**Dokumentation: Automatisierte Testfall-Generierung mit KI (Ergebnisse und nächste Schritte)**

**1. Ziel des Projekts**

Das Ziel dieses Projekts ist es, mithilfe von Künstlicher Intelligenz (KI) automatisch Testfälle für ECUs zu generieren. Dabei wird ein LLM (Large Language Model) wie Llama2:7b genutzt, um JSON-Daten in vollständige Testfallbeschreibungen umzuwandeln. Zusätzlich werden alle Komponenten in einer Docker-Umgebung ausgeführt, um eine wiederholbare und skalierbare Lösung zu gewährleisten.

**2. Vorgehensweise**

**2.1 Einrichtung der Arbeitsumgebung**

1. **Docker-Installation:** Docker wurde installiert, um die LLM-Modelle in einer isolierten Umgebung auszuführen. Die Installation wurde erfolgreich überprüft:
2. docker --version
3. # Ausgabe: Docker version 27.4.0, build bde2b89
4. **Start des Ollama-Containers:** Ein Docker-Container mit Ollama wurde gestartet, um das KI-Modell bereitzustellen:
5. docker run --name ollama-container -p 11434:11434 -d ollama/ollama
6. **Installation des Llama2:7b-Modells:** Das Llama2:7b-Modell wurde im Container installiert:
7. docker exec -it ollama-container ollama pull llama2:7b
8. **Prüfung des LLM-Servers:** Die Funktionalität des LLM-Servers wurde getestet:
9. curl -X POST http://localhost:11434/v1/completions -H "Content-Type: application/json" -d '{"model": "llama2:7b", "prompt": "Was ist 2 + 2?"}'
10. # Ausgabe: 2 + 2 = 4

**2.2 Entwicklung des Python-Skripts zur Testfall-Generierung**

Ein Python-Skript wurde erstellt, um JSON-Daten zu lesen, einen Prompt für das LLM zu generieren und eine Anfrage an den LLM-Server zu senden. Hier ist der aktuelle Python-Code:

import json

import requests

def generate\_test\_case(json\_file, url):

# JSON-Datei lesen

with open(json\_file, "r") as file:

data = json.load(file)

# Wichtige Informationen extrahieren

ecu = data["ECU"]

mechanism = data["Mechanism"]

seed = data["Details"]["Seed"]

key = data["Details"]["Key"]

# Prompt erstellen

prompt = f"""

Test Case Security Access $27

Test Item: Security Access

• Goal: To successfully test the UDS Service Secure Access $27 mechanism in the ECU ({ecu}).

Init Steps

Init Step 1: Prepare Tools, Documentation, and ECU for Security Access Testing

• Action: Gather the Holon Diagnostic Tool (or CANoe with DLL), the communication matrix, ECU documentation, UDS documentation, and the ECU's corresponding cryptographic key. Ensure the ECU is connected, communication is stable, and switch to a session which Security Access is enabled (e.g., Programming Session via UDS Service $10).

• Expected Result: All tools and documents are ready, the ECU is connected, communication is stable, and the diagnostic session is configured to permit Security Access.

Run Steps

Run Step 1: Verify Access Denied for Secured Functions Before Security Access

• Action: Attempt to execute a secured function (e.g., RequestDownload using UDS Service $34) to confirm it is inaccessible before completing the Security Access process.

• Expected Result: The secured function cannot be executed. The ECU responds with a negative response code (e.g., NRC 0x33 = securityAccessDenied).

Run Step 2: Request Seed (Challenge) from the ECU

• Action: Send a Seed request using UDS Service $27 with subfunction 0x01 to request a challenge.

• Expected Result: ECU responds with a positive response (SID 0x67, subfunction 0x01) and sends the Seed (Seed = Random Number = Challenge) back to the tool.

Run Step 3: Calculate the response Using the Seed

• Action: Use the Seed/challenge provided by the ECU and the cryptographic algorithm in the tool and the cryptographic key to calculate the response.

• Expected Result: A valid response is calculated based on the ECU’s challenge.

Run Step 4: Send the response to the ECU

• Action: Submit the calculated response to the ECU using UDS Service $27 with subfunction 0x02.

• Expected Result: ECU responds with a positive response (SID 0x67, subfunction 0x02), granting access to secured functions.

Run Step 5: Verify Access Granted for Secured Functions

• Action: Attempt to execute the same secured function (e.g., RequestDownload using UDS Service $34) after completing Security Access.

• Expected Result: The secured function is successfully executed, confirming that Security Access has been granted.

Shutdown Step 1: Logout from the Programming Session

• Action: Send a DiagnosticSessionControl (0x10) request with SubFunction Default Session (0x01) to switch the ECU from the current Programming Session to the Default Session.

• Expected Result: The ECU is logged out of the Programming Session.

"""

# Anfrage an den LLM-Server senden

response = requests.post(

url,

headers={"Content-Type": "application/json"},

json={"model": "llama2:7b", "prompt": prompt},

timeout=300 # Timeout in Sekunden

)

# Ausgabe des Ergebnisses

if response.status\_code == 200:

print("Erstellter Testfall:")

print(response.json()["choices"][0]["text"])

else:

print(f"Fehler: {response.status\_code}")

print(response.text)

# JSON-Datei und URL angeben

generate\_test\_case("example.json", "http://localhost:11434/v1/completions")

**3. Ergebnisse**

1. **Erfolgreiche Bereitstellung der Infrastruktur:**
   * Docker läuft stabil.
   * Llama2:7b wurde erfolgreich installiert und getestet.
2. **Python-Skript:**
   * Das Skript generiert basierend auf JSON-Daten Prompts und kommuniziert erfolgreich mit dem LLM-Server.
   * Timeout wurde implementiert, um Hänger zu vermeiden.
3. **Testfall-Generierung:**
   * Die generierten Testfälle entsprechen dem vorgegebenen Template und decken die notwendigen Schritte ab.

**4. Herausforderungen**

1. **Timeouts:** Das ursprüngliche Skript hatte keinen Timeout, wodurch es bei Problemen mit dem Server hängen blieb.
2. **Server-Konfiguration:** Die Ladezeit des Modells im Container war mitunter lang.
3. **Modell-Skalierung:** Sicherstellen, dass das Modell auch bei höherem Datenvolumen performant bleibt.

**5. Nächste Schritte**

1. **Fehlerbehebung und Optimierung:**
   * Regelmäßige Überprüfung der Docker-Container auf Stabilität.
   * Zusätzliche Tests mit verschiedenen JSON-Datensätzen durchführen.
2. **Automatisierung der Infrastruktur:**
   * Skripte erstellen, um die Container automatisch zu starten und zu überwachen.
3. **Validierung der generierten Testfälle:**
   * Sicherstellen, dass alle generierten Testfälle den Standards entsprechen und keine Lücken aufweisen.
4. **Skalierung der Lösung:**
   * Testen mit größeren Modellen und umfangreicheren JSON-Daten.

Ende der Dokumentation.