THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD

CodeWars: Using LLMs for Vulnerability Analysis in Cybersecurity Education

Arunima Chaudhary[1], Walter Colombo[1], Amir Javed[1]
Junaid Haseeb[2], Vimal Kumar[2], Richard Larsen[2]
1 Cardiff University (UK) 2 The University of Waikato (New Zealand)

Can AI Models be Used to Teach Cyber Security?

Dual approach – Cardiff / Waikato

Can AI model generate secure respecting the **Secure Software Development (SSD)** standards that can be used in Cyber Security education ?

Open Worldwide Application Security Project (OWASP)

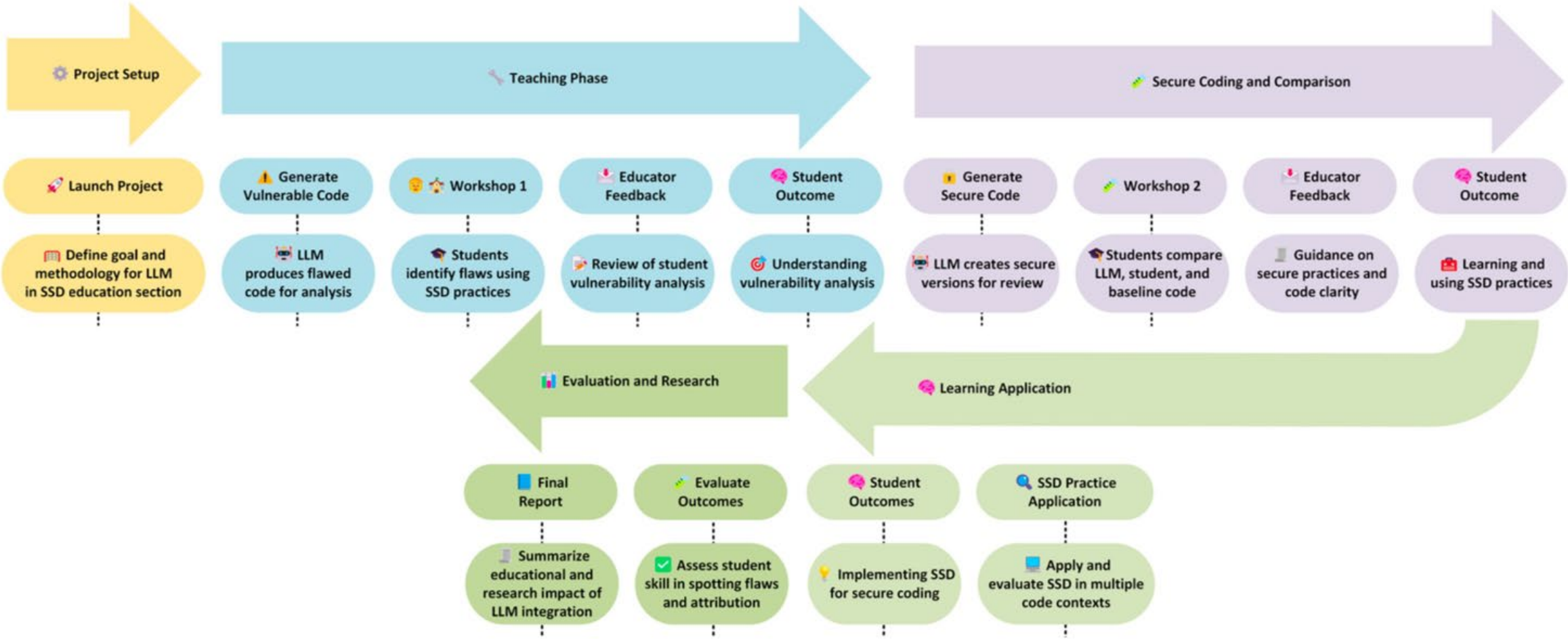
https://owasp.org/Top10/A00_2021_Introduction/
https://owasp.org/Top10/A03_2021-Injection/

Can AI models produce code specifically targeted for teaching, e/g including correct examples of vulnerabilities (security errors)?

Motivation and Objectives

- Investigate the integration of LLMs into Secure Software Development (SSD) education by developing case studies that evaluate adherence to SSD practices against the Open Worldwide Application Security Project (OWASP) Top 10 (2021) guidelines.
- Examine how students identify vulnerabilities in LLM and human-generated code.
- Evaluate students' ability to distinguish between LLM-generated and human-written code, and analyse the factors influencing their authorship judgments.
- Explore student perceptions, preferences, and ease of understanding different code origins.
- Evaluate the pedagogical effectiveness of LLMs for secure coding instruction.

Project Framework



Case Study 1 (Student Generated)

THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD

```java
// function for finding user by email
public User findByEmail (String email) {
// SQL query to find user by email
String sql = "SELECT * FROM users WHERE email = ?";

try {
    // Using queryForobject to directly return the result as a single User object
    return jdbcTemplate.queryForObject(sql, (rs, rowNum) -> {
        User user new User();
        user.setId(rs.getInt(columnLabel: "id"));
        user.setEmail(rs.getString(columnLabel: "email"));
        user.setPassword(rs.getString(columnLabel:"password"));
        user.setFullname(rs.getString(columnLabel:"fullname"));
        user.setDateOfBirth(rs.getObject(columnLabel:"dob", type: LocalDate.class));
        user.setRoleId(rs.getInt(columnLabel: "role_id"));
        user.setOrganization(rs.getString(columnLabel: "organization"));
        return user;
    }, email);
} catch (EmptyResultDataAccessException e) {
// return null if no user is found with the email
return null;
}
```

Cardiff study

Ask LLMs to generate '*secure code*'

Prompt Engineering – Chain of Thought - Few Shot Learning

The sequence of prompts included:

