Fortgeschrittene Themen der IT-Sicherheit

LLMs zu bestehenden Softwareprojekten hinzufügen

vorgelegt von

Moritz Hausmann (Matrikelnummer: 84367)

Studiengang IT-Sicherheit
Semester 7

Hochschule Aalen

Hochschule für Technik und Wirtschaft

Betreut durch Prof. Dr. Roland Hellmann

12.07.2024

Erklärung

Ich versichere, dass ich die Ausarbeitung mit dem Thema „LLMs zu bestehenden Softwareprojekten hinzufügen“ selbstständig verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt habe. Die Stellen, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, sind in jedem einzelnen Fall unter Angabe der Quelle als Entlehnung (Zitat) kenntlich gemacht. Das Gleiche gilt für beigefügte Skizzen und Darstellungen.

Aalen, den 12. Juli 2024	Moritz Hausmann
Ort, Datum	Autor

Kurzfassung

In dieser Arbeit soll dem Leser das Werkzeug an die Hand gegeben werden, seine bestehenden Applikationen und Software-Projekte mit einem LLM (Large Language Model) funktional zu erweitern. Hierbei wird:

1. in Grundlagen eingeführt
2. ein Überblick über mögliche Anwendungsfälle gegeben
3. erklärt, welche Form von Künstlicher Intelligenz was kann und was nicht
4. am Beispiel eines bestehenden Projektes gezeigt, wie ein LLM hinzugefügt werden kann

Das Projekt, welches um ein LLM erweitert werden soll, ist das IoT (Internet of Things)-Honeypot-Framework **Sofah**[1]. An dieser Stelle soll das LLM zur Generierung von Daten für den Honeypot verwendet werden. Dieses wird im Verlauf nur beiläufig erwähnt, allerdings wird für die Übungen ein speziell für diesen Zweck entwickeltes Projekt verwendet, um die Komplexität zu reduzieren.

[1] https://github.com/sofahd

Inhaltsverzeichnis

Akronyme

API	Application Programming Interface
FQDN	Fully Qualified Domain Name
IoT	Internet of Things
KI	Künstliche Intelligenz
LLM	Large Language Model
ML	Machine Learning
NLP	Natural Language Processing
NN	Neuronale Neztze
RAG	Retrieval Augmented Generation

Abbildungsverzeichnis

1. Einleitung

Häufig steht man vor der Frage, ob und falls ja, wie man sein Softwareprojekt mit künstlicher Intelligenz erweitern kann. Die potenziell erreichbare Wertsteigerung und die damit verbundenen Möglichkeiten sind enorm. Doch muss auch besonders beachtet werden, für welche Anwendungsfälle, welche Form der KI geeignet ist und welche nicht.

1.1 Motivation

Künstliche Intelligenz ist in aller Munde und **das** Thema der Innovation der letzten Jahre. Doch will man ja nicht nur sein Projekt mit dem neuesten Hype versehen, sondern einen echten Mehrwert schaffen. Denn langfristig wird sich nur durchsetzen, was auch wirklich einen Nutzen bringt. Stupide Assoziierung mit KI wird hier nicht ausreichen.

1.2 Problemstellung und -abgrenzung

Die grundlegende Problemstellung ist: *Wie kann ich mein bestehendes Projekt mit einem LLM erweitern?*. Hierbei müssen die folgenden Teilprobleme gelöst werden:

- Welche Form von KI ist für meinen Anwendungsfall geeignet?
- Wie erweitere ich (auf einer technischen Ebene) mein bestehendes Projekt?
- Wie kann ich sicherstellen, dass mein LLM auch wirklich einen Mehrwert bringt?

Hierbei werden andere Formen von KI, wie z. B. bildgenerative Modelle oder Machine Learning, zwar kurz angeschnitten um auch hier mögliche Anwendungsfälle aufzuzeigen, jedoch liegt der Fokus dieser Arbeit auf LLMs, und deren Integration.

2. Grundlagen

Im Folgenden werden Grundlagen zu Künstlicher Intelligenz, insbesondere LLMs, gelegt. Außerdem werden Grundbegriffe zum Verständnis der verwendeten Techniken erläutert.

2.1 Künstliche Intelligenz

Grundsätzlich beschreibt der Begriff, dem Sinne folgend eine Technologie, welche es Computern erlaubt menschliche Intelligenz zu simulieren und so Probleme zu lösen.[1]

KI (Künstliche Intelligenz) als Feld der Informatik wird häufig in einem Atemzug mit Machine Learning und Deep Learning genannt. Dies ist insofern korrekt, als dass KI sehr wohl als Überbegriff auch über diese beiden Felder verstanden werden kann.[2]

Auf die Charakteristika von verschiedenen Formen von KI, wird in den folgenden Abschnitten eingegangen.

2.1.1 Machine Learning

ML (Machine Learning) ist eine Unterkategorie von KI, bei welcher Computer die Fähigkeit erlangen, aus Daten zu lernen. Hierbei wird der Computer befähigt zu *lernen* indem sich auf Basis von Daten Algorithmen anpassen. Dies ist besonders nützlich, wenn die Datenmenge zu groß ist, um von Menschen verarbeitet zu werden.[2]

Technische Funktion

Um ein ML-Modell zu trainieren, werden 3 Funktionen definiert:

- **Zielfunktion:** Funktion, welche das gewünschte Ergebnis beschreibt
- **Approximationsfunktion:** Funktion, welche im Rahmen das Trainings abgeändert wird, es soll sich hier an die Zielfunktion angenähert werden.
- **Kosten- oder Fehlerfunktion:** Diese Funktion misst die Abweichung zwischen der Approximationsfunktion und der Zielfunktion.

[3]

Man spricht hierbei auch von einem **Optimierungsproblem**, da die Kostenfunktion minimiert werden soll.

2.1.2 Deep Learning

Deep Learning kann als eine spezielle Form von ML verstanden werden, wobei hier eine komplexe Struktur von Algorithmen verwendet wird, welche nach dem Vorbild des menschlichen Gehirns aufgebaut sind. Diese Struktur bezeichnet man als NN (Neuronale Neztze).

Der größte Unterschied und auch damit ein Vorteil von Deep Learning gegenüber anderen Machine Learning-Methoden ist, dass hierbei die Klassifizierung der Parameter, nach denen optimiert wird, also *gelernt* wird, automatisch erfolgt. Dies ist besonders nützlich, wenn die Datenmenge zu groß ist, um von Menschen verarbeitet zu werden, oder die exakte Struktur der Daten nicht bekannt ist, respektive keine offensichtlich sinnvollen Parameter im Vorfeld definiert werden können.[2]

Technische Funktion

Auf einer technischen Ebene funktioniert Deep Learning grundsätzlich ähnlich wie Machine Learning. Allerdings gilt es hier zu unterscheiden, dass die bei Deep Learning verwendeten Algorithmen deutlich komplexer als bei anderen Machine Learning-Methoden sind.[2]

Deep Learning beschreibt hierbei Algorithmen, welche Daten mit einer logischen Struktur verarbeiten können, wobei Sie der Art ähneln, mit welcher Menschen Schlüsse ziehen.

