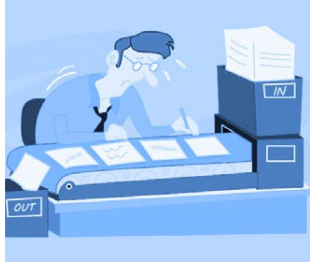
Heterogeneous Software



Various Analysis Tasks



Tune ←



- Significant Manual Effort
- Thus, Not Scalable

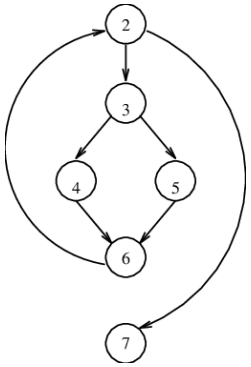
Challenges of Rule-Driven Approaches

Human Expert

Hand-Curate →

Rules and Heuristics

How-to →



Represent a Program

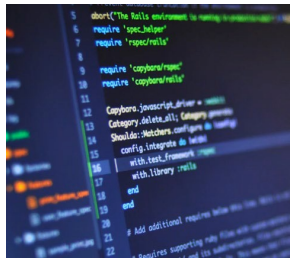
Input: $kill(B)$ and $gen(B)$ for every basic bloc B.

Output: $in(B)$ and $out(B)$ for every basic bloc B.

for each B **repeat**
  $out(B) := gen(B)$
while changes to any $out(B)$ occur **repeat**
  $in(B) := \bigcup_{B' \in \mathsf{pred}(B)} out(B')$
  $out(B) := gen(B) \bigcup (in(B) \setminus kill(B))$

Design Analysis rules

Can we have a general-purpose program representation
that can scale across heterogeneous software and different analysis tasks?



Abundant Public Code

Abundant Compute

→

We can learn general purpose
representation and analysis rules.

Training Code LMs

- Open-source code e.g., GitHub
- Discussion Boards e.g., StackOverflow
- Crawled & Hand Curated problems with tests
  - APPS, HumanEval
- RLHF style Compiler feedback

