



LLM-gestuetzte-Softwareentwicklung-Labor

aus dem Studiengang LLM-gestuetzte-Softwareentwicklung

an der Hochschule Esslingen University of Applied Sciences Campus
Esslingen Flandernstraße

von

Tim Jauch

Annabel Heberle

10.01.2026

Matrikelnummer, Kurs:
763086, SWB
770677, SWB

Betreuer an der HSE: Prof. Dr. Jörg Nitzsche, Dr. Stefan Kaufmann, Clemens Morbe

Selbstständigkeitserklärung

Wir versichern hiermit, dass wir unsere Arbeit mit dem Thema:

LLM-gestuetzte-Softwareentwicklung-Labor

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass alle eingereichten Fassungen übereinstimmen.

Esslingen, 10.01.2026

T. Jauch

Tim Jauch

Annabel Heberle

Annabel Heberle

Inhaltsverzeichnis

1 Einleitung	1
1.1 Vorstellung des Projekts	1
1.2 Zielsetzung und Relevanz	1
1.3 Aufbau der Arbeit	1
2 Projektbeschreibung	2
2.1 Problemstellung und Anforderungen	2
2.2 Zielgruppe und Personas	2
3 Softwarearchitektur und Technologie	3
3.1 Gesamtübersicht der Architektur	3
3.2 Frontend/Benutzeroberfläche	4
3.3 Backend, Datenbank und Schnittstellen	4
3.4 LLM-Integration (MCP, Wrapper, API)	4
3.5 Sicherheits- und Authentifizierungsmechanismen	5
3.6 Sonstiges	5
4 Implementierung	6
4.1 Vorgehensmodell und Projektmanagement	6
4.2 Code-Generierung und Entwicklung	6
4.3 Datenmodelle und Schnittstellenimplementierung	7
4.3.1 Domain-Model Diagramm	7
4.3.2 Schnittstellenimplementierung	8
4.4 Versionskontrolle, Build- und CI-Prozesse	8
5 Einsatz der LLMs und Prompt Engineering	10
5.1 Auswahl der LLMs und Dienste	10
5.2 Aufbau und Dokumentation der Prompts	11
5.2.1 Prompt Grundstruktur	11
5.2.2 Prompt-Vorgehen	12
5.3 Generierte Assets (Code, Bilder, Audio, Video, Text)	12
5.3.1 Stammdaten in JSON-Format	12
5.3.2 Bildgenerierung	13
5.4 Fehleranalyse und Optimierung	15
5.5 Qualitätssicherung bei der LLM-Nutzung	16
6 Anpassungen nach der Präsentation	17
6.1 Frontend	17

6.2 Backend	17
6.3 Allgemein	18
7 Erfahrung, Herausforderungen und Reflexion	19
7.1 Positive Erfahrungen und Erfolgsgeschichten	19
7.2 Schwierigkeiten und Problemstellung	19
7.3 Grenzen und Risiken von LLMs	20
7.4 Lessons Learned und Empfehlungen	20
7.5 Zukunft der LLM-gestützten Entwicklung	20
8 Zusammenfassung und Fazit	22
8.1 Zusammenfassung der Arbeit	22
8.2 Persönliche Reflexion und Fazit	22

Abbildungsverzeichnis

Abbildung 1	Architektur der Software	3
Abbildung 2	Domain Model Diagramm	7
Abbildung 4	Der Spinosaurus	14
Abbildung 5	Der Spinosaurus	14
Abbildung 7	Dinosaurier vor Promptanpassung	14
Abbildung 8	Dinosaurier nach Promptanpassung mit Ergänzung: „Nicht fusioniert“	14
Abbildung 10	Zu realistischer Dinosaurier 1	15
Abbildung 11	Zu realistischer Dinosaurier 2	15

Codeverzeichnis

Code 1 Prompt für die Erstellung der DinoData.json	12
Code 2 Auszug aus dem Dinosaurier-Datensatz	13
Code 3 Promt für die Generierung eines Bildes	13

1 Einleitung

Stack Attack: All or nothing!

1.1 Vorstellung des Projekts

Stack Attack ist ein Softwareprojekt, welches mit Hilfe von LLMs entwickelt und dokumentiert wird. Das Ziel des Projekts ist es, eine Online-Version des beliebten Quartett-Kartenspiels zu erstellen. Hierbei gilt es, den Aufwand bei der Erstellung der Software mithilfe von LLMs zu minimieren und gleichzeitig eine qualitativ hochwertige Software zu entwickeln.

1.2 Zielsetzung und Relevanz

Ziel dieses Softwareprojekts ist der Einsatz von LLMs zur Unterstützung bei der Softwareentwicklung und die Integration in selbige. Durch den Einsatz von LLMs soll der Entwicklungsprozess effizienter gestaltet werden, mit gleichbleibender Qualität.

Wichtig und relevant ist das Arbeiten mit LLMs, da von einer Steigerung der Produktivität ausgegangen wird. Dadurch kann der Aufwand und somit in der realen Welt Kosten verringert werden. Außerdem ist der Vergleich von statischem Code, welcher eine Entscheidung trifft und dynamischen Entscheidungen durch LLMs besonders hinsichtlich Performance und Qualität beachtenswert.

1.3 Aufbau der Arbeit

Nach einer Projektbeschreibung folgt die Softwarearchitektur und im Zuge der Implementierung wird noch detaillierter auf den Einsatz der LLMs und Prompt-Engineering eingegangen. Abschließend geht es noch um Erfahrungen, Herausforderungen, Reflexion, sowie eine Zusammenfassung und das Fazit.

2 Projektbeschreibung

2.1 Problemstellung und Anforderungen

Die Problemstellung besteht darin, eine Online-Version des Kartenspiels Quartett zu erstellen und gegen eine LLM spielen zu können. Die Anforderungen umfassen die Erstellung einer benutzerfreundlichen Oberfläche, die Implementierung der Spiellogik und die Integration einer LLM als Gegner.

2.2 Zielgruppe und Personas

Die Zielgruppe sind Kinder und Erwachsene, welche gerne Quartett spielen und Dinos mögen. Für die Kinder steht das Spielen und Lernen im Vordergrund, während Erwachsene konkret erfahren können, was kleine LLM-Modelle leisten können. Dabei kann die Qualität und Performance ganz gut beobachtet werden.