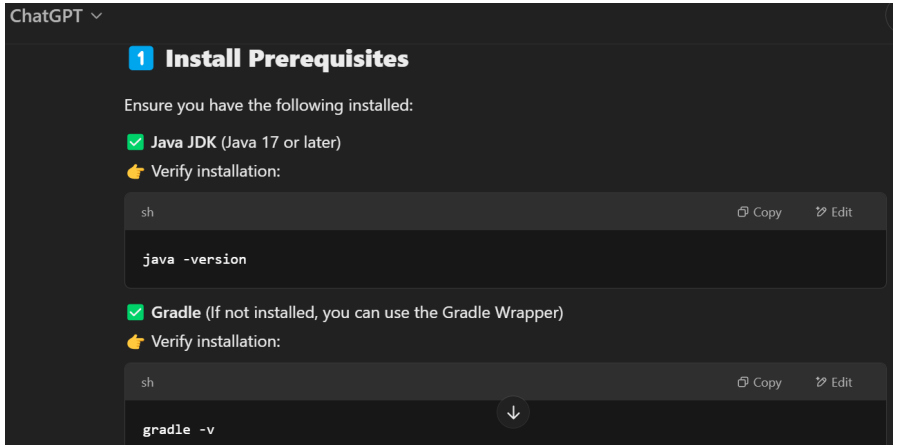
- Application Setup
- Configuring the application using Gradle.
- Implementing a responsive login page with attention to data privacy and security

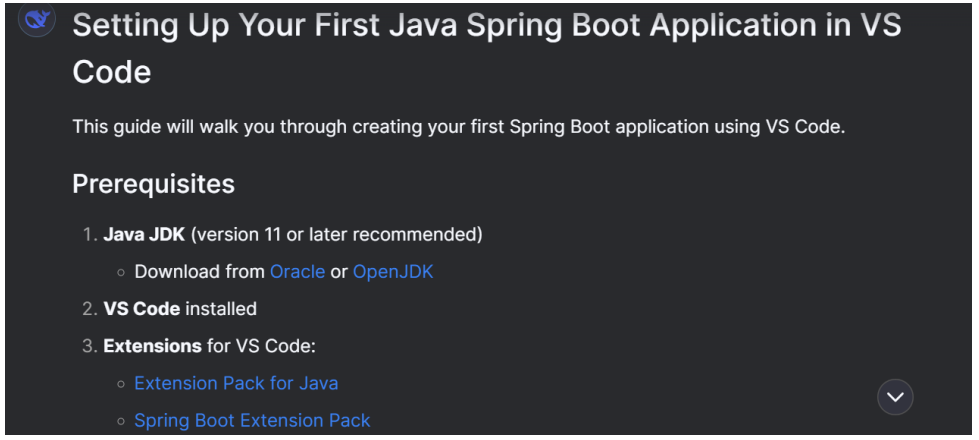
The generated outputs (ChatGPT, DeepSeek, Gemini) were first assessed for functional completeness, deployment feasibility, correct application layer structure in adherence to the SSD practices when prompted with client requirements.

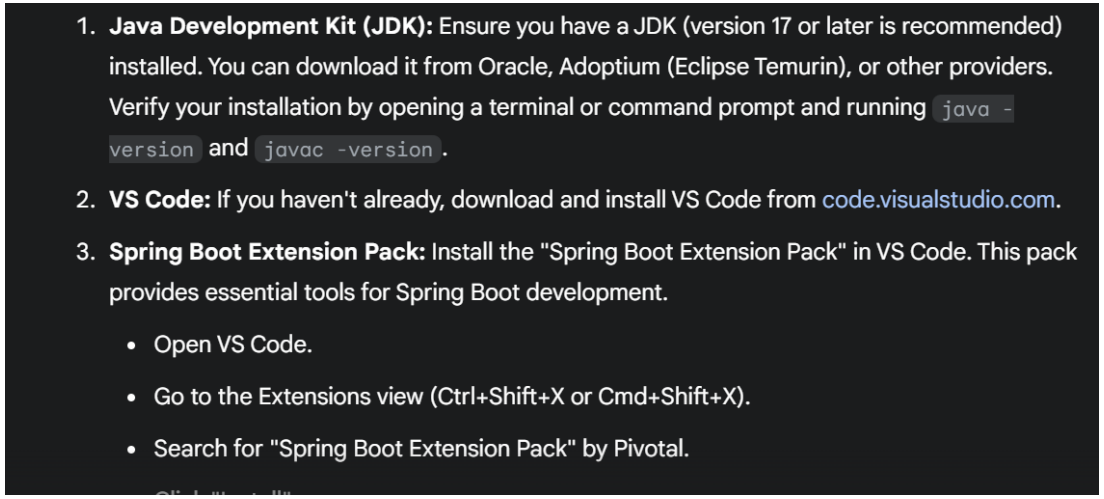
To develop a chain of thought reasoning for the language models.

At first it was tested on how well versed these models were when asked to set up the application.

Prompt-1 "Help me set up my first java spring boot application using vs code"







Followed by models' response another prompt was given for choosing the building tool for the application

Prompt-2 "How can I do this using gradle"

At this point of the research, we were able to fine tune the model to be able to deduce the correct working application directory structure and establish correct implementation of the coding practice across these models.

Finally, we navigated the model towards developing a login page code for us using the following prompt.

Prompt-3 "Now help me create a login page, ……. Additional Considerations: Ensure the app has a responsive design to allow access from mobile devices. Data privacy and security, especially around user accounts and media uploads, must be maintained. The app should be easy to navigate for non-technical users, with an intuitive interface. "

Results

After evaluating the code generated across ChatGpt, DeepSeek and Gemini it was established that **DeepSeek** had the most logically developed application code in terms of the completion and communication between the different application layers and generated code the most reliable one for deployment.

it was provided with a bootstrap's responsive grid system which was in adherence to our prompt for a *"responsive design to allow access from mobile devices" and* generated a CSS style sheet which was completely overlooked by the other models

