HAW HAMBURG

Masterarbeit

Tom Krause

Entwurf einer Softwarearchitektur zur Integration von Anwendungsfällen in eine Plattform zur Datensynthese

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Tom Krause

Entwurf einer Softwarearchitektur zur Integration von Anwendungsfällen in eine Plattform zur Datensynthese

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Ulrike Steffens
Zweitgutachter: Prof. Dr.-Ing. Lars Hamann

Eingereicht am: 07.02.2024

Tom Krause

Thema der Arbeit

Entwurf einer Softwarearchitektur zur Integration von Anwendungsfällen in eine Plattform zur Datensynthese

Stichworte

Softwarearchitektur, Maschinelles Lernen, Datensynthese

Kurzzusammenfassung

Diese Arbeit betont die Bedeutung der Generierung synthetischer Daten für unzureichende Datensätze, insbesondere im Bereich Künstliche Intelligenz. Die Plattform DaFne, welche in einem Forschungsprojekt entwickelt wird, bietet umfassende Ansätze zur Generierung synthetischer Daten. Die Forschungslücke besteht in der Identifikation und Entwicklung geeigneter Architekturmuster für solche Datensynthese-Plattformen. Ziel dieser Arbeit ist die Entwicklung eines flexiblen Architekturentwurfs, der die Funktionalitäten und Anwendungsfälle von DaFne integriert und für zukünftige Funktionen erweiterbar ist. Dazu wird, basierend auf der ersten Forschungsfrage, die Eignung vorhandener Architekturmuster für Maschinelles Lernen untersucht. Mit der zweiten Forschungsfrage wird anschließend die Entwicklung eines eigenen Entwurfs angestrebt. Durch die Analyse der Architekturmuster kann belegt werden, dass die einzelnen Muster durch ihre individuellen Vorteile eine Grundlage dafür schaffen, eine Plattform zur Datensynthese zu entwickeln. Dabei können gemeinsame Vorteile und Schwachstellen identifiziert werden. Zusätzlich stellt sich heraus, dass ein Microservice-Entwurf mit einem Gateway anzustreben ist. Der resultierende Plattform-Entwurf ermöglicht die nahtlose Integration verschiedener Anwendungsfälle und Funktionalitäten, impliziert jedoch Optimierungspotenziale, die zu berücksichtigen sind.

Tom Krause

Title of Thesis

Design of a software architecture for the integration of use cases into a platform for data synthesis

Keywords

Software Architecture, Machine Learning, Data Synthesis

Abstract

This work emphasises the importance of generating synthetic data for insufficient data sets, especially in the field of artificial intelligence. The DaFne platform, which is being developed in a research project, offers comprehensive approaches for generating synthetic data. The research gap is the identification and development of suitable architectural patterns for such data synthesis platforms. The aim of this work is to develop a flexible architectural design that integrates the functionalities and use cases of DaFne and can be extended for future functions. Based on the first research question, the suitability of existing architecture patterns for machine learning is analysed. The second research question then aims to develop an own design. By analysing the architectural patterns, it can be proven that the individual patterns create a basis for developing a platform for data synthesis due to their individual advantages. Common advantages and weaknesses can be identified. In addition, it turns out that a microservice design with a gateway is desirable. The resulting platform design enables the seamless integration of different use cases and functionalities, but implies optimisation potential that needs to be taken into account.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Die Generierung von synthetischen Daten ist relevant, um Daten in unzureichender Quantität und Qualität zu ergänzen. Diese synthetischen Daten können dann für die Entwicklung und Erforschung von Methoden der Künstlichen Intelligenz für Anwendungsfälle genutzt werden, wie z.B. im Finanzsektor [ADM+20] oder im Kontext von Smart Cities [JYVDS18]. Um die Datensynthese durchführen zu können, wird in [KKZS22] DaFne vorgestellt. DaFne ist eine Plattform, welche verschiedene Ansätze zur Datengenerierung ganzheitlich integrieren kann. Die derzeitige Forschungslücke besteht darin, Architekturmuster zur Entwicklung einer solchen Plattform zur Datensynthese zu identifizieren und zu entwickeln. Der in [KKZS22] vorgestellte Ansatz, inklusive deren Funktionalitäten, ist durch die hier getätigte Arbeit mittels einer geeigneten Softwarearchitektur vollständig konzeptionell abzudecken. Die Architekturvorlagen, welche im Rahmen dieser Arbeit identifiziert und erarbeitet werden, können dann zukünftig für Plattformen des Maschinellen Lernens (ML) mit ähnlichen Anforderungen wiederverwendet werden oder auch für generische ML-Plattformen.

1.2 Ziel der Arbeit

Die vorliegende Arbeit verfolgt das Ziel, einen Architekturentwurf für eine Plattform zur Generierung synthetischer Daten zu entwickeln, der die hier vorgestellte Funktionalitäten und Anwendungsfälle nahtlos integrieren kann. Dabei ist sicherzustellen, dass der entstehende Entwurf erweiterbar ist, um zukünftige Funktionalitäten abzubilden. Hierbei dient das Projekt in [KKZS22] als Ausgangspunkt. Um einen Architekturentwurf für eine Datensynthese-Plattform zu erstellen, werden bestehende Architekturmuster für Maschinelles Lernen vorgestellt und analysiert. Die erste Forschungsfrage lautet dabei wie folgt:

Inwiefern sind die bestehenden Architekturmuster als Referenzarchitekturen für eine Plattform zur Datensynthese geeignet? Daraufhin wird ein eigener Architekturentwurf für die Plattform entwickelt. Durch die Konzeption dieser Plattform wird die zweite Forschungsfrage behandelt: **Wie kann ein Architekturentwurf aussehen, welcher diverse Anwendungsfälle nahtlos integrieren kann?**

1.3 Aufbau der Arbeit

Die Arbeit ist in acht Kapitel unterteilt. In Kapitel 2 erhalten Leser:innen eine Einführung in die Softwarearchitekturen für Maschinelles Lernen, Microservices und in die Funktionsweise der Datensynthese. Anschließend folgt in Kapitel 3 eine Auflistung ähnlicher Arbeiten, die den Entwurf von ML-Plattformen thematisieren. Kapitel 4 charakterisiert die zu entwickelnde Plattform, einschließlich der Funktionalitäten, exemplarischer Anwendungsfälle sowie potenzieller Anwender:innen. Die Konzeption für die entwickelte Softwarearchitektur wird in Kapitel 5 erläutert, indem Anforderungen identifiziert und bestehende ML-Architekturmuster analysiert werden. Zusätzlich werden mögliche Komponenten für den Entwurf identifiziert. Die im Rahmen dieser Arbeit zu entwickelnde Softwarearchitektur wird in Kapitel 6 vorgestellt, inklusive der Integration der Funktionalitäten/Anwendungsfälle. Kapitel 7 untersucht den entwickelten Architekturentwurf hinsichtlich der zuvor definierten Anforderungen. Diese Arbeit schließt mit einem Fazit und einem Ausblick auf zukünftige Forschungsarbeiten in Kapitel 8.

2 Stand der Forschung

Dieses Kapitel widmet sich den Grundlagen von Softwarearchitekturen im Kontext des Maschinellen Lernens, Microservices und der Datensynthese. Im Fokus steht zunächst die Dokumentation von Softwarearchitekturen sowie eine eingehende Analyse der Vor- und Nachteile von Microservices. Zur Konzeption einer Architektur für Maschinelles Lernen werden zusätzlich die spezifischen Herausforderungen dieser Softwarearchitekturen behandelt. Abschließend wird das Konzept der Datensynthese thematisiert, um im weiteren Verlauf der Arbeit einen Entwurf zu entwickeln, der den Prozess der Datensynthese abbilden kann.

2.1 Softwarearchitektur

2.1.1 Allgemein

Softwarearchitekturen können durch diverse Definitionen beschrieben werden. Dabei existieren Merkmale, die jedoch allgemein für alle Softwarearchitekturen gelten. Eine Softwarearchitektur strukturiert ein System und zerlegt es in Teile, welche in Beziehung zueinander stehen. Um das Zusammenspiel dieser Teile zu kennzeichnen, werden Schnittstellen konzipiert [Zör15].

2.1.2 Dokumentation

2.1.2.1 arc42

Zur Dokumentation der hier entwickelten Architektur wird arc42 verwendet, was als Muster zur Architekturdokumentation dient. Dazu ist arc42 in einzelne Sektionen aufgeteilt [SHa]. Zur Dokumentation der Architektur für eine Plattform zur Datensynthese liegt

der Fokus in dieser Arbeit auf der Bausteinsicht und Laufzeitsicht. Zusätzlich werden Entwurfsentscheidungen dokumentiert.

Bausteinsicht

Die Bausteinsicht visualisiert die statische Zerlegung eines Softwaresystems in einzelne Bausteine. Bei den Bausteinen handelt es sich zum Beispiel um Module, Komponenten, Subsysteme, Klassen, Schnittstellen, Pakete. Zusätzlich dokumentiert die Bausteinsicht die Beziehungen der einzelnen Bausteine untereinander. Die unterschiedliche Granularität wird dabei durch Ebenen unterschiedlicher Detaillierung (Level) gewährleistet. Je höher das Level ist, desto detaillierter ist die Darstellung der einzelnen Bausteine [SHb].

Laufzeitsicht

Die Laufzeitsicht visualisiert das Verhalten der einzelnen Bausteine zur Laufzeit. Dadurch können die einzelnen Bestandteile des Systems dabei beobachtet werden, wie beispielsweise wichtige Anwendungsfälle ausgeführt werden oder wie die Interaktion mit Nachbarsystemen realisiert wird [SHc].

2.1.3 Microservices

2.1.3.1 Allgemein

Der Architekturstil Microservices zerlegt ein Softwaresystem in kleinere, eigenständige Teile. Dieser Ansatz verfolgt das Ziel, eine Alternative zu monolithischen Systemen zu bieten, indem einzelne Funktionalitäten des Systems über separate Services bereitgestellt werden [Sta15]. Die einzelnen Services nutzen als Module einzelne Programme, die als eigene Prozesse laufen [Wol18]. Die Modularisierung von Fachlichkeiten über Services wird durch Protokolle realisiert, über die die einzelnen Services kommunizieren. Jeder Service läuft dabei in seiner eigenen Ablaufumgebung. Das Management sowie die Datenverwaltung der einzelnen Services sind dezentral.

2.1.3.2 Vorteile

Microservices bieten diverse Vorteile [Sta15]. Im Allgemeinen stellen Microservices ein geeignetes Konzept zur Modularisierung dar. Bei herkömmlichen Architekturen entstehen oftmals ungewollte Abhängigkeiten, was die Wartbarkeit eines Systems beeinträchtigt.

