AI Assisted Coding

Week3 – Monday

Name: Pruthagwin Challuri

Batch: 07

Roll No.: 2303A51951

Lab Experiment: Ethics, Privacy & Transparency in AI-Assisted Coding

---

Task Description #1: Privacy in API Usage (Weather API)

1) Prompt to be given (Humanized – Zero Shot)

Generate a Python program that connects to a weather API and fetches weather data securely without exposing API keys directly in the code.

---

2) Code

(a) Original AI-Generated Code (Insecure)

```
import requests

API_KEY = "12345ABCDE"
city = "London" url =
f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_KEY}"

response = requests.get(url) print(response.json())
```

(b) Secure Version Using Environment Variables

```
import requests import os
```

```
API_KEY = os.getenv("WEATHER_API_KEY") city =
"London"

url =
f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_KEY}"
response = requests.get(url)

print(response.json())
```

---

3) Output

```
Weather Data Retrieved Successfully
Temperature: 18°C
Condition: Clear Sky
```

---

4) Remarks

- **Original code hardcodes API keys, which is a security risk.**
- **Secure version uses environment variables, protecting sensitive data.**
- **Demonstrates importance of privacy in API usage.**
- **Secure coding practices reduce risk of key leakage.**

Task Description #2: Privacy & Security in File Handling

1) Prompt to be given (Humanized)

Generate a Python program that stores user details such as name, email, and password in a file.

---

2) Code

```
(a) Original AI-Generated Code (Insecure) def
store_user(name, email, password):     with
open("users.txt", "a") as file:
        file.write(f"{name},{email},{password}\n") (b)
Revised Secure Code (Password Hashing) import
hashlib


def store_user(name, email, password):
    hashed_password = hashlib.sha256(password.encode()).hexdigest()     with
open("users.txt", "a") as file:
        file.write(f"{name},{email},{hashed_password}\n")
```

---

3) Output

User data stored successfully.

Password stored in encrypted (hashed) format.

---

4) Remarks

- **Original version stores passwords in plain text, a major privacy risk.**
- **Revised version uses SHA-256 hashing.**
- **Hashing ensures passwords are not directly readable.**
- **Improves compliance with basic security standards.**

Task Description #3: Transparency in Algorithm Design (Armstrong Number)

1) Prompt to be given (Humanized)

Generate a Python function to check whether a number is an Armstrong number and explain the code line by line using comments.

---

2) Code def is_armstrong(num):

```
   # Convert the number to string to extract digits     digits =
str(num)


   # Find the number of digits     power
= len(digits)


   # Initialize sum
total = 0


   # Calculate sum of digits raised to power     for
digit in digits:
       total += int(digit) ** power


   # Check if the calculated sum equals the original number     if
total == num:
       return "Armstrong Number"
else:
       return "Not an Armstrong Number"
```

---

3) Output

Input: 153 → Output: Armstrong Number

Input: 123 → Output: Not an Armstrong Number

---

4) Remarks

- Code is fully transparent with line-by-line comments.
- Explanation matches actual functionality.
- Easy for beginners to understand.
- Improves trust and clarity in AI-generated algorithms.

Task Description #4: Transparency in Algorithm Comparison

1) Prompt to be given (Humanized)

Generate Python code for QuickSort and BubbleSort.

Include comments explaining step-by-step how each algorithm works and

highlight their differences.

---

2) Code

```
# Bubble Sort Algorithm def
bubble_sort(arr):    n = len(arr)    for i in
range(n):        for j in range(0, n - i - 1):
if arr[j] > arr[j + 1]:              # Swap
elements              arr[j], arr[j + 1] = arr[j
+ 1], arr[j]    return arr



# Quick Sort Algorithm def
quick_sort(arr):    if len(arr) <= 1:
return arr    pivot = arr[0]    left = [x
for x in arr[1:] if x <= pivot]    right = [x
for x in arr[1:] if x > pivot]    return
quick_sort(left) + [pivot] +
quick_sort(right)
```

---

3) Output

Input: [5, 3, 8, 1]

Bubble Sort Output: [1, 3, 5, 8]

Quick Sort Output: [1, 3, 5, 8]

---

4) Remarks

- **Bubble Sort is simple but inefficient ($O(n^2)$).**
- **Quick Sort is faster and efficient (O(n log n)).**
- **Comments clearly explain internal logic.**
- **Transparency helps understand performance differences.**

Task Description #5: Transparency in AI Recommendations

1) Prompt to be given (Humanized)

Generate a Python-based product recommendation system that not only suggests products but also provides clear reasons for each recommendation.

---

2) Code def recommend_product(user_interest):

```
  products = {
     "laptop": "Recommended because you are interested in technology and productivity.",
     "headphones": "Recommended because you enjoy music and audio quality.",
     "smartwatch": "Recommended because you are interested in fitness tracking."
  }

  if user_interest in products:
     return products[user_interest]     else:
     return "No recommendation available for the given interest."
```

---

3) Output

Input: laptop

Output: Recommended because you are interested in technology and productivity.

4) Remarks

- **Recommendations include clear explanations.**
- **Enhances user trust and transparency.**
- **Logic is easy to understand and modify.**
- **Explainable AI improves decision acceptance.**

Overall Conclusion

- **AI-generated code must be reviewed for privacy risks.**
- **Transparency improves trust and understanding.**
- **Secure handling of data and explainable logic are essential.**
- **Ethical AI-assisted coding leads to safer and better software.**