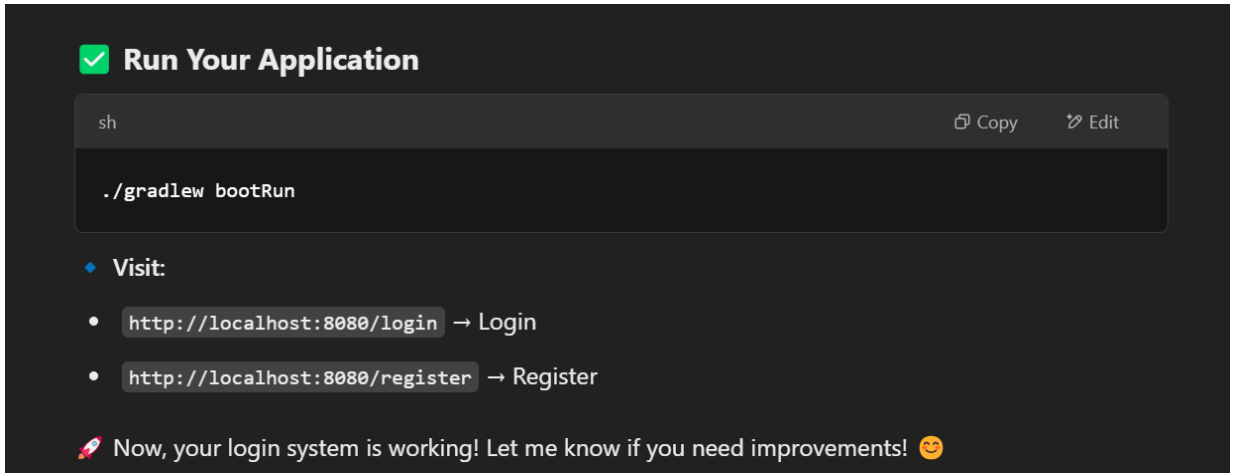
Furthermore, while other language models responses mentioned deployment measures, DeepSeek evaluated its response more towards feature completion of the prompt, thus establishing dominance in reasoning over its counter parts.

Gemini

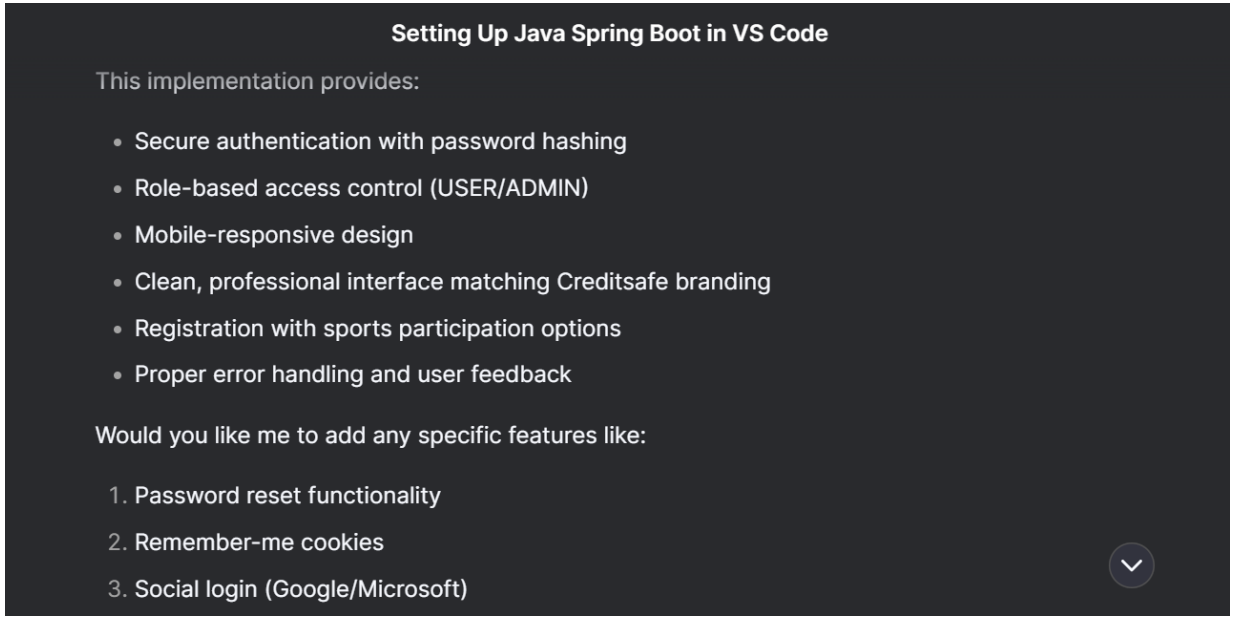
8. Run the application:

Run `./gradlew bootRun`

ChatGpt



DeepSeek



Therefore, DeepSeek generated code was deployed and assessed for secure coding across the OWASP guidelines.

Case Study 2 (LLM Generated)

THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD

```java
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.password. PasswordEncoder;
import org.springframework.stereotype.Service;

@Service
public class UserService implements UserDetailsService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    public UserService (UserRepository userRepository, PasswordEncoder passwordEncoder) {
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
    }

    @Override
    public UserDetails loadUserByUsername (String email) throws UsernameNotFoundException {
        return userRepository.findByEmail(email)
            .orElseThrow(() -> new UsernameNotFoundException(msg: "User not found"));
    }

    public User registerUser (User user) {
        user.setPassword (passwordEncoder.encode(user.getPassword()));
        return userRepository.save(user);
    }
}
```

Cardiff Study

- Code Snippets evaluated against the Open Worldwide Application Security Project (OWASP) guidelines:
  - *Positives:* Password Storage, Role-based Access Control, Secure Session Management, Security Based Authentication
  - *Negatives:* Lack of Password Policy Enforcement and Account Lockout Mechanisms, Absence of Password Length and Complexity Validation, No Handling of Failed Authentication Attempts.