Von LLM erstellte passende Personas: Persona 1: Dino-begeistertes Kind

Alter: 9-12 Jahre; Motivation: Spielen, Lernen, Spaß an Dinos und Gewinnen. Ziele: Neue Dino-Arten entdecken, Spielregeln schnell verstehen und Erfolgsergebnisse im Spiel. Technik: Tablet/Smartphone, einfache Bedienung, wenig Geduld für komplexe Menüs. Bedürfnisse: Klare Anleitung, kurze Texte, große Buttons, bunte Visuals, schnelle Antworten, unkompliziert. Schmerzpunkte: Zu viel Text, langsame Ladezeiten, unklare Regeln.

Persona 2: Technik-affiner Erwachsener (z.B. Elternteil oder Lehrkraft)

Alter: 30-45 Jahre; Motivation: Qualität und Performance kleiner LLM-Modelle beurteilen, Lern-/Unterhaltungswert für Kinder testen. Ziele: Sehen, wie stabil und fix die LLM-Antworten sind, Inhalte auf Korrektheit prüfen, didaktischen Nutzen für Unterricht/Spieleinsätze bewerten. Technik: Desktop/Laptop/Tablet; Bedürfnisse: Übersichtliches Monitoring (Antwortzeiten, Qualität), nachvollziehbare Quellen/Erklärungen, einfache Einrichtung (Docker/Setup). Schmerzpunkte: unklare Modellgrenzen, Halluzinationen ohne Kennzeichnung, fehlende Logs/Statistiken, aufwendiges Setup.