Eine weitere Abgrenzung zum konventionellen Machine Learning ist, dass bei Deep Learning die Algorithmen in der Lage sind, selber festzulegen, welche Eigenschaften (Features) der Daten für die Klassifizierung relevant sind. Bei klassischerem Machine Learning müssen diese Features im Vorfeld händisch definiert werden. Diese Eigenschaft nennt man auch *automatic feature generation* (automatische Generierung von Eigenschaften). [2]

2.1.3 Neuronale Netze

Neuronale Netze sind Deep Learning bzw. Machine Learning Modelle, welche Entscheidungen auf eine dem menschlichen Gehirn nachgebildeten Art treffen.

Ein neuronales Netz besteht aus einer Eingabeschicht, einer oder mehreren versteckten Schichten und einer Ausgabeschicht. Jeder Knoten in einer Schicht ist mit jedem Knoten in der nächsten Schicht verbunden. Die Verbindungen haben Gewichte, welche während des Trainings angepasst werden, um die Genauigkeit des Modells zu verbessern.

Neuronale Netze lernen aus Trainingsdaten und verbessern ihre Genauigkeit mit der Zeit. Sie sind leistungsstark in der Datenklassifikation und -clustering und finden Anwendung in der Spracherkennung und Bildverarbeitung. Ein bekanntes Beispiel ist der Suchalgorithmus von Google.[4]

2.1.4 Large Language Models

LLMs sind umfangreiche Modelle, welche in der Lage sind, verschiedene Aufgaben der natürlichen Sprachverarbeitung zu bewältigen. Um dies zu erreichen, werden Sie durch Deep Learning trainiert. Dabei erlernen die Modelle statistische Beziehungen zwischen Sätzen und Wörtern sowie dem Kontext. Dies führt zu großen Wahrscheinlichkeitsnetzen, welche schlussendlich das sind, was ein LLM *ausmacht*.[5]

Technische Funktion

Ein LLM entsteht aus einem langwierigen Deep Learning-Prozess, bei welchem ein NN trainiert wird, um Texte zu generieren. Hierbei wird der NN mit einer großen Menge an Texten trainiert. Dieser Prozess kann mehrere Wochen dauern und ist sehr rechenintensiv.

Bei der Generation von Texten wird eine Eingabe ein sog. *Prompt* in das System übergeben und dann auf Basis dieses Prompts Text generiert. Vereinfacht gesagt funktioniert das LLM so, dass es auf Basis des Prompts eine Wahrscheinlichkeitsverteilung über die nächsten Wörter generiert und dann das wahrscheinlichste Wort auswählt. Dieser Prozess wird dann iterativ wiederholt, bis der gewünschte Text generiert wurde.

Tokenization

Genauer betrachtet wird zunächst die Prompt in Token zerlegt, vorverarbeitet und dann erzeugt das LLM mit jeder Iteration ein Token. Hierbei kann es sich um ein Wort, ein Teil eines Wortes oder sonstige Textbausteine wie Zahlen oder Satzzeichen handeln. Auf der rechten Seite ist ein Schaubild mit Beispiel zu sehen. Hierbei wird durch die verschiedenen Farben des Beispiels klar, wie Token aufgeteilt werden.

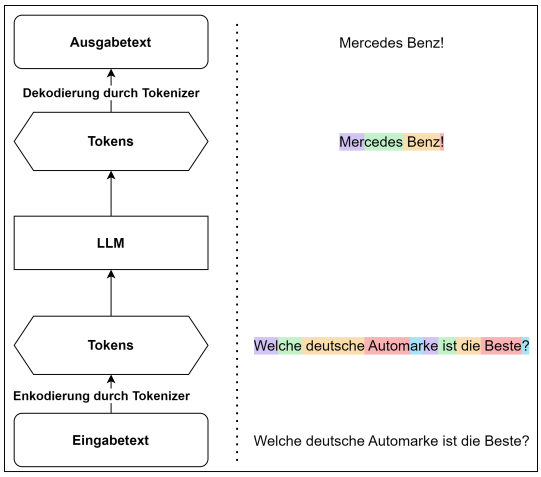

Abbildung 2.1: Schaubild zur Tokenization

2.1.5 Generative Pre-trained Transformer

Generative, vor-trainierte Transformatoren sind eine Art Large Language Models, welche als architektonische Grundlage den meisten modernen LLM dient.

Hierbei basiert das Modell auf der Transformer-Architektur.

Relevant ist, dass die Modelle mit einer großen ungelabelten Datenmenge vor-trainiert wurden, um eine Vielzahl von Sprachmustern zu erkennen und zu reproduzieren. Die erste Implementierung dieser Art stammt von OpenAI und wurde bereits im Jahre 2018 vorgestellt.[5]

Transformer

Transformer sind eine Form von Neuronale Netze, welche speziell für die Verarbeitung von Texten zum Erlernen von NLP (Natural Language Processing) entwickelt wurden. Hierbei war die Entwicklung der Transformer ein echter Durchbruch, da zuvor verwendete Modelle nicht den nötigen Kontext in der Größe bereitstellen konnten, um Texte zu generieren.

Zuvor funktionierte der von den Transformern implementierte Schritt auf Basis von En- und Decodern, welche Eingaben in kompakte mathematische Darstellungen umwandeln, und dann wieder zurück in Texte. Da dies sequenziell erfolgte, war dieser Schritt bei großen Texten sehr rechenintensiv. Die Transformer hingegen können auf Basis von Attention Mechanismen, also der Fokussierung auf bestimmte Teile des Textes, den Kontext besser erfassen. Man bezeichnet diesen Mechanismus als *self-attention*, bzw. *Selbstaufmerksamkeit*.

Weiter erlauben Transformatoren eine Multimodalität, also die Verarbeitung von mehreren Arten von Informationen, wie z. B. Text und Bildern. Dies ermöglicht erst die Generierung von Texten, welche Bilder beschreiben, oder auch die Generierung von Bildern, welche Texte beschreiben.[6]

2.1.6 Modalität

Modalität beschreibt die Art und Weise, wie Informationen in einem System repräsentiert werden. Hierbei wird unterschieden zwischen:

- **monomodal:** Eine Art von Informationen wird repräsentiert, z. B. Text

- **multimodal:** Mehrere Arten von Informationen, wie bei gpt-4. z. B. Text, welcher eingegebenes Bild beschreibt

2.1.7 Function Calling

Function Calling (zu deutsch: Funktionsaufruf) ist eine Technik, welche besonders im Kontext von LLMs verwendet wird. Hierbei wird das LLM ermächtigt, während der Generierung von Texten auch Funktionen aufzurufen, wie z. B. Python Code zu generieren und dann auszuführen oder auch API (Application Programming Interface)-Aufrufe zu tätigen. Dies ist besonders nützlich, wenn das LLM in der Lage sein soll, beispielsweise mathematische Probleme zu lösen oder auch auf externe Daten zuzugreifen. [7]

3. Einsatzmöglichkeiten von LLMs

Nachdem die Grundlagen gelegt wurden, soll nun aufgezeigt werden, was bei dem Einsatz von LLM in bestehenden Softwareprojekte zu beachten ist. Hierbei wird aufgezeigt, welche Anwendungsfälle sich besonders gut für LLMs eignen und in welchen Fällen andere Formen von KI besser geeignet sind.