Mixed Expertise Cohort – Postgraduates but not only specific to Cyber Security

Case Study 3 (Human Generated- Control Case)

THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD

```python
def index():
            if current_time - failed_attempts[ClientIP]['time'] < BLOCK_TIME:
                return "<pre>Too many failed attempts. Try again later in 10 minutes.</pre>", 403
            else:
                del failed_attempts[ClientIP]
        if 'Submit' in request.args:
            user_id = request.args.get('id', '').strip();password = request.form.get('password', '')

            if not re.fullmatch(r'\d+', user_id):
                return "<pre>Invalid ID format.</pre>", 400
            try:
                conn = get_db_connection()
                with conn.cursor() as cursor:
                    query = "SELECT first_name, last_name, password_hash FROM users WHERE user_id = %s"
                    cursor.execute(query, (user_id,));result = cursor.fetchone()
                    if result:
                        first_name, last_name, stored_hash = result
                        if bcrypt.checkpw(password.encode('utf-8'), stored_hash.encode('utf-8')):
                            if ClientIP in failed_attempts:
                                del failed_attempts[ClientIP]
                            output = f"ID: {user_id}<br>First name: {html.escape(first_name)}<br>Surname: {html.escape(last_name)}<br>"
                            return render_template_string(f'<pre>{output}</pre>')
                        else:
                            if ClientIP not in failed_attempts:
                                failed_attempts[ClientIP] = {'count': 1, 'time': current_time}
                            else:
                                failed_attempts[ClientIP]['count'] += 1
                                failed_attempts[ClientIP]['time'] = current_time
                        return "<pre>No user found.</pre>", 404
            except mysql.connector.Error as err:
                return f"<pre>Database error: {html.escape(str(err))}</pre>", 500
            finally:
                conn.close()
        return """<form method='GET'><input type='text' name='id' required pattern='\\d+'>
                    <input type='submit' name='Submit' value='Submit'></form>"""
if __name__ == '__main__':
        app.run(debug=False, host=HOST, port=PORT, ssl_context=('server.crt', 'server.key'))
```

1) Attribution: is the Code Human or LLM Generated?

- This revealed a clear trend from participant comments:
  - *Untidy / Messy* code and presence of *Errors* was attributed to Humans
  - Presence of *Comments*, *Tidy, Structured* and *Well-Formatted* code with very few or no errors was attributed to AI

Attribution: is the Code Human or LLM Generated?

28. Among the three Case Studies, which ones do you believe were written by a human?	29. Please specify the reason for your answer.	30. Among the three Case Studies, which ones do you believe were written by a LLM?	31. Please specify the reason for your answer.
Case Study 1;Case Study 2	Case Study 1: Apparently Human due to pragmatic but suboptimal design (e.g., SELECT *, incorrect COLUMN_LABEL syntax) and usual human errors.	Case Study 3	It LLM-generated because of it's unusual syntax, mixed languages, simplistic logic, and lack of production-ready design.
Case Study 1;Case Study 3	Study 1, code style is not consistent with Java standards, comments don't follow correct capitalization, unhashed credentials would not be typical in modern	Case Study 2	Strict adherence to standards
Case Study 3	GPT's will not ommit any necessary headers and goves full code. Case study 3 seems to lack the coherence and missinf few important headers. The naming	Case Study 1	Well formatted code with typical naming of variables and attributes. Addition of comments in the code increases the chance
Case Study 2	AI tools tend to make a lot of issues related to usage of legacy libraries, but make lesser synractical errors. Case Study 2 has a few syntactical errors.	Case Study 1. Case Study 3	Generic variable names used. Some poor practices like no parametrised queries Skeleton code returned in Cse
Case Study 3	The least fatal errors and counted.	Case Study 3	seems too advanced to be human development.
Case Study 2	Logic in Case study 2 wasn't the soundest in code flow and I thought contained a syntax error	Case Study 1 and Case Study 3	Knew how to code correctly looking but maybe a few weird design errors
Case Study 1;Case Study 3	The code seem more inconsistent in Line 1&3 in how it his styled and written which I think points to being made by a human	Case Study 2	Clean looking code

28. Among the three Case Studies, which ones do you believe were written by a human?	29. Please specify the reason for your answer.	30. Among the three Case Studies, which ones do you believe were written by a LLM?	31. Please specify the reason for your answer.
Case Study 1;Case Study 3	jjkjk	Case Study 2	jkjkkj
Case Study 2	errors	Case Study 3	almost no errors
Case Study 1;Case Study 2	Some errors were identified because of a syntax error or a stray whitespace, which in my experience has been caused by human error, the artificially generated code has	Case Study 3	I was unable to find any glaringly obvious errors
Case Study 2	A lot messier	Case Study 3	More consistent, neat code
Case Study 2;Case Study 3	The error seems to be manual error	Case Study 1	the comment format looks more AI generated
Case Study 2;Case Study 3	Iâ€™d choose Case Study 2 (Python Flask) due to the glaring security flaws (e.g., using GET for a login form) and practical oversights (e.g., no session management),	Case Study 3	Case Study 3: Most likely (structured, polished, unusual design choices like HTML responses).
Case Study 2	It was the most conventional secure development practices	Case Study 3	Does not use consstent software development practices
Case Study 1;Case Study 3	Case Study 1 and 3 is inconsistent, probably human	Case Study 2	It looks clean, proper indentation is used.
Case Study 2	The format feels more human and has a better understanding on the structure of the code, even though they import the large library	Case Study 1	Has comments and only targets on the specific task without thinking the entire code base
Case Study 2	Spacing seems human	Case Study 3	
Case Study 2;Case Study 3	There are a lot of human errors or things that AI just wouldn't generate or will fix	Case Study 1	The comments and code seems to be AI and then was went over by a human to make it seem like their own