3 Softwarearchitektur und Technologie

3.1 Gesamtübersicht der Architektur

Mermaid Chart der Softwarearchitektur in Github

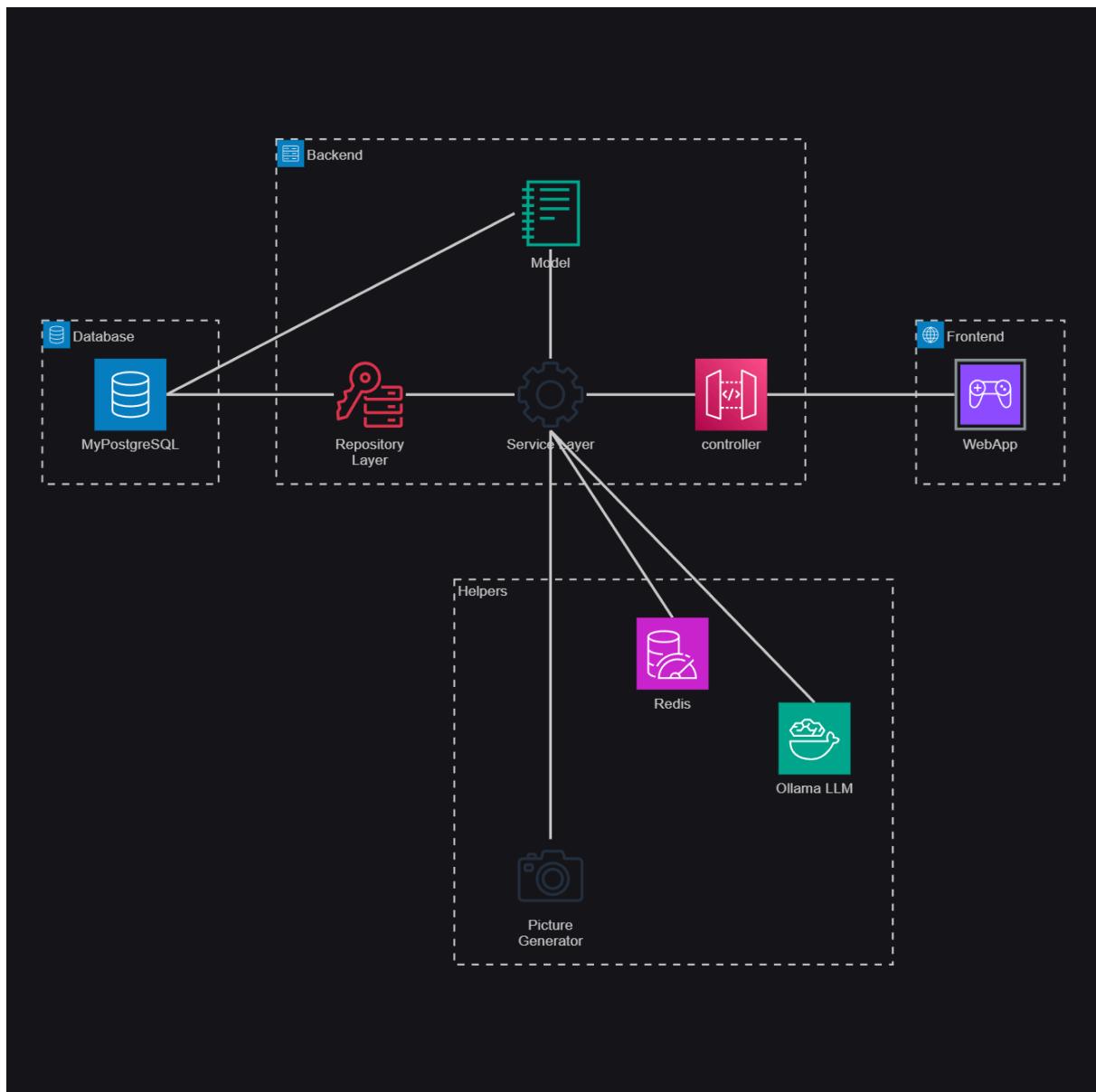


Abbildung 1 – Architektur der Software

3.2 Frontend/Benutzeroberfläche

Beim ersten Aufruf blockiert ein kleines Dialogfenster das Spiel, bis ein Spielername eingegeben und gespeichert ist. Name und Profil bleiben für spätere Sitzungen erhalten. Von dort lässt sich jederzeit ein neues Match über den Startknopf beginnen. Während des Spiels stehen oben die aktuellen Kartenzahlen beider Decks, darunter die oberste Spielerkarte mit Name, Bild, Gruppencode und den sechs spiel relevanten Attributen Lifespan, Length, Speed, Intelligence, Attack und Defense, die per Klick zum Vergleich ausgewählt werden. Ist der Mensch am Zug, sind die Buttons aktiv. Gewinnt die KI eine Runde und das Spiel läuft weiter, spielt sie nach kurzer Bedenkzeit automatisch erneut. Auf Wunsch kann bis zu dreimal pro Match der Joker verwendet werden, bei welchem der KI-Gegner für den Menschen eine Wahl trifft. Nach jedem Zug zeigt die Oberfläche das Rundenergebnis, beide Werte und die aktualisierten Deckgrößen. Ist das Spiel vorbei, erscheint ein Game Over Bereich mit Gesamtsieger und Play Again Option. Lade und Fehlersituationen machen sich durch deaktivierte Buttons und eine sichtbare Fehlermeldung bemerkbar. Typescript und React bilden das Fundament.

3.3 Backend, Datenbank und Schnittstellen

Im Backend läuft eine Java Spring Boot Anwendung, die ihre REST API bereitstellt und den Spielablauf steuert. Persistente Daten wie Dinosaurier und Spieler landen in PostgreSQL, während laufende Matches im schnellen Redis Cache gehalten werden. Die Infrastruktur startet über Docker, inklusive Adminer zur Datenbankinspektion. Datenbankänderungen werden mit Flyway versioniert. Für Entscheidungen im Spiel fragt der Server ein Ollama Modell (phi3:mini) an, und greift so auf LLM Unterstützung zurück. Insgesamt bildet das Backend das Fundament zwischen Frontend, Datenbank und KI Helfer.

3.4 LLM-Integration (MCP, Wrapper, API)

Die LLM-Integration sitzt im LlmClientService.java: Das Backend ruft über ein konfiguriertes RestTemplate den Ollama-Container an, der das Modell phi3:mini bereitstellt. Für jede Entscheidung baut der Service einen klaren Prompt mit den sechs zulässigen Attributen (lifespan, length, speed, intelligence, attack, defense) und schickt ihn an die Chat-API des Containers. Falls die LLM eine unbrauchbare Antwort liefert, was bei LLMs eine zu berücksichtigende Möglichkeit ist, so greift die Fallback-Logik. Fallback-Logik wählt den höchsten Wert, sollte auch dies nicht funktionieren, so wird einfach das Angriffs-Attribut ausgewählt. Das Timeout

ist auf 2 Minuten (120000 ms) gesetzt, wodurch phi3:mini die Inferenz durchzuführen und eine Antwort zu generieren. Kommt etwas Unerwartetes zurück oder gar nichts, greift die bereits erwähnte Fallback-Logik, sodass das Spiel auch ohne LLM-Antwort weiterläuft.

3.5 Sicherheits- und Authentifizierungsmechanismen

Im Rahmen dieses Projektes wurden keine speziellen Sicherheits- oder Authentifizierungsmechanismen implementiert, da es sich um eine einfache Spielanwendung handelt, die keine sensiblen Daten verarbeitet. Von anderen kleinen Online-Games ist die Funktionsweise bekannt, dass Username und minimale Spieldaten über den Erfolg gespeichert werden, ohne je ein Passwort festzulegen. Ziel war, dass einem User ein am laufendes Spiel zugeordnet werden kann und auf einer Profil-Page Informationen über die Anzahl an Siegen und Verlusten angezeigt werden können. Lokal werden Spielername und ID als einziges gespeichert, sodass diese Ziele ohne Authentifizierung erreicht werden können.

3.6 Sonstiges

Die Dokumentation erfolgt mit Typst und Mermaid Chart wird verwendet um Diagramme zu erstellen.

4 Implementierung

4.1 Vorgehensmodell und Projektmanagement

Während dieses Projektes wurden keine festen Vorgehensmodelle und Projektmanagement-Tools eingesetzt und stattdessen mehr auf flexibles Arbeiten gesetzt. Dies ist bewusst und kein Fehler. Das Team hat sich dazu entschieden, um mehr Zeit für die Generierung von Code zu haben, da bei kleinen Projekten eine mündliche Planung im Rahmen von Meetings für ausreichend befunden wurde. In wöchentlichen Meetings wurden die nächsten Schritte geplant, Aufgaben verteilt und der Fortschritt überprüft. Während der Entwicklung wurde der main Branch geschützt, sodass nur Änderungen eines überprüften Pull Requests durch ein anderes Teammitglied in diesen zusammengeführt werden konnten. Dadurch konnte eine höhere Codequalität gewährleistet werden.

4.2 Code-Generierung und Entwicklung

Der grundlegende Ablauf dieses Projektes erfolgte sehr klassisch. Nach Planen des Projekts und der Features wurde mit dem Set-up des Projektes begonnen. Im Anschluss wurden die persistente Datenbank und das Backend eingerichtet. Nach anfänglicher Konfiguration und Testen wurde anhand von abgesprochenen User Stories zunächst die Grundfunktionen des Spiels implementiert. Hierbei wurde der Fokus auf die Spiellogik und die Kommunikation zwischen Frontend und Backend gelegt. Die Backendfunktionen, also die Spiellogik ist das Herzstück des Projekts und wurde dementsprechend auch so behandelt. Das Frontend wurde dann häufig, abhängig von dem Backend, mit LLMs zunächst testmäßig generiert. Anschließend wurde die LLM-Integration implementiert und getestet. Während der Entwicklung wurden immer wieder kleinere Anpassungen vorgenommen, um die Benutzerfreundlichkeit zu verbessern und Fehler zu beheben. Das Frontend wurde größtenteils nach Abschließen aller Funktionalitäten neu aufgebaut und optimiert.