3.1 Format

Um KI in ein bestehendes Projekt zu integrieren, muss zunächst definiert werden, in welchem Format die Daten vorliegen und welche Daten erzeugt werden sollen. Anhand dieser Information kann dann entschieden werden, welche Form von KI grundsätzlich geeignet ist.

3.1.1 Ausgabeformat

Zunächst muss definiert werden, in welchem Format die Daten erzeugt werden sollen, beziehungsweise noch genauer, welche Form der Daten generiert werden sollen.

- **Text:**
  - **Native Ausgabe:** Hierbei wird der Text des LLMs direkt verwendet. Hierzu kann ein LLM verwendet werden.
  - **Indirekte Ausgabe:** Hierbei wird der Text, also die Ausgabe des LLMs, als Zwischenschritt verwendet. Beispiel: Mermaid-Diagramm-Code generieren und Nutzer nur das Diagramm anzeigen. Hier ist bereits Vorsicht geboten, da kleinere Fehler oder Abweichungen im generierten Code zu großen Fehlern im Endprodukt führen können.
- **Bild:** Da ein LLM nicht direkt Bilder generieren kann, ist es hier nicht direkt geeignet, was allerdings häufiger passiert ist, dass ein LLM Text generiert, welcher dann als Prompt an eine Bildgenerierende KI weitergegeben wird.
- **Weitere Medienformen:** Wenn es Medienformen gibt, welche sich nicht mit einer der obigen Kategorien beschreiben lassen, so ist es unwahrscheinlich, dass ein LLM hierfür geeignet ist.

3.1.2 Eingabeformat

Neben der in Ausgabeformat genannten Ausgabe muss auch die Eingabe definiert werden. Hierbei ist zu beachten, dass die meisten LLMs, welche aktuell quelloffen verfügbar sind, nicht multimodal sind.

3.2 LLM Schwächen und Stärken

Die oben eingeführten Modelle eignen sich besonders zur Generierung von Texten, neben der Erzeugung kann auch eine Zusammenfassung von Texten gut funktionieren. Zu beachten gilt hierbei immer, dass LLMs auch Halluzinationen erzeugen können, da das LLM kein echtes *Verständnis* hat, wozu es eigentlich schreibt. Das heißt, dass bei der Erzeugung eben lediglich Wahrscheinlichkeiten berechnet und so immer das nächst wahrscheinliche Wort generiert wird. Hierbei kann es dann zu inhaltlichen Fehlern kommen, welche besonders tückisch sind, da sie in guter Grammatik und Syntax verpackt und teilweise auch von wahren Fakten umschlossen sind. Daher geben sich viele verantwortliche Nutzer selbst die Regel, die KI-generierten Texte nicht zu verwenden, wenn man nicht selbst die Expertise hat, um den Inhalt zu überprüfen.

Mit dem Wissen um die Tatsache, dass LLMs nur Wahrscheinlichkeiten berechnen und keine echte Intelligenz besitzen, kann auch direkt auf eine weitere Schwäche geschlossen werden. Denn LLMs können, auch wenn dies viele Nutzer von Ihnen erwarten, überhaupt nicht rechnen. Das heißt, dass sie nicht in der Lage sind, mathematische Probleme zu lösen oder auch nur einfache Rechnungen durchzuführen. Denn auch hierbei handelt es sich um ein Problem, welches nicht durch Wahrscheinlichkeiten gelöst werden kann, sondern durch exakte Berechnungen. Die großen kommerziellen Anbieter wie OpenAI lösen dieses Problem, indem sie via Function Calling dem LLM die Möglichkeit geben, selbst geschriebenen Python Code auszuführen. Dieser Code kann dann für die Berechnung von mathematischen Problemen genutzt werden.

Eine nicht von der Hand zu weisende Schwäche ist die Ressourcenintensität, welche das Betreiben eines LLMs mit sich bringt. Hierbei ist zu beachten, dass ein LLM mit 8 Milliarden Parametern wie **llama3** schon eine sehr bereits 7 GB Arbeitsspeicher benötigt, um geladen werden zu können. Es gibt zwar deutlich kleinere und weniger leistungsfordernde Modelle, diese haben jedoch häufig auch eine deutlich geringere Leistung in der Generierung von Texten.

Beispielhaft wurde hier mit dem Modell **qwen2:0.5b** mit nur einer halben Milliarde Parametern ein sehr kleines Modell getestet.

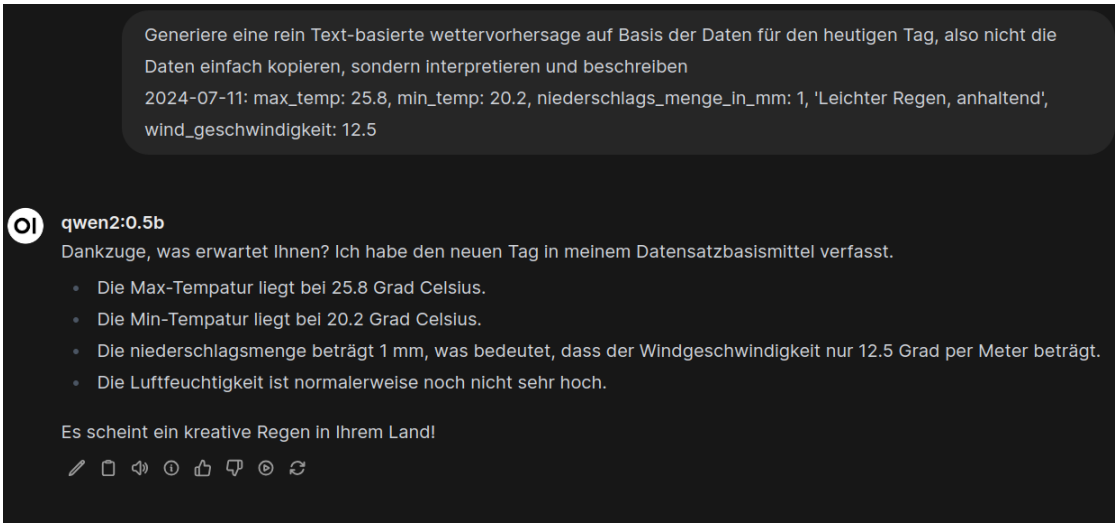

Abbildung 3.1: Fehler von qwen2:0.5b

An dieser Stelle wird deutlich, dass sich zumindest dieses Modell nicht für die Generierung von Texten eignet, wie es hier versucht wurde. Das Problem kann auch daraus erwachsen sein, dass das Modell von dem chinesischen Unternehmen *Alibaba Cloud*, Tochtergesellschaft der *Alibaba Group* entwickelt wurde und daher die Fehler in deutscher Sprache und Grammatik naheliegender sind. Anzumerken gilt hierbei dennoch, dass laut den Angaben der Entwickler das Modell auch für die deutsche Sprache geeignet sein sollte.[8]

4. Integration in das bestehende Honeypot-Framework Sofah

Im Folgenden soll gezeigt werden, wie ein LLM in das Honeypot-Framework **Sofah** integriert werden kann.