Durch die klar definierten Schnittstellen, welche die Kommunikation zwischen Microservices realisieren, sind weniger ungewollte Abhängigkeiten in Software-Systemen zu finden. Weitere Vorteile von Microservices bestehen in der Flexibilität hinsichtlich der Technologieauswahl sowie eine Reduktion ungewollter Abhängigkeiten zwischen einzelnen Services. Des Weiteren können Microservices leichter ersetzt werden. Dies ist der Fall bei Services, welche dieselbe Schnittstelle anbieten ohne dabei jegliche Technologie zu übernehmen. Diese Eigenschaft ermöglicht, zusammen mit dem Konzept der Modularisierung, eine nachhaltige Software-Entwicklung, da sich Services zum Beispiel durch ihre Technologiefreiheit und Ersetzbarkeit dafür eignen, alte Bestandteile aus Systemen leichter zu entfernen. Folglich kann die Zerlegung eines Systems in kleine Einheiten eine erhöhte Wartbarkeit und Flexibilität gewährleisten. Die Geschwindigkeit in der Entwicklung wird dabei erhöht. Dies ist der Fall, da einzelne Teams unabhängig und parallel an Funktionalitäten arbeiten können. Dadurch können einzelne Services schneller in die Produktion gebracht werden. Ein weiterer Vorteil besteht in der Skalierbarkeit von Microservices. Einzelne Services können unabhängig skaliert werden, ohne das Gesamtsystem zu skalieren [Wol18]. Durch ihre Eigenschaften eignen sich Microservices zusätzlich zur Umsetzung von Continuous Delivery [Wol16], wodurch Code mithilfe von Automatisierung schneller veröffentlicht werden kann, da einzelne Services unabhängig voneinander bereitgestellt werden können [Wol18].

2.1.3.3 Herausforderungen

Die Herausforderungen bezüglich Microservices sind bei dem Entwurf der hier zu entwickelnden Software-Architektur zu berücksichtigen. Eine Herausforderung besteht in versteckten Beziehungen, welche zwischen Services existieren können. Die Umstrukturierung der Architektur ist herausfordernd, wodurch die Verschiebung von Funktionalitäten schwierig ist. Des Weiteren kann der Betrieb eines Systems, welches aus vielen einzelnen Microservices besteht, komplex sein. Das liegt daran, dass die Services bereitgestellt, überwacht und betrieben werden müssen. Die notwendigen Anforderungen für die Infrastruktur sind dabei zu berücksichtigen. Zusätzlich handelt es sich bei einem System aus Microservices um ein verteiltes System. Folglich können Aufrufe zwischen Services zu Überlastungen eines Netzwerks führen, was eine geringere Effizienz implizieren kann [Wol18]. Weitere Herausforderungen sind die erhöhte Komplexität beim Testen der einzelnen Module. Zusätzlich kann es schwierig sein, eine korrekte Interaktion der Services zur Laufzeit zu garantieren. Des Weiteren besteht eine Herausforderung darin, die Konsistenz

der Daten zu gewährleisten, wenn jeder Service nur einen geringen Teil der Gesamtdaten verwaltet [Sta15].

2.1.4 Softwarearchitektur für Maschinelles Lernen

2.1.4.1 Herausforderungen

Allgemein weisen Softwaresysteme für Maschinelles Lernen Merkmale auf, die in herkömmlichen Anwendungen nicht zu finden sind. Diese sind beim Entwurf und bei der Implementation dieser Systeme zu berücksichtigen [SHG+14, Tri18, WUKG19]. Diese Merkmale implizieren Herausforderungen für Entwickler:innen, welche sich kategorisieren lassen. In der Kategorie des Entwurfs existieren diverse Herausforderungen. ML-Anwendungen benötigen eine überdurchschnittliche Anzahl von Daten, was bei dem Entwurf als möglicher Engpass berücksichtigt werden muss. Dies ist der Fall, da das Systemverhalten durch die hohe Datenverarbeitung unvorhersehbar sein kann. Des Weiteren sollte der Entwurf der ML-Systeme flexibel sein, um den Änderungen von Algorithmen und Bibliotheken gerecht zu werden [RRK+19]. Beim Entwurf der ML-Anwendungen treten allgemein diverse Herausforderungen im Zusammenhang mit der Datenverarbeitung, Abhängigkeiten, Modellen, Infrastruktur, Werkzeugen, Algorithmen auf. Im Folgenden wird genauer auf diese individuellen Herausforderungen eingegangen.

Wie bereits beschrieben, stellt der Umgang mit hohen Datenmengen eine Herausforderung dar. Diese sind unerlässlich für ML-Systeme, da ML-Anwendungen nur mittels Daten lernen können. Der herausfordernde Umgang mit Daten bezieht sich dabei auf das Erfassen, Zugreifen, Sammeln, Bereinigen sowie Transformieren der Daten [ABB+19, BJS19, CPM+20, FMBO20, FHI+20, HBEB16, KZDB17, LRB+19, SAM+17].

Weitere Herausforderungen sind mit Modellen identifizierbar. Dazu gehören die Erstellung des Modells wie das Trainieren, Evaluieren und Bereitstellen der Modelle. Dies kann daran liegen, dass die Entwicklung dieser Modelle oftmals mehrere Iterationen und Anpassungen benötigen [ABB+19, PQW+20]. Besonders herausfordernd sind dabei das Verständnis der Modelle hinsichtlich des Verhaltens und der Struktur [ABB+19, ABCFB18, HBEB16, XQMZ19, ZYF+19] sowie die Wiederverwendung von Modellen (Anpassung der Modelle für individuelle Probleme) [ZGM+19, BJS19].

Des Weiteren stellen Abhängigkeiten eine Herausforderung dar, welche beim Entwurf von ML-Systemen zu berücksichtigen sind. Wie in herkömmlichen Softwaresystemen

sollten eine hohe Kohäsion (Bindung innerhalb einer Komponente) sowie eine geringe Kopplung (Bindung zwischen Komponenten) beim Entwurf erreicht werden [Lar12]. Dies ist beim Entwurf von ML-Systemen komplex. Zwar ist es möglich, Daten und das zugehörige Verhalten modular zu kapseln, jedoch weisen diese Module Abhängigkeiten zu externen Daten/Modulen auf [SHG+15]. Die mögliche Konsequenz besteht darin, dass sich Änderungen im Modell negativ auf abhängige Komponenten auswirken können [BVC19, HBEB16, LRB+19].

Der Umgang mit Werkzeugen und Infrastruktur zur Entwicklung der ML-Systeme stellen Entwickler:innen vor weiteren Herausforderungen. Dies liegt an der Vielzahl unterschiedlicher inkompatibler Werkzeuge, was zu Diskrepanzen in der Entwicklung führen kann [KZDB17, ZXZ+20]. Folglich kann es sinnvoll sein, eine geeignete Infrastruktur mit benötigter Software zu schaffen [ABB+19]. Dies kann durch Kapselung der Software, zum Beispiel durch Docker [Inca], realisiert werden. Aufgrund des zuvor beschriebenen iterativen Charakters von Modellen sollte die Infrastruktur, durch Eigenschaften wie Skalierbarkeit und Flexibilität, das Experimentieren mit Modellen ermöglichen [ABCFB18, GWRL19, LRB+19].

2.2 Datensynthese

2.2.1 Allgemein

Bei der Datensynthese handelt es sich um einen Vorgang, bei dem künstliche Daten generiert werden. Dieser Vorgang zeichnet sich dabei durch verschiedene Datentypen aus, wie z.B. reale Daten, aus denen dann neue (synthetische) Daten generiert werden können. Hierbei wird ein Modell entwickelt, um die Struktur und Beziehungen der realen Daten zu lernen und daraufhin synthetische Daten erzeugen zu können [EEMH20]. Dieser Vorgang ist in Abbildung 2.1 dargestellt. Synthetische Daten werden in vielen Anwendungsdomänen eingesetzt, zum Beispiel in der Bioinformatik [CHBJ+18, KAK+17], im Natural Language Processing [SHB16, XWL+17] oder im Bereich Smart Cities [AZM15].

2.2.1.1 Generative Adversarial Networks

Generative Adversarial Networks stellen einen Ansatz dar, mit dem mittels Maschinellen Lernens Strukturen und Muster eines Datensatzes erkannt werden können, um in einem

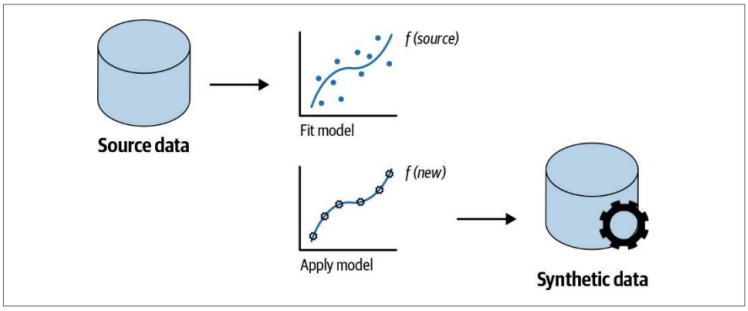


Abbildung 2.1: Der Prozess der Datensynthese. Quelle: [EEMH20]

weiteren Schritt synthetische Daten generieren zu können. Die Generative Adversarial Networks bestehen dabei aus zwei konkurrierenden neuronalen Netzen, dem Generator und dem Diskriminator. Der Generator erstellt die Daten, welche dann vom Diskriminator als reale Daten oder künstliche Daten klassifiziert werden müssen. Das Ziel besteht darin, synthetische Daten zu erstellen, welche der Diskriminator nicht von den echten Daten unterscheiden kann [GPAM+20]. Eine Visualisierung für den methodischen Ansatz von Generative Adversarial Networks ist in Abbildung 2.2 dargestellt. Dabei ist zu erwähnen, dass der Generator keinen direkten Zugriff auf reale Ursprungsdaten besitzt, sondern durch die Interaktion mit dem Diskriminator lernt. Der Diskriminator hat direkten Zugriff auf die synthetischen Abbildungen sowie auf die realen Daten [CWD+18]. Die generativen Modelle, mit denen neue Daten synthetisiert werden können, eignen sich für verschiedene Anwendungsbereiche. Mögliche Anwendungsbereiche sind zum einen die Datenerweiterung [BSD+17], semantische Bildbearbeitung [ZKSE16] sowie die Klassifikation von Trainingsdaten [RMC15]. In der vorliegenden Arbeit dienen Generative Adversarial Networks als Grundlage für die Datensynthese, welche in die Plattform zu integrieren sind.

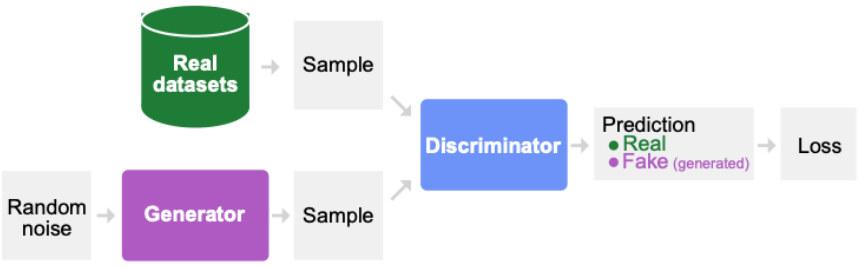


Abbildung 2.2: Der methodische Ablauf eines Generative Adversarial Network. Quelle: [KTC+18]

3 Ähnliche Arbeiten

Dieses Kapitel legt den Fokus auf Arbeiten, welche eine Plattform für das Maschinelle Lernen konzipieren. Dabei handelt es sich ebenfalls um Arbeiten, welche über den Schritt der Konzeption hinausgehen und durch diverse Anwendungsfälle in der Praxis getestet sind.