4.3 Datenmodelle und Schnittstellenimplementierung

4.3.1 Domain-Model Diagramm

Bei den Datenmodellen wird zwischen persistenten und nicht persistenten Daten unterschieden. Persistente Daten werden in der PostgreSQL Datenbank gespeichert, während die nicht persistenten Daten in Redis gespeichert werden oder nur In-Memory verwendet werden. Zu den persistenten Daten zählen die Spieler Daten aus der Player Entität und die einzelnen Dinosaurier. Nicht persistent sind die Matchstates, welche Informationen über ein laufendes Spiel enthalten und nach Beenden eines Spiels wieder gelöscht werden, sowie die Dino-Karten die aus der Dinosaurier Entität entstehen und innerhalb des Matchstates gespeichert werden. Das Domain-Model Diagramm zeigt die Verbindung zwischen den Datenmodellen.

Mermaid Chart des Domain Model Diagramms in Github

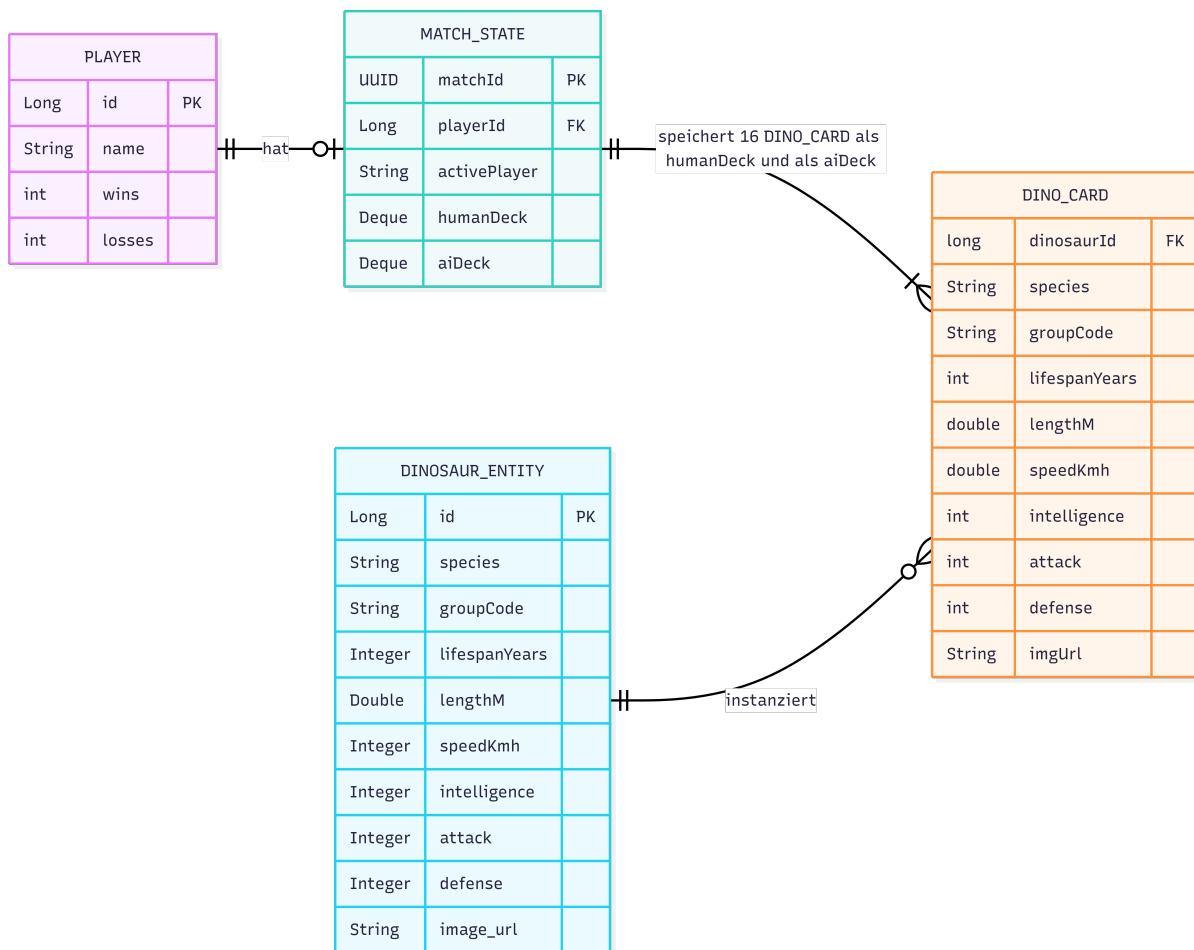


Abbildung 2 – Domain Model Diagramm

4.3.2 Schnittstellenimplementierung

Methode	Endpunkt	Payload	Beschreibung	Daten
POST	/players	CreatePlayer-Request	Eintrag eines neuen Spielers	DB
GET	/players/{id}	-	Abrufen Spielerdaten	DB
POST	/players/{id}/result	Id, outcome	Spielergebnis eines Spielers dokumentieren	DB
GET	/dinosaurs	-	Abrufen aller Dinosaurier	DB
GET	/dinosaurs/{id}	-	Abrufen eines Dinosauriers	DB
POST	/matches/start	PlayerId	Erstellen eines MatchStates	Redis
POST	/matches/{matchId}/play	PlayCardRequestDto	Spielen einer Karte & Kartenvergleich	Redis
POST	/matches/{matchId}/suggest-attribute	-	Joker: AI spielt für User	Redis

4.4 Versionskontrolle, Build- und CI-Prozesse

Für die Versionskontrolle wurde Git in Kombination mit GitHub eingesetzt. Der gesamte Quellcode sowie die Projektdokumentation wurden in einem zentralen Repository verwaltet. Die Entwicklung erfolgte überwiegend auf Feature-Banches, die nach erfolgreicher Implementierung und lokaler Prüfung in den Hauptbranch (main) integriert wurden. Durch diese Vorgehensweise konnten einzelne Änderungen klar nachvollzogen und isoliert entwickelt werden.

Commits wurden in kleinen, thematisch zusammenhängenden Schritten durchgeführt, um insbesondere bei der LLM-gestützten Codegenerierung eine transparente Entwicklungshistorie zu gewährleisten.

Die Build-Prozesse wurden lokal durchgeführt und sind klar zwischen Frontend und Backend getrennt. Im Frontend erfolgt der Buildprozess über Node.js und npm. Das Backend wird über den Maven Wrapper gestartet. Für die genauen Befehle hilft ein Blick in die README-Datei.