4.1 Sofah

Sofah (**S**peedy **O**pen **F**ramework for **A**utomated **H**oneypot-development) ist ein Framework, welches universell für IoT-Geräte eingesetzt werden können soll. Hierbei ist der Anwendungsfall, dass mit einer IP-Adresse und einer Liste an Endpunkten voll automatisiert ein Honeypot generiert wird, welcher ununterscheidbar von dem originalen Gerät ist.[9]

4.2 Anwendungsfälle

Im Folgenden soll auf mögliche Anwendungsfälle von LLM eingegangen werden.

4.2.1 Generierung von Daten

Ein Anwendungsfall, welcher besonders für Honeypots auf der Hand liegt, ist die Generierung von Daten. Hierbei kann ein LLM genutzt werden, um z. B. Log-Dateien zu generieren, welche dann von Angreifern ausgelesen werden können. Dies ist besonders nützlich, da so auch ungewöhnliche Angriffe, welche nicht in der Datenbank des Honeypots hinterlegt sind, erkannt werden können.

Hier kann man tatsächlichen Mehrwert schaffen, ohne einen Anwendungsfall zu betrachten, welcher unrealistisch ist oder mit anderen Formen von KI besser gelöst werden könnte.

4.2.2 Generierung von Antworten

Ein weiterer Anwendungsfall, welcher sich anbietet, ist die Generierung von Antworten auf Anfragen. Gerade bei unerwarteten Anfragen, auf welche der Honeypot zuvor nur mit einer statischen generellen Fehlerseite antworten konnte, könnte so deutlich verbessert werden.

Allerdings ist dies aus mehreren Gründen schwieriger in der Umsetzung.

- **Ressourcen/Rechenzeit:** die Generierung kann erst nach Eingehen der Anfrage des Angreifers starten, da sie auf dessen Anfrage basiert. Steht nun kein sehr schnelles, sehr leistungsfähiges LLM zur Verfügung, kann es passieren, dass die Antwort zu lange dauert und der Angreifer Verdacht schöpft.

- **Fehleranfälligkeit:** Da zu erwarten ist, dass der Angreifer die Antwort des Honeypots automatisiert auswertet und eine sehr spezifische Antwort erwartet, um fortzufahren, kann es hier sehr gut passieren, dass die Antwort zwar ähnlich dem ist, was der Angreifer erwartet, aber einer expliziten Prüfung nicht standhält.

Diese beiden Probleme existieren bei der Generierung von Daten nicht in der Form, da zum einen die Generierung der Daten im Vorfeld geschehen kann und zum anderen die Daten nicht so spezifisch sein müssen wie die Antworten.

4.2.3 Interpretation von Anfragen/Log-Daten

Weiter wäre es natürlich vorstellbar, mit einem LLM die Anfragen von Angreifern zu interpretieren und so auch ungewöhnliche Anfragen zumindest in der Aufbereitung der Daten zu erkennen. Hierbei könnte das LLM z. B. die Anfragen in Kategorien einteilen und so auch ungewöhnliche Anfragen erkennen.

Allerdings ist dies auch ein eher schwieriger Anwendungsfall, welcher vermutlich durch klassisches Machine Learning besser gelöst werden könnte.

4.3 Integration

Die Integration eines LLMs in ein bestehendes Projekt ist grundsätzlich nicht schwer, allerdings gibt es einige Dinge zu beachten.

4.3.1 Technische Voraussetzungen

Zunächst muss sichergestellt werden, dass ein LLM zur Verfügung steht und der Honeypot darauf zugreifen kann.

Es gilt sich die Frage zu stellen, ob das LLM selbst gehostet werden soll oder ob ein externer Dienstleister wie OpenAI genutzt werden soll. Hierbei ist zu beachten, dass externe Dienstleister in der Regel kostenpflichtig sind und auch die Daten, welche an den Dienstleister gesendet werden, nicht immer sicher sind. Auf der anderen Seite sind lokale Modelle auch nicht perfekt und benötigen eine große Menge an Rechenleistung. Daher ist hier eine wohlüberlegte Abwägung notwendig.

Falls man sich zum selbst hosten entscheidet, kann das Software-Projekt `ollama`, welches auch in der Übung zum Einsatz kommt, empfohlen werden. Hierbei handelt es sich um ein Open-Source-Projekt, welches es ermöglicht, ein LLM lokal zu hosten, und dann über eine API anzusprechen. Ein angeschlossenes Repository, welches die meisten Open-Source LLM enthält, macht die Verwendung besonders nutzerfreundlich, da mit einem Befehl neue Modelle heruntergeladen und installiert werden können.

4.3.2 Sicherheit

Ein weiterer wichtiger Punkt ist die Sicherheit. Hierbei ist zu beachten, dass das LLM nicht dazu genutzt werden sollte, um sensible Daten zu generieren oder zu verarbeiten. Besonders bei der Verwendung von externen Dienstleistern ist hier Vorsicht geboten, da die Daten, welche an den Dienstleister gesendet werden, je nach AGB des Dienstleisters weiterverarbeitet werden oder zum Training genutzt werden können.

4.3.3 Implementierung

Da Sofah in Python implementiert ist, fällt die Integration besonders leicht, da man vorgefertigte `pip`-Module nutzen kann, um mit dem LLM zu interagieren. Für OpenAI[1] gibt es ein Modul, genauso wie für das `ollama`-Projekt[2].

Allerdings sollten auch andere Programmiersprachen keine Schwierigkeiten bereiten, da die meisten LLMs über eine API angesprochen werden können und so auch in anderen Sprachen genutzt werden können.

Für eine beispielhafte Implementierung im größeren Detail soll die Übung 5 betrachtet werden.

[1] https://platform.openai.com/docs/libraries
[2] https://pypi.org/project/ollama/

5. Übung

Im Folgenden werden Sie

1. in die Lage versetzt, unterscheiden zu können, ob und falls ja welche Form von KI für Ihren Anwendungsfall geeignet ist
2. ihr erstes Open-Source LLM zu starten, und selber die ersten Texte generieren
3. in ein bestehendes Softwareprojekt ein LLM integrieren.

5.1 Aufgabe 0: Technische Vorbereitung

Um die folgenden Aufgaben bearbeiten zu können, müssen Sie zunächst einige Grundlagen schaffen. Zunächst sei vorausgeschickt, dass diese Aufgaben auf den meisten Linux-Systemen problemlos laufen sollten, da alle Ausführungen mit Docker containerisiert vonstattengehen. Allerdings wurden die Aufgaben lediglich auf einem aktuellen Kali-Linux getestet.

Beachten Sie, dass Ihr System eine größere Menge an Arbeitsspeicher benötigen wird, um größere LLMs laden und bereitstellen zu können. Hierbei sollten Sie als absolutes Minimum 4 GB Arbeitsspeicher zur Verfügung haben. Um auf der sicheren Seite zu sein, sollten Sie allerdings eher 8-16 GB Arbeitsspeicher zur Verfügung stellen.