Markov et al. [MWK+22] stellen allgemeine Prinzipien sowie eine Architektur einer Plattform für das Maschinelle Lernen vor. Die Plattform, namens Looper, integriert Schnittstellen zur Entscheidungsfindung sowie für Feedback für Nutzer:innen. Dabei wird der gesamte Lebenszyklus des Maschinellen Lernens von der Datensammlung bis zur Ableitung von Maßnahmen durch Looper abgebildet. Im Jahr 2021 konnten mit Looper bereits 440 bis 1.000 ML-Modelle verwaltet und zwischen 4 bis 6 Millionen Echtzeitentscheidungen getroffen werden.

Ein Vergleich von zwei End-to-End-ML-Plattformen findet in [MAG+23] statt. Dieser Vergleich dient dazu, langfristige Ziele für die Entwicklung einer neuen End-to-End-ML-Plattform zu definieren, bei der eine ML-Automatisierung skalierbar ist und eine geeignete Systemintegration gegeben ist.

Ein Ansatz, bei dem die Orchestrierung der einzelnen ML -Komponenten fokussiert wird, ist in [BBC+17] zu finden. Der dort konzipierte Ansatz TensorFlow Extended (TFX) ist eine Plattform für das Maschinelle Lernen, welche auf TensorFlow basiert und dabei Komponenten für folgende Aufgaben abdeckt: Datenanalyse, Datentransformation, Datenvalidierung, Modellerstellung, Modelltraining, Modellevaluation und Modellbereitstellung. Die Plattform TFX, welche bei Google [Incc] implementiert ist, kann durch die Integration dieser einzelnen Aufgaben mittels Komponeten in eine Plattform wesentliche Vorteile erzielen. Bei einer Fallstudie konnte dies belegt werden, da sich schnellere Produktionen (z.B. von Applikationen), Verringerung von benutzerdefiniertem Code sowie schnellere Installationen von Applikationen festellen lassen. Mögliche Ursachen dafür sind effizientere Datenanalysen und Modellanalysen durch TFX.

Flux [SZC+17] realisiert mittels einer Microservice-Architektur einen Ansatz zur beschleunigten Überwachung und Ausführung von ML-Modellen. Ziel dabei ist es, Modelle in das Produktionssystem eines Unternehmens zu integrieren. Flux ist dabei in der Lage, Modelle im Batch-Modus zu verarbeiten oder als einzelne Services für Anwendungsfälle in Echtzeit bereitzustellen.

Agrawal et al. [AAB+19] haben ein Datenverwaltungssystem namens MLdp entwickelt. Der Vorteil liegt im Vergleich zu herkömmlichen ML-Systemen darin, dass MLdp ein flexibles Datenmodell für jegliche Art von Daten anbietet. Des Weiteren ermöglicht MLdp eine etablierte Versionsverwaltung und damit eine Reproduzierbarkeit von Experimenten des Maschinellen Lernens. Zusätzlich sind herkömmliche ML-Bibliotheken integriert. Weitere Vorteile sind effiziente Iterationen durch ML-Experimente.

In [RNGS19] wird eine Plattform namens Overton entwickelt mit der primären Aufgabe, maschinelle Lernsysteme in der Überwachung innerhalb der Produktion zu unterstützen. Overton automatisiert dabei den Zyklus der Modellerstellung,- bereitstellung und -überwachung. Dies wird durch Abstraktionsprinzipien erreicht. Somit besteht das Ziel von Overton darin, dass Entwickler:innen sich mit Maschinellem Lernen auf einer höheren Ebene befassen können, um beispielsweise Anwendungen zu erstellen ohne den Programmcode zu schreiben. Anwendungen, welche auf Overton basieren, haben bereits Billionen von Datensätzen verarbeitet und dabei die Fehlerquote verglichen mit Produktionssystemen stark minimiert.

In [HDB] ist eine Plattform zu finden, welche von Uber [Ince] entwickelt und genutzt wird. Die Plattform mit dem Namen Michelangelo ist dabei in der Lage, den gesamten ML-Lebensyklus abzudecken und maschinelle Lernlösungen in der Größenordnung von Uber anzubieten. Michelangelo dient Uber dazu, Vorhersagen für die stark genutzen Online-Dienste (z.B. Vorhersage von Lieferzeiten) zu entwickeln. Konzeptionell besteht Michelangelo aus Open-Source-Komponenten und Komponenten, welche selbst entwickelt sind. Bei den Open-Source-Komponenten handelt es sich um Apache Hadoop [Foua], Apache Spark [Fouc] inklusive der Bibliothek MLlib [Foud], Apache Samza [Foub], XGBoost [Incb] und Tensorflow [Incd].

4 Charakterisierung der Plattform

Um potenzielle Anforderungen an eine Plattform zur Datensynthese zu definieren, werden in diesem Abschnitt die Plattform und ihre grundlegenden Funktionalitäten sowie Anwendungsfälle beschrieben, welche im Plattform-Entwurf zu berücksichtigen sind. Mögliche Nutzer:innen der Plattform werden dafür zusätzlich identifiziert.

4.1 Funktionalitäten

Die grundlegenden Überlegungen bezüglich der Plattform-Funktionalitäten stammen aus [KKZS22] und sind in Abbildung 4.1 zusammengefasst. Das Ziel besteht darin, Nutzer:innen diverse Funktionalitäten zu bieten, um eine Datensynthese für tabellarische Daten durchführen zu können. Dabei soll der gesamte Prozess der Datensynthese abgebildet werden. Folglich ist es möglich, Input-Daten selbst hochzuladen oder auszuwählen. Mögliche Daten sind z.B. öffentliche Daten aus dem Kontext Smart Cities sowie Daten, welche von der Plattform angeboten werden. Im nächsten Schritt bietet die Plattform vielfältige Möglichkeiten zur Generierung synthetischer Daten sowie die Evaluation synthetischer/realer Daten. Die generierten Daten können dann verschiedenen Anwender:innen zur Verfügung gestellt werden. Zusätzlich ist es möglich, bereits trainierte Modelle des Maschinellen Lernens zu speichern, um diese dann später für weitere Generierungen zu nutzen.

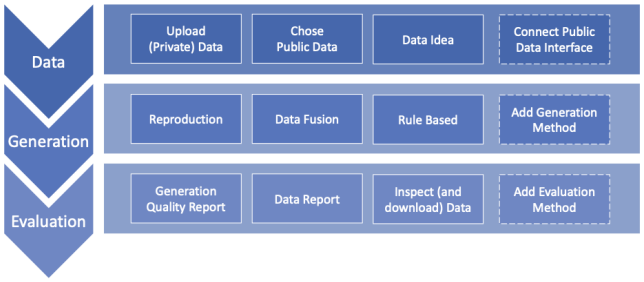
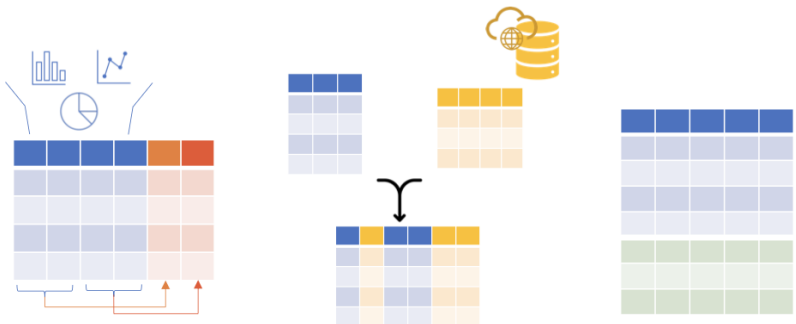


Abbildung 4.1: Übersicht der Plattform-Funktionalitäten. Quelle: [KKZS22]

Die Plattform ist in der Lage, diverse Datentypen (z.B. Geodaten, Zeitstempel, etc.) durch ein Datenmodell abzudecken, welche im Kontext von Smart Cities nützlich sein können. Hinzu kommt, dass es autorisierten Nutzer:innen möglich ist, vorhandene Daten durch öffentliche Daten zu erweitern. Für diesen Vorgang sind geeignete Schnittstellen zu integrieren. Die Methoden zur Datengenerierung, welche durch die Plattform angeboten werden und auch zukünftig kombinierbar sein sollen, sind in Abbildung 4.2 visualisiert. Diese werden im weiteren Verlauf genauer erläutert.



(a) Rule Based (b) Fusion (c) Reproduction

Abbildung 4.2: Methoden zur Datengenerierung durch die Plattform. Quelle: [KKZS22]

a) **Regelbasierte Datengenerierung**:

Die regelbasierte Datengenerierung dient dazu, sowohl eigene Spalten mit ausgewählten Datentypen, Verteilungen und Merkmalswerten (Abbildung 4.2a in blau) zu definieren und Spalten mit bestimmten Merkmalen zu generieren (abhängig von anderen Spalten, visualisiert in orange und rot). Ein Zugriff auf Input-Daten ist dabei nicht notwendig. Des Weiteren kann die regelbasierte Datengenerierung

auf bestehende Spalten im Datensatz angewandt werden, um diesen durch weitere Spalten zu erweitern.

b) **Datenfusion**:

Die Datenfusion kann mehrere Datensätze miteinander verbinden. Dies ist in Abbildung 4.2b zu sehen. Ein Szenario kann darin bestehen, einen eigenen Datensatz (blau) mit öffentlichen Daten (gelb) zu kombinieren. Folglich kann dadurch der Informationsgehalt eines ersten Datensatzes mit einem weiteren Datensatz erweitert werden. Als Ausprägung der Datenfusion wird im weiteren Verlauf ein Anwendungsfall betrachtet, bei dem tabellarische Brückendaten mit Wetterdaten angereichert werden. Somit werden zu den Spalten der Brückendaten zusätzliche Wetter-Informationen hinzugefügt. Domänen-Expert:innen können dadurch genaure Informationen über den Zustand einer Brücke durch Wetterdaten erhalten.

b) **Reproduktion von Daten**:

In Abbildung 4.2c ist die letze Variante zur Generierung von Daten zu sehen, die Reproduktion von Daten. Dabei können zu einem bestehenden Datensatz (blau) neue Zeilen (grün) hinzugefügt werden, die hinsichtlich der Merkmale und Beziehungen zwischen den Spalten den ursprünglichen Daten ähneln.

4.2 Exemplarische Anwendungsfälle

Im weiteren Verlauf werden exemplarische Anwendungsfälle, die zusätzlich zu den Funktionalitäten durch die Plattform realisierbar sein sollen, dargestellt.