Only few exceptions argued that humans would instead generate '*better*' code

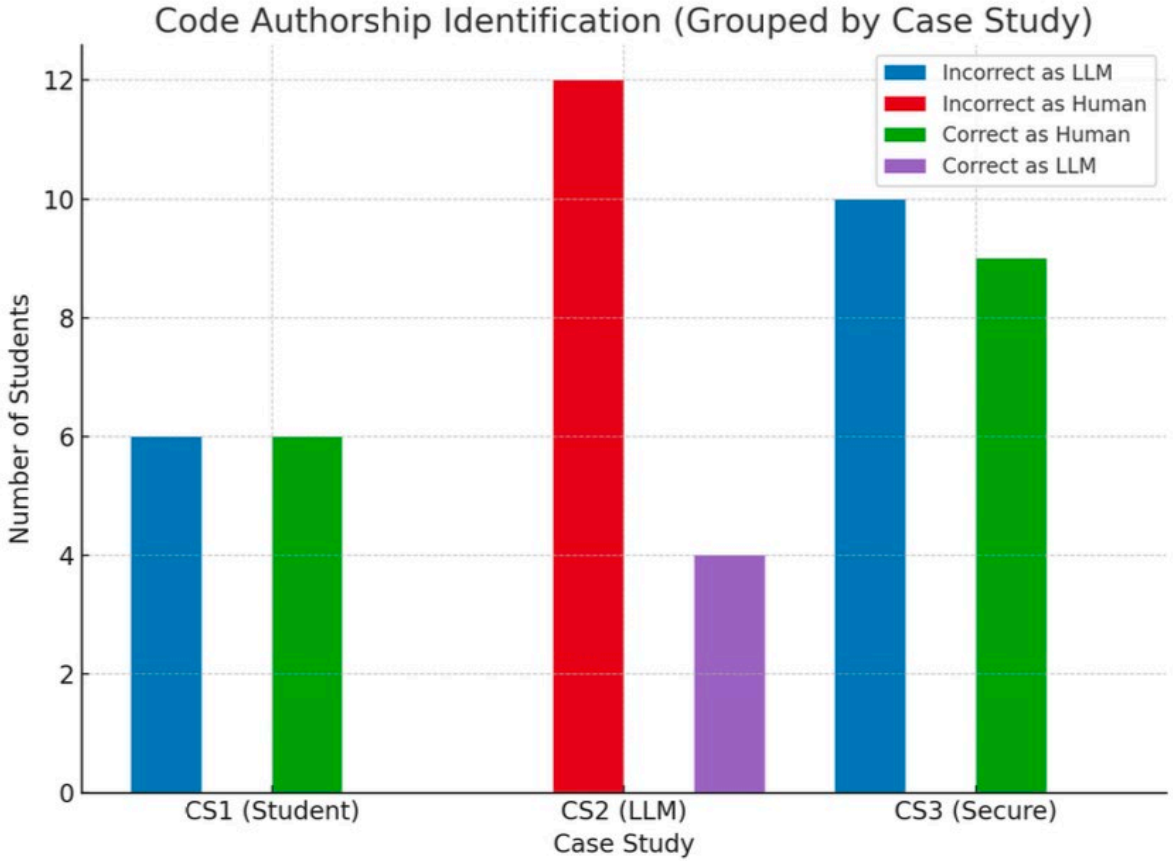
28. Among the three Case Studies, which ones do you believe were written by a human?	29. Please specify the reason for your answer.	30. Among the three Case Studies, which ones do you believe were written by a LLM?	31. Please specify the reason for your answer.
Case Study 1;Case Study 2	Case Study 1: Apparently Human due to pragmatic but suboptimal design (e.g., SELECT *, incorrect COLUMN_LABEL syntax) and usual human errors.	Case Study 3	It LLM-generated because of it's unusual syntax, mixed languages, simplistic logic, and lack of production-ready design.
Case Study 1;Case Study 3	Study 1, code style is not consistent with Java standards, comments don't follow correct capitalization, unhashed credentials would not be typical in modern	Case Study 2	Strict adherence to standards
Case Study 3	GPT's will not ommit any necessary headers and goves full code. Case study 3 seems to lack the coherence and missinf few important headers. The naming	Case Study 1	Well formatted code with typical naming of variables and attributes. Addition of comments in the code increases the chance
Case Study 2	AI tools tend to make a lot of issues related to usage of legacy libraries, but make lesser synractical errors. Case Study 2 has a few syntactical errors.	Case Study 1. Case Study 3	Generic variable names used. Some poor practices like no parametrised queries Skeleton code returned in Cse
Case Study 3	The least fatal errors and counted.	Case Study 3	seems too advanced to be human development.
Case Study 2	Logic in Case study 2 wasn't the soundest in code flow and I thought contained a syntax error	Case Study 1 and Case Study 3	Knew how to code correctly looking but maybe a few weird design errors
Case Study 1;Case Study 3	The code seem more inconsistent in Line 1&3 in how it his styled and written which I think points to being made by a human	Case Study 2	Clean looking code

28. Among the three Case Studies, which ones do you believe were written by a human?	29. Please specify the reason for your answer.	30. Among the three Case Studies, which ones do you believe were written by a LLM?	31. Please specify the reason for your answer.
Case Study 1;Case Study 3	jjkjk	Case Study 2	jkjkkj
Case Study 2	errors	Case Study 3	almost no errors
Case Study 1;Case Study 2	Some errors were identified because of a syntax error or a stray whitespace, which in my experience has been caused by human error, the artificially generated code has	Case Study 3	I was unable to find any glaringly obvious errors
Case Study 2	A lot messier	Case Study 3	More consistent, neat code
Case Study 2;Case Study 3	The error seems to be manual error	Case Study 1	the comment format looks more AI generated
Case Study 2;Case Study 3	Iâ€™d choose Case Study 2 (Python Flask) due to the glaring security flaws (e.g., using GET for a login form) and practical oversights (e.g., no session management),	Case Study 3	Case Study 3: Most likely (structured, polished, unusual design choices like HTML responses).
Case Study 2	It was the most conventional secure development practices	Case Study 3	Does not use consstent software development practices
Case Study 1;Case Study 3	Case Study 1 and 3 is inconsistent, probably human	Case Study 2	It looks clean, proper indentation is used.
Case Study 2	The format feels more human and has a better understanding on the structure of the code, even though they import the large library	Case Study 1	Has comments and only targets on the specific task without thinking the entire code base
Case Study 2	Spacing seems human	Case Study 3	
Case Study 2;Case Study 3	There are a lot of human errors or things that AI just wouldn't generate or will fix	Case Study 1	The comments and code seems to be AI and then was went over by a human to make it seem like their own

This however led to misattributions:

In Cardiff the perceived presence of a syntax error in the LLM generated code (CS2) led to attribution to humans

The CS2 code was in general perceived as more untidy and with more errors, possibly because of a comparison with the control case aimed to be errorless and fully adhering to SSD standards



- *Case study 1 (Student-written)* Correctly identified as human by 6, misclassified as LLM by 6 – 33.3% correct attribution.
- *Case study 2 (LLM-generated)* Misclassified as human by 12; only 4 correctly identified it as LLM.
- *Case study 3 (Secure baseline)* Misclassified as LLM by 10; correctly identified as human by 9.