Zusätzlich ermöglichte Docker eine reproduzierbare Ausführung der gesamten Systemumgebung, einschließlich Datenbank, Redis, Adminer und LLM-Service. Dadurch konnte das Projekt unabhängig von der lokalen Entwicklungsumgebung konsistent betrieben werden.

Auf einen vollautomatisierten Continuous-Integration-Prozess (CI) wurde im Rahmen dieses Projekts bewusst verzichtet. Aufgrund des begrenzten Projektumfangs und des prototypischen Charakters lag der Fokus auf der funktionalen Entwicklung und der Integration von LLMs. Die eingesetzten Build- und Versionskontrollmechanismen bilden jedoch eine solide Grundlage, um zukünftig automatisierte Tests, Linting oder Build-Pipelines (z. B. über GitHub Actions) zu ergänzen.

Der Einsatz von LLMs bei dem Schreiben von Commit-Nachrichten wurde getestet, allerdings stellten wir schnell fest, dass diese nicht so gut verständlich waren wie menschliche und legten dies somit ab.

5 Einsatz der LLMs und Prompt Engineering

5.1 Auswahl der LLMs und Dienste

Bei der Entwicklung des Codes wurden die KI-Agenten Codex und GitHub Copilot verwendet, die der Unterstützung während des Programmierens dienten. Ein zentrales Ziel des Projekts war die Implementierung von Large Language Models (LLMs) in zwei verschiedenen Anwendungsbereichen: Erstens die automatisierte Generierung von Dinosaurier-Abbildungen für die Quartettkarten über eine externe API und zweitens die Realisierung eines KI-Gegners als Kernfunktion des Spiels. Bei der Wahl dieser LLMs war das ausschlaggebendste Kriterium, dass sie kostenlos verwendbar sein mussten, was sich schnell als Herausforderung erwies.

Um den KI-Gegener zu realisieren, wurde das kleine lokale Modell phi3:mini gewählt. Die Bereitstellung des Modells erfolgt über Ollama, ein Framework zur Ausführung verschiedener Sprachmodelle auf lokaler Hardware. Wobei um die Skalierbarkeit zu erhöhen Ollama in einen Docker-Container gekapselt wurde.

Ollama fungiert hierbei als Schnittstelle, über die das Backend Spielzustände an das phi3:mini Modell sendet und darauf Entscheidungen der KI zurückhält. Phi3:mini hat sich für dieses Projekt geeignet, da es sowohl kostenlos, sowie relativ kompakt und somit von unserer Hardware stemmbar war.

Die Wahl eines API-Endpunktes war etwas eingeschränkt, da nicht viele kostenlos zur Verfügung stehen. Prinzipiell war die Wahl allerdings nicht von zu großer Bedeutung, da der Code so strukturiert wurde, dass lediglich der API-Endpunkt geändert werden muss, um ein anderes Modell nach Belieben zu verwenden. Über Huggingface-Inference-API wurde das Modell Stable Diffusion XL (SDXL) 1.0 von Stabilityai für die Dinosaurier-Bildgenerierung angebunden. Huggingface ist ein tokenbasierter Service, der API-Endpunkte für verschiedene Modelle zur Verfügung stellt. Das SDXL Modell erzeugt eine hohe Ausgabequalität und ist aufgeteilt in ein Base- und ein Refiner-Modell. Das Base-Modell erzeugt die erste Grundlage des Bildes und kann optional durch das Refiner-Modell hinsichtlich Detailgrad und Prompttreue optimiert werden. Da Huggingface tokenbasiert ist und somit die Bildgenerierung limitiert war, erwies

sich SDXL als effizient, da das Base-Modell ausreichend war und gezielt einzelne Bilder durch den Refiner optimiert werden konnten, um das Token-Kontingent zu schonen.

5.2 Aufbau und Dokumentation der Prompts

Der Aufbau der Prompts folgte einem einheitlichen Grundschema. Zunächst wurde der Rolle des Modells ein klarer Kontext gegeben (z. B. Attributwahl). Anschließend wurde die konkrete Aufgabe beschrieben und durch explizite Einschränkungen ergänzt. Ziel war es, den Interpretationsspielraum des Modells möglichst gering zu halten und reproduzierbare Ergebnisse zu erzielen.

Bei der Attributwahl wurde das Modell angewiesen, ausschließlich eines von sechs vorgegebenen Attributen zurückzugeben und keine erläuternden Texte zu generieren. Zusätzlich wurden die Kartendaten in strukturierter Form übergeben, um Fehlinterpretationen zu minimieren. Nach der Präsentation des Projekts wurde der Prompt weiter verfeinert, indem zusätzliche Informationen wie Minimal- und Maximalwerte der Attribute ergänzt wurden, um die Entscheidungsqualität zu verbessern.

5.2.1 Prompt Grundstruktur

Für die Codegenerierung hatten wir ein festes Schema für die Prompts das wir meistens angewendet haben.:

- 1. Kontext** (Worum geht es? Bsp.: React Frontend, Java Backend, LLM Integration)
- 2. Aufgabe** (Was soll generiert werden? Bsp.: Ein Button der eine REST-API GET Anfrage macht)
- 3. Einschränkungen** (Was muss beachtet werden? Bsp.: Nutze Typescript, aber verändere so wenig Code wie möglich)
- 4. Erwartetes Ausgabenformat** (Wie will ich das Ergebnis haben? Bsp.: Nur Code, oder JSON)
- 5. Fehler, Beispiele oder spezielle Hinweise (Optional)** (Was muss zusätzlich beachtet werden? Bsp.: Error oder Beispielcode)

5.2.2 Prompt-Vorgehen

Bei der Code-Generierung wurde ein iteratives Prompting-Vorgehen angewendet. Dabei wurden durch das LLM erzeugte Codevorschläge manuell überprüft, getestet und bei Bedarf durch gezielte Rückfragen oder Präzisierungen weiterentwickelt. Dieser Prozess folgt dem Human-in-the-Loop-Prinzip, bei dem der Mensch die Kontrolle über Entwurfsentscheidungen, Korrekturen und finale Implementierungen behält. Insbesondere bei komplexeren Codeabschnitten erwies sich dieses Vorgehen als notwendig, um fehlerhafte oder unpassende Vorschläge zu identifizieren und zu korrigieren.