4.2.1 Generierung von synthetischen Nachbarschaften

Die allgemeine Idee der Generierung von synthetischen Nachbarschaften besteht darin, die Modellierung von Städtedaten mit Hilfe Künstlicher Intelligenz zu ermöglichen. Diese Modellierung basiert auf der Erzeugung synthetischer tabellenähnlicher Daten (DataFrames). In diesem Fall können die Nutzer:innen künstliche Nachbarschaften auf der Grundlage eines leeren Layouts und eines anschließend trainierten maschinellen Lernmodells erzeugen. Dies ist in Abbildung 4.3 zu sehen. Darüber hinaus kann diese allgemeine Idee

der Modellierung von Städten oder Stadtvierteln verwendet werden, um andere Anwendungsfälle zu definieren, die für verschiedene Interessengruppen nützlich wären. Durch die Generierung synthetischer Nachbarschaften können beispielsweise Unternehmer:innen entscheiden, welches Gebäude in einem Stadtviertel, basierend auf der Umgebung, am besten für die Umsetzung ihrer Geschäftsidee geeignet ist. Ein Beispiel dafür sind hoch frequentierte Fußwege, an denen Unternehmer:innen ein Café eröffnen könnten [Vas].

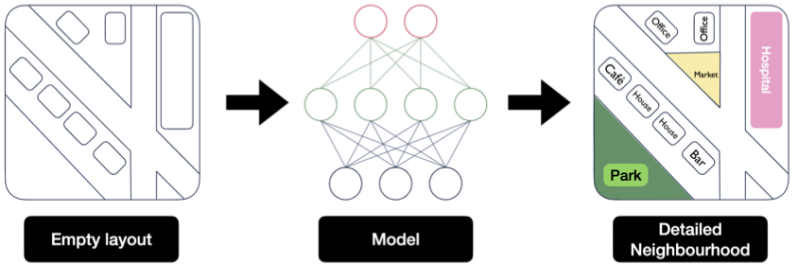


Abbildung 4.3: Generierung von synthetischen Nachbarschaften. Quelle: [Vas]

4.2.2 Generierung synthetischer Fußgängerrouten

In [GN22] wird der letzte zu betrachtende Anwendungsfall vorgestellt, der in die Plattform zur Datensynthese zu integrieren ist. Das grundlegende Vorhaben besteht darin, Daten mithilfe von agentenbasierten Simulationen in Verbindung mit verstärktem Lernen zu generieren. Die Integration eines verstärkenden Lern-Rahmenwerks auf Basis von Multi-Agenten-Systemen ermöglicht es, das Mobilitätsverhalten von Bürger:innen in urbanen Umgebungen zu erfassen und zu verstehen. Der Algorithmus, welcher dazu verwendet wird, ist in Abbildung 4.4 schematisch abgebildet. Dabei werden mögliche Aktionen eines Agenten als Input-Datei gespeichert, welche der Agent dann ausführen kann. Der Agent agiert mit seiner Umgebung und erhält eine Belohnung von seiner Umgebung. Das Ergebnis, welches die beste Aktion eines Agenten beinhaltet, wird dann in einer CSV-Datei gespeichert.

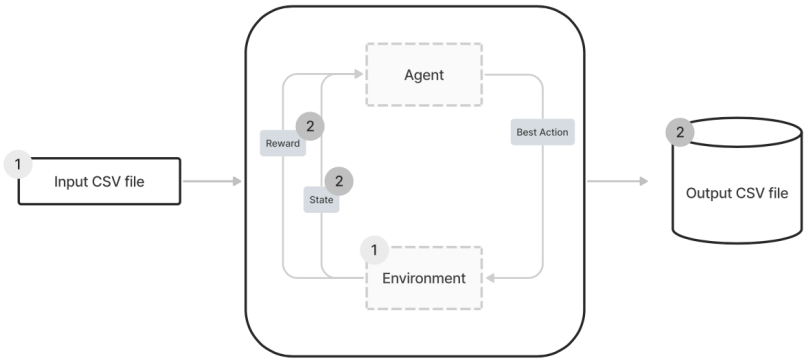


Abbildung 4.4: Algorithmus des verstärkten Lernens für Simulationen von Fußgängerrouten. Quelle: [GN22]

4.3 Benutzerrollen

Für die zuvor beschriebenen Funktionalitäten und Anwendungsfälle werden folgende Benutzerrollen identifiziert:

- **Domänen-Expert:innen**: Nutzer:innen, die Daten in einem Anwendungskontext benötigen, um beispielsweise Simulationen auszuwerten, aber keine oder nur geringe Kenntnisse über Maschinelles Lernen oder Datengenerierung haben.
- **Datenwissenschaftler:innen**: Diese Nutzer:innen benötigen Daten für Anwendungsfälle der Künstlichen Intelligenz, haben aber gewisse Kenntnisse über ML-Algorithmen und deren Parametrisierung.
- **Daten-Ingenieur:innen**: Benutzer:innen, die befugt sind, neue Datenquellen zu integrieren oder zu verbinden.
- **Entwickler:innen von generativen ML-Modellen**: Benutzer:innen, die berechtigt sind, die Plattform durch Hinzufügen neuer generativer Modelle zu erweitern.
- **Entwickler:innen von Evaluierungsmethoden**: Autorisierte Benutzer:innen, welche die Auswertungsmethoden für generierte Daten erweitern.
- **Administrator:innen**: Administrator:innen sind verantwortlich für die Autorisierung von Benutzern, das Löschen von Daten, Modellen und Methoden und haben dabei Zugriff auf alle Dienste der Plattform.

Die zu entwickelnde Plattform umfasst zusammenfassend eine breite Palette von Benutzerrollen. Die vielfältigen Rollen, die über verschiedene Organisationen verteilt sind, betonen die Notwendigkeit einer integrativen Architektur. Die Herausforderungen dabei liegen nicht nur in der technischen Integration der Funktionalitäten/Anwendungsfälle, sondern auch in der Zusammenarbeit unterschiedlicher Fachkenntnisse der einzelnen Anwender:innen. Die zu entwickelnde Architektur muss folglich flexibel sein, um den Anforderungen verschiedener Nutzerprofile sowie der Integration einzelner Funktionalitäten gerecht zu werden.

5 Konzeption der Architektur

5.1 Anforderungsanalyse

Dieses Kapitel liefert einen Überblick über die Anforderungen an den Architekturentwurf für eine Plattform zur Datensynthese. Die Anforderungen werden dabei in nichtfunktionale und funktionale Anforderungen unterteilt. Die formulierten Anforderungen stützen sich auf Erkenntnisse aus den Herausforderungen von Softwarearchitekturen für Maschinelles Lernen. Darüber hinaus werden sie durch selbst definierte Anforderungen ergänzt, die auf der Charakterisierung der Plattform in Kapitel 4 basieren.

Bei den nichtfunktionalen Anforderungen handelt es sich um grundlegende Anforderungen, welche bei der Konzeption der Architektur zu berücksichtigen sind. Diese Anforderungen zielen darauf ab, eine zukünftige Implementation des Systems zu erleichtern, die Anpassbarkeit, Skalierbarkeit und Wartbarkeit des Systems zu verbessern.

Die funktionalen Anforderungen sind den Funktionalitäten und Anwendungsfällen, die durch die Plattform abgedeckt werden, zuzuordnen. Fokussiert wird dabei der Vorgang der synthetischen Datengenerierung und somit das Maschinelle Lernen.

5.1.1 Nichtfunktionale Anforderungen

1.) Skalierbarkeit:

Die Softwarearchitektur sollte skalierbar sein, um die Kapazitäten der Plattform für Nachfragen/Lasten anpassen zu können. Folglich sollte die Plattform mit eine/-r Nutzer/-in sowie mit mehreren Nutzer:innen zufriedenstellend funktionieren und Spitzen sowie Einbrüche im Lastverhalten bewältigen. Des Weiteren kann es notwendig sein, einzelne Funktionen innerhalb der Plattform zu skalieren, welche stärker nachgefragt sind, um die Nachfrage nach einer Anwendungsfunktion zu decken.

2.) Erweiterbarkeit:

Die Architektur sollte so entwickelt werden, dass sie das Hinzufügen weiterer Algorithmen zur Datengenerierung nahtlos ermöglicht. Dies dient auch dazu, mögliche weitere Anwendungsfälle in der Zukunft durch die Plattform realisieren zu können.

3.) Flexibilität:

Die Flexibilität der Architektur bezieht sich sowohl auf verschiedene Nutzerrollen, die unterschiedliche Kenntnisse hinsichtlich der Parametrisierung von Verfahren zur Datensynthese haben als auf verschiedene Technologien zur Datensynthese, die integriert werden können. Nutzer:innen sollen, basierend auf den Kenntnissen, persönliche Bereiche innerhalb der Plattform zur Verfügung gestellt werden. Des Weiteren soll der Entwurf flexibel sein, um Hinzufügen oder Entfernen von Komponenten, wie z.B. ML-Bibliotheken, gerecht zu werden.

4.) Überwachung:

Eine weitere Anforderung an die Plattformarchitektur ist die Möglichkeit der Protokollierung. Damit soll Transparenz über die Zustände der einzelnen Dienste, z.B. Auslastung oder Trainingszustände von Modellen und deren Ergebnissen, geschaffen werden.

5.) Sicherheit:

Um die Plattform vor externen Manipulationen oder Angriffen zu schützen, ist Sicherheit eine weitere Anforderung. Dies betrifft die Authentifizierung der Nutzer:innen sowie die Arbeit mit den Daten selbst (z.B. bei der Datensynthese oder Speicherung der Daten). Folglich sollen nur autorisierte Nutzer:innen Zugriff zur Plattform erhalten.

6.) Wartbarkeit:

Der Architektur-Entwurf sollte so konzipiert sein, dass die Wartbarkeit für Entwickler:innen angstrebt wird, um beispielsweise auf Fehler im System schnell reagieren zu können. Somit ist ein Architekturmuster zu wählen, was sich durch Wartbarkeit auszeichnet.

5.1.2 Funktionale Anforderungen

7.) Vermeidung von Engpässen:

Die Architektur muss so konzipiert sein, dass mögliche Engpässe bei der Datenverarbeitung bezüglich der Grenzen für Rechenleistung berücksichtigt werden. Dabei sollten z.B. Überlegungen getroffen werden, ob Begrenzungen hinsichtlich der Datenmenge für die Datensynthese integriert werden.

8.) Integration von ML-Komponenten:

Komponenten, die das Maschinelle Lernen ermöglichen, sind über geeignete Schnittstellen zu integrieren. Eine einheitliche Integrationsstrategie sollte dabei verfolgt werden.

9.) Trainingszustand der Modelle:

Durch die Architektur und die zukünftige Implementation sollten Anwender:innen der Plattform Informationen über den Zustand der Modelle haben. Dies soll eine Transparenz im Vorgang der Datengenerierung schaffen.

10.) Modellmanagement:

Durch den Architektur-Entwurf soll der komplette Vorgang des Modellmanagements abgebildet werden können. Dazu gehören folgende Bestandteile: Erstellung, Training, Evaluation, Bereitstellung des Modells.

11.) Wiederverwendung:

Bereits trainierte Modelle des Maschinellen Lernens sollten wiederverwendbar sein und somit auf der Plattform gespeichert werden. Dies ist nützlich, da Modelle mit einer hohen Performanz und überdurchschnitlichen Kennzahlen für neue Datensätze genutzt werden könnten.

12.) Datenvielfalt:

Unterschiedliche tabellarische Datentypen müssen durch die Plattform integriert werden können. Die Vorverarbeitung dieser Daten soll in den jeweiligen Modellen stattfinden.