5.1.1 Installation von Docker

Um die folgenden Aufgaben bearbeiten zu können, benötigen Sie sowohl Docker als auch Docker Compose. **Achtung:** Beachten Sie, dass Sie `docker compose` und nicht `docker-compose` installieren müssen. Da die ältere Version (mit Bindestrich) nicht mehr unterstützt wird.

Überprüfen Sie zunächst, ob Sie bereits `docker` und/oder `docker compose` installiert haben, indem Sie folgenden Befehle ausführen:

Listing 5.1: Überprüfen der Installation von Docker

```
1 docker --version
```

Listing 5.2: Überprüfen der Installation von Docker Compose

```
1 docker compose version
```

In beiden Fällen sollten Informationen zur Version ausgegeben werden.

Falls bereits bei 5.1 eine Fehlermeldung ausgegeben wird, können Sie Docker mit

Listing 5.3: Installation von Docker

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
    docker-buildx-plugin
```

installieren. Falls `apt` hier nicht alle benötigen Pakete findet, wenden Sie sich an die offizielle Dokumentation von Docker unter https://docs.docker.com/engine/install/debian/#install-using-the-repository, **Für Kali Linux:** https://www.kali.org/docs/containers/installing-docker-on-kali/#installing-docker-ce-on-kali-linux.

Falls 5.2 fehlschlägt, können Sie Docker Compose mit

Listing 5.4: Installation von Docker Compose

```
sudo apt-get update
sudo apt-get install docker-compose-plugin
```

installieren. Beachten Sie hierbei, dass Sie `docker-compose-plugin` und nicht `docker-compose` installieren müssen. Auch hier gilt, dass falls `apt` die benötigten Pakete nicht findet, Sie sich an die offizielle Dokumentation von Docker unter https://docs.docker.com/engine/install/debian/#install-using-the-repository wenden sollten (**Bei Kali Linux:** https://www.kali.org/docs/containers/installing-docker-on-kali/#installing-docker-

Beachten Sie, dass alle hier angegebenen Links für die Installation von Docker auf Debian basierten Systemen (bzw. Kali-Linux) sind. Falls Sie ein anderes System verwenden, sollten Sie sich an die entsprechende offizielle Dokumentation von Docker wenden.

5.1.2 Herunterladen der benötigten Dateien

Falls nicht bereits geschehen, sollten Sie nun die benötigten Dateien herunterladen. Hierzu können Sie das Repository unter https://github.com/mrtzhsmnn/LLM_Exercise klonen.

Nutzen Sie hierzu am einfachsten den Befehl

Listing 5.5: Klonen des Repositories

```
git clone https://github.com/mrtzhsmnn/LLM_Exercise.git
```

in einem Ordner, in welchem Sie den Ordner mit den Übungsunterlagen speichern möchten.

5.1.3 Starten der Container

Nachdem Sie die Dateien heruntergeladen haben, können Sie den Container starten. Hierzu wechseln Sie in den Ordner, in welchem Sie das Repository geklont haben, und führen den Befehl

Listing 5.6: Starten des Containers

```
1 sudo docker compose up -d
```

aus. Hierbei wird der Container im Hintergrund gestartet, was einige Zeit (etwa 15 Minuten) in Anspruch nehmen wird, daher ist es ratsam sich in der Zwischenzeit um Aufgabe 1: Theorie zu kümmern.

5.2 Aufgabe 1: Theorie

Um zu verstehen, welche Anwendungsfälle sich besonders gut für LLMs eignen, und in welchen Fällen andere Formen von KI besser geeignet sind, sollten Sie sich zunächst mit den Grundlagen von LLMs vertraut machen. Hierzu können Sie die Grundlagen lesen.

5.2.1 Aufgabe 1.1: Anwendungsfälle

Im Folgenden erhalten Sie eine Liste an exemplarischen Anwendungsfällen, bei welchen Sie dann, basierend auf den Informationen aus Grundlagen entscheiden sollen, ob ein LLM für diesen Anwendungsfall geeignet ist, oder ob eine andere Form von KI besser geeignet wäre. Bitte begründen Sie Ihre Entscheidung.

Nennen Sie für jeden Fall das Ein- und Ausgabeformat, bevor Sie begründen, welche Form von KI für diesen Anwendungsfall am geeignetsten ist.

- Sie haben eine Wetter App und wollen auf Basis der Wetterdaten für die nächste Stunde einen Text generieren.
- Um neue Nutzer für Ihre Wetter-App zu gewinnen wollen Sie täglich für jede größere Stadt ein Bild generieren, welches das Wetter des kommenden Tages versinnbildlicht. Diese Bilder wollen Sie dann auf Social Media verbreiten.
- Sie wollen mit einer KI-Lösung gleichbleibende Mathematische Probleme lösen.
- Für einen Dienst, welcher Malware-Reports darstellt, wollen Sie KI einsetzen um Reports für den Leser zusammenzufassen.
- Sie wollen eine KI einsetzen, um automatisch zwischen harmlosen und besorgniserregenden Ereignissen in einer Log-Datei zu unterscheiden, um besorgniserregende Ereignisse direkt an einen Analysten weiterzuleiten.

5.2.2 Aufgabe 1.2: Schwächen und Stärken

Nennen Sie jeweils zwei Schwächen und zwei Stärken von LLMs. Nennen Sie bei einer der Schwächen auch einen Ansatz, mit welchem diese Schwäche mitigiert werden kann.

5.3 Aufgabe 2: Erste Schritte mit einem LLM

Im Optimalfall sollten alle Container in der Zwischenzeit gestartet sein. Um dies zu überprüfen, können Sie sich mit dem Befehl

Listing 5.7: Überprüfen der Container