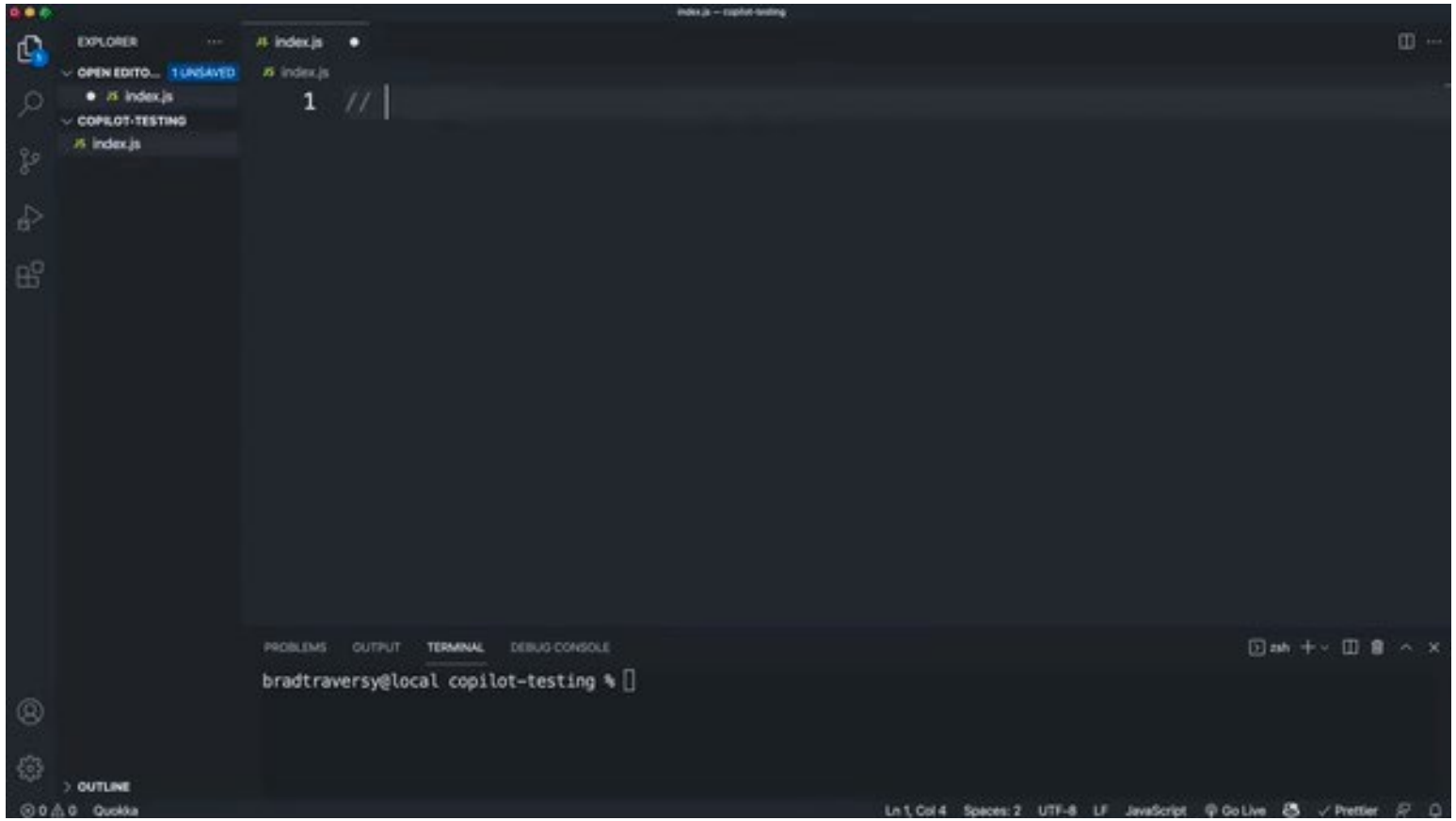
AI-powered Software Development

OpenAI
ChatGPT

GitHub
Copilot

AWS
CodeWhisper

Project IDX
Google



Demo

https://gist.github.com/ufarooq/bce5e3f195e2988c997d55c8e1a41ca7

Security Concerns of LLM generated code – I

- LLMs may be trained over potentially insecure or buggy code
  - May reproduce those insecurities/bugs
  - Consider the "MD5" hash algorithm once widely used for security
    - E.g., password encryption
  - MD5 has been cryptographically broken – should not be used
  - Code examples with MD5 remain available online, thus LLMs learn to (incorrectly) suggest MD5 for hashing passwords

Security Concerns of LLM generated code – II

- Code which may be secure in isolation may become insecure depending on the sequence execution
  - Consider storing text in a buffer
  - Can be performed safely using functions such as **`snprintf`**

```
int main() {
    char buffer[80];
    int n;
    n = snprintf(buffer, 80, "This is a string. It is being stored in a buffer.");
    printf("The string is: %s\n", buffer);
    return 0;
}
```

  - However, if buffer was just free-d, then the same line of code calling **`snprintf`** would result in a use-after-free

Evaluating code security

- Static Analysis
  - detect security-related bugs at compile-time
- Run-time analysis
  - debuggers and sanitizers like ‘Address Sanitizer’ and ‘Undefined Behavior Sanitizer’ (UBSAN)
- Fuzzers
  - run the program on concrete, randomly generated inputs

A majority of LLMs treats code as text!

```
for i in people.data.users:
    response = client.api.statuses.user_timeline.get(screen_name=i.scre
    print 'Got', len(response.data), 'tweets from', i.screen_name
    if len(response.data) != 0:
        ltdate = response.data[0]['created_at']
        ltdate2 = datetime.strptime(ltdate,'%a %b %d %H:%M:%S +0000 %Y'
        today = datetime.now()
        howlong = (today-ltdate2).days
        if howlong < daywindow:
            print i.screen_name, 'has tweeted in the past' , daywindow,
            totaltweets += len(response.data)
            for j in response.data:
                if j.entities.urls:
                    for k in j.entities.urls:
                        newurl = k['expanded_url']
                        urlset.add((newurl, j.user.screen_name))
        else:
            print i.screen_name, 'has not tweeted in the past', daywind
```

Limitations: Lacks Understanding of Program Semantics

Original Function name

```
def remove_lowercase(str1):
    """
    Write a function to remove lowercase
    substrings from a given string.
    >>> remove_lowercase("PYTHon")
    ('PYTH')
    >>> remove_lowercase("FInD")
    ('FID')
    >>> remove_lowercase("STRinG")
    ('STRG')
    """
    return "".join([i for i in str1 if i.isupper()])
```

Original completion

Perturbed function name

```
def removeLowercase(str1):
    """
    Write a function to remove lowercase
    substrings from a given string.
    >>> removeLowercase("PYTHon")
    ('PYTH')
    >>> removeLowercase("FInD")
    ('FID')
    >>> removeLowercase("STRinG")
    ('STRG')
    """
    str2 = str1.lower()
    return str2
```

New completion

CodeGen-16B-mono is correct on original prompt but fails when function name is perturbed.

Wang et al., AWS AI Lab, "ReCode: Robustness Evaluation of Code Generation Models." ACL 2023.." ACL 2023.

Lacks Understanding of Program Semantics [ICLR'24]