13.) Datenqualität:

Eine Datenqualität ist im Vorgang der Datensynthese, basierend auf statistischen Metriken, anzustreben. Dies bezieht sich zunächst auf die Qualität möglicher Input-Daten sowie auf die Durchführung eines Qualitätstests für synthetisch generierte Daten.

14.) Generierung eines Berichts:

Diese Anforderung dient der zuvor beschriebenen angestrebten Datenqualität. Nach einer erfolgreichen Datengenerierung sollten Nutzer:innen der Plattform einen Bericht über die synthetischen Daten (z.B. Datenverteilung, Metriken etc.) erhalten.

15.) Metadatenschema:

Ein Metadatenschema soll Anwender:innen der Plattform zur Verfügung gestellt werden, um mehr Informationen über einzelne Datensätze zu gewinnen. Dabei kann es sich z.B. um Metadaten der einzelnen ML-Modelle handeln, welche zur Datensynthese verwendet werden.

5.2 Musteranalyse

In diesem Kapitel werden existierende Softwarearchitekturen für Maschinelles Lernen vorgestellt und anschließend auf ihre Eignung als Referenzarchitekturen für den Entwurf einer Plattform zur Datensynthese untersucht.

Um eine Analyse der folgenden Architekturmuster zu tätigen, dienen die in Kapitel 5.1 identifizierten nichtfunktionalen und funktionalen Anforderungen. Basierend auf diesen Anforderungen werden Begründungen geliefert, warum diese Muster als Referenzarchitekturen ausgewählt werden. Gleichzeitig sind dabei potenzielle Schwachstellen der Architekturmuster zu erläutern, um Optimierungspotenziale im eigenen Entwurf berücksichtigen zu können. In der folgenden Analyse werden nicht alle einzelnen Anforderungen im Detail behandelt. Stattdessen liegt der Fokus auf den Anforderungen, welche durch die Vorstellung der Muster in der zugehörigen Literatur beantwortet werden können.

5.2.1 Architekturmuster für Maschinelles Lernen

Data Lake

Das erste Muster ist der Data Lake, welches in diversen Quellen thematisiert wird, unter anderem in [Gol16]. Das Ziel besteht dabei darin, Daten im Rohformat auf unbestimmte Zeit zu sammeln und diese Daten dann in Echtzeit zu verarbeiten. Die Daten können dabei aus vielfältigen Quellen stammen und unterschiedliche Formate aufweisen. Der Vorteil durch Einbindung von Data Lakes besteht zum Beispiel darin, dass die veschiedenen Daten parallel gelesen werden können und der Zugang zu Maschinellem Lernen gewährleistet werden kann [Gol16]. Im Gegensatz zu einem Data Warehouse müssen die diversen Daten nicht erst strukturiert werden, um dann im nächsten Schritt abgespeichert zu werden. Der Data Lake bietet jedoch ebenfalls die Vorteile eines Data Warehouse [KG]. Ein Vorteil ist dabei die effiziente Verarbeitung von Datenabfragen [MB00]. Der Engpass, welcher durch die vorzeitige Strukturierung der Daten im Data Warehouse entsteht, wird beim Data Lake vermieden, was durch den Verzicht auf Datentransformationen realisiert wird [KG]. Die Abbildung 5.1 zeigt die Struktur eines Data Lake mit den zuvor beschriebenen Eigenschaften. Dort ist ebenfalls zu sehen, dass die Daten in verschiedenen Formaten (unstrukturiert/strukturiert) gespeichert sind, um anschließend von Analyseanwendungen genutzt zu werden.

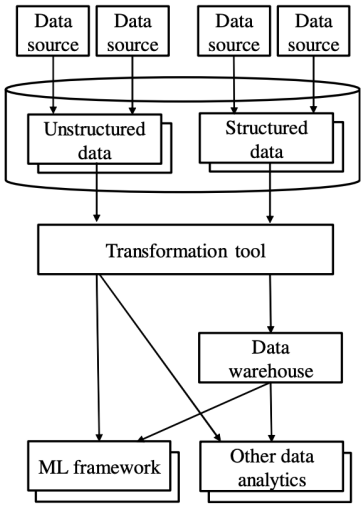


Abbildung 5.1: Data Lake - Architektur. Quelle: [KG]

Der Data Lake ist dazu in der Lage, stukturierte/unstrukturierte Daten zu verarbeiten ohne dass ein Schema vordefiniert werden muss [Gol16]. Somit ist Anforderung 12

erfüllt. Dies kann für den Entwurf einer Plattform zur Datensynthese vorteilhaft sein, da diverse Daten im Rohzustand genutzt und später bei Bedarf verarbeitet werden. Des Weiteren können dadurch in Zukunft weitere Anwendungsfälle, ohne Beschränkungen auf bestimmte Datenstrukturen, durch eine potenzielle Plattform realisiert werden. Die Abbildung 5.1 zeigt, dass bei der Konzeption dieser Architektur bereits ML-Komponenten berücksichtigt werden (Anforderung 8, Anforderung 10). Folglich ist der Data Lake als Grundlage für die Datenverarbeitung von Input-Daten geeignet, um im nächsten Schritt ML-Verfahren (z.B. Datensynthese) auf diese Daten anzuwenden. Die Prinzipien der Skalierbarkeit sind bei Data Lakes gewährleistet. Data Lakes können unterschiedlich skaliert werden. Eine Möglichkeit ist die vertikale Skalierung, indem mehr CPU, RAM oder Speicher zur Datenverarbeitung aufgerüstet werden sowie die horizontale Skalierung, bei der mehr Server oder Knoten hinzugefügt werden [Mat17]. Damit ist die Anforderung 1 ebenfalls erfüllt.

Die Wartbarkeit hinsichtlich wachsender Datenmengen kann bei Data Lakes komplex werden, insbesondere bei der Sicherstellung der Datenqualität [TLT23]. Die Erfüllung der Anforderung 6 stellt sich somit als herausfordernd dar. Ein Grund dafür ist das in Kapitel 5.1 beschriebene Metadatenschema und dessen Verwaltung (Anforderung 15). Dafür wird ein standardisiertes Vorgehen benötigt [NZM+19]. Die Vorstellung des Musters in [Gol16] impliziert weitere Herausforderungen für die funktionalen Anforderungen. Ein Beispiel dafür sind ML-Komponenten, die zwar in der Architektur berücksichtigt werden, jedoch werden diese nicht genauer beschrieben. Das genaue Vorgehen zum Trainieren großer Datenmengen durch die Architektur wird nicht thematisiert. Somit wird auch nicht klar inwiefern weitere Anforderungen, wie z.B. die Wiederverwendung von Modellen, erfüllt werden kann.

Das Muster des Data Lake bietet zusammenfassend eine solide Grundlage für ML-Anwendungen zur Verarbeitung großer Datenmengen. Dennoch bedarf es spezifischer Anpassungen für einzelne Anwendungsfälle. Diese Anpassungen sollten nicht nur die reine Datenvorverarbeitung berücksichtigen, sondern auch einen Fokus auf die Anwendung von Verfahren des Maschinellen Lernens legen, die in einem eigenen Architekturentwurf thematisiert werden müssen.

Trennung der Geschäftslogik von ML-Modellen

Dieses Architekturmuster stammt aus [Yok19]. Das Ziel des Musters besteht darin, die allgemeine Geschäftslogik einer ML-Anwendung von den tatsächlichen ML-Modellen zu trennen. Das Problem besteht oftmals darin, dass die Geschäftslogik von den Ergebnissen

der Modelle abhängt. Die ML-Modelle können aus diversen Gründen fehlschlagen, was negative Folgen für die Geschäftslogik impliziert. Folglich ist es sinnvoll, die Modelle des Maschinellen Lernens von der tatsächlichen Geschäftslogik zu trennen. Dadurch können Modelle geändert werden ohne Auswirkungen auf die Geschäftslogik. Visualisiert wird der Ansatz in Abbildung 5.2. Die einzelnen Komponenten sind einzelnen Ebenen zugeordnet (Präsentationsebene, Logikebene, Datenebene). Zusätzlich werden die Komponenten zur Datenverarbeitung (unterer Teil des Bildes) den ML-Komponenten zugeordnet. Dadurch wird eine klare Trennung zwischen der Geschäftslogik und den dazugehörigen ML-Anwendungen realisiert [Yok19].

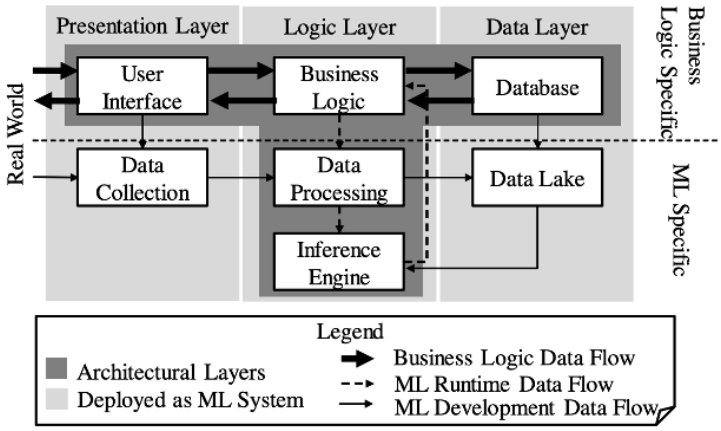


Abbildung 5.2: Trennung der Geschäftslogik von ML-Modellen. Quelle: [Yok19]

Das Muster ist aus diversen Gründen als Referenzarchitektur geeignet. Der erste allgemeine Vorteil besteht in der Trennung der Logik des Anwendungsfalls von ML-Modellen. Dadurch ermöglicht dieses Muster die Änderungen von Modellen ohne Auswirkungen auf die Logik des individuellen Anwendungsfalls. Um die gewünschte Transparenz in Trainingsphasen der ML-Modelle zu erhalten, wird ein Logging integriert. Dadurch ist die Anforderung 4 erfüllt. Große Datenmengen können durch die Integration eines Data Lake verarbeitet werden (Anforderung 12), wodurch das Muster hinsichtlich der Datenverarbeitung ebenfalls eine Skalierbarkeit impliziert (Anforderung 1). Modelle, die bereits trainiert sind, sind wiederverwendbar. Das Muster stellt eine Grundlage für ML-Systeme dar, da bereits ML-Komponenten im Entwurf integriert sind, wie z.B. die Inference Engine (Anforderung 8). Dies erleichtert das geforderte Modellmanagement aus Anforderung 10, da Komponenten des Maschinellen Lernens nahtlos integriert werden. Zusätzlich ist das Muster so konzipiert, dass klar definierte Schnittstellen zwischen ML-Komponenten sowie herkömmlichen Komponenten vorgesehen sind. Dies erhöht die Wartbarkeit, da z.B.

Änderungen an Komponenten durchgeführt werden können ohne die bestehende Funktionalität zu gefährden (Anforderung 6). Die 3-Schichten-Architektur deckt zusätzliche Eigenschaften, wie z.B. die Integration einer Benutzeroberfläche ab, um Nutzerfreundlichkeit zu gewährleisten [Yok19].

