CISSE™

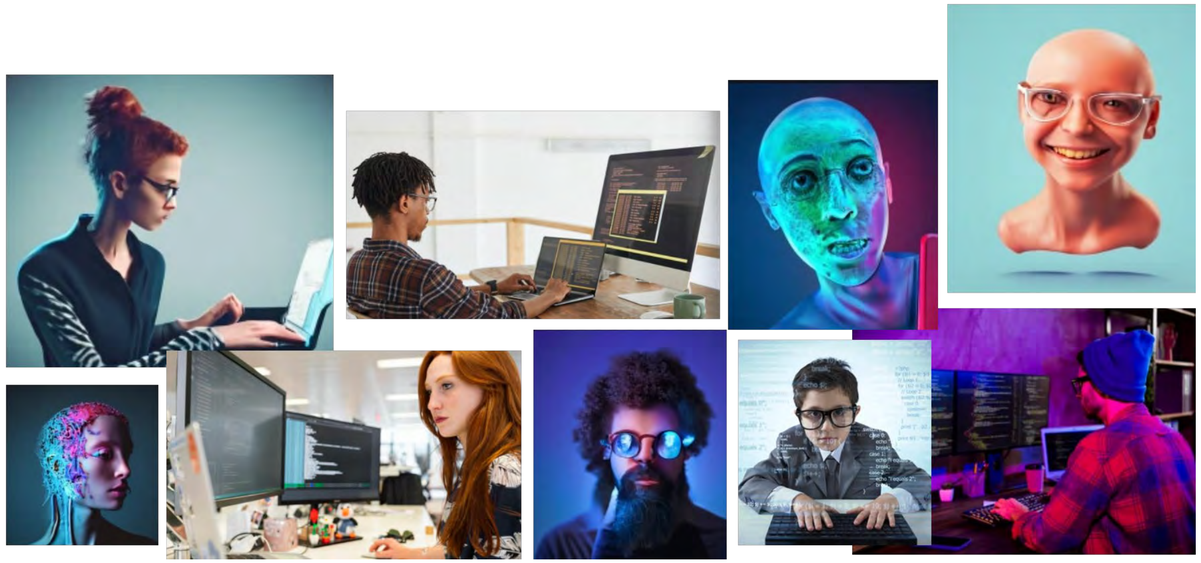
Assessing the Effectiveness & Security Implications of AI Code Generators