Waikato Study

Ask LLMs to generate '*code embedding specific vulnerabilities*' from SSD practices

Buffer Overflow – String Formatting – Command Injection

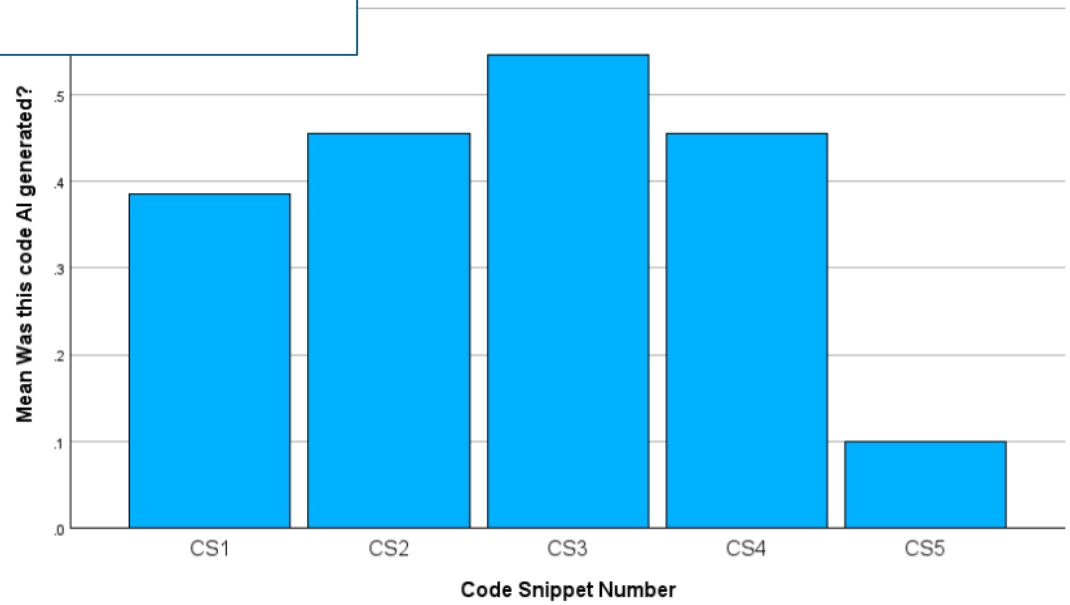
- *Guided Model:*
  - Participants were first introduced to core SSD principles and provided with the predefined list of vulnerabilities expected to be present in the code
  - Asked to analyze the snippets and map each to vulnerabilities from the list
  - Aligns with the *scaffolding* educational theory

Cohort of Cyber Security postgraduates

This however led to misattributions:
In Waikato we observed a similar trend than in the Cardiff workshop: more errors and more untidiness led to attribute the code to human standards

As LLM generated code was intentionally designed to include a (larger) number of specific vulnerabilities and vice versa the human code was produced by an expert researcher with over 20 years-experience the code was (marginally) perceived as more human than AI generated (CS1 & CS2)

CS5 was instead correctly attributed as it presented very untidy and wrongly formatted code and was largely attributed to humans



Waikato study: Total 14 participants

- CS1 & CS2: AI-generated. (Gemini)
- CS3, CS4 & CS5: Human-written.
- CS3 (Human-written): Most frequently misclassified as LLM-generated.
- CS5 (Poorly formatted human code): Most confidently identified as human-written.

Authorship Attribution

The attribution process was primarily driven by **superficial and appearance features** such as **style, tidiness, and presence of (syntax) errors**

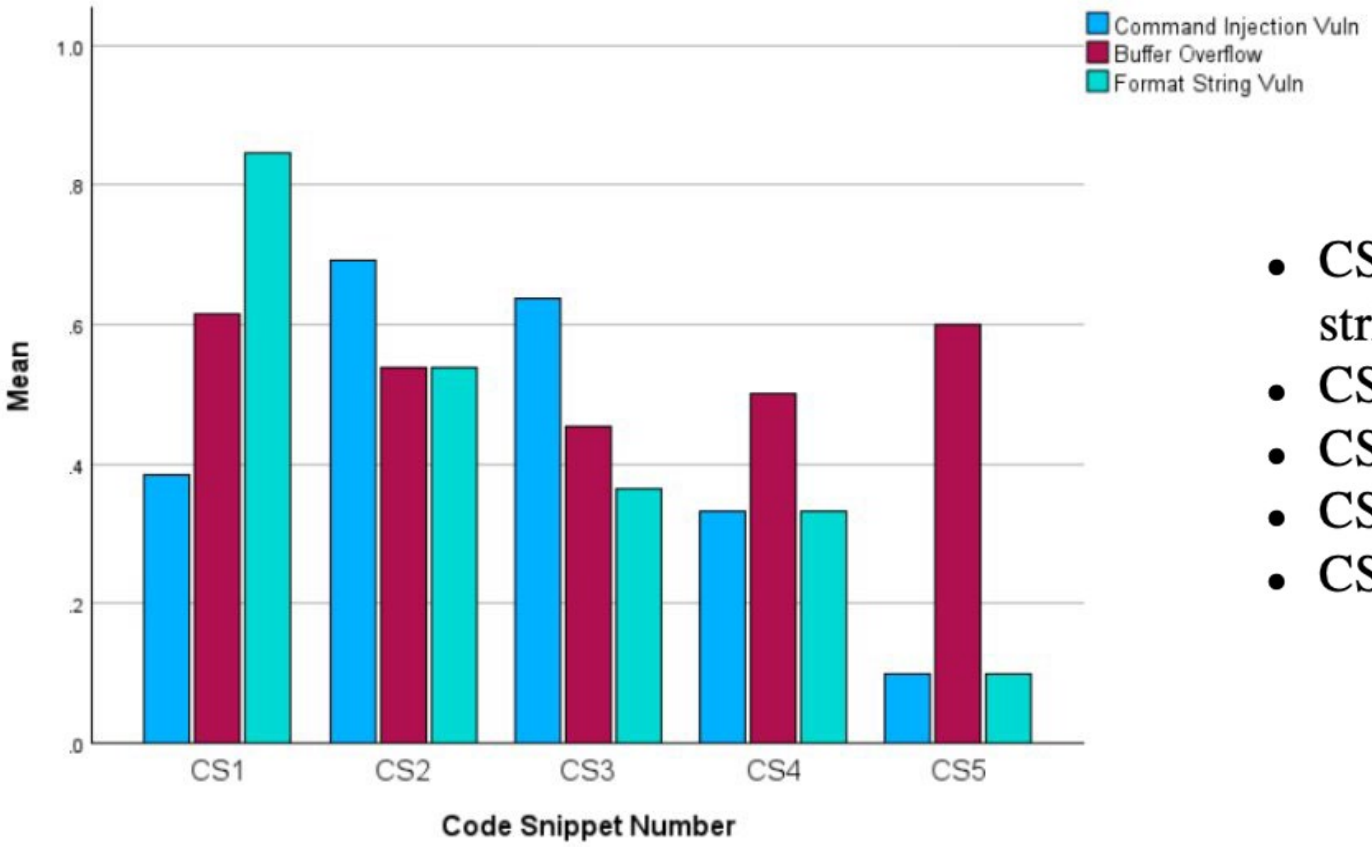
Instead of making judgements based on the **adherence to SSD principles**