Das vorliegende Muster weist dennoch bezüglich der Anforderungen aus Kapitel 5.1 einige Schwachstellen auf, die genauer betrachtet werden sollten. Nicht alle Anforderungen können auf Erfüllung überprüft werden. Erstens werden mögliche Einschränkungen für Datenformate nicht beschrieben, was zu Unsicherheiten der Handhabung einzelner Datenformate (Input-Daten) führen kann. Dies stellt eine Schwachstelle dar, wenn detaillierte Analysen und Berichterstattungen über die Leistung und Qualität der Modelle, z.B. für eine Datensynthese, erforderlich sind (Anforderung 13). Ein weiterer Punkt, der nicht ausreichend thematisiert wird, ist ein mögliches Metadatenschema der Input-Daten (Anforderung 15). Dasselbe gilt für die restlichen Anforderungen aus Kapitel 5.1.

Zusammenfassend stellt das Muster eine solide Grundlage für die Entwicklung von ML-Anwendungen dar. Um jedoch eine Vielzahl von Anwendungsfällen umsetzen zu können, ist es erforderlich, die zuvor identifizierten Schwachstellen zu adressieren und zusätzliche funktionale Anforderungen zu berücksichtigen. Das aktuelle Muster allein berücksichtigt diese Anforderungen nicht, es bietet jedoch die Möglichkeit, durch die Integration zusätzlicher Komponenten entsprechend erweitert zu werden.

Lambda-Architektur für Maschinelles Lernen

Die Lambda-Architektur für Maschinelles Lernen wird in [LMNVGDB20] vorgestellt. Das Ziel der Lambda-Architektur besteht darin, ML-Awendungen in Echtzeit anzuwenden ohne auf Genauigkeit der Vorhersagen zu verzichten. Weitere Treiber sind dabei die Gewährleistung einer hohen Vorhersagegeschwindigkeit, Datenvalidierung sowie eine Konsistenz der Vorhersageergebnisse. In traditionellen ML-Anwendungen werden Daten nach dem ETL-Prinzip verarbeitet. Das heißt, dass die Daten extrahiert, transformiert und dann in einen Speicher (Data Warehouse) geladen werden [LMNVGDB20]. In herkömmlichen Anwendungen wird oftmals Online Transactional Processing (OLTP) genutzt, um anschließend Online Analytical Processing (OLAP) mit Daten durchzuführen [ESHEB11]. OLTP dient dazu, Daten zu aggregieren, um diese Daten dann anschließend mit OLAP zu analysieren [Pla09]. Das Problem bei diesem Vorgang besteht darin, dass Verzögerungen im OLTP auftreten können und somit erst zeitversetzte Analysen mit OLAP möglich sind. Folglich ist die Echtzeitanwendung nicht immer gewährleistet. Die Lösung besteht darin, die herkömmliche Lambda-Architektur zu übernehmen und in diese Architektur

ML-Modelle im Speed Layer und im Batch Layer zu integrieren (Abbildung 5.3). Das ML-Modell im Batch Layer ist verantwortlich für die Verarbeitung großer Datensätze und das Modell im Speed Layer realisiert die Erzeugung einer Vorhersage in Echtzeit. Im Speed Layer wird der ETL-Prozess für Datenströme durchgeführt, um dann ein Modell trainieren zu können. Anschließend werden die Ergebnisse dann über den Serving Layer bereitgestellt. Im Batch Layer werden die Input-Daten verarbeitet und dann für den Data Lake oder das Data Warehouse kategorisiert. Mit einem Schwellenwert wird dann bestimmt, ob eine Menge an Input-Daten erreicht ist oder eine bestimmte Zeit, um dann einen ML-Prozess für die Daten zu starten. Durch diese Kombination der herkömmlichen Lambda-Architektur und der ML-Komponenten kann den Treibern dieser Architektur entsprochen werden.

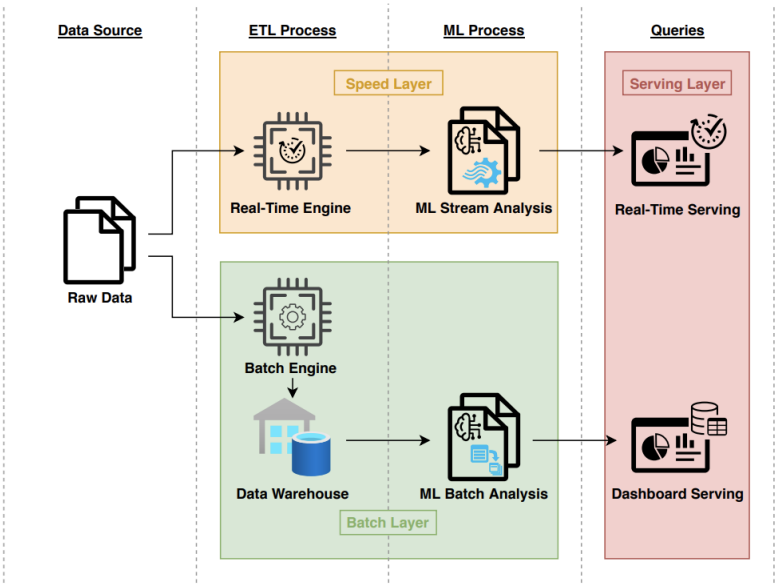


Abbildung 5.3: Lambda-Architektur für Maschinelles Lernen. Quelle: [LMNVGDB20]

Die Lambda-Architektur für Maschinelles Lernen fungiert als relevante Referenzarchitektur, insbesondere im Kontext von ML-Anwendungen in Echtzeit. Die Integration der herkömmlichen Lambda-Architektur und zusätzlicher ML-Prozesse ermöglicht die Realisierung von Echtzeitanalysen im Rahmen des Maschinellen Lernens. Ein wesentlicher Vorteil besteht dabei in der Flexibilität, indem zwei separate ML-Modelle in jeder Schicht der Architektur eingesetzt werden. Die Skalierbarkeit der einzelnen Schichten der Lambda-Architektur stellt ein weiteres Argument dar. Die unabhängige Skalierbarkeit des Batch- und Speed-Layer ermöglicht eine effiziente Ressourcennutzung und die Bewältigung von

Lastspitzen (Anforderung 1) [LMNVGDB20]. Die Flexibilität beinflusst auch die Datenverarbeitung und die technologische Vielfalt innerhalb der Schichten, wodurch unterschiedliche Datenformate (Anforderung 12) und Technologien integriert werden können (Anforderung 3). Zusätzlich ist eine einfache Implementierung von Aktualisierungen oder Verbesserungen ohne Beeinträchtigung der Echtzeitanalysen möglich [FM]. Die Integration von ML-Prozessen in dieser Architektur, einschließlich der jeweiligen Komponenten, bietet einen weiteren Mehrwert. Dies ermöglicht die effektive Nutzung von Maschinellem Lernen in Echtzeit und erleichtert das Modellmanagement durch die Integration von ML-Komponenten (Anforderung 8, Anforderung 10). Darüber hinaus führt die Aufrechterhaltung des Batch-Layer zu einem größeren Datendurchsatz, was bei umfangreichen Datenmengen und komplexen Analysen von Vorteil ist [LMNVGDB20].

Wie bereits in Kapitel 5.2.1 beschrieben, ist dieses Muster so konzipiert, dass die herkömmliche Lambda-Architektur um insgesamt zwei ML-Modelle, ein Modell im Batch-Layer sowie ein Modell im Speed-Layer, erweitert wird. Bei der herkömmlichen Lamdba-Architektur existieren diverse Herausforderungen. Eine Herausforderung besteht darin, Dateninkonsistenzen zwischen den beiden Schichten zu vermeiden, da die Synchronisation der beiden Schichten komplex ist. Somit ist die Sicherstellung der Datenqualität herausfordernd (Anforderung 13). Eine weitere Konsequenz, welche aus der Trennung zwischen Batch- und Speed-Layer resultiert, ist der erhöhte Wartungsaufwand innerhalb der Gesamtarchitektur, da Verarbeitungen in beiden Schichten verarbeitet werden und Veränderungen somit in beiden Schichten koordiniert werden müssen. Die Implementation der Lambda-Architektur erfordert somit eine hohe Expertise [JM17]. Durch die Erweiterung der herkömmlichen Lambda-Architektur um ML-Modelle werden die zuvor beschriebenen Herausforderungen, wie z.B. der Wartungsaufwand sowie die Koordination beider Schichten, verstärkt.

Die Lambda-Architektur für Maschinelles Lernen dient primär als Konzept zur Datenverarbeitung. Das vorliegende Architekturmuster bietet jedoch, durch das Hinzufügen von ML-Prozessen, eine Grundlage für ML-Anwendungen. Weitere funktionale Anforderungen, zur Konzeption einer Plattform für die Datensynthese, müssten zusätzlich bei der Anwendung des Musters berücksichtigt werden.

Kappa-Architektur für Maschinelles Lernen

Die Kappa-Architektur für Maschinelles Lernen, dargestellt in Abbildung 5.4, wird in [PKG] beschrieben. Diese Architektur strebt ein Ziel an, das dem der Lambda-Architektur für Maschinelles Lernen ähnelt. Dieses Ziel besteht darin, genaue Vorhersagen in Echtzeit durch ML-Anwendungen zu realisieren. Der Unterschied bei der Kappa-Architektur für Maschinelles Lernen besteht im Gegensatz zur Lambda-Architektur darin, die Komplexität der Pipeline weiter zu reduzieren, wenn die Vorhersage in Echtzeit im Gegensatz zur Datenanalyse priorisiert wird. Eine Schwäche der zuvor vorgestellten Lambda-Architektur ist es, dass nach Abschluss des Prozesses im Batch Layer und der Fusion mit den Vorhersageergebnissen Inkonsistenzen auftreten können, welche dann an den Serving Layer übergeben werden. Des Weiteren müssen im Entwurf und der Wartung der Lambda-Architektur unterschiedliche Werkzeuge genutzt werden, was zu einer höheren Komplexität führt. Folglich kann es schwierig sein, neue Funktionalitäten hinzuzufügen sowie Fehler zu beheben ohne Beeinträchtigungen für die gesamte Architektur zu verursachen. Ein Lösungsansatz liefert dafür die Kappa-Architektur für Maschinelles Lernen. Diese Architektur reduziert die Komplexität, indem der Batch Layer entfernt wird. Dadurch wird nur der ETL-Prozess sowie die Analyse durch ML-Komponenten im Speed Layer beibehalten. Hierbei werden die Daten nicht im Batch Layer verarbeitet, sondern im Speed Layer. Dabei sollte ein Prozess etabliert werden, wie zum Beispiel das Sammeln aller Input-Daten in der zeitlichen Reihenfolge, um die Daten dann anschließend nochmal im Speed Layer zu aktualisieren. Am Ende werden die Daten dann im Serving-Layer aktualisiert und zur Verfügung gestellt.

