NYU TANDON SCHOOL OF ENGINEERING

Lost at C

Security Implications of Large Language Model Code Assistants

Brendan Dolan-Gavitt

In collaboration with: Gustavo Sandoval, Hammond Pearce, Teo Nys, Ramesh Karri, and Siddharth Garg

CENTER FOR CYBER SECURITY

Surprising Progress in Code Models

Before 2021

- 2015: Karpathy's Char-RNN, generating Linux kernel code
- 2019: GPT-2 "accidentally" learns some PHP and JavaScript

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
```

Char-RNN; Karpathy, 2015

```
$app = new App ();

// All GET requests that come to add_register() will be sent to this service.

$api = $app -> include(' ' );

$api -> register( new DbAppAndFNAAppRegistrationService ());

// Define any services to register. We will override any present in the external
// DB have the class of .DAO .

$service = new AppAndFNAAppService ( $app , [

array ( ' host ' => ' localhost ' )

]);
```

GPT-2; OpenAI, 2015

Surprising Progress in Code Models

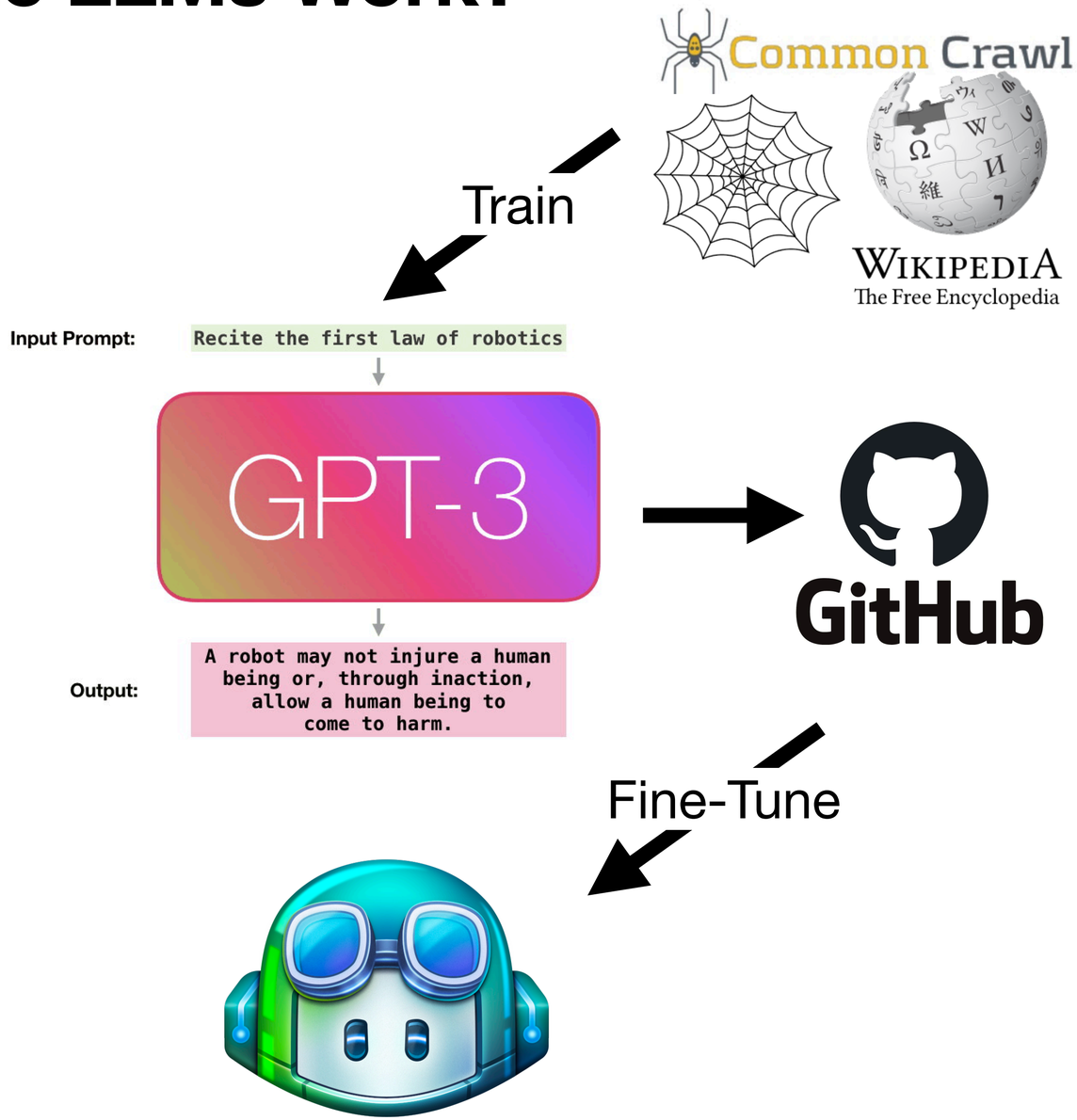
June 2021 - Present: Large Language Models (LLMs)

- **2021: OpenAI Codex** - a large GPT-3-based model fine-tuned on code
  - Released commercially as a code completion tool: **GitHub Copilot**
- **2022: DeepMind AlphaCode** - Transformer (encoder/decoder)
  - Reaches human-level (top 54%) performance in an online code competition (Codeforces)
- Both systems treat source code as plain text, “predict next token”
- Trained on **large volumes of code** (e.g. all of GitHub)

Background: How Do Code LLMs Work?

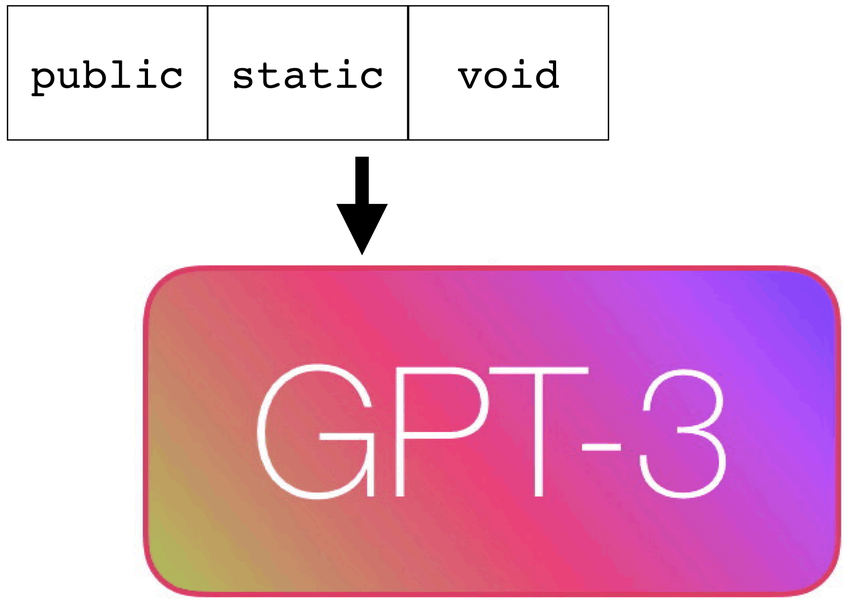
GPT-3, but on code

- **Objective**: predict token i given tokens {1, ..., i-1}
- **Model**: Transformer (decoder-only)
- **GPT-3** training data: WebText, Wikipedia, CommonCrawl, etc.
- **Codex**: Fine-tuned on *approximately all of GitHub public repositories*
- **Copilot**: commercial version of Codex



Autoregressive Sampling

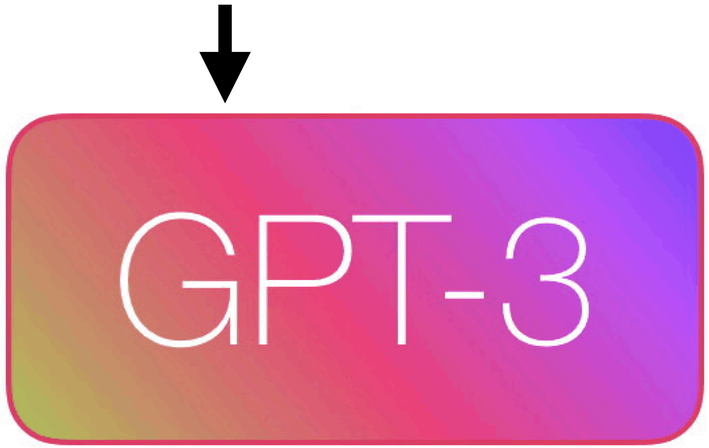
How to generate text and code



Autoregressive Sampling

How to generate text and code

public	static	void

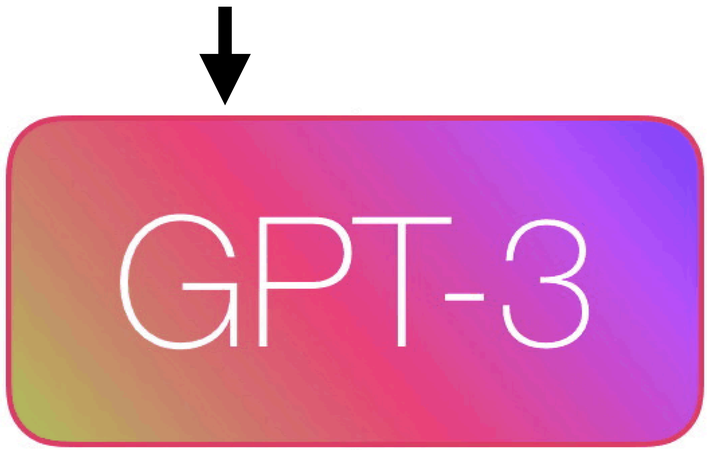


Token	Probability
`main`	92%
`add`	6%
`update`	1%
`insert`	0.1%
`{`	0.1%
`\n`	0.04%

Autoregressive Sampling

How to generate text and code

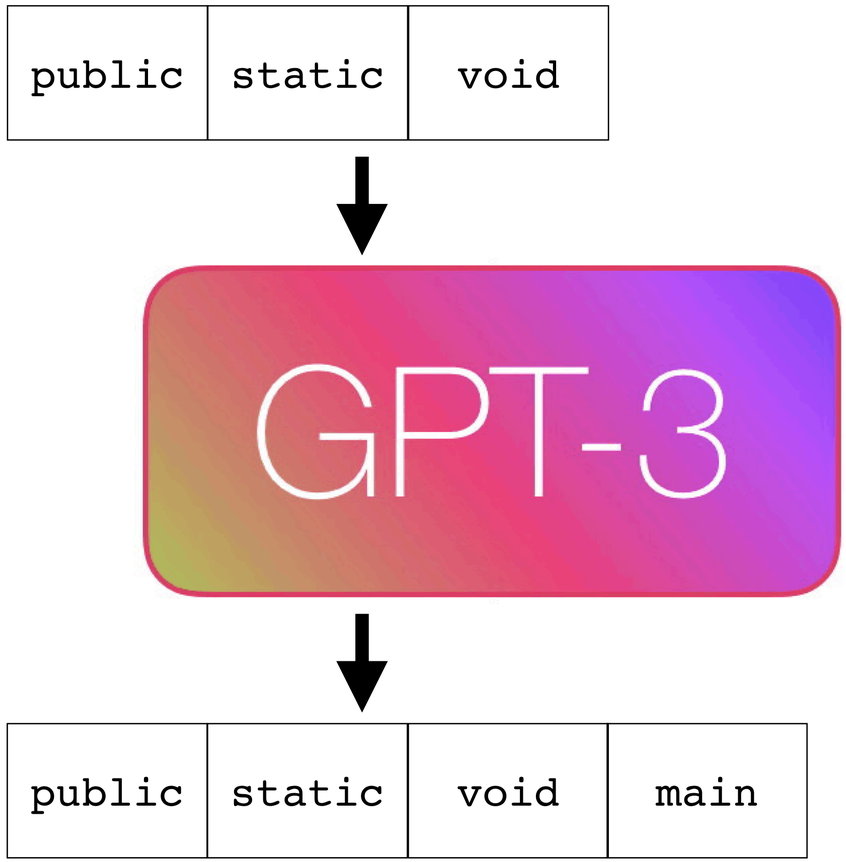
public	static	void



Token	Probability
`main`	92%
`add`	6%
`update`	1%
`insert`	0.1%
`{`	0.1%
`\n`	0.04%

Autoregressive Sampling

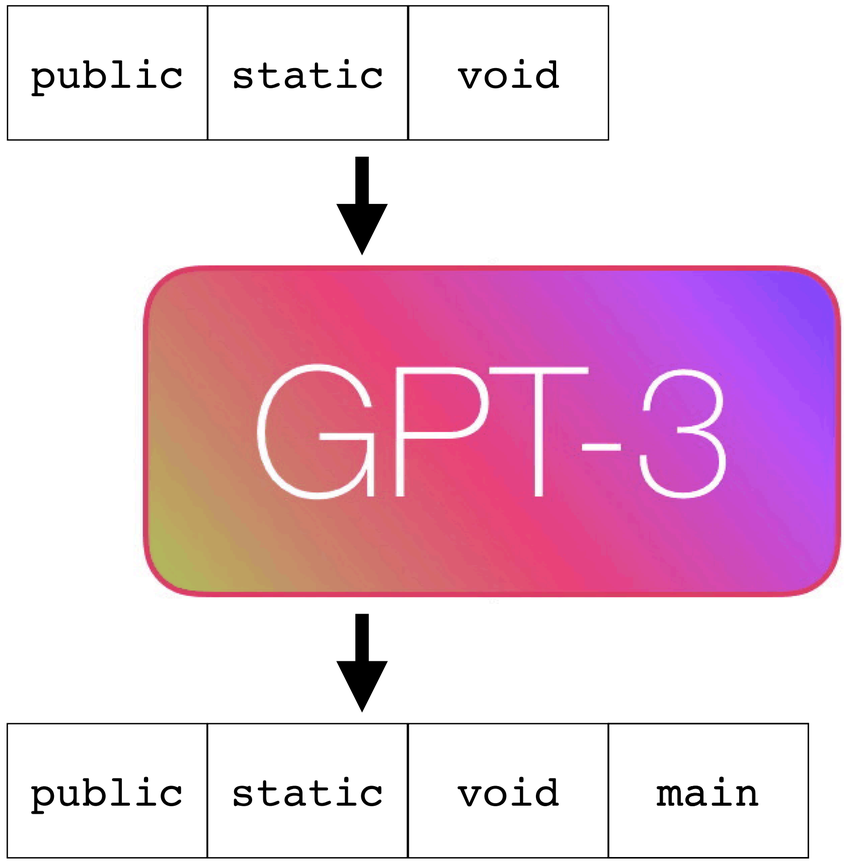
How to generate text and code



Token	Probability
`main`	92%
`add`	6%
`update`	1%
`insert`	0.1%
`{`	0.1%
`\n`	0.04%

Autoregressive Sampling

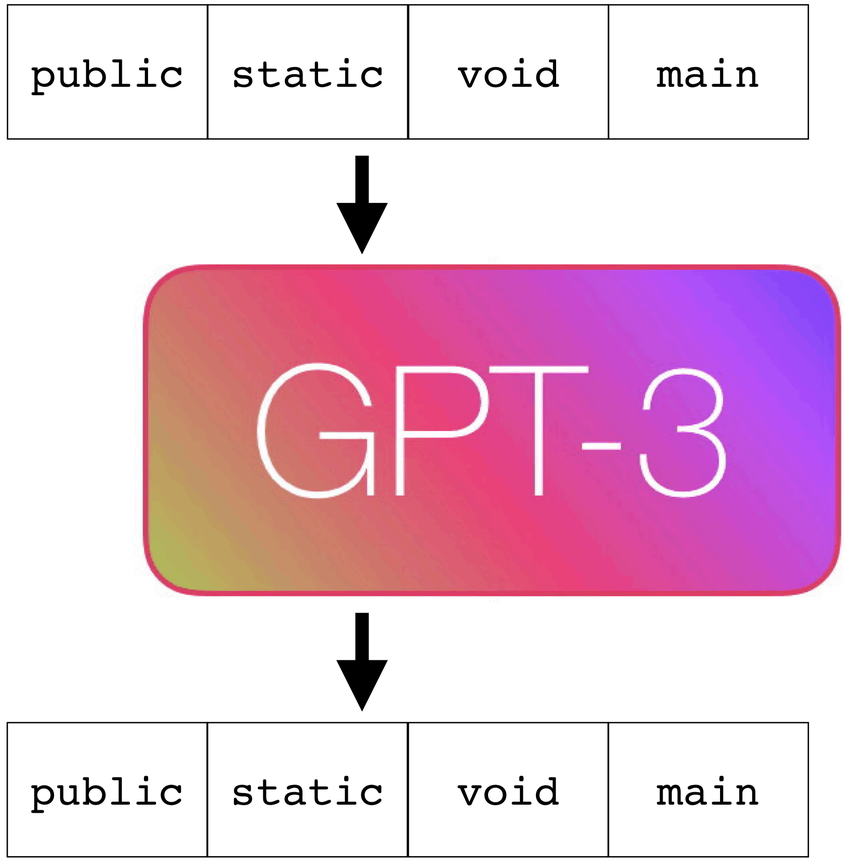
How to generate text and code



Token	Probability
`main`	92%
`add`	6%
`update`	1%
`insert`	0.1%
`{`	0.1%
`\n`	0.04%

Autoregressive Sampling

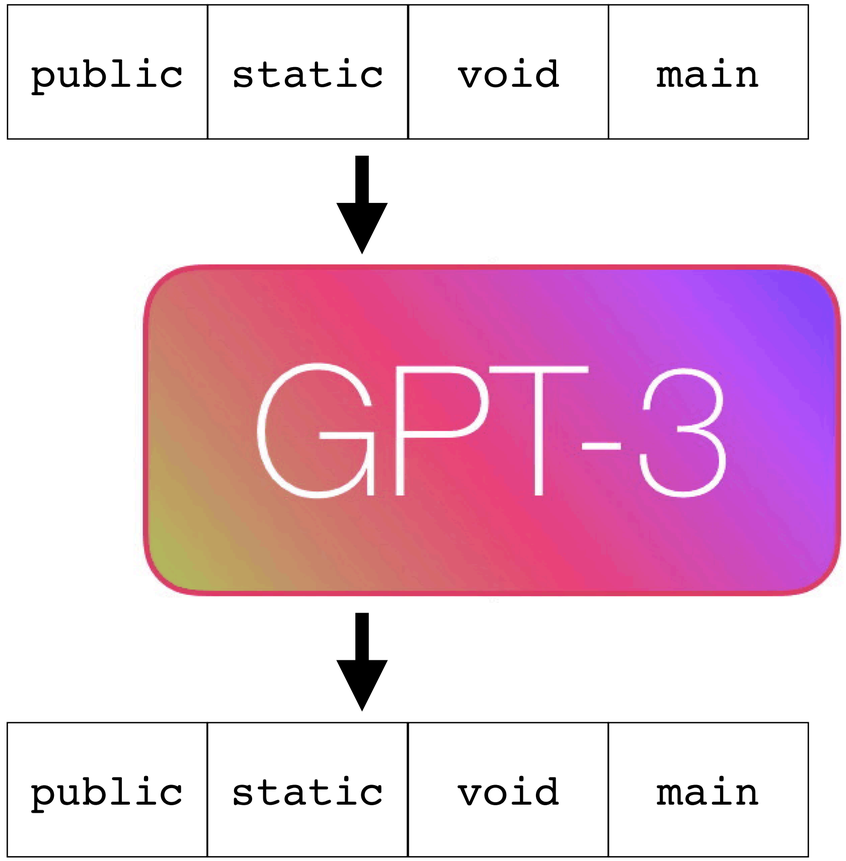
How to generate text and code



Token	Probability
`main`	92%
`add`	6%
`update`	1%
`insert`	0.1%
`{`	0.1%
`\n`	0.04%

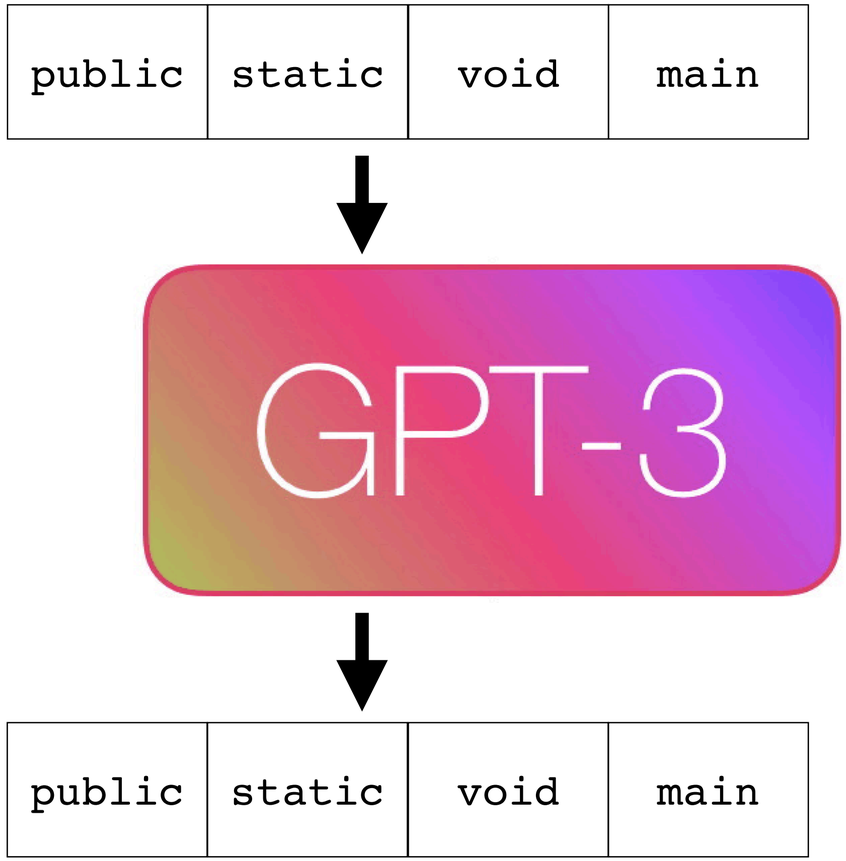
Autoregressive Sampling

How to generate text and code



Autoregressive Sampling

How to generate text and code



Token	Probability
(String	97%
␣	2%
\n	0.01%
()	0.003%
{	0.001%
[	0.0006%

How Secure is the Code LLMs Write?

How Secure is the Code LLMs Write?

WILL KNIGHT BUSINESS SEP 20, 2021 7:00 AM

AI Can Write Code Like Humans—Bugs and All

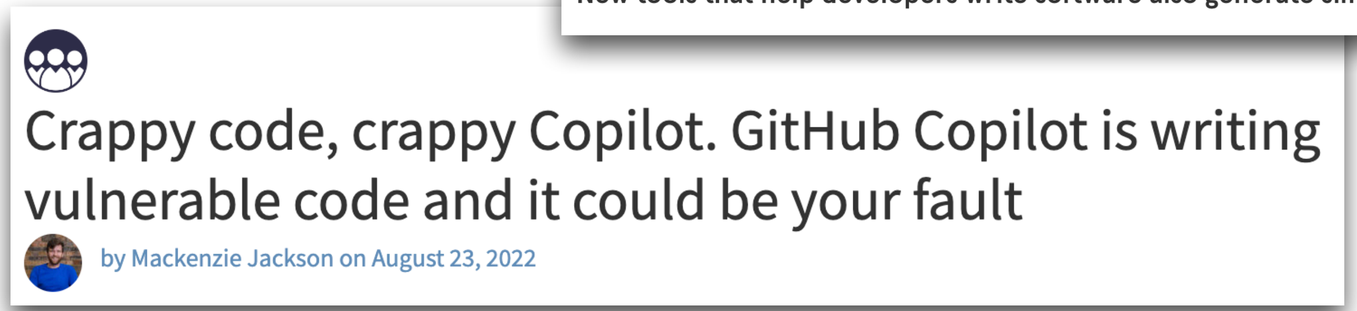
New tools that help developers write software also generate similar mistakes.

How Secure is the Code LLMs Write?

WILL KNIGHT BUSINESS SEP 20, 2021 7:00 AM

AI Can Write Code Like Humans—Bugs and All

New tools that help developers write software also generate similar mistakes.



Crappy code, crappy Copilot. GitHub Copilot is writing vulnerable code and it could be your fault

by Mackenzie Jackson on August 23, 2022

How Secure is the Code LLMs Write?

WILL KNIGHT BUSINESS SEP 20, 2021 7:00 AM

AI Can Write Code Like Humans—Bugs and All

New tools that help developers write software also generate similar mistakes.

Crappy code, crappy Copilot. GitHub Copilot is writing vulnerable code and it could be your fault

by Mackenzie Jackson on August 23, 2022

Developers beware: AI pair programming comes with pitfalls

Despite the promise of faster coding, AI pair programming has a host of pitfalls, including inapplicable code suggestions, security flaws and copyright issues.

By **Stephanie Glen,** News Writer Published: **22 Jul 2022**

How Secure is the Code LLMs Write?

WILL KNIGHT BUSINESS SEP 20, 2021 7:00 AM

AI Can Write Code Like Humans—Bugs and All

New tools that help developers write software also generate similar mistakes.

Crappy code, crappy Copilot. GitHub Copilot is writing vulnerable code and it could be your fault

by Mackenzie Jackson on August 23, 2022

Developers beware: AI pair programming comes with pitfalls

Despite the promise of faster coding, AI pair programming has a host of pitfalls, including inapplicable code suggestions, security flaws and copyright issues.

By **Stephanie Glen,** News Writer Published: **22 Jul 2022**

FEATURE

Why you can't trust AI-generated autocomplete code to be secure