2) Vulnerabilities Detection

- **Was AI able to generate examples of code containing vulnerabilities**
  - that were useful for educational purposes in a cyber security context ?
  - that were correctly identified by students ?

Waikato Study

Generated AI code targeting specific vulnerabilities



- CS1: All three vulnerabilities (buffer overflow, format string, command injection).
- CS2: Format string and command injection.
- CS3: Buffer overflow and command injection.
- CS4: Buffer overflow and format string.
- CS5: Two buffer overflows

Waikato Study

- Participants were able to correctly identifying vulnerabilities also - and even better - within the AI generated code (over 60% accuracy - case 5 was an exception - *possible double counting)*
- ***Average accuracy****:* number of selections on average that correctly identified vulnerabilities present in the code

Code Snippet	CI (%)	BO (%)	FS (%)	Accuracy (%)
1	38	61	85	61.33
2	70	53	53	61.50
3	64	45	37	54.50
4	35	50	35	42.50
5	10	60	10	60.00

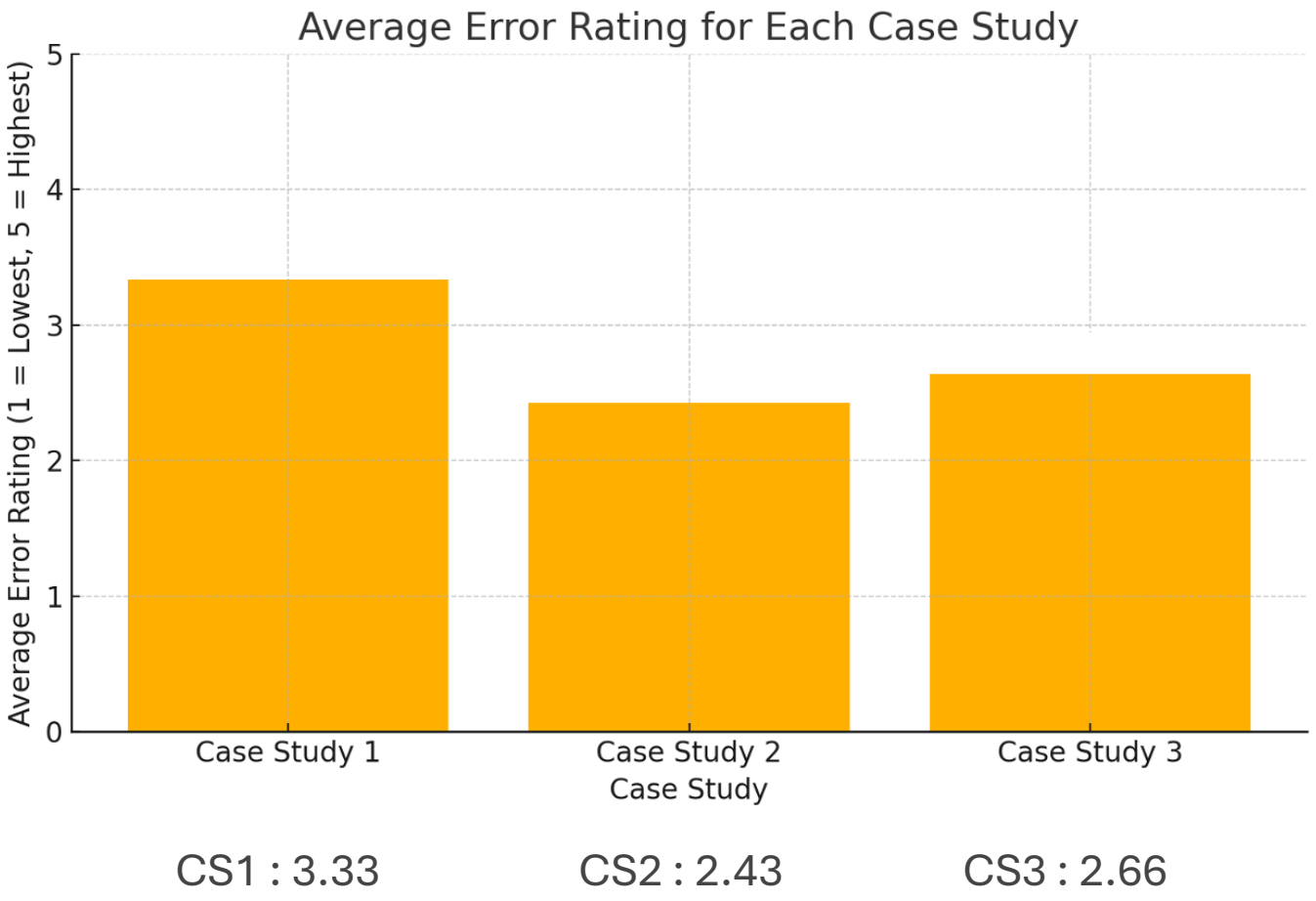
Cardiff Study

- Introduced the ***error rating score*** by asking participants to provide an error rating for each code snippets on a scale 1 to 5 (1 being error-free and 5 many errors)