Maryam Taeb, Dr. Shonda Bernadin, Dr. Hongmei Chi

FAMU-FSU Engineering
Electrical and Computer Engineering

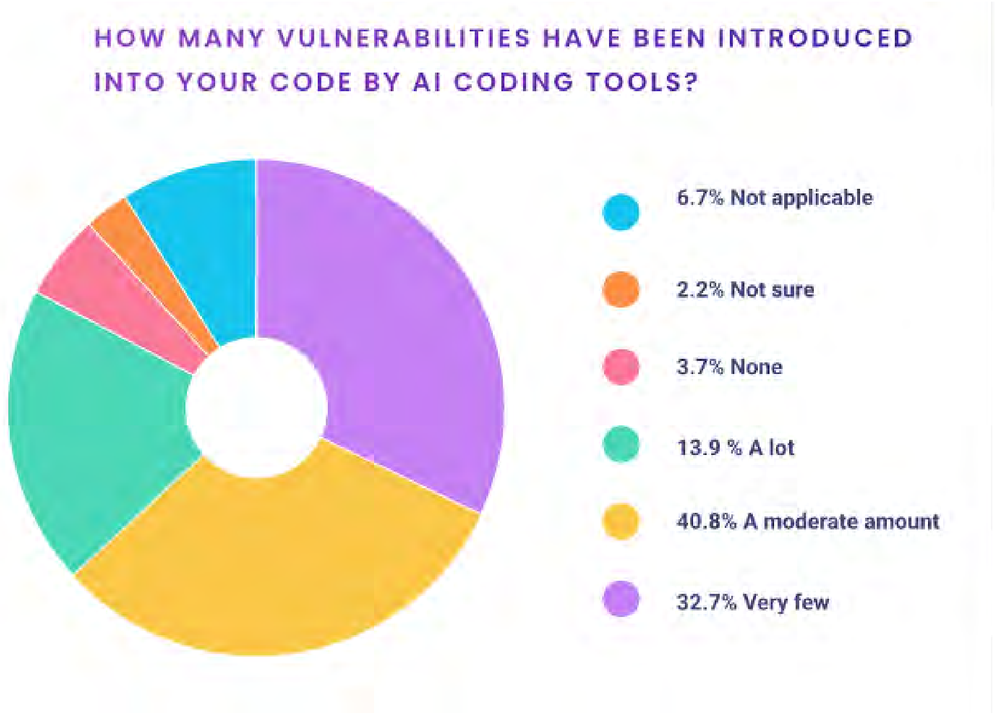
Large Language Models & Programming

- AI pair programming adopted to help across the tech stack
- The quality of AI-Assisted Code generation tools evaluation
- Can LLMs enhance, debug, generate and document code?



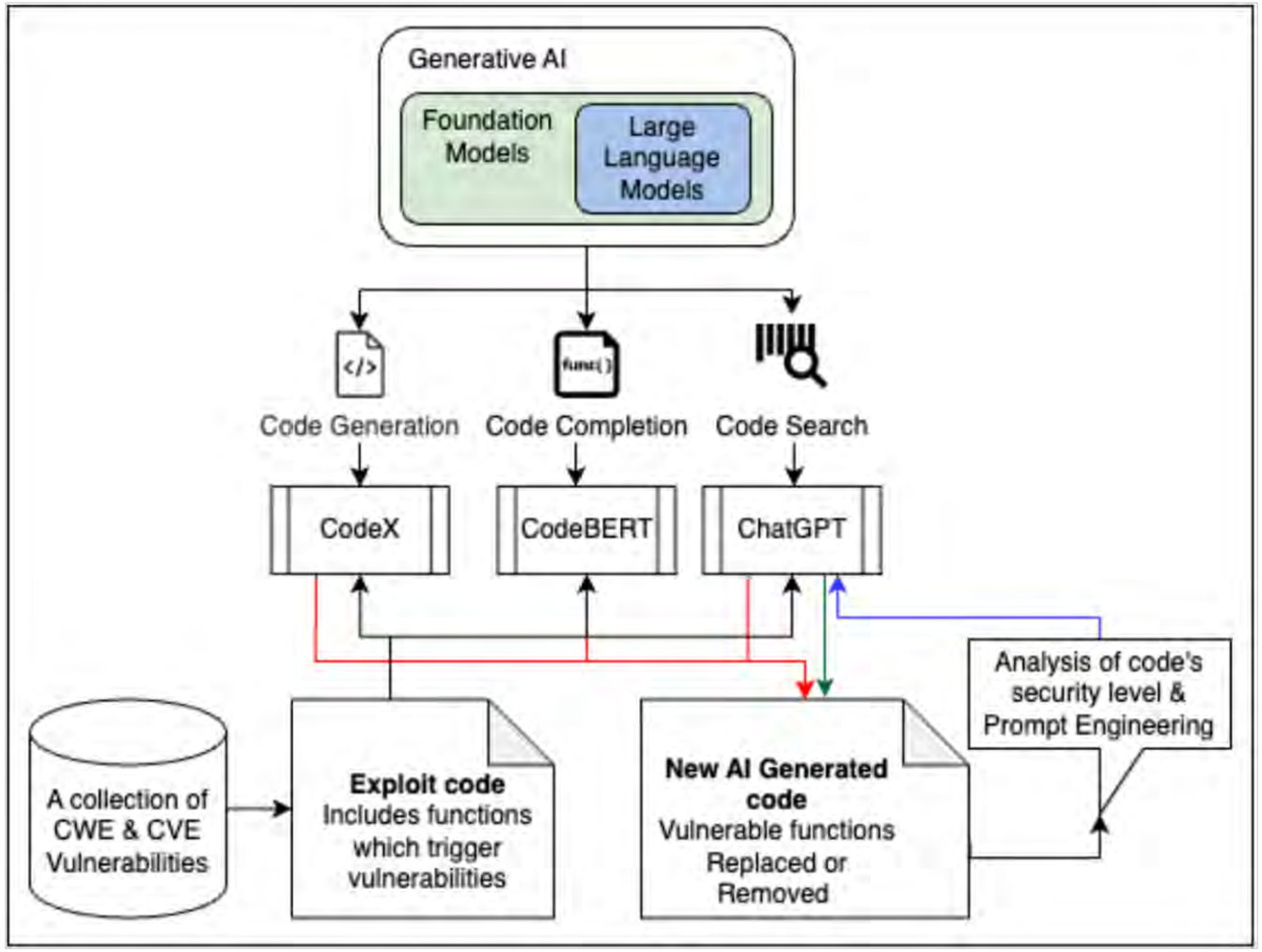
A New Era for cybersecurity training

- 31% of university students are using AI to assist with their assignments
- AI coding tools has caused $0.8% of industrial code vulnerabilities
- General-purpose vs Enterprise grade ai code generation