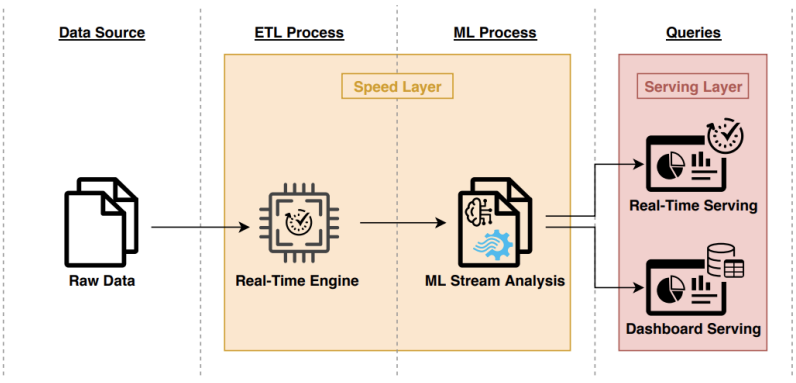


Abbildung 5.4: Kappa-Architektur für Maschinelles Lernen. Quelle: [PKG]

Die Konstruktion dieses Musters ähnelt der zuvor vorgestellten Lambda-Architektur. Die grundlegende Kappa-Architektur wird um ML-Komponenten erweitert (Anforderung 8).

Die Vorteile der Kappa-Architektur für Maschinelles Lernen sind analog zu den Vorteilen der Lambda-Architektur mit dem Unterschied, dass die Kappa-Architektur keinen Batch-Layer besitzt. Folglich ist die Komplexität der Datenverarbeitung geringer und die Wartbarkeit höher [PKG]. Dadurch können die Anforderung 6 sowie die Anforderung 14 von der Kappa-Architektur unterstützt werden.

Wie bei herkömmlichen Systemen zur Verarbeitung von Datenströmen, liegt auch bei der Kappa-Architektur eine Herausforderung darin, Strategien zur Sicherstellung der Datenqualität (Anforderung 13) zu definieren und anzuwenden [DL20]. Eine Ursache dafür ist z.B. die Echtzeitverarbeitung, bei der ebenfalls Inkosistenzen der Daten in Echtzeit behandelt werden müssen. Zwar weist die Kappa-Architektur eine geringere Komplexität als die Lambda-Architektur auf, jedoch kann es aufgrund der Echtzeitverarbeitung herausfordernd sein, eine historische Datenverarbeitung zu realisieren [Lip23].

Die Kappa-Architektur für Maschinelles Lernen dient zusammenfassend vorrangig zur Datenverarbeitung. Das vorliegende Architekturmuster bietet jedoch, durch das Hinzufügen von ML-Prozessen, eine Grundlage für ML-Anwendungen. Funktionale Anforderungen, z.B. für die Synthetisierung von Daten, müssten durch eine Anpassung des Musters integriert werden.

Gateway-Routing-Architektur für Maschinelles Lernen

Das Problem in diversen ML-Systemen besteht in der engen Kopplung zwischen dem Front-End-Client, der Geschäftslogik und den ML-Komponenten. Dies impliziert eine geringe Flexibilität sowie Skalierbarkeit, da Änderungen nur komplex umzusetzen sind. Des Weiteren kann es schwierig sein für einzelne Services Endpunkte einzurichten, wenn der Client verschiedene Services nutzt. Ein Lösungsansatz für diese Probleme ist die Gateway-Routing-Architektur (Abbildung 5.5). Diese verfolgt das Ziel, eine enge Kopplung zwischen Komponenten für das Maschinelle Lernen und der Geschäftslogik zu minimieren. Ermöglicht wird das durch die Installation eines Gateway, welches Anfragen an entsprechende Services weiterleitet. Die Geschäftslogik wird durch einen Service realisiert, der nach außen hin als Schnittstelle dient. Die Integration eines Gateway in ML-Anwendungen impliziert dabei diverse Vorteile. Durch die lose Kopplung zwischen Clients und Services sind die Details der Services während der Nutzung irrelevant. Des Weiteren wird eine bessere Benutzerfreundlichkeit erreicht, da der Client mehrere ML-Services nutzen kann ohne die Endpunkte zu verwalten. Das Gateway kann zusätzlich als einziger Zugangspunkt sowie Kontrollpunkt zur Authentifizierung innerhalb des Systems genutzt werden, wodurch die Sicherheit erhöht wird [Yok19].

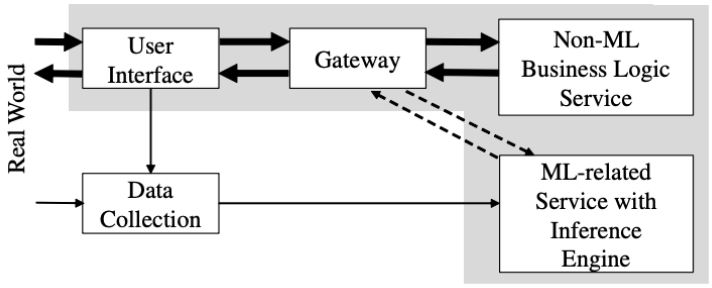


Abbildung 5.5: Gateway-Routing-Architektur für Maschinelles Lernen. Quelle: [Yok19]

Das vorliegende Architekturmuster beinhaltet diverse Vorteile. Die Integration einer Machine Learning Engine steht im Zentrum dieses Entwurfs, wodurch das Modellmanagement durch die Integration benötigter Komponenten in das Gesamtsystem eingebunden werden kann. Dadurch können die Anforderungen 8 und 10 erfüllt werden. Durch die Integration eines Gateway zeichnet sich die Architektur durch ihre Skalierbarkeit aus, indem eine lose Kopplung von Services und Clients ermöglicht wird. Die Skalierbarkeit wird dadurch unterstützt, indem über das Gateway Anfragen auf mehrere Services verteilt werden können (Anforderung 1). Die lockere Kopplung von Services und Clients fördert zusätzlich die Erweiterbarkeit des Systems, wodurch neue Services nahtlos integriert werden können (Anforderung 2). Ein weiterer Aspekt ist die Sicherheit, indem Gateways als Kontrollpunkte für Authentifizierung dienen, wodurch die Anforderung 5 unterstützt wird. Das Architekturmuster zeichnet sich durch Modularität aus, die durch die Nutzung von Services erreicht wird. Diese modulare Struktur erleichtert die Containerisierung und ermöglicht ebenfalls Skalierbarkeit einzelner Komponenten. Ein weiteres Merkmal ist die Berücksichtigung der Datenvielfalt, die beispielsweise durch die Integration eines Data Lake ermöglicht wird [Yok19]. Folglich können diverse Datentypen mit diesem Architekturmuster verarbeitet werden (Anforderung 12). Diese Vielfalt an Datenquellen impliziert Möglichkeiten für umfassende Analysen und gewährleistet die Integration diverser Anwendungsfälle.

Die vorliegende Architektur impliziert ebenfalls Herausforderungen. Auch bei diesem Muster sind funktionale Anforderungen schwer zu überprüfen, da Details zum Maschinellen Lernen nicht beschrieben werden. Ein Beispiel ist die Generierung eines Analyse-Berichts. Ein weiteres Beispiel ist die Gewährleistung von Datenqualität der Input-Daten (Anforderung 13). Dasselbe gilt für ein mögliches Metadatenschema, um eine einheitli-

che Verwaltung von Metadaten zu ermöglichen (Anforderung 15). Durch die modulare Struktur könnten diese aber zukünftig integriert werden.

Durch das integrierte Gateway und den modularen Aufbau kann das Muster als Basis dazu dienen, eine Plattform zur Datensynthese zu entwickeln. Weitere funktionale Anforderungen, die zur Realisierung der Synthetisierung von Daten notwendig sind, müssen bei der Nutzung des Musters berücksichtigt werden.

Microservice-Architektur für Maschinelles Lernen

Herkömmliche Anwendungen für Maschinelles Lernen bestehen aus verschiedenen Frameworks, die unabhängig voneinander entwickelt werden. Folglich sind Frameworks für Maschinelles Lernen, inklusive der Schnittstellen, heterogen. Die Nutzung und Integration dieser Frameworks gestaltet sich somit komplex, da die einzelnen Frameworks nicht kompatibel sein können oder nicht lokal ausführbar sind. Ein Lösungsansatz dafür wäre die Bereitstellung der Frameworks über Microservices (dargestellt in Abbildung 5.6). Dies impliziert eine bessere Zugänglichkeit, da Microservices über simple Protokolle zugänglich sind. Zusätzlich können Anwender:innen von ML-Systemen, z.B. Data Scientists, die Systeme nutzen ohne die Frameworks zu installieren oder zu warten. Hinzu kommt, dass Microservices von diversen, konkurrierenden Anwendungen genutzt werden können [WUKG20].

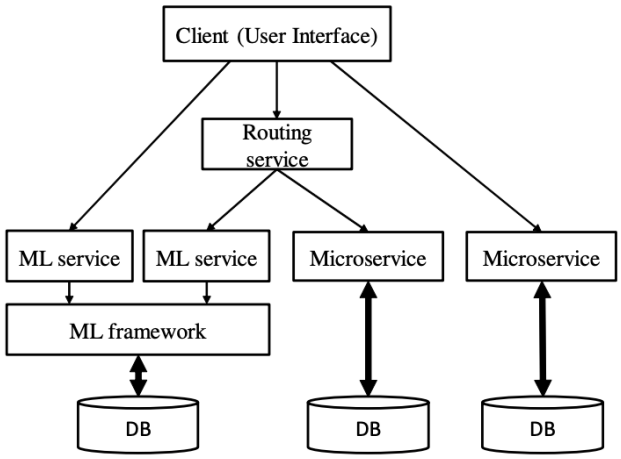


Abbildung 5.6: Microservice-Architektur für Maschinelles Lernen. Quelle: [WUKG20]

Grundsätzlich gelten die Vorteile analog zu den Vorteilen von Microservices aus Kapitel 2.1.3.2. Da diese Microservice-Architektur um ML-Services erweitert wird, sind weitere

Vorteile identifizierbar. Ein Vorteil besteht darin, dass zunächst Verfahren des Maschinellen Lernens integriert werden. Dadurch können Modelle verwaltet werden. Folglich unterstützt das Muster die Anforderungen 8 und 10. Die Microservice-Architektur für Maschinelles Lernen integriert zusätzlich einen Service, welcher Routing ermöglicht, was die Erweiterbarkeit fördert, da eine Integration weiterer Services sowie deren Kommunikation untereinander ermöglicht wird (Anforderung 2). Des Weiteren werden in dieser architektonischen Konzeption heterogene ML-Bibliotheken als Microservices implementiert. Dies ermöglicht eine verbesserte Zugänglichkeit und eine unkomplizierte Kommunikation durch klare und definierte Schnittstellen zwischen den einzelnen Services. Ein weiterer Vorteil dieses Ansatzes besteht darin, dass die ML-Rahmenwerke nicht lokal installiert werden müssen, was die Nutzung dieser Services vereinfacht und flexibler gestaltet (Anforderung 3). Ein weiterer Aspekt besteht darin, dass die implementierten ML-Services nicht nur als Schnittstellen zu den verschiedenen Rahmenwerken fungieren, sondern auch eigenständige Module bereitstellen, die bestimmte ML-Modelle oder Algorithmen kapseln. Dies ermöglicht eine bessere Integration von ML in verschiedene Anwendungen, indem die Komplexität der zugrunde liegenden Frameworks abstrahiert wird [WUKG20]. Durch die Nutzung eines Gateway kann diese Architektur dazu dienen, diverse Backends zu integrieren und somit verschiedene Anwendungsfälle abbilden. Des Weiteren ist durch die Nutzung von Services eine Containerisierung gewährleistet.