5.3 Generierte Assets (Code, Bilder, Audio, Video, Text)

Im Projekt wurden verschiedene Assets generiert, dazu gehören: Code, Stammdaten in Form von JSON und Bilder von Dinos für die Karten.

5.3.1 Stammdaten in JSON-Format

Die Basis dieses Projektes bilden die Dinosaurierdaten, aus welchen die einzelnen Karten im Quartettspiel erstellt werden. Da keiner der Teammitglieder ausgiebiges Wissen über Dinosaurier hatte, beauftragten wir ein LLM über einen Prompt ein JSON-Objekt zu erstellen, welches später dazu verwendet wurde, diese Daten in die Datenbank zu übertragen.

Create a JSON-Object that contains 32 dinosaurs. Choose 32 dinosaurs where a group of 4 are in a similar dinosaur family, where each dinosaur has a group value, consisting of the group number and a letter. For example 1A-1D. Each dinosaur has the 6 attributes lifespan, length, speed, intelligence, attack and defense. Choose fitting values for these attributes in range 0 to 100, by looking up characteristic traits of the dinosaurs.

Code 1 – Prompt für die Erstellung der DinoData.json

```
[JSON
  {
    "species": "Tyrannosaurus rex",
    "group": "1A",
    "lifespan_years": 28,
    "length_m": 12.3,
    "speed_kmh": 27,
    "intelligence": 58,
    "attack": 98,
    "defense": 70
  },
]
```

Code 2 – Auszug aus dem Dinosaurier-Datensatz

5.3.2 Bildgenerierung

Für die Bildgenerierung wird in einem Startup-Service überprüft, ob ein Dinosaurier in der Datenbank noch keinen Bildvermerk hat und im Falle dessen, für diesen Dinosaurier ein Prompt an das Modell gesendet.

```
String prompt = "Create an image of the dinosaur species: " PROMPT
  + d.getSpecies()
  + ". Style: Cute anime/cartoon with bright/strong colors."
  + "Include characteristic features for each dinosaur and a "
  + "background matching its natural habitat. Make the dinosaur "
  + "easy to recognize by using some realistic characteristic "
  + "features. The image should be suitable for a quartett "
  + "card game, similar in layout to a Pokémon card.;"
```

Code 3 – Promt für die Generierung eines Bildes

Der obige Prompt wurde im Allgemeinen verwendet, um die Bilder zu generieren und erzielte im Großen und Ganzen die besten Ergebnisse. Allerdings gab es Einzelfälle, für die der Prompt angepasst werden musste, damit ein passendes Bild entsteht. Aufgrund der begrenzten Tokens war das Experimentieren mit verschiedenen Prompts eingeschränkt. Die unteren Bilder zeigen gute Ergebnisse, die mit diesem Prompt generiert wurden.



Abbildung 4 – Der Spinosaurus

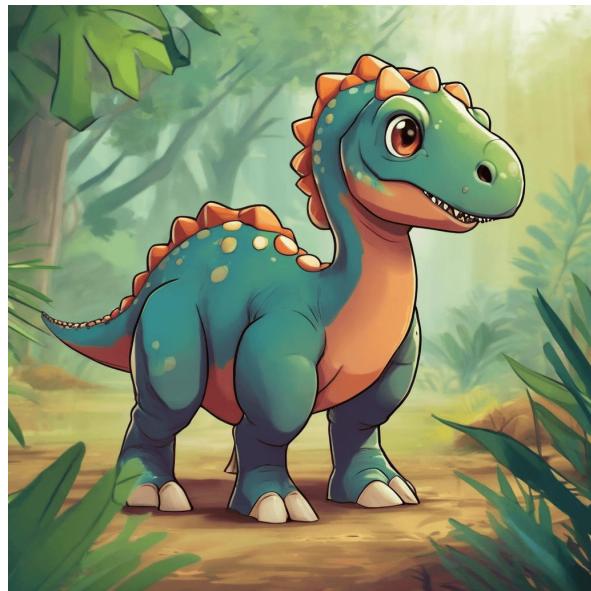


Abbildung 5 – Der Spinosaurus

Aber auch mit diesem Prompt wurden gelegentlich fehlerhafte Bilder generiert, beispielsweise Dinosaurier mit zu vielen oder zu wenigen Gliedmaßen, die ineinander verschmolzen waren. Um diese Bilder zu verbessern wurde der obige Prompt für Einzelfälle angepasst.



Abbildung 7 – Dinosaurier vor Promptanpassung



Abbildung 8 – Dinosaurier nach Promptanpassung mit Ergänzung: „Nicht fusioniert“

Bevor allerdings der obige Prompt verwendet wurde, gab es einen großen Fehlschlag, der die Dinosaurier zu realistisch aussehen lassen mit einem unschönen Farbschema. Dieser

Prompt wurde durch drei Schlüsselworte „cute“, „bright/strong colors“ und „some realistic characteristic features“ erweitert. Ergebnisse ohne diese Erweiterungen zeigen die unteren Abbildungen auf.



Abbildung 10 – Zu realistischer Dinosaurier

1



Abbildung 11 – Zu realistischer Dinosaurier

2

5.4 Fehleranalyse und Optimierung

Die Fehleranalyse erfolgte überwiegend durch manuelles Testen und das gezielte Durchspielen typischer Spielabläufe. Dabei wurden sowohl die Benutzeroberfläche als auch die Backend-Logik überprüft, indem komplett Matches gespielt und verschiedene Spielsituationen nachvollzogen wurden. Auf diese Weise konnten fehlerhafte Zustandswechsel, unerwartete Reaktionen auf API-Aufrufe sowie Darstellungsprobleme im Frontend identifiziert und behoben werden.

Zusätzlich zur praktischen Erprobung wurde der Backend-Code anhand des vorhandenen Wissens über die eingesetzte Programmiersprache und das Framework überprüft. Durch das Lesen und Analysieren der Codeabschnitte konnten potenzielle Fehlerquellen, unklare Logik und ungünstige Implementierungsentscheidungen erkannt und verbessert werden. Dieses Vorgehen erwies sich insbesondere bei der Spiellogik und der Verarbeitung von LLM-Antworten als hilfreich.