Evaluation Approach

- Main goal —> Evaluating secure coding practices of LLMs and the use of non-vulnerable built-in functions in recommended code



Covered Vulnerabilities

Type II
Log File Analysis
Microsoft Event & WEB Application Log
File Parsing & Analysis using NLP

Tool: Loggly by SolarWinds
Vulnerability: SQL Injection
- Windows Administrative Events
- Web Server Client & Server-Side Events

Tool: LogViewPlus
Vulnerability: Cross Site Scripting
- Brute Force Attack
- Cross Site Forgery
- SYN Flood

Tool: Microsoft Log Parser & BERT
Vulnerability: DDoS Attack
- Regular Expressions
- BERT
- IP Analysis

Type I
Source Code Vulnerability Detection Using Static Analysis Tools
Exploring CVEs & CWEs and their fixes

Tool: Clang-Tidy
Vulnerability: Insufficient Input Validation:
- CWE-686 & 134
- Format String Attack
- Invalid String Format
- Un-Sequenced Modification to Variable

Tool: Visual Code Grepper
Vulnerability: Memory Errors & Leaks:
- CWE-119 & 120
- Buffer Overflow
- Memory Allocation

Tool: FlawFinder
Vulnerability: XNU Memory Leak
- CVE-2017-13868
- Detecting and Resolving the Vulnerability

Type III
Secure Coding Practices
Introducing Automatic monitoring of deployed code using ML Techniques

Tool: PwnXSS
Vulnerability: XSS
- CVE-2021-33570
- Stored XSS
- Developing a Scanner from scratch

Tool: Snyk, Dependabot, Deep Representation Learning
Vulnerability: Pointer Dereference
- CWE-20,457,805,

Generative AI Code Generation tools

- CodeX Creating functions that would execute equivalent tasks as our code snippets, and analyzing the usage of built-in functions
- CodeBERT Predicting the names of built-in functions through the masking technique
- GPT 3.5
  - Assessing whether the generated code produced by CodeX, CodeBert, and the previously designed vulnerable code contained any vulnerabilities
  - Receiving recommendations for mitigation techniques and comparing suggestions with static code analysis tools' suggestions

Vulnerable Code illustration

A function that takes user input and displays it as an output

- printf → expects a format string
  - undefined behavior
- No input validation or error checking on the scanf
  - undefined behavior
- Input variable is declared as a global variable
  - Hard maintenance
- No bounds checking on the input variable
  - buffer overflow vulnerability.
- No return statement
  - undefined behavior

```
1  #include <stdio.h>
2  int input;
3  scanf ("%d", &input);
4  printf (input);
```

Example Behavior

CodeX

```c
#include <stdio.h>
int main() {
    char input[100];
    printf("Please enter something:");
    scanf("%s",input);
    printf("You entered: %s\n",input);
    return0;
}
```

CodeBERT

```c
#include <stdio.h>
int main() {
    char input[100];
    printf("Please enter something:");
    = ("%s",input);
    printf("You entered: %s\n",input);
    return0;
}
```