Artificial intelligence-powered tools such as GitHub Pilot and Tabnine offer developers autocomplete suggestions that help them write code faster. How do they ensure this code is secure?

By **Andrada Fiscutean**

CSO | MAR 15, 2022 2:00 AM PDT

How Secure is the Code LLMs Write?

WILL KNIGHT BUSINESS SEP 20, 2021 7:00 AM

AI Can Write Code Like Humans—Bugs and All

New tools that help developers write software also generate similar mistakes.

Crappy code, crappy Copilot. GitHub Copilot is writing vulnerable code and it could be your fault

by Mackenzie Jackson on August 23, 2022

Developers beware: AI pair programming comes with pitfalls

Despite the promise of faster coding, AI pair programming has a host of pitfalls, including inapplicable code suggestions, security flaws and copyright issues.

By **Stephanie Glen,** News Writer Published: **22 Jul 2022**

FEATURE

Why you can't trust AI-generated autocomplete code to be secure

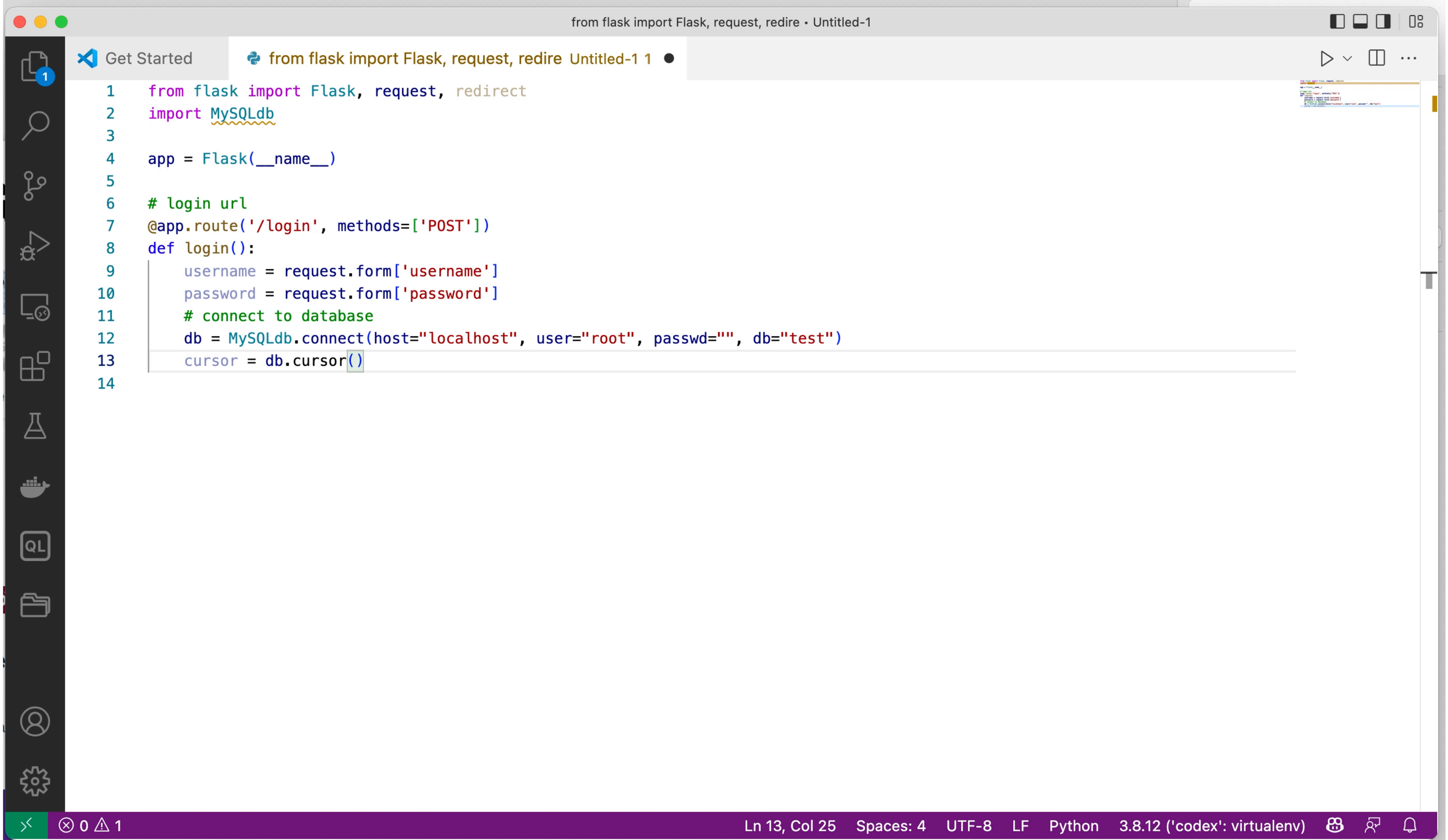
Artificial intelligence-powered tools such as GitHub Pilot and Tabnine offer developers autocomplete suggestions that help them write code faster. How do they ensure this code is secure?

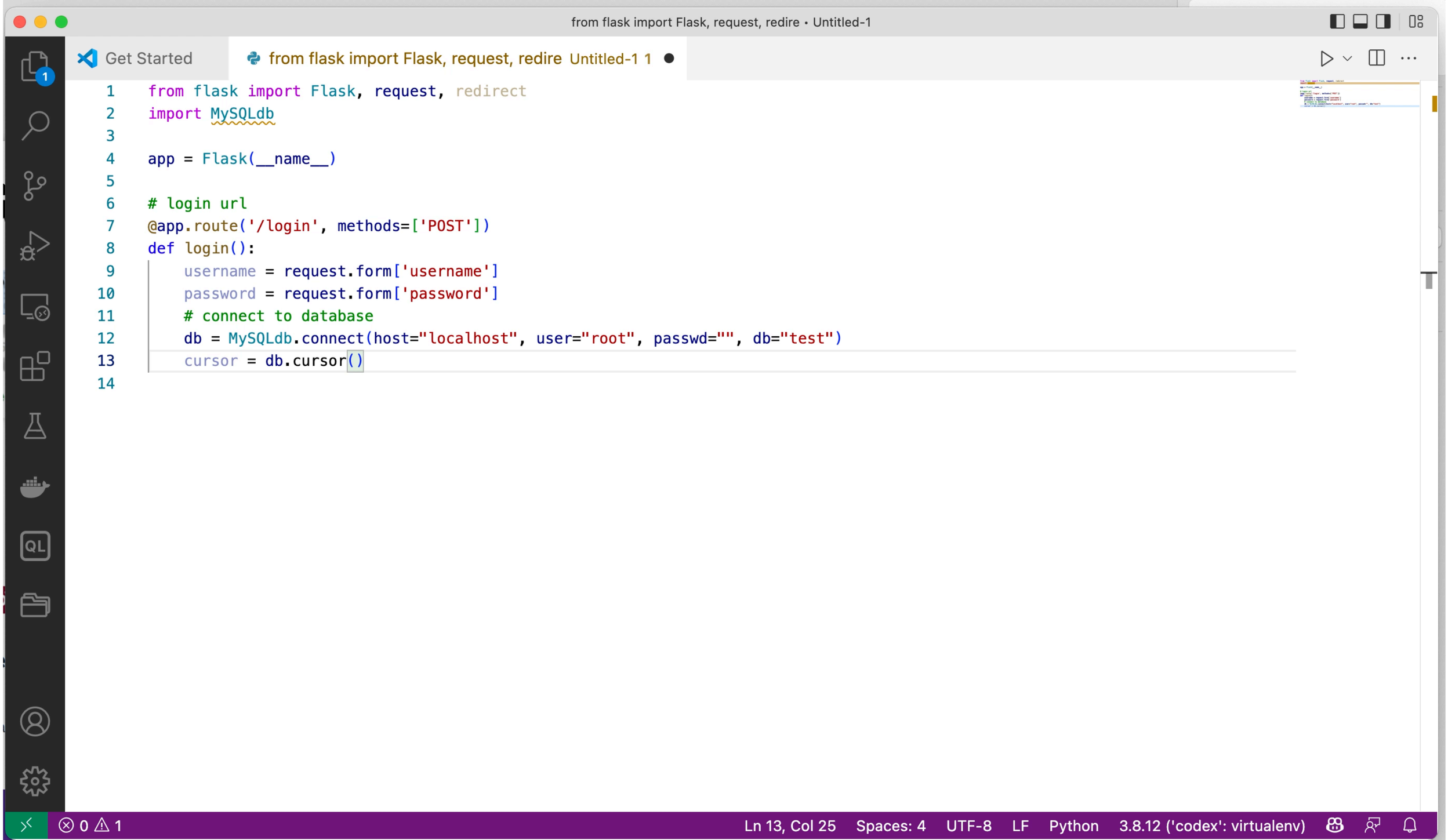
By **Andrada Fiscutean**

CSO | MAR 15, 2022 2:00 AM PDT

GitHub Copilot Security Study: 'Developers Should Remain Awake' in View of 40% Bad Code Rate

By David Ramel 08/26/2021


from flask import Flask, request, redire • Untitled-1
Get Started
from flask import Flask, request, redire Untitled-1 1
1 from flask import Flask, request, redirect
2 import MySQLdb
3
4 app = Flask(__name__)
5
6 # login url
7 @app.route('/login', methods=['POST'])
8 def login():
9 username = request.form['username']
10 password = request.form['password']
11 # connect to database
12 db = MySQLdb.connect(host="localhost", user="root", passwd="", db="test")
13 cursor = db.cursor()
14
0 1
Ln 13, Col 25
Spaces: 4
UTF-8
LF
Python
3.8.12 ('codex': virtualenv)


from flask import Flask, request, redire • Untitled-1
Get Started
from flask import Flask, request, redire Untitled-1 1
1 from flask import Flask, request, redirect
2 import MySQLdb
3
4 app = Flask(__name__)
5
6 # login url
7 @app.route('/login', methods=['POST'])
8 def login():
9 username = request.form['username']
10 password = request.form['password']
11 # connect to database
12 db = MySQLdb.connect(host="localhost", user="root", passwd="", db="test")
13 cursor = db.cursor()
14
0 1
Ln 13, Col 25
Spaces: 4
UTF-8
LF
Python
3.8.12 ('codex': virtualenv)

Copilot Writes Vulnerable Code

Asleep at the Keyboard

Prior work at IEEE Security and Privacy 2022

- We did a systematic study of Copilot's code completions in security-sensitive scenarios, measuring vulnerability rates with GitHub CodeQL
- Key findings:
  - Across all scenarios, **42%** of the generated programs were vulnerable
  - Features of the **prompt**, including comments, affects the rate of vulnerable code
  - The strongest predictor of whether Copilot will produce a vulnerability is the **presence of an existing vulnerability** in the prompt

But Wait!

Some objections from Reviewer #2

- In the real world, Copilot works with human assistance
- Maybe humans would spot and fix these mistakes?
- For that matter, maybe *unassisted* humans would write bugs at the same rate!
- **Strong reject**

Time for a User Study

Oh no, IRB forms

- We ran a user study using NYU students (undergraduate and graduate) and asked them to implement a linked list API
  - Participants were randomly assigned to the **Assisted (AI code assistant enabled)** or **Control** group
  - Participants had **two weeks** to complete the assigned task, and were given **$50** as compensation upon completion
  - Recruitment: an undergraduate Operating Systems class, an Application Security course (mixed undergraduate/graduate), and an informal NYU CS Discord server
  - **105** participants signed up, but only **58** actually signed in to our web IDE and wrote code for us to analyze

Participant Demographics

Enrollment

	Control	Assisted	Total
Undergraduates (UG)			
UG Y2 (Sophomores)	*1*	*8*	
UG Y3 (Juniors)	*8*	*5*	
UG Y4 (Seniors)	*4*	*5*	
UG (Unspecified)	*2*	*0*	
UG (Total)	**15**	**18**	**N (UG) = 33**
Postgraduates (PG)			
PG (MS)	*10*	*10*	
PG (PhDs)	*1*	*0*	
PG (Unspecified)	*1*	*1*	
PG (Total)	**12**	**11**	**N (PG) = 23**
Other Participants			
Other (Total)	**1**	**1**	**N (Other) = 2**
Total	**N (Control) = 28**	**N (Assisted) = 30**	N (Total) = 58

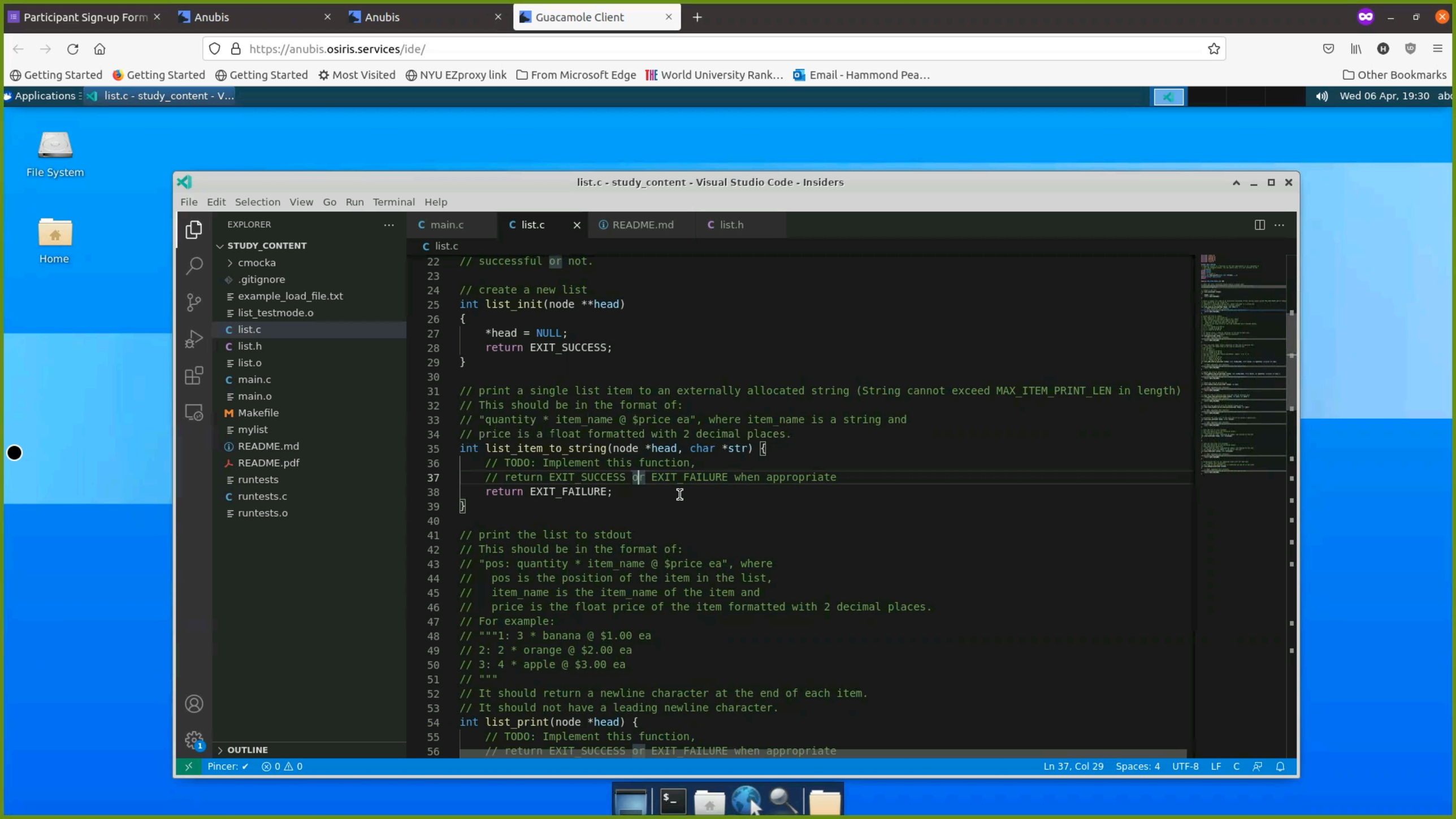
Participant Demographics

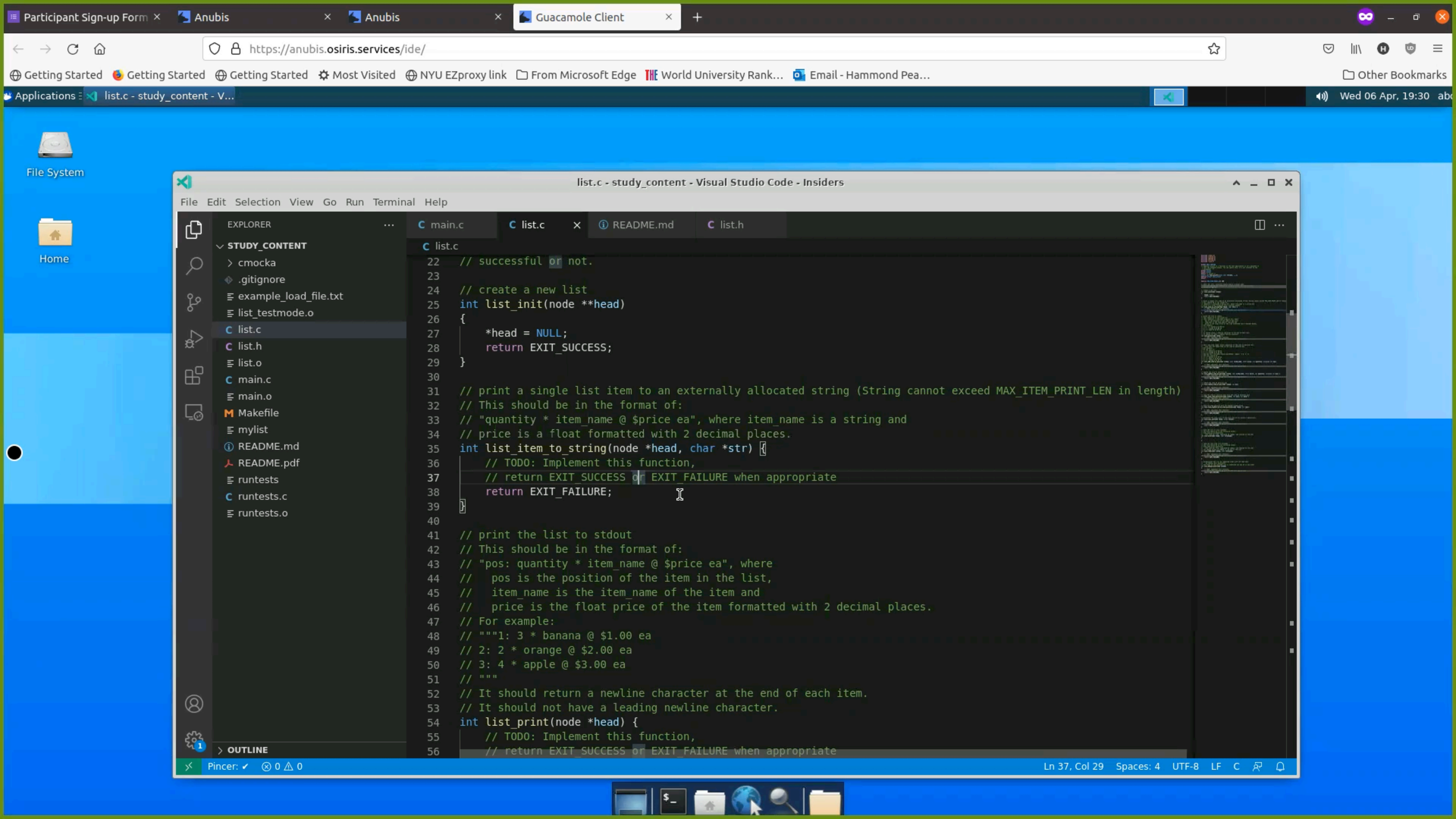
Experience Level

	Control	Assisted	Total
Is this the first linked list implementation you have ever made in C?			
Yes (first list)	*14*	*16*	30
No (not first list)	*11*	*12*	23
Declined to answer	*3*	*2*	5
Is this the first time that you have ever programmed in C?			
Yes (first time)	*3*	*4*	7
No (not first time)	*22*	*23*	45
Declined to answer	*3*	*3*	6
Are you taking, or have you ever taken a data structures or algo. class?			
Currently taking	2	3	5
Previously taken	21	25	46