Ein weiterer Schwerpunkt lag auf der Behandlung fehlerhafter oder unvollständiger Daten. Stammdaten in Form von JSON-Dateien wurden manuell kontrolliert, um fehlende Felder oder unplausible Werte zu erkennen. Die zugehörigen Bilder wurden visuell geprüft, um mögliche Halluzinationen oder inhaltliche Abweichungen bei KI-generierten Assets zu identifizieren.

Im Zusammenhang mit der LLM-Integration traten vereinzelt unerwartete Antwortformate oder inhaltlich unpassende Ergebnisse auf. Diese wurden durch Validierungslogik, klare Vorgaben im Prompt sowie durch Fallback-Mechanismen abgefangen. Die Optimierung erfolgte iterativ, indem sowohl Code als auch Prompts auf Basis der gemachten Beobachtungen angepasst wurden.

5.5 Qualitätssicherung bei der LLM-Nutzung

Besonders bei der Nutzung der KI-Agenten musste auf Richtigkeit und Qualität des Codes geachtet werden. Um Qualität zu verbessern, erhielten die Agenten bei Verwendung nur kleinere Aufgabenpäckchen, da das Ergebnis des Outputs dadurch verbessert werden konnte. Überreichen mehrerer Aufgaben gleichzeitig führte dazu, dass die Agenten eine Aufgabe besser und die andere schlechter beziehungsweise gar nicht bearbeiteten. Des Weiteren wurden die eingefügten Codeabschnitte überprüft und getestet, bevor diese beibehalten wurden.

6 Anpassungen nach der Präsentation

Im Zuge der Präsentation gab es wertvolles Feedback, welches in die Weiterentwicklung des Projekts einfloss. Basierend auf den Anregungen der Präsentation, sowie eigener Ideen wurden folgende Anpassungen vorgenommen:

6.1 Frontend

Zum Zeitpunkt der Präsentation hatte das Frontend Ausbaupotential. Ursprünglich wurde nach jedem Spielzug ein Feld angezeigt, in welchem der Gewinner, die verglichenen Attribute und deren Werte angezeigt wurden, nicht aber die gesamte Gegnerkarte. Durch Anregung aus dem Feedback wurde das Frontend weitgehend optimiert, sodass nun auch die Karte des KI-Gegners angezeigt wird. Während jeder Matchrunde wird die Karte dabei verschwommen angezeigt und nach Matchentscheidung offenbart, sodass der User die Karten vergleichen kann.

Um die User-Experience zu optimieren, wurden des Weiteren Animationen der Kartenränder eingefügt, um ganz einfach Matchgewinner und Verlierer zu erkennen. Darüber hinaus wurde das Layout des Frontends angepasst, um ansprechender zu sein. Zusätzlich wurde ein Responsive-Design eingeführt, sodass das Spiel auf kleinen und großen Bildschirmen funktioniert. Hinzu kommt eine Userseite auf welcher ein User Profil angezeigt wird, welches ein Diagramm über die totalen Gewinne und Verluste anzeigt.

6.2 Backend

Damit der KI-Gegner eine fundierte Entscheidung treffen kann, müssen dem LLM die möglichen Minimum- und Maximumwerte der Attribute bekannt sein, wie dem Feedback der Präsentation entnommen wurde. Daher wurde der Prompt, der an das LLM-Modell weitergebracht wird, in dieser Hinsicht optimiert. Bei erstmaliger Optimierung erhielten alle Attribute die Minimum- und Maximumwerte 0 bis 100.

Eine detaillierte Analyse der zugrundeliegenden DinoData.json Datei ergab jedoch, dass die tatsächlichen Wertebereiche stark variieren. So liegt beispielsweise der Wert für „Speed“ kon-

stant zwischen 18 und 60, während das Attribut „Angriff“ Werte zwischen 40 und 98 aufweist. Ohne diese Information würde die KI eine Fehlentscheidung treffen: Ein Geschwindigkeitswert von 60 stellt das absolute Maximum dar und garantiert einen Sieg, wohingegen ein Angriffswert von 70 lediglich im mittleren Bereich liegt. Bei Beibehalten der starren Grenzen von 0-100 im Prompt, würde das Modell die Siegwahrscheinlichkeit von Geschwindigkeit niedriger einschätzen als die von Angriff, was fehlerhaft wäre. Um die Entscheidungsqualität zu optimieren, wurden die Minimum- und Maximumwerte eines jeden Attributs einzeln im Prompt definiert. Erst durch diesen Kontext ist das Modell dazu in der Lage, eine fundierte Entscheidung zu treffen. Diese Promptanpassung funktioniert allerdings nur, da es in dieser Version nur ein zu spielendes Dino-Set gibt.

6.3 Allgemein

Im Anschluss an die Präsentation erfolgte neben der Optimierung der Features eine umfassende Optimierung der Codestruktur und der Namenskonventionen. Ein kritischer Punkt war die ursprüngliche Architektur des Frontends, bei der die gesamte Programmlogik in einer einzigen, unübersichtlichen Datei konzentriert war.

Um die Wartbarkeit und Lesbarkeit zu verbessern, wurde ein Refactoring durchgeführt. Dabei wurde die Logik in modulare Komponenten unterteilt und eine klare Verzeichnisstruktur eingeführt.

7 Erfahrung, Herausforderungen und Reflexion

7.1 Positive Erfahrungen und Erfolgsgeschichten

Durch den gezielten und bewussten Einsatz von LLMs ist ein schneller Einstieg in neue Technologien möglich gewesen, insbesondere auch bei der Technologienfindung hat der Einsatz der LLMs sich als positiv erwiesen. Bei simplen Fehlern konnten LLMs schnell Abhilfe schaffen und den Entwicklungsprozess beschleunigen.

Als überwiegend positiv hat sich auch die Frage nach Beispielen herausgestellt, wenn es beispielsweise darum ging, wie man eine neue Technologie einsetzt.