```
1 sudo docker ps -a
```

alle Container anzeigen lassen. Hierbei sollten Sie die drei laufenden Container `open-webui`, `ollama`, `weather` sehen.

Häufige Fehler

Falls Sie hierbei eine Fehlermeldung erhalten, kann dies daran liegen, dass die Ports welche von den im `docker-compose.yml` definierten Containern verwendet werden sollten, bereits von anderen Prozessen in Verwendung sind. Besonders im Falle des Port `80`, welcher von der open-Webui verwendet wird, kann dies häufig vorkommen. Hier können Sie dem folgenden Vorgehen nachgehen.

Zunächst mit `sudo lsof -i :80` ermitteln, welche Prozesse den Port `80` verwenden. Hierbei wird auch die `pid` des Prozesses angezeigt, sowie der Name.

Ist der Name bekannt, kann es sich lohnen zu recherchieren, wie man den entsprechenden Service hinter dem Prozess stoppen kann. Andernfalls kann man auch einfach den Prozess mit `kill -9 <pid>` beenden. Hierbei ist zu beachten, dass dies den Prozess sofort beendet, und nicht auf eine saubere Art und Weise. Daher sollte dies nur gemacht werden, wenn der Prozess sicher nicht mehr benötigt wird.

5.3.1 Aufgabe 2.1: Open-Webui

Öffnen Sie die Web-UI, indem Sie in Ihrem Browser `localhost` eingeben. Hierbei sollten Sie sich dann bei einer graphischen Benutzeroberfläche wiederfinden, welche Ihnen die Möglichkeit gibt, mit verschiedenen LLMs zu interagieren.

Exemplarisch sind Ihnen hier drei Modelle bereitgestellt, welche sich deutlich in der Größe und in der Resourcenintensität unterscheiden.

1. **qwen2:0.5b** Ein sehr kleines Modell, mit lediglich einer halben Milliarde Parameter, sehr wenig ressourcenintensiv, dafür allerdings nicht sehr leistungsstark.
2. **moondream** Ein ebenfalls kleines Modell mit 1.4 Milliarden Parametern, welches allerdings schon deutlich mehr Leistung bietet.
3. **llama3** Ein deutlich größeres Modell mit 8 Milliarden Parametern, welches allerdings auch deutlich mehr Ressourcen benötigt, und Ihre Hardware vermutlich schon an die Grenzen bringen wird, sollten Sie versuchen dieses Modell nicht mit einer Grafikkarte zu betreiben.

Die Unterschiede in der Ressourcenintensität werden Sie besonders in der Dauer bemerken, welche die Modelle benötigen, um eine Antwort zu generieren. Sie werden mit Sicherheit auch in der Lage sein, die Unterschiede in der Qualität der generierten Texte zu bemerken.

Interagieren Sie mit den Modellen, indem Sie ihnen verschiedene Prompts geben. Hierbei sollten Sie versuchen, die Unterschiede in der Qualität der generierten Texte zu bemerken.

5.4 Aufgabe 3: Wetter-App

In dieser Aufgabe werden Sie eine bestehende Python Web-App um ein LLM erweitern. Hierbei soll das LLM genutzt werden, um auf Basis der Wetterdaten für die nächste Stunde einen Text zu generieren.

5.4.1 Aufgabe 3.1: Funktionstest und Vertrautmachen mit dem Code

Sofern die Container wie in Aufgabe 0: Technische Vorbereitung beschrieben gestartet sind, können Sie unter `http://localhost:8080` die Wetter-App aufrufen. Hierbei sollten Sie eine einfache Web-App sehen, bei welcher Sie Land und Stadt eingeben können, und dann die Wetterdaten abrufen können. Diese Funktionalität ist bereits vollständig implementiert, wobei die API-Endpunkte in der Datei `app.py` definiert sind, welche Sie im Ordner `weather/src` finden. Im selben Ordner finden Sie dann auch die Datei `weather.py`, welche die eigentliche Logik der App enthält.

Machen Sie sich mit dem Code der Web-App vertraut, bis Sie die folgenden Fragen beantworten können.

- Welche Funktion werden in der Datei `app.py` definiert?
- Welche Funktion werden in der Datei `weather.py` definiert?
- In welcher Methode müssen Sie den Code ergänzen, um das LLM zu integrieren?
- Welche Parameter erwartet die Web-App, bei Abfrage des Wetters?

- Wie können Sie intern auf die API von `ollama` zugreifen?

5.4.2 Aufgabe 3.2: Integration des LLM

Um nun zu verstehen, wie Sie das LLM in die Web-App integrieren können, sollten Sie zunächst nachvollziehen, wie Sie mit dem LLM als API interagieren können. Hierzu lesen Sie bitte die Dokumentation von https://github.com/ollama/ollama/blob/main/docs/api.md durch. (**Hinweis**: besonders die *Generate a Completion* Sektion ist für Sie relevant).

Dann müssen Sie noch das Prompt-Engineering verstehen. Da Sie an dieser Stelle leider sehr von der Geschwindigkeit des LLM eingeschränkt sind, müssen Sie beachten, dass die Prompt nicht zu groß sein darf, da die Verarbeitungsdauer des LLMs mit der Länge des Prompts steigt.

Weiter müssen Sie sich auf Basis ihrer vorausgegangenen Experimente mit den verschiedenen Modellen entscheiden, welches Modell Sie für die Wetter-App verwenden wollen.

5.4.3 Aufgabe 3.2.1: Implementierung

Implementieren Sie nun die Integration des LLM in die Web-App. Schlussendlich soll von der Methode `_get_ai_text` in der Datei `weather.py` ein Text generiert werden, welcher das Wetter des Kommenden Tages beschreibt. Die Methode ist bereits mit Header, und Rückgabetypen definiert, Sie müssen lediglich den Code ergänzen, um das LLM zu integrieren.

Hierbei können Sie das `requests` pip-Modul verwenden. Beachten Sie, dass Sie dringend den `timeout`-Wert des `requests.post`-Aufrufs setzen sollten, da das LLM sehr lange brauchen kann, um eine Antwort zu generieren, und der Standardwert wesentlich zu kurz ist.

Hinweise

- Der Container `ollama` exponiert keine Ports, daher können Sie nicht über `localhost`, oder `127.0.0.1` auf die API zugreifen. Allerdings ist der `weather` container im selben Netzwerk wie der `ollama` Container.
- Es kann hilfreich sein, auf die Logs des `weather` Containers zuzugreifen, um zu sehen, ob der `requests.post`-Aufruf erfolgreich war. Hierbei können Sie den Befehl `sudo docker logs weather` verwenden.
- Beachten Sie, dass der `requests.post`-Aufruf eine `json`-Datei erwartet, und auch eine solche zurückgibt. Hierbei können Sie die `json`-Bibliothek von Python verwenden, um die Daten in das richtige Format zu bringen.

- Stellen Sie sich die Frage, ob sie einen Header Wert für den Post-Request setzen sollten, und wenn ja, welchen Wert dieser haben sollte.
- Sie erhalten mit der Methode das gesamte dictionary mit allen Wetterdaten. Beachten Sie, dass Zeitstempel in diesem Dict mit deutscher Zeitzone und nicht mit UTC-Zeitzone angegeben sind.

Wenn Sie den Code von `weather.py` erfolgreich ergänzt haben, können Sie die Web-App neu starten, indem Sie den Container `weather` neu starten. Hierzu können Sie den Befehl

Listing 5.8: Neustarten des Containers