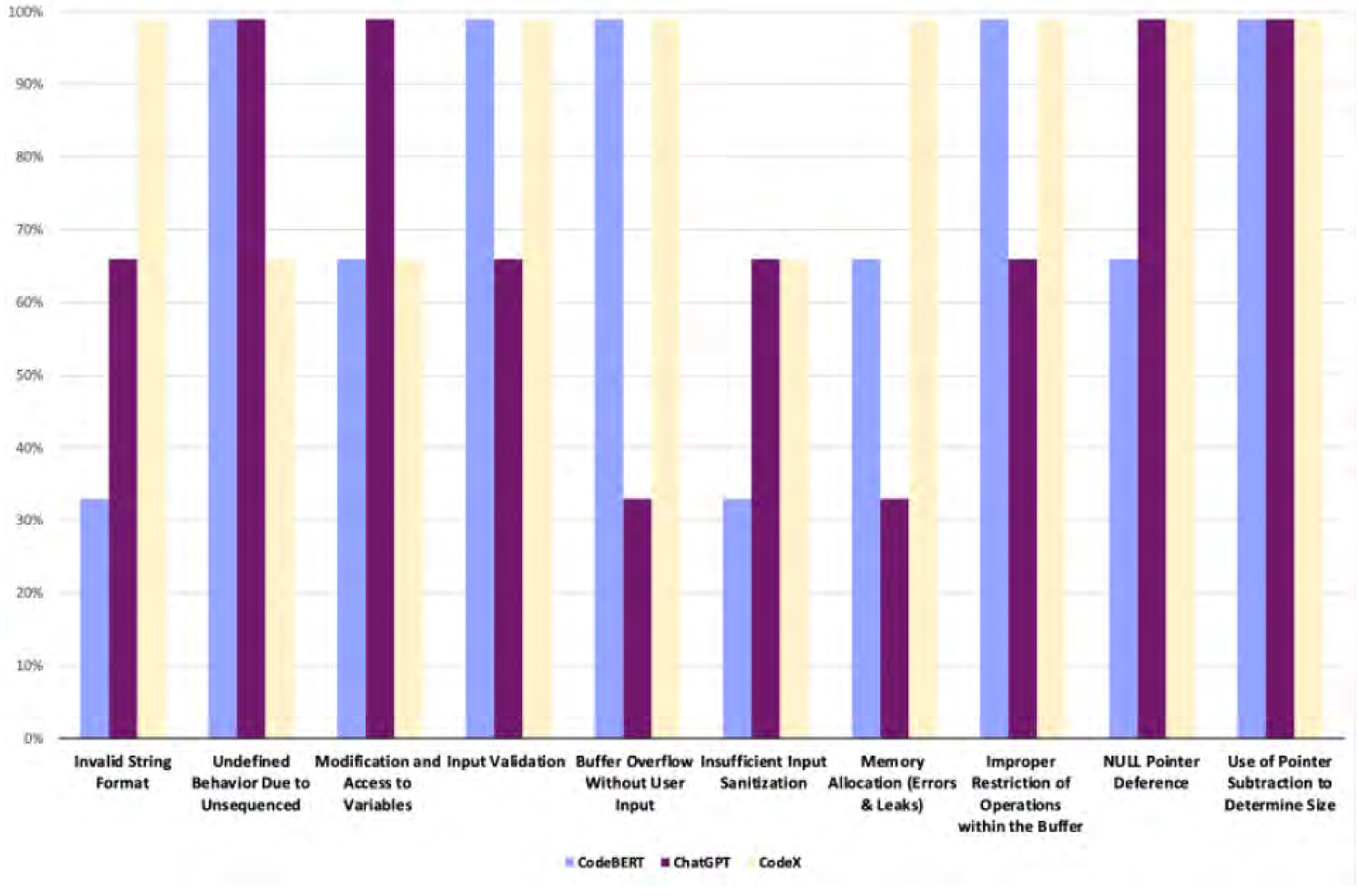
GPT3.5

```c
#include <stdio.h>
int main() {
    char userInput [100];
    fgets (userInput, 100, stdin);
    printf ("You entered: %s", userInput);
}
```

GPT 3.5 vulnerability mitigation recommendations

```c
#include <stdio.h> #include <stdlib.h>
int main() {
    int input;
    if (scanf ("%d", &input) != 1)
    {
        printf("Errror: invalid input \n");
        return 1;
    }
    printf("%d\n", input);
    return 0;
}
```

Results


100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Invalid String Format
Undefined Behavior Due to Unsequenced
Modification and Access to Variables
Input Validation
Buffer Overflow Without User Input
Insufficient Input Sanitization
Memory Allocation (Errors & Leaks)
Improper Restriction of Operations within the Buffer
NULL Pointer Deference
Use of Pointer Subtraction to Determine Size
CodeBERT
ChatGPT
CodeX

Discussion

- CodeX had the highest code generation capability, generating code
- GPT 3.5 had relatively lower code generation capabilities but excelled in explaining potential vulnerabilities, commenting on the code, and analyzing log files
- CodeBERT weak Performance in terms of built-in function suggestion

- Best use case of AI-Assisted Code generation tools is in Acceleration mode when Programmer already knows what they want to do next

Thank you for your time

Maryam Taeb