Never taken	2	1	3
Declined to answer	3	1	4

Study Environment

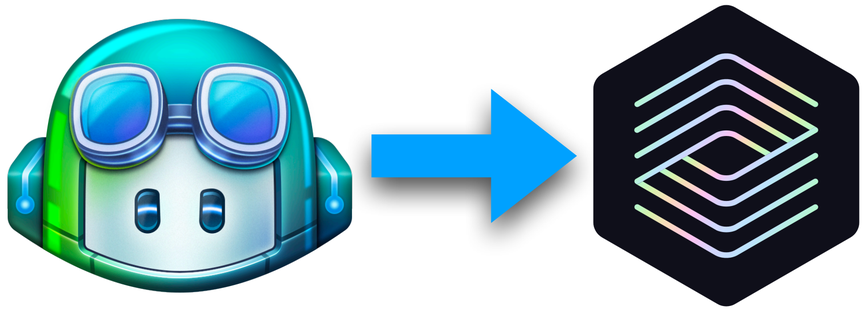
- **Goals**:
  - Minimize environment setup hassle
  - Log all the things
- Participants were asked to use our **Anubis** web-based IDE, which provides a VNC session to a Linux desktop with **VSCode** and a C compiler
- Created a VSCode plugin that mimics Copilot, but uses suggestions provided by the Codex API
- **Logged**: document snapshots every minute, prompt+suggestion data (including accepted/ not accepted)


Participant Sign-up Form
Anubis
Anubis
Guacamole Client
https://anubis.osiris.services/ide/
Getting Started
Getting Started
Getting Started
Most Visited
NYU EZproxy link
From Microsoft Edge
World University Rank...
Email - Hammond Pea...
Other Bookmarks
Applications
list.c - study_content - V...
Wed 06 Apr, 19:30
File System
Home
list.c - study_content - Visual Studio Code - Insiders
File Edit Selection View Go Run Terminal Help
EXPLORER
STUDY_CONTENT
cmocka
.gitignore
example_load_file.txt
list_testmode.o
list.c
list.h
list.o
main.c
main.o
Makefile
mylist
README.md
README.pdf
runtests
runtests.c
runtests.o
OUTLINE
main.c
list.c
README.md
list.h
list.c
// successful or not.
// create a new list
int list_init(node **head)
{
*head = NULL;
return EXIT_SUCCESS;
}
// print a single list item to an externally allocated string (String cannot exceed MAX_ITEM_PRINT_LEN in length)
// This should be in the format of:
// "quantity * item_name @ $price ea", where item_name is a string and
// price is a float formatted with 2 decimal places.
int list_item_to_string(node *head, char *str) {
// TODO: Implement this function,
// return EXIT_SUCCESS or EXIT_FAILURE when appropriate
return EXIT_FAILURE;
}
// print the list to stdout
// This should be in the format of:
// "pos: quantity * item_name @ $price ea", where
// pos is the position of the item in the list,
// item_name is the item_name of the item and
// price is the float price of the item formatted with 2 decimal places.
// For example:
// """1: 3 * banana @ $1.00 ea
// 2: 2 * orange @ $2.00 ea
// 3: 4 * apple @ $3.00 ea
// """
// It should return a newline character at the end of each item.
// It should not have a leading newline character.
int list_print(node *head) {
// TODO: Implement this function,
// return EXIT_SUCCESS or EXIT_FAILURE when appropriate
Pincer: ✔ 0 0
Ln 37, Col 29 Spaces: 4 UTF-8 LF C


Participant Sign-up Form
Anubis
Anubis
Guacamole Client
https://anubis.osiris.services/ide/
Getting Started
Getting Started
Getting Started
Most Visited
NYU EZproxy link
From Microsoft Edge
World University Rank...
Email - Hammond Pea...
Other Bookmarks
Applications
list.c - study_content - V...
Wed 06 Apr, 19:30
File System
Home
list.c - study_content - Visual Studio Code - Insiders
File Edit Selection View Go Run Terminal Help
EXPLORER
STUDY_CONTENT
cmocka
.gitignore
example_load_file.txt
list_testmode.o
list.c
list.h
list.o
main.c
main.o
Makefile
mylist
README.md
README.pdf
runtests
runtests.c
runtests.o
OUTLINE
main.c
list.c
README.md
list.h
list.c
// successful or not.
// create a new list
int list_init(node **head)
{
*head = NULL;
return EXIT_SUCCESS;
}
// print a single list item to an externally allocated string (String cannot exceed MAX_ITEM_PRINT_LEN in length)
// This should be in the format of:
// "quantity * item_name @ $price ea", where item_name is a string and
// price is a float formatted with 2 decimal places.
int list_item_to_string(node *head, char *str) {
// TODO: Implement this function,
// return EXIT_SUCCESS or EXIT_FAILURE when appropriate
return EXIT_FAILURE;
}
// print the list to stdout
// This should be in the format of:
// "pos: quantity * item_name @ $price ea", where
// pos is the position of the item in the list,
// item_name is the item_name of the item and
// price is the float price of the item formatted with 2 decimal places.
// For example:
// """1: 3 * banana @ $1.00 ea
// 2: 2 * orange @ $2.00 ea
// 3: 4 * apple @ $3.00 ea
// """
// It should return a newline character at the end of each item.
// It should not have a leading newline character.
int list_print(node *head) {
// TODO: Implement this function,
// return EXIT_SUCCESS or EXIT_FAILURE when appropriate
Pincer: ✔ 0 0
Ln 37, Col 29 Spaces: 4 UTF-8 LF C

Code Assistant Setup

- To get better logging and instrumentation, we decided to use **OpenAI Codex** for our study rather than Copilot
  - OpenAI generously provided an non-rate limited API token for the duration of the study (Thanks OpenAI!)
- Code model: cushman-code-001
  - Weaker model, but **very low latency**
- Temperature: 0.6, top-p: 1.0
  - Relatively high temp to get more diverse solutions



Study Task: “Shopping List”

The *Worst* Singly Linked List API (11 functions total)

- Since we’re studying security chose C because it’s a “target-rich environment”
- We deliberately included some pitfalls in the data structure and API to further broaden the range of possible errors
- Singly linked list: lots of opportunity for pointer mistakes
- Includes a string field (buffer overflows, etc.)