```
1 sudo docker compose weather up --force-recreate --remove-
    orphans --build -d
```

verwenden. (Hierzu müssen Sie im Ordner sein, in welchem Sie auch das `docker-compose.yml` gespeichert haben.)

5.4.4 Aufgabe 3.3: Testen

Testen Sie schlussendlich Ihre Implementierung, indem sie die Web-App aufrufen und die Wetterdaten für eine Stadt abfragen. Hierbei sollte dann ein Text generiert werden, welcher das Wetter des kommenden Tages beschreibt. Lassen Sie sich hierbei nicht beunruhigen, wenn die Generation des Textes sehr lange dauert, dies ist im Anbetracht der begrenzten Ressourcen des LLM normal.

6. Lösungen

Im Folgenden wird auf die Lösungen zu den Aufgaben eingegangen.

6.1 Lösung 1: Theorie

Grundlagen gibt dem Leser die nötigen Informationen, um die Aufgaben zu lösen.

6.1.1 Lösung 1.1: Anwendungsfälle

- Sie haben eine Wetter App und wollen auf Basis der Wetterdaten für die nächste Stunde einen Text generieren.
  - **Eingabeformat:** Text als Prompt, mit Wetterdaten
  - **Ausgabeformat:** Text
  - **Geeignete Form von KI:** hier lässt sich hervorragend ein LLM einsetzen, um auf Basis der Wetterdaten einen Text zu generieren.
- Um neue Nutzer für Ihre Wetter-App zu gewinnen wollen Sie täglich für jede größere Stadt ein Bild generieren, welches das Wetter des kommenden Tages versinnbildlicht. Diese Bilder wollen Sie dann auf Social Media verbreiten.
  - **Eingabeformat:** Text, mit den Beschreibungen des Wetters des kommenden Tages
  - **Ausgabeformat:** Bild, welches o. g. Beschreibung visualisiert
  - **Geeignete Form von KI:** An dieser Stelle lässt sich eine bildgenerierende KI, wie beispielsweise `dall-e-2`[10] einsetzen. Zu beachten ist, hierbei dass eventuell noch ein LLM benötigt wird, um von den Wetterdaten auf einen Text zu kommen, welcher sich besser als Prompt für die Bildgenerierung eignet.
- Sie wollen mit einer KI-Lösung gleichbleibende Mathematische Probleme lösen.
  - **Eingabeformat:** Zahlen
  - **Ausgabeformat:** Zahlen
  - **Geeignete Form von KI:** Hier eignet sich keine KI so richtig, die meisten LLM sind nicht in der Lage zu rechnen. Da allerdings die Probleme gleichbleibend sind, könnte man hier simpel eine Funktion schreiben, welche die Probleme löst. Falls die Mathematischen Probleme allerdings variieren, könnte man vermutlich mit Machine Learning eine Lösung finden.
- Für einen Dienst, welcher Malware-Reports darstellt, wollen Sie KI einsetzen um Reports für den Leser zusammenzufassen.

  - **Eingabeformat:** Text (Malware-Report)
  - **Ausgabeformat:** Text (Zusammenfassung)
  - **Geeignete Form von KI:** Hier eignet sich ebenfalls ein LLM sehr, um auf Basis des Malware-Reports eine Zusammenfassung zu generieren. Eventuell ist es noch ratsam RAG (Retrieval Augmented Generation) zu verwenden, um die Chance auf Halluzinationen zu verringern.
- Sie wollen eine KI einsetzen, um automatisch zwischen harmlosen und besorgniserregenden Ereignissen in einer Log-Datei zu unterscheiden, um besorgniserregende Ereignisse direkt an einen Analysten weiterzuleiten.
  - **Eingabeformat:** Text (Log-Nachricht)
  - **Ausgabeformat:** Score (harmlos oder besorgniserregend, bzw. Wahrscheinlichkeit)
  - **Geeignete Form von KI:** Hier ist klassisches Machine Learning die am besten geeignete Form von KI, da es sich um ein klassisches Klassifizierungsproblem handelt.

6.1.2 Lösung 1.2: Schwächen und Stärken

- **Stärken:**
  - **Generierung von Texten:** LLMs sind besonders gut darin, Texte zu generieren, und können dies in vielen Fällen besser als Menschen.
  - **Zusammenfassung:** LLMs können auch sehr gut Texte zusammenfassen, und so lange Texte auf wenige Sätze reduzieren.
- **Schwächen:**
  - **Rechnen:** LLMs sind nicht in der Lage zu rechnen, und können daher keine mathematischen Probleme lösen.
  - **Halluzinationen:** LLMs können Halluzinationen erzeugen, und so falsche Informationen generieren. Dies ist besonders gefährlich, da die generierten Texte oft sehr überzeugend sind.

Für das Problem der Halluzinationen, kann man mit einem RAG die Wahrscheinlichkeit reduzieren, dass Halluzinationen auftreten.

Um das Problem der Rechenunfähigkeit zu lösen, kann man via Function Calling dem LLM beispielsweise ermöglichen, Python Code auszuführen, und so mathematische Probleme zu lösen.

6.2 Lösung 2: Erste Schritte mit einem LLM

Im Folgenden wird die Lösung zu den Aufgaben in Aufgabe 2: Erste Schritte mit einem LLM dargestellt.

6.2.1 Lösung 2.1: Open-Webui

Die Web-UI ist ein einfaches Frontend, welches es dem Nutzer ermöglicht, komfortabel mit verschiedenen, (in diesem Fall) von `ollama` bereitgestellten LLMs zu interagieren.

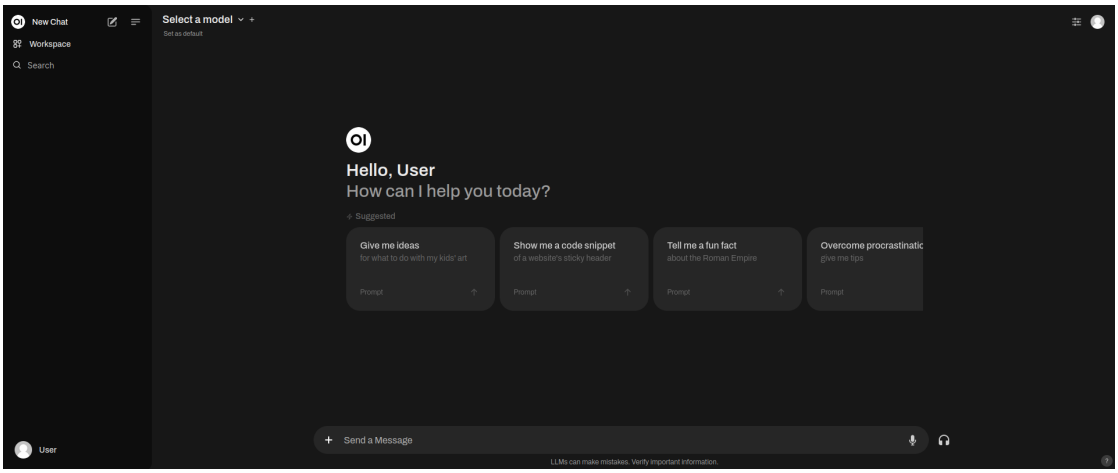

Abbildung 6.1: Open-Webui

Auf 6.1 ist die Web-UI zu sehen. Hierbei können links oben mit ”select a model”das entsprechende Sprachmodell ausgewählt werden, mit welchem man interagieren möchte. Beim Ausprobieren der verschiedenen Modelle, sollte dem Bearbeiter auffallen, dass die Qualität der generierten Texte mit der Größe des Modells (also der Anzahl der Parameter) steigt, allerdings auch die Ressourcenintensität, und somit die Zeit welche zum Generieren einer Antwort benötigt wird.

Man kann die drei bereitgestellten Modelle entsprechend anordnen:

1. qwen2:0.5b
2. moondream
3. llama3

Wobei mit Aufsteigender Zahl, die Anzahl der Parameter, die Rechenintensität, und die Qualität der generierten Texte steigt. Mit abnehmender Zahl wird die Geschwindigkeit der Modelle Steigen.

6.3 Lösung 3: Wetter-App

Für diese Übung wird der Bearbeiter mit der Aufgabe betraut, eine Wetter-App um ein LLM zu erweitern, um auf Basis der Wetterdaten für die nächste Stunde einen Text zu generieren.

6.3.1 Lösung 3.1: Funktionstest und Vertrautmachen mit dem Code

- Welche Funktion werden in der Datei `app.py` definiert? -> Die Flask-Api mit ihren Endpunkten!
- Welche Funktion werden in der Datei `weather.py` definiert? -> Die Logik der Wetter-App
- In welcher Methode müssen Sie den Code ergänzen, um das LLM zu integrieren? -> In der `_get_ai_text`-Methode der `Weather`-Klasse
- Welche Parameter erwartet die Web-App, bei Abfrage des Wetters? -> Land und Stadt, in Namen `country` und `city`
- Wie können Sie intern auf die API von `ollama` zugreifen? -> Über das docker-Netzwerk, da der `ollama`-Container keine Ports exponiert, daher einfach `ollama` als FQDN (Fully Qualified Domain Name) verwenden.

6.3.2 Lösung 3.2: Integration des LLM

Grundsätzlich soll der Bearbeiter hier anhand der Dokumentation von `ollama` die API verstehen, und dann in der Lage sein, die API in die Wetter-App zu integrieren. Hierbei sollte der Bearbeiter auch die Problematik des Prompt-Engineering verstehen, besonders im Kontext der begrenzten Rechenressourcen des LLM.

6.3.3 Lösung 3.2.1: Implementierung

Die Implementierung in `weather.py`, in der `_get_ai_text`-Methode sollte wie folgt aussehen:

Listing 6.1: Implementierung der _get_ai_text-Methode