Dieses Muster ist auf den grundlegenden Prinzipien von Microservices aufgebaut. Folglich gelten analog die Herausforderungen, die im Abschnitt 2.1.3.3 beschrieben werden. Diese sind bei dem Entwurf einer Plattform zur Datensynthese zu berücksichtigen. Um die gewünschten Funktionalitäten (Kapitel 4.1) zu realisieren und den funktionalen Anforderungen gemäß Kapitel 5.1 zu entsprechen, ist die Implementierung entsprechender Services erforderlich. Dies schließt beispielsweise Evaluierungsdienste ein, die zur Analyse künstlich generierter Daten auf der Plattform dienen. Ebenso sind Strategien zu integrieren, die das Metadatenschema umsetzen.

Die Microservice-Architektur für Maschinelles Lernen impliziert die meisten Vorteile, da sie einerseits auf der grundlegenden Idee von Microservices basiert und zusätzlich ML-Services integriert, welche die Umsetzung der Datensynthese ermöglichen können. Die Umsetzung weiterer funktionaler Anforderungen ist durch die Integration einzelner Services zu realisieren.

Zwischenfazit

Auf Basis der Musteranalysen kann die erste Forschungsfrage beantwortet werden. Die Untersuchung der präsentierten Muster verdeutlicht, dass bereits Ansätze existieren, die eine geeignete Grundlage für die Konzeption einer Datensynthese-Plattform schaffen. Dafür sind folgende Argumente festzuhalten: Die einzelnen Muster bieten, wie zuvor beschrieben, individuelle Vorteile. Dabei stellt sich heraus, dass folgende Muster für die reine Datenverarbeitung geeignet sind: Data Lake, Lambda-Architektur für Maschinelles Lernen und Kappa-Architektur für Maschinelles Lernen. Die Anforderungen aus Kapitel 5.1 können bei der Analyse aller Muster bereits teilweise erfüllt werden. Auffällig ist, dass alle analysierten Muster die Anforderungen 1,8,10 und 12 unterstützen. Folglich sind die Muster skalierbar, integrieren ML-Komponenten und ermöglichen ein Modellmanagement. Zusätzlich unterstützen die vorliegenden Muster eine Datenvielfalt hinsichtlich der Datenverarbeitung. Des Weiteren sind gemeinsame Schwachstellen der Muster aufgefallen. Bei keinem der Muster können Informationen geliefert werden, ob z.B. die Anforderungen 13 (Datenqualität) oder 15 (Metadatenschema) erfüllt werden können. Eine weitere Gemeinsamkeit, die sich zwischen den Mustern feststellen lässt, ist der Sicherheitsaspekt, der bei keinem Muster explizit behandelt wird, außer bei der Gateway-Routing-Architektur. Zudem ist bei allen Mustern zu beachten, dass Entwickler:innen eine Strategie zur Gewährleistung der Datenqualität anwenden müssen, um hochwertige ML-Analysen durchzuführen. Die Erfüllung weiterer Anforderungen kann dabei bei allen Mustern nicht anhand der Beschreibung dieser Muster beantwortet werden. Dies ist im weiteren Verlauf bei einem eigenen Architekturentwurf zu berücksichtigen.

Wie bereits in der Einleitung beschrieben, besteht das Ziel dieser Arbeit darin, einen Architekturentwurf zu konzipieren, der Funktionalitäten/Anwendungsfälle integrieren kann und auch zukünftig erweiterbar ist. Für den weiteren Verlauf wird sich deshalb zunächst dagegen entschieden, den Fokus auf folgende Architekturen zu legen: Data Lake, Lambda-Architektur für Maschinelles Lernen und Kappa-Architektur für Maschinelles Lernen. Ein Grund dafür ist der reine Fokus auf die Datenverarbeitung dieser Architekturmuster. Des Weiteren kann die Integration von Kappa oder Lambda, aufgrund der zusätzlichen Komplexität, Skalierungsherausforderungen implizieren.

Zusammenfassend zeigt die Analyse der Muster, dass die Microservices-Architektur für Maschinelles Lernen die meisten positiven Auffälligkeiten aufweisen kann. Dies ist der Fall, da neben den herkömmlichen Vorteilen von Microservices (Kapitel 2.1.3.3) zusätzliche ML-Services integriert werden. Die Erfüllung weiterer funktionaler Anforderungen,

wie z.B. das Modellmanagement, kann durch die Integration der notwendigen Services realisiert werden. Die Vorteile der Microservices werden im weiteren Verlauf durch die Integration eines Gateway ergänzt, analog zu der Gateway-Routing-Architektur für Maschinelles Lernen, um zusätzlich ein Sicherheitsverfahren durch eine Autorisierung zu unterstützen. Somit dient eine Microservice-Architektur mit einem integrierten Gateway als Referenzarchitektur für eine Datensynthese-Plattform. Die beiden kombinierten Muster eignen sich besonders zur Integration von Funktionalitäten in eine Plattform. Die Gründe dafür sind die zuvor beschriebenen Eigenschaften der Skalierbarkeit, Flexibilität, Modularität, Erweiterbarkeit und Sicherheit. Dabei ist eine nahtlose Integration neuer Services in die Architektur möglich, um den Großteil der Anforderungen durch den Entwurf zu erfüllen.

5.3 Identifikation von Komponenten

Aufgrund der zuvor analysierten Muster wird eine Softwarearchitektur konzipiert, die auf Services basiert. Diese Services bestehen aus Komponenten, welche im weiteren Verlauf beschrieben werden.

Auf Basis der in Kapitel 4.1 vorgestellten Funktionalitäten und Anwendungsfällen (Kapitel 4.2) und den damit einhergehenden Anforderungen aus Kapitel 5 werden im weiteren Verlauf Komponenten identifiziert. Diese Komponenten werden kategorisch unterschieden. Bei den Kernkomponenten handelt es sich um die Komponenten, welche zur Umsetzung der reinen Plattform-Funktionalitäten (Reproduktion, Fusion, regelbasierte Datengenerierung) dienen. Die anwendungsspezifischen Komponenten dienen zur Umsetzung der exemplarischen Anwendungsfälle.

5.3.1 Kernkomponenten

Komponenten für die regelbasierte Datengenerierung

Die grundlegenden Überlegungen zur Aufteilung in Komponenten und deren Funktionalitäten stammen aus [BgK23]. Folgende Komponenten realisieren den Service der regelbasierten Datengenerierung:

Komponente	**Funktionalität**
TableViewInterface	Repräsentiert die Benutzeroberfläche zur Ansicht von Tabellendaten.
SchemaViewInterface	Die Benutzeroberfläche zur Schema-basierten Datengenerierung.
ColumnManager	Bereitstellung aller Spalten inklusive Schemaoperationen.
ColumnUpload	Ermöglicht das Hinzufügen einer Spalte.
TableUpload	Ermöglicht das Hinzufügen aller Spalten einer Tabelle.
GeneratorEngine	Durchführung der Datengenerierung.
RuleManager	Bereitstellung der verfügbaren Generations-Regeln.
FileManager	Bereitstellung sämtlicher Dateien.
FileUpload	Realisiert das Hochladen einer Datei.
FileDownload	Realisiert das Herunterladen einer Datei.

Tabelle 5.1: Identifikation & Beschreibung der Komponenten zur regelbasierten Datengenerierung.

Komponenten für die Datenfusion

Die Komponentenaufteilung und ihre Funktionalitäten für den Service der Datenfusion werden hierbei in Anlehnung an [AM23] betrachtet.

Komponente	Funktionalität
API	Implementierung von Endpunkten zur Anfragenverarbeitung/Abrufen von Ergebnissen/Anfragen.
Application	Implementation der Anwendungsfälle sowie Koordination der Komponenten-Interaktionen.
State	Verwaltung von Anfragebearbeitungszuständen; Speicherung von Ergebnissen; Aktualisierungen der Zustände während der Anfrageverarbeitung.
CSV-Processing	Lesen/Schreiben von CSV-Dateien; Extraktion von Daten aus CSV-Dateien; Konvertierung zwischen CSV-Format und internem Datenformat.
Fusion	Durchführung der Datenfusion.
External Data	Integration einer externen Schnittstelle zur Abfrage & Konvertierung fremder Daten in ein internes Datenformat.

Tabelle 5.2: Identifikation & Beschreibung der Komponenten zur Datenfusion.

Komponenten für die Reproduktion

Die hier vorgestellten Komponenten sind in ihrer ursprünglichen Form in [KKZS22] beschrieben und in [Kra23] modifiziert. Der Service für die Reproduktion setzt sich aus folgenden Kompnenten zusammen:

Komponente	Funktionalität
GraphicalUserInterface	Interaktion der Nutzer:innen mit der Anwendung; Parametrisierung der Algorithmen; Einsicht der Generierungsergebnisse.
API-Gateway	Authentifizierung und Eintrittspunkt in diverse Backend-Services.
DataManagement	Datenverwaltung inklusive Datenintegration, Datenharmonisierung, Metadatenmanagement.
GenerationOrchestrator	Ausführung der Services nach Reihenfolge & Abhängigkeiten, Koordination der Generierungsschritte mittels der hier beschriebenen Komponenten (inklusive Datenmanagement, Modellmanagement etc.).
ModelManagement	Modellverwaltung inklusive hinzufügen, parametrisieren, entwickeln, speichern und trainieren der Modelle.
GenerativeModel	Modell, welches die Datengenerierung durchführt.
EvaluationManager	Verwaltung der Evaluations-Methoden inklusive hinzufügen von Methoden; Einsicht der evaluierten Generierungsergebnisse.
EvaluationMethod	Repräsentiert die Evaluations-Methode basierend auf statistischen Metriken.

Tabelle 5.3: Identifikation & Beschreibung der Komponenten zur Reproduktion.

5.3.2 Anwendungsspezifische Komponenten

Komponenten zur Generierung von synthetischen Nachbarschaften

Die folgenden Komponenten für den Service zur Generierung synthetischer Nachbarschaften inklusive möglicher Überlegung zur Integration in eine Plattformarchitektur stammen aus [Kra23].

Komponente	Funktionalität
NeighbourhoodGenerationOrchestrator	Start des Generierungsprozesses; Bereitstellung der Generierungsergebnisse; Bereitstellung bereits trainierter ML-Modelle.
GraphModelManager	Verwaltung der bereits trainierten Modelle; Training neuer Modelle; Ausgabe der bereits trainierten/neu trainierten Modelle.
NeighbourhoodGenerator	Erhalt des Input-Graphen; Erstellung einer Liste von Knoten & Vorhersage zur Nutzung der einzelnen Knoten.
GraphConverter	Erhalt der Input-Daten; Erzeugung eines Adjazengraphen mit Gebäude als Knoten; Erzeugung eines Graphen als Ausgabe.
LabelSelector	Erhalt der Knotenliste aus dem Neighbourhood Generator; Erzeugung von Beschriftungen der einzelnen Knoten für die passende Kategorie.
NodeFinder	Erzeugung einer Liste mit Kategorie-Werten für jeden Knoten.
NeighbourhoodGenerationDataManager