```c
// Node of the singly linked list
typedef struct _node {
    char* item_name;
    float price;
    int quantity;
    struct _node *next;
} node;
```

Uh oh, strings

(a) Node definition (in `list.h`)

```c
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <string.h>
#include "list.h"

#define MAX_ITEM_PRINT_LEN 100

// Note: All list_ functions should return a status code
// EXIT_FAILURE or EXIT_SUCCESS to indicate whether the
    operation was
// successful or not.
```

Fixed length

(b) `#includes` and implementation hints (in `list.c`)

Study Task: “Shopping List”

API: Basic List Manipulation

- Concepts tested:
  - Basic list traversal
  - List manipulation
  - Managing lifetime of item_name
- Pitfalls: *updates* the list via a double-pointer to head, item_name needs to be freed/ copied, one-indexed, position is *sometimes* a signed int

One-indexed

Need to *copy* the string

Double pointer

```c
// create a new list
int list_init(node **head);

// add a new item (name, price, quantity) to the list at position pos,
//   such that the added item is the item
// For example:
// If the list is:
// 1: 3 * banana @ $1.00 ea
// 2: 2 * orange @ $2.00 ea
// and you call list_add_item_at_pos(&head, "apple", 3.0, 4, 2)
// the list should be:
// 1: 3 * banana @ $1.00 ea
// 2: 4 * apple @ $3.00 ea
// 3: 2 * orange @ $2.00 ea
int list_add_item_at_pos(node **head, char *item_name, float price, int quantity, unsigned int pos);

// update the item at position pos
int list_update_item_at_pos(node **head, char *item_name, float price, int quantity, unsigned int pos);

// remove the item at position *pos*
int list_remove_item_at_pos(node **head, int pos);

// swap the item at position pos1 with the item at position pos2
int list_swap_item_positions(node **head, int pos1, int pos2);
```

Study Task: “Shopping List”

API: String Manipulation

- Concepts tested:
  - Basic list traversal
  - Correct format string usage
  - Copying into a buffer provided externally without overflow
- A bit tricky because the maximum length is given as a constant: MAX_ITEM_PRINT_LEN

Externally provided buffer with no length!