```
1 def _get_ai_text(self, weather_data:dict) -> str:
2     """
3     This method will be used to actually implement the AI-Text
          generation
4     Here, we'll run the request to the ollama api, and before
          we do that do the prompt-engineering
```

```
5
6       :param weather_data: The weather data for which the AI-Text
            should be generated
7       :type weather_data: dict
8       :return: The AI-Text as string
9       """
10
11      ret_str = ""
12
13      today_date = datetime.now(pytz.timezone('Europe/Berlin')).
            strftime('%Y-%m-%d')
14      prompt = f"Beschreibe das Wetter für Heute, den {today_date
            } in {weather_data['city']}, {weather_data['country']},
            auf deutsch! "
15      prompt = prompt + f"Halte dich Kurz!.\nmax_temp: {
            weather_data['daily'][today_date]['max_temp']}, min_temp
            : {weather_data['daily'][today_date]['min_temp']}, "
16      prompt = prompt + f"niederschlag: {weather_data['daily'][
            today_date]['niederschlag']}, wetter: {weather_data['
            daily'][today_date]['wetter']}, "
17      prompt = prompt + f"wind_geschwindigkeit: {weather_data['
            daily'][today_date]['wind_geschwindigkeit']},
            wind_richtung: {weather_data['daily'][today_date]['
            wind_richtung']}"
18
19      payload = {
20          "model": "llama3",
21          "prompt": prompt,
22          "stream": False
23      }
24
25      headers = {
26          "Content-Type": "application/json"
27      }
28
29      response = requests.post("http://ollama:11434/api/generate"
            , headers=headers, data=json.dumps(payload), timeout
            =120)
30      if response.status_code != 200:
31          ret_str = "Es ist ein Fehler aufgetreten, bitte
                versuchen Sie es später erneut."
32      else:
33          ret_str = response.json()['response']
```

```
34
35     return ret_str
```

Hierbei wird zunächst das heutige Datum (in Deutscher Zeitzone) ermittelt, dann ein Prompt generiert, welches die Wetterdaten des heutigen Tages enthält. Hierbei wird sichergestellt, dass die Prompt nicht zu lang ist, da dies die Rechenzeit des LLM erhöhen würde. Weiter wird dann ein `payload` vorbereitet, welcher das Modell, den Prompt, und den Stream-Modus enthält. Hier ist es wichtig, dass der Stream modus auf `false` gesetzt ist, um zu vermeiden, dass `ollama` jedes Token einzeln zurückgibt, was es in der Verarbeitung deutlich weniger einfach machen würde.

Dann wird der `post`-Request an die `ollama`-API gesendet. Wichtig ist hier, dass der `timeout`-Parameter des `post`-Requests auf eine deutlich erhöhte Zahl gesetzt wird. Entsprechend der Dokumentation wird hierbei ein Statuscode von 200 erwartet. Wenn dies nicht der Fall ist, wird eine Fehlermeldung statt eines KI-Textes zurückgegeben. Ansonsten wird der generierte Text zurückgegeben. Hierzu wird der Dokumentation folgend auf das zu erwartende `json`-Objekt zugegriffen.

Literaturverzeichnis

[1] *What is Artificial Intelligence (AI)? | IBM*, en-us, Aug. 2023. [Online]. Verfügbar: https://www.ibm.com/topics/artificial-intelligence (besucht am 12. 07. 2024).

[2] A. Wolfewicz, *Deep Learning vs. Machine Learning – What's The Difference?*, en, Blog, Feb. 2023. [Online]. Verfügbar: https://levity.ai/blog/difference-machine-learning-deep-learning (besucht am 07. 07. 2024).

[3] T. M. Mitchell, *Machine Learning* (McGraw-Hill series in computer science), en. New York: McGraw-Hill, 1997, ISBN: 978-0-07-042807-2.

[4] *What is a Neural Network? | IBM*, en-us, Okt. 2021. [Online]. Verfügbar: https://www.ibm.com/topics/neural-networks (besucht am 12. 07. 2024).

[5] *Large Language Model (LLM) | Learn how to interact with OpenAI models*, en. [Online]. Verfügbar: https://microsoft.github.io/Workshop-Interact-with-OpenAI-models/llms/ (besucht am 30. 06. 2024).

[6] *Was sind Transformatoren? – Transformatoren in der künstlichen Intelligenz erklärt – AWS*, de-DE. [Online]. Verfügbar: https://aws.amazon.com/de/what-is/transformers-in-artificial-intelligence/ (besucht am 08. 07. 2024).

[7] A. mistral, *Function calling | Mistral AI Large Language Models*, en, Documentation. [Online]. Verfügbar: https://docs.mistral.ai/capabilities/function_calling/ (besucht am 12. 07. 2024).

[8] *qwen2*. [Online]. Verfügbar: https://ollama.com/library/qwen2 (besucht am 11. 07. 2024).

[9] M. Hausmann, *SOFAH*, en, Documentation, Juni 2024. [Online]. Verfügbar: https://github.com/sofahd (besucht am 12. 07. 2024).

[10] *DALL · E 2 | OpenAI*. [Online]. Verfügbar: https://openai.com/index/dall-e-2/ (besucht am 11. 07. 2024).