7.2 Schwierigkeiten und Problemstellung

Falschinformationen durch LLMs und unklare Grenzen der LLMs haben zu Schwierigkeiten und teils längeren Umwegen geführt. Diese Umwege mussten dann menschlich korrigiert werden und die Fehlerbehebung fand dann mithilfe der traditionellen Fehlersuche statt (Google, Stackoverflow, Reddit usw.). Als problematisch hat sich deshalb erwiesen, dass die LLM bei fast jeder Frage eine Antwort generiert, auch wenn diese falsch ist oder die Antwort unbekannt ist. Daher wurde auch beispielsweise gezielt eine Überprüfung der Antworten von phi3:mini vorgenommen, sowie deren Fallback-Logik, damit das Spiel nicht einfach in der Mitte eines Durchlaufs stehen bleibt.

Zusätzlich konnten die KI-Agenten mit kleineren Aufgabenpäckchen deutlich besser umgehen, sodass es auch vorkam, dass diese das Ausführen von Anforderungen verweigerten aufgrund des Umfangs. Außerdem kam es des Öfteren vor, dass Anweisungen nicht genau befolgt wurden und ungefragt unerwünschte Sachen hinzugefügt wurden. Besonders bei der Optimierung des Frontends erwies sich dies als problematisch. Während das Grundgerüst sehr gut mit der Hilfe von KI-Agenten eingerichtet werden konnte, war die Änderung dessen deutlich schwieriger. Besonders in spezifischen Stilwünschen und Anforderungen an das Responsive-Design, sodass eigenhändiges Optimieren schneller und effizienter war.

7.3 Grenzen und Risiken von LLMs

Besonders beim Debugging und der Fehlersuche wurden die Schwächen und Grenzen der LLMs deutlich. Hierbei ist ein Kreislauf des Todes bei Problemen nicht unüblich. Exemplarische Darstellung dieser Prompts:

- „Bitte löse das Problem ...“
- „Ja, klar ich hier die verbesserte Version:“

Dabei resultierte die Ausgabe oft in einer exakten Reproduktion des Quellcodes, einer fehlerbehafteten Variante des ursprünglichen Codes oder in vollständig neu generiertem Code, der wiederum neue Fehler aufweisen kann

Deutlich wurde die Fehlerhäufigkeit bei größerem Scope, also z.B. langen Codeabschnitten. Hierbei ist unserer Beobachtung nach die Fehlerquote deutlich höher als bei kleineren Codeabschnitten.

7.4 Lessons Learned und Empfehlungen

LLMs sind fantastisch beim Einstieg in neue Technologien, Brainstorming, einfachen kreativen Aufgaben, allerdings nicht als alleinige Lösung geeignet.

Ist aber ein bereits fertiges Backend vorhanden und man möchte sich die Arbeit für das Frontend sparen, so ist der Einsatz von LLMs durchaus sinnvoll, solange man genaue Angaben an die Struktur und Anforderungen des Frontends mitgibt. Wichtig ist dabei in kleinen Schritten vorzugehen und klare Anweisungen zu geben.

Ist man in einer Technologie versiert/erfahren, so kann man sich den Einsatz von LLMs durchaus sparen, da der Aufwand den man für die Qualitätssicherung und Fehlerbehebung betreiben muss, den Nutzen übersteigt.

7.5 Zukunft der LLM-gestützten Entwicklung

Kurzfristig: Aus unserer Sicht dienen LLMs hauptsächlich als Unterstützung für Entwickler, nicht als Ersatz (Effizienzsteigerung, Ideenfindung, Dokumentationserstellung).

Mittel- bis langfristig: In naher Zukunft sehen wir LLMs durchaus als hauptsächlichen Entwickler (Code-Generierung) und derzeitige Entwickler später eher als Architekten und Qualitätsmanager.

8 Zusammenfassung und Fazit

8.1 Zusammenfassung der Arbeit

Im Rahmen des Projekts ‚Stack Attack‘ wurde eine Web-Anwendung eines Dinosaurier-Quartettspiels entwickelt. Spieler treten gegen einen KI-Gegner an, der anhand seiner Kartendaten das zu vergleichende Attribut wählt.

8.2 Persönliche Reflexion und Fazit

Die Arbeit an diesem Projekt bot einen wertvollen Einstieg in das Arbeiten und den Umgang mit KI-Agenten und vor allem Erkenntnisse darüber, welche KI-Agenten den einzelnen Teammitgliedern als Unterstützung besser gefallen haben. Im Großen und Ganzen konnten sich die Teammitglieder hierbei darauf einigen, dass GitHub Copilot deutlich effizienter ist und besonders die gestellten Problemstellungen und Aufgaben besser versteht und bearbeitet als Codex, zumindest nach persönlichem Empfinden des Teams.

Des Weiteren war es interessant LLMs auf verschiedene Weisen in das Projekt einfließen zu lassen. Es wurde erstmals mit Modellen des lokalen Frameworks Ollama gearbeitet und dabei festgestellt, dass selbst kleine Modelle wie phi3:mini doch ziemlich viel Rechenleistung benötigen, was die Geschwindigkeit des Modells stark beeinträchtigt. Es war interessant zu sehen, wie lange selbst eine simple Antwort dauert, bei begrenzten lokalen Ressourcen. In einem Quartettspiel ist die Antwortzeit für einen angenehmen Spielfluss wichtig, dieser ist aktuell nicht für einen release geeignet. Wir haben uns dennoch bewusst dafür entschieden, die bestehende Architektur beizubehalten, um einerseits Ressourcen zu schonen, andererseits Erfahrungen mit der Nutzung lokaler LLM-Modelle zu sammeln und zugleich aufzuzeigen wie viel effizienter statischer Code im Vergleich sein kann.

Auch bei der Einbindung eines API-Endpunktes war es schade, dass der Prompt nicht unbegrenzt oft angepasst werden konnte, da die limitierten Tokens zur Bildergenerierung im Hinterkopf behalten werden mussten. Dieses Problem hätte vielleicht umgangen werden können, in dem lokale Modelle aus Ollama verwendet worden wären. Da die Einbindung der

Bildergenerierung allerdings zuerst stattfand und das nicht mit Gedanken an den danach eingefügten KI-Gegner, war diese Option zu dem Zeitpunkt nicht ersichtlich gewesen. Dennoch war es interessant LLMs auf 2 verschiedene Wege einzubauen.