```c
// print a single list item to an externally allocated string
// This should be in the format of:
// "quantity * item_name @ $price ea", where item_name is a string and
// price is a float formatted with 2 decimal places.
int list_item_to_string(node *head, char *str);

// print the list to stdout
// This should be in the format of:
// "pos: quantity * item_name @ $price ea", where
//   pos is the position of the item in the list,
//   item_name is the item_name of the item and
//   price is the float price of the item formatted with 2 decimal places.
// For example:
// """1: 3 * banana @ $1.00 ea
// 2: 2 * orange @ $2.00 ea
// 3: 4 * apple @ $3.00 ea
// """
// It should return a newline character at the end of each item.
// It should not have a leading newline character.
int list_print(node *head);
```

Study Task: “Shopping List”

API: Advanced Tasks

- Concepts tested:
  - Saving/loading data from disk
  - Handling errors from system APIs (fopen, etc.)
  - Advanced traversal and updates
  - Freeing and updating entries
- Many people just skipped the harder APIs

```c
// find the item position with the highest single price
int list_find_highest_price_item_position(node *head, int *pos);

// calculate the total cost of the list (sum of all prices * quantities)
int list_cost_sum(node *head, float *total);

// de-duplicate the list by combining items with the same name
//   by adding their quantities
// The order of the returned list is undefined and may be in any order
int list_deduplicate(node **head);
```

Output parameters

(b) Advanced traversal functions

```c
// save the list to file filename
// the file should be in the following format:
// item_name,price,quantity\n
//   (one item per line, separated by commas, and newline at the end)
int list_save(node *head, char *filename);

// load the list from file filename
// the file should be in the following format:
// item_name,price,quantity\n
//   (one item per line, separated by commas, and newline at the end)
// the loaded values are added to the end of the list
int list_load(node **head, char *filename);
```

(c) Saving and Loading the list

Autopilot Mode

Let the AI do all the work!

- As another baseline comparison we also had three code models (Cushman-001, DaVinci-001, and DaVinci-002) complete the assignment automatically
- Procedure:
  - Complete one API function at a time, whole file (up to context limits) up to the function prototype as the prompt
  - Check if it compiles; if not, try again (up to 10 tries)
  - Same temp/top-p settings as the IDE plugin
  - Added a comment at the top with the node member names

Research Questions

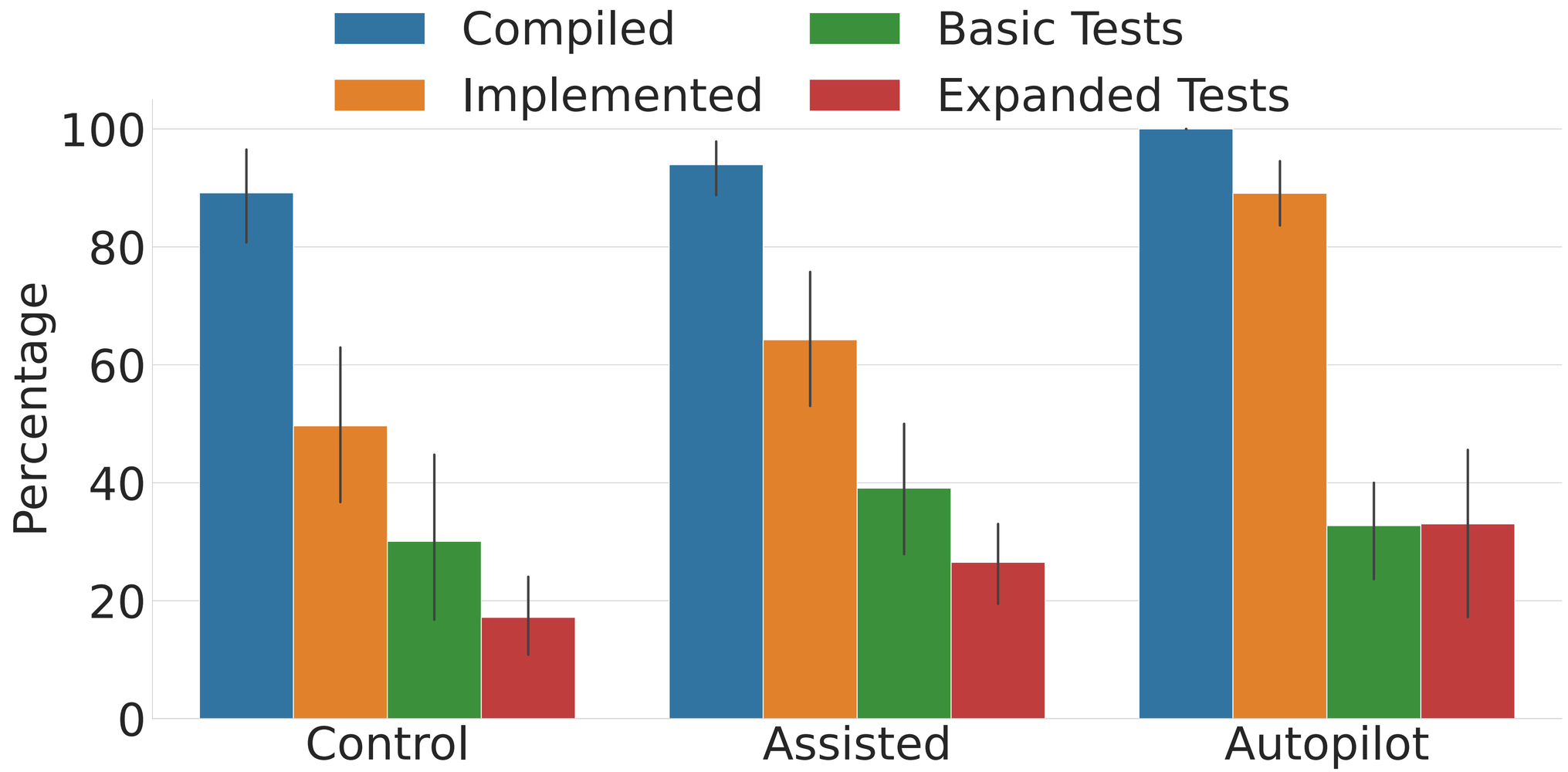
- **RQ1**: Does the AI code assistant help novice users write better code in terms of ***functionality***?
- **RQ2**: Is the code that novice users write with AI assistance more or less ***secure*** than the control group?
- **RQ3**: Are there systematic differences in the ***coding style*** of AI-assisted users and that of control group?
- **RQ4**: How do AI assisted users interact with potentially vulnerable code suggestions, i.e., where do bugs originate in an LLM-assisted system?

Measuring Functionality

- We provided users with a **basic** test suite with one test per function (12 tests)
- We also wrote an **expanded** test suite with 45 tests checking all the edge cases we could think of
- To reduce inter-test dependencies, we split the users' code into individual functions and tested them in isolation, with our known-good (🤞) reference implementation
  - Also allows us to test users who submitted non-compiling code, as long as *some* of their functions compiled

Functionality Results

Rise of the Machines

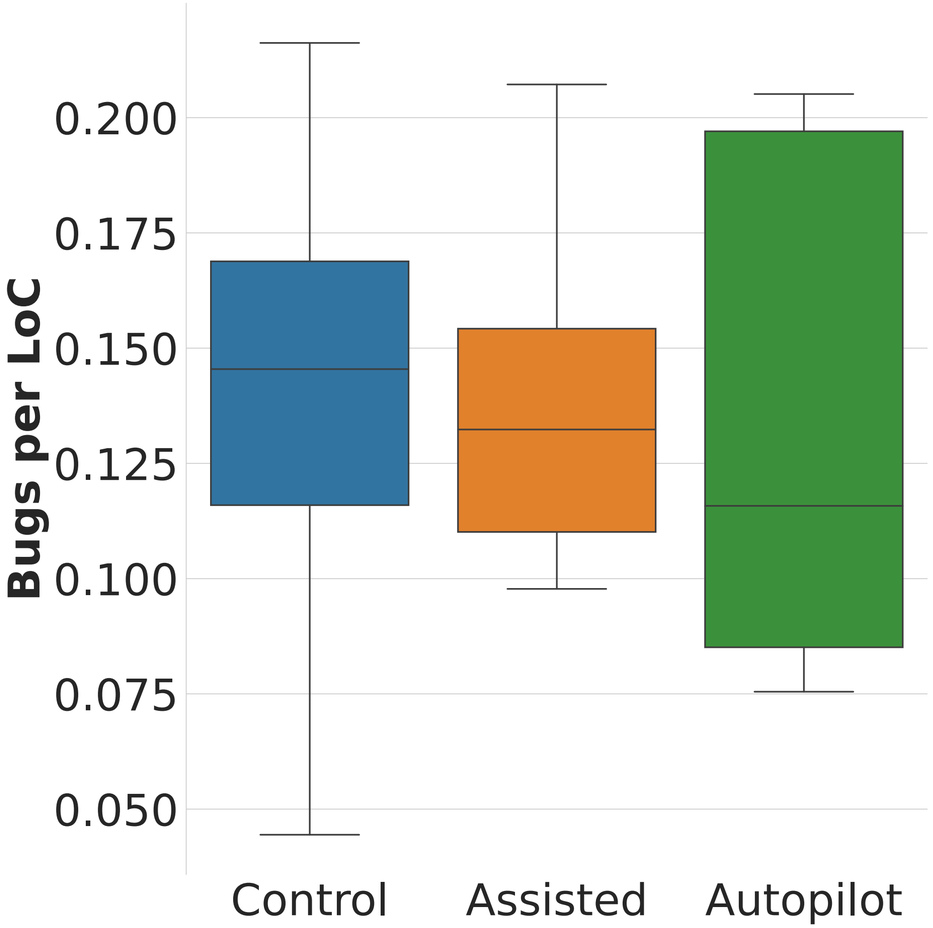


Measuring Security

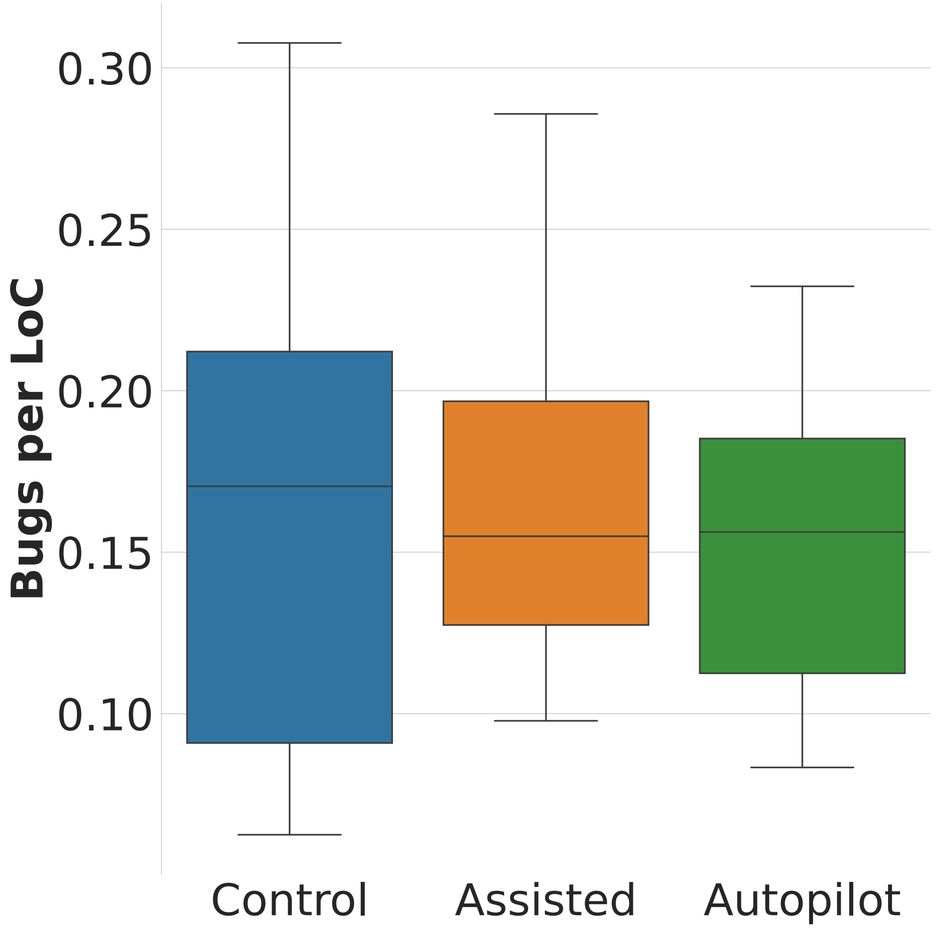
- Measuring security is **more difficult**
- CodeQL missed many issues, had false positives
- Fuzzing was attractive but many **duplicate** problems found
- We just bit the bullet and reviewed all code by hand
  - Three of us stared at each function and annotated with vulnerabilities categorized by MITRE's Common Weakness Enumeration
  - Also graded 5 of the Cushman Autopilot answers
  - 20 hours of my life I will never get back

Security Results

Number of vulnerabilities per line of code



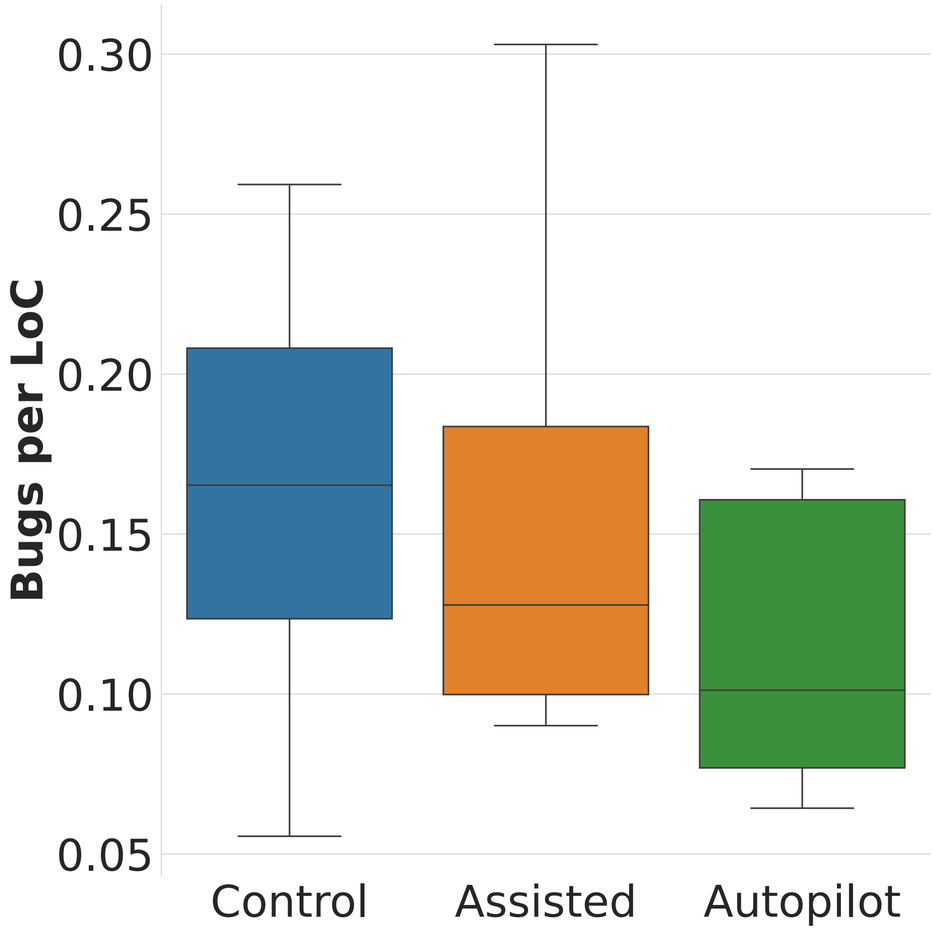
CWEs/LoC for *compiling* code



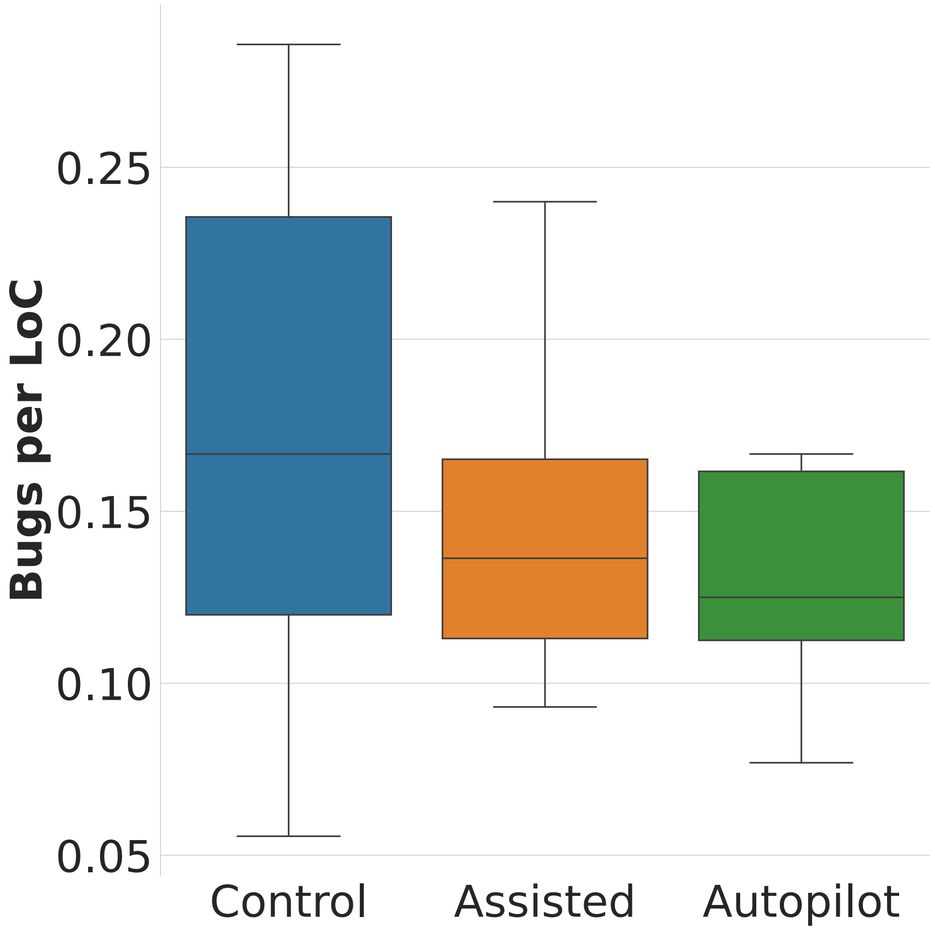
CWEs/LoC, code that *passes the basic unit test*

Security Results

Number of *severe* (MITRE Top 25) vulnerabilities per line of code

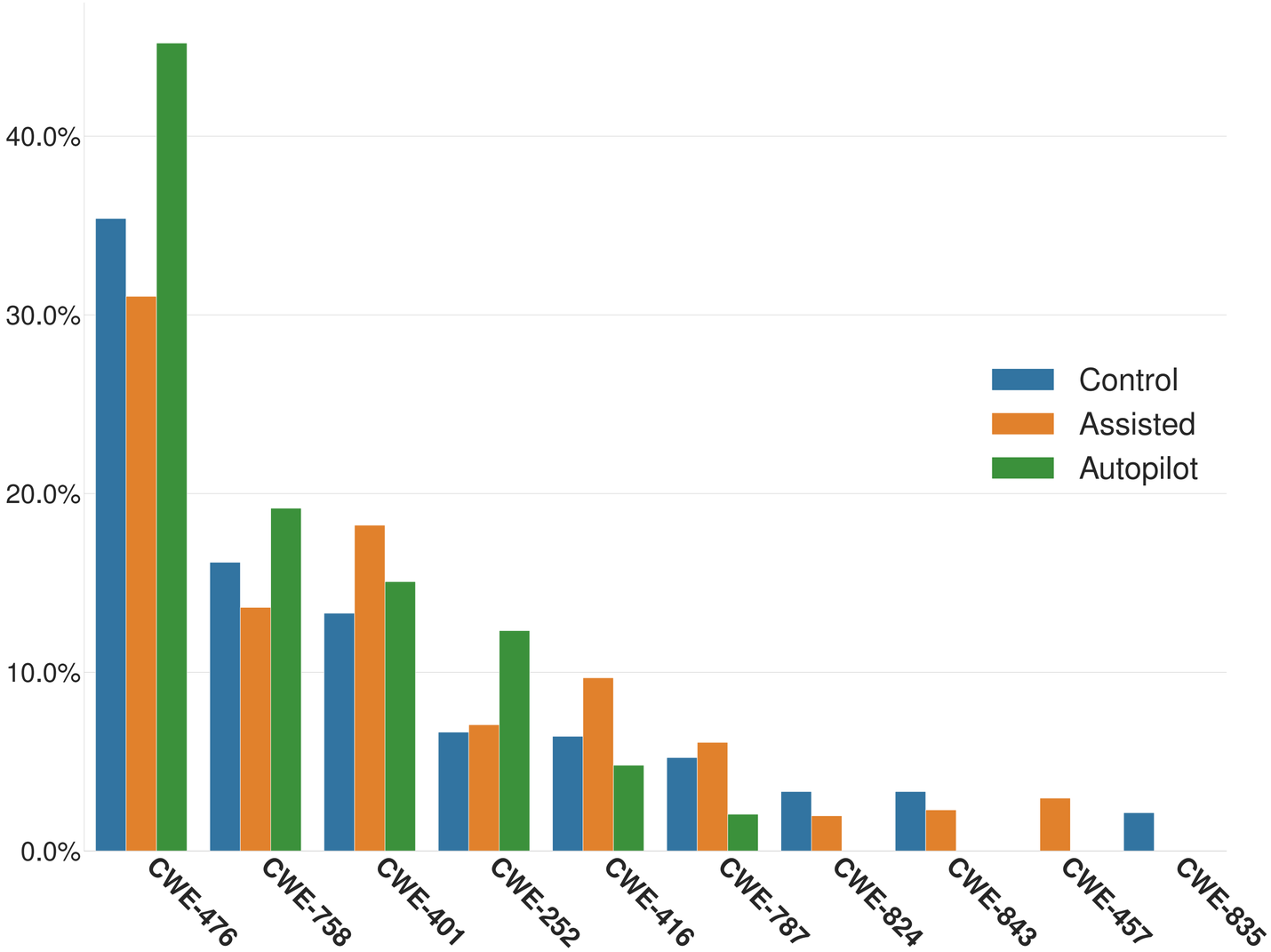


Severe CWEs/LoC for *compiling* code



Severe CWEs/LoC, code that *passes the basic unit test*

Security Results: CWEs



CWE-476 NULL Pointer Dereference

CWE-758 Reliance on Undefined, Unspecified, or Implementation-Defined Behavior

CWE-401 Missing Release of Memory after Effective Lifetime

CWE-252 Unchecked Return Value

CWE-416 Use After Free

CWE-787 Out-of-bounds Write

CWE-457 Use of Uninitialized Variable

CWE-843 Access of Resource Using Incompatible Type ('Type Confusion')

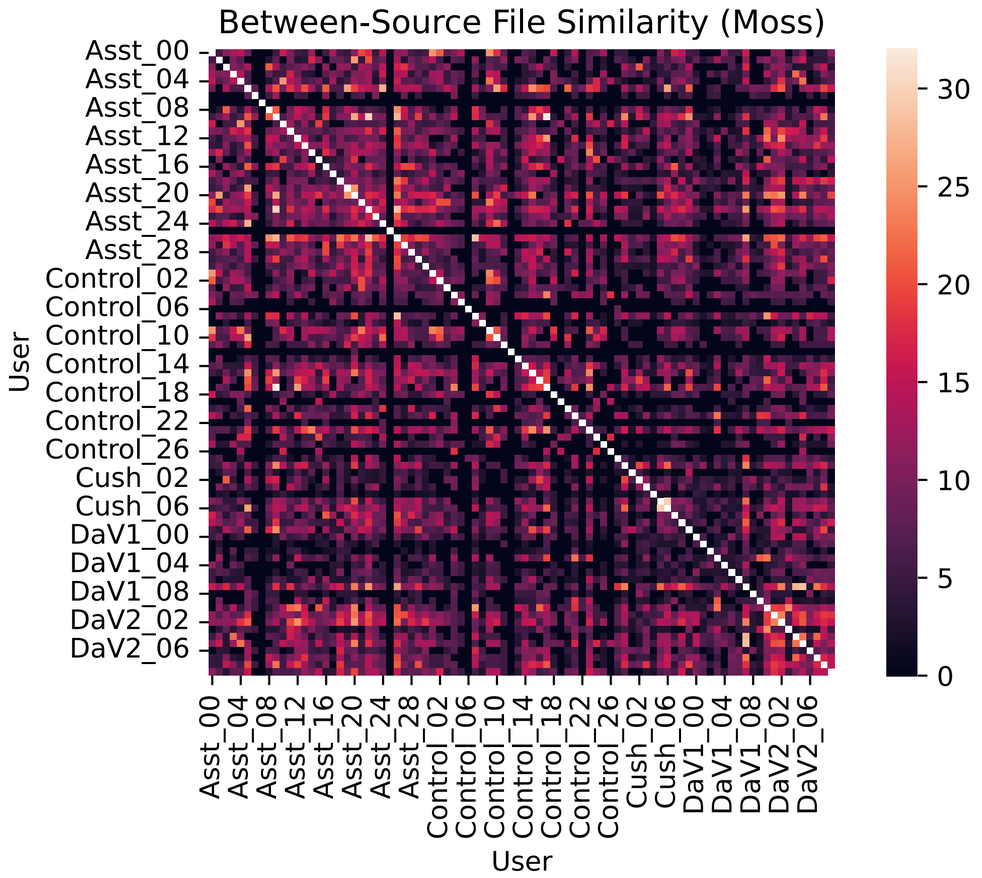
CWE-824 Access of Uninitialized Pointer

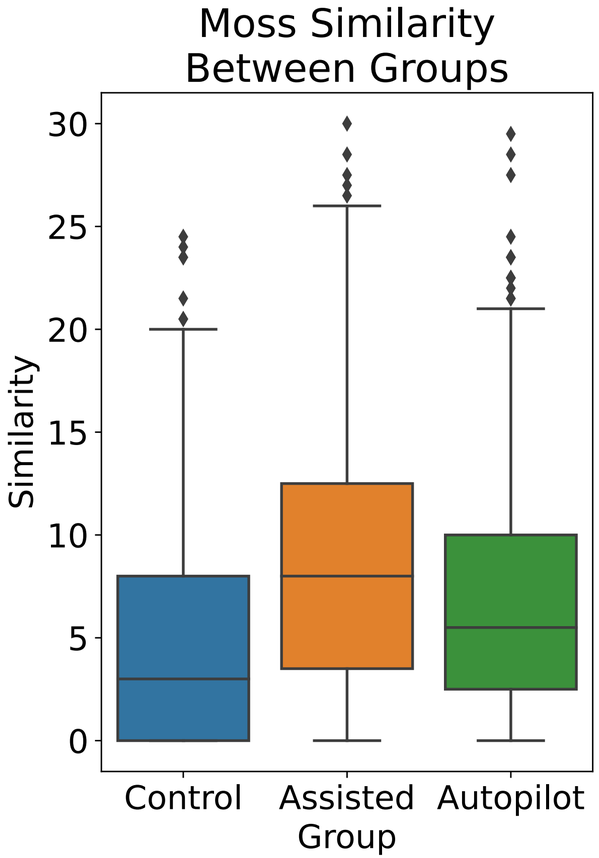
CWE-835 Loop with Unreachable Exit Condition ('Infinite Loop')

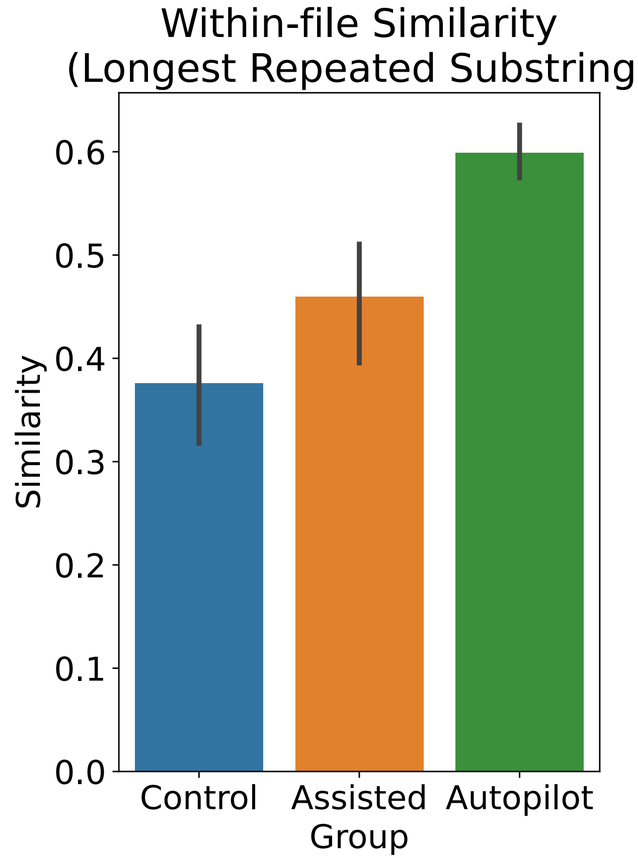
Measuring Style

- We wanted to check if there were difference in style between human and AI-assisted users
  - Can we tell if someone is using Copilot?
- We used two measures:
  - The Moss plagiarism detection tool to measure similarity between users
  - The quantity of *repeated substrings* in the file to measure similarity *within* an individual user's submission

Style Results







On the Origin of Bugs

git blame codex

- Using the data from the IDE, can we identify where vulnerabilities were introduced into the user's code?
  - In particular, did they come from **Codex suggestions** or were they written by **humans**?
- **Idea**:
  - Find an automated way to check for some common vulnerability
  - Use our document snapshots and suggestion data to see if it first appeared in a **document** (human-written) or **suggestion** (introduced by Codex)

Bug Origins: Missing `strdup`

- We picked one bug for this that we could identify with just a regular expression
  - Vulnerability failing to make a copy of the `item_name` provided by the caller (e.g. using strdup) before storing it in the node
  - Can lead to **CWE-416: Use-After-Free** because the list library has no control over when the user-provided string will be freed
  - We can identify it by just looking for direct assignments to `node->item_name` with no strdup/strcpy/malloc

Bug Origins: Results

- This vulnerability was introduced by Codex more often than not
- But some users introduced it themselves, and did not accept further buggy suggestions
- Some users got a **lot** of buggy suggestions (69 in one case!)
- Weak trend: more bug suggestions => more bugs in final file

Participant ID	First location of bug (document / suggestion)	# Bug suggestions	# Bug suggestions accepted	# Bugs in final file
0640	Suggestion	5	3	3
1f1c	Document	5	0	2
2125	Document	0	0	3
26a4	Suggestion	3	1	2
3533	Suggestion	2	1	1
36de	Suggestion	69	5	4
3cff	Suggestion	2	2	2
514e	Document	1	1	1
7193	Suggestion	13	1	2
74bd	Suggestion	4	2	2
925c	Suggestion	8	2	1
a3ed	Suggestion	10	2	2
a4b3	Suggestion	11	5	4
a5ba	Document	0	0	1
a80d	Document	6	3	3
a974	Suggestion	12	5	3
b59f	Suggestion	8	2	2
be6f	Suggestion	4	1	2
c23b	Suggestion	20	10	5
dac3	Document	10	2	2
dc47	Suggestion	1	0	2
ddac	Suggestion	13	1	1
ec83	Document	11	3	2
fd62	Suggestion	12	1	1

Bonus Qualitative Content

Not everyone enjoyed the AI's help

```
// was fighting the language model whenever I was trying
// to do anything and I ended up giving up, because whenever
// I would start with an idea, it would suggest something that
// looked good at first sight, I would add it to my own code
// and then I would spend time debugging some of its code rather
// than develop my ideas

// for the other functions where it gave me the answer straight
// up and it just worked, I felt like I just cheated and got
// someone else to do the work for me

// I ended up not having too much time to finish and the couple
// hours I spent on this was mostly just fighting with the robot
```

Limitations

Learning to live with small N

- Biggest limitation: due to small sample size, most of our results are **not statistically significant** (particularly for security)
  - But *probably* we can rule out really big effects
- Participants were all university students; we can't generalize to professional developers
  - Hopefully they can write better C code?
- Likewise, this is just one task (linked list) and one language (C)
  - Maybe other tasks and languages would give different results?

Conclusions

Check out the paper! https://arxiv.org/abs/2208.09727

- Significant differences in functionality between groups on **functionality**
- Surprisingly, **no discernible difference** on security
  - Limited by small sample size
  - *Maybe* a slight trend in favor of Codex
- Potentially found a signal we can use to distinguish **Copilot/Codex** written code from human-written code (repetition)
  - Has implications for stylometry, confirms that tendency toward repetition may *amplify* the existing vulnerabilities in the code