- Students identified the maximum number of vulnerabilities in the human-written code (23 mentions in total) reflecting typical novice-level security oversights

- These clustered around:
  - *Weak / No Hashing* (4)
  - *Plaintext Password Storage* (3)
  - *SQL injection* (3)

Cardiff Study



- Student code was correctly identified as containing more vulnerabilities
- LLM generated code showed the lowest error ratings *(LLM were specifically asked to generate secure code)*
- Some errors were however identified even in the control case – may be due to the unfamiliar formatting (*numerous comments and notes)*

Further Considerations

THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD

A. Correlation between code complexity and perceived LLM generation ✓

B. Significance of comments and code structure when detecting if the code was LLM generated ✓

C. Correlation between vulnerabilities detected and code authorship X

see for example the Cardiff results where LLM generated code was perceived as containing the lowest number of security errors / vulnerabilities while being largely attributed to humans

Conclusion

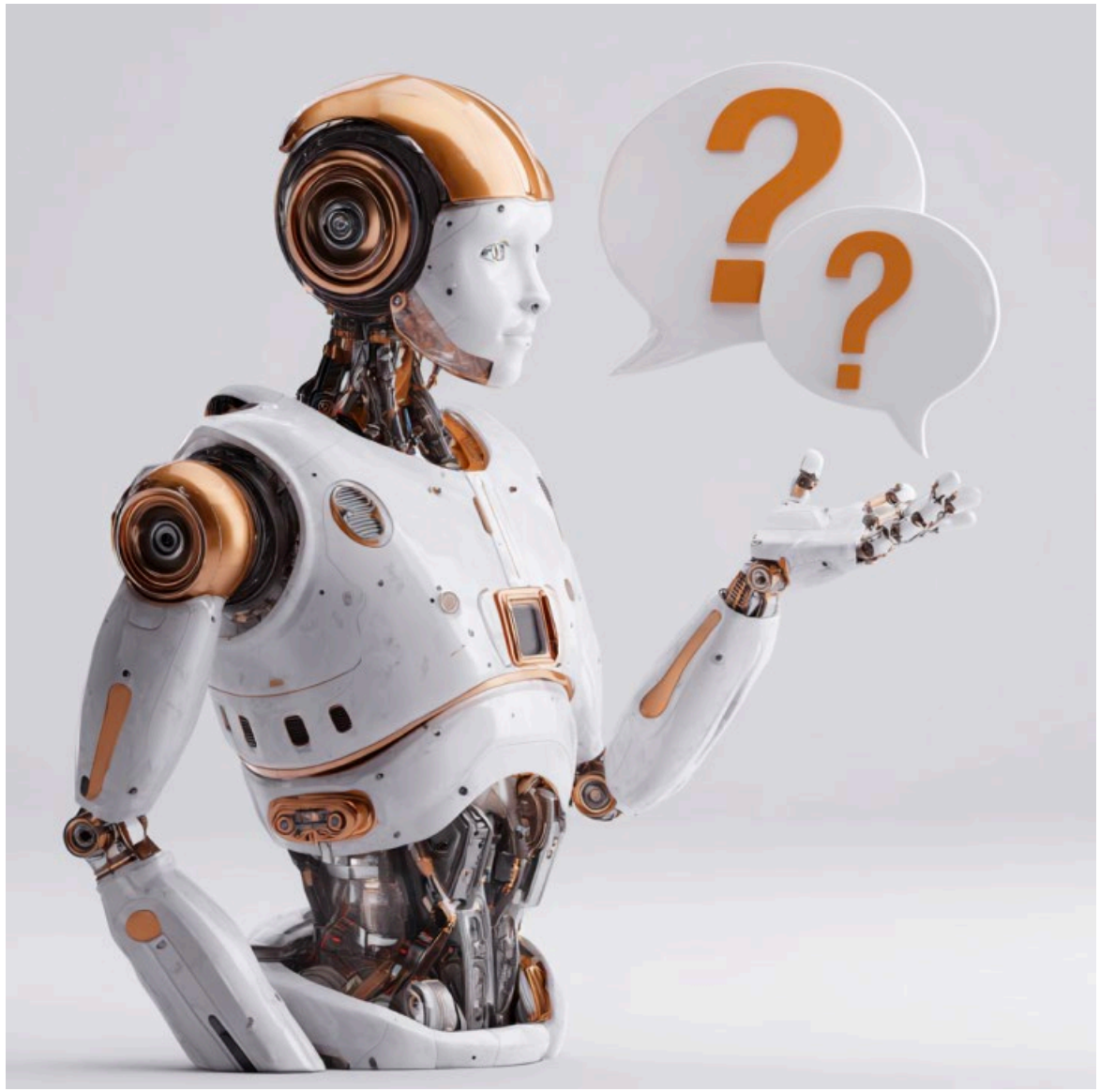
- AI seems to be already able to generate code that can be used in cybersecurity education, embedding correctly a range of vulnerabilities, with focus on SSD standards
- Towards a ***Reusable and Scalable Model for Integrating LLMs into Secure Software Development in Cybersecurity Education***
- ***Developing a Novel Framework*** that include human and AI generated elements for Cyber Security Education (AI generated code that follow a specific methodology in adherence to SSD standards)
- There are a number of misattribution examples primarily due to the reliance on superficial features and styling issues rather than SSD compliance

Future Work

- ***Expand Vulnerability Coverage*** to include a wider range of SSD security errors mapped to standards (OWASP Top 10 - CWE/SANS Top 25), thus giving students richer exposure to real-world security errors

- ***Gamification*** to increase engagement introducing features such as *scoring*, *timed challenges*, and *team competitions* to promote deeper understanding and long-term retention

- Explore if LLMs can serve not only as generators but also as **AI *Evaluators*,**



THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD

Thank you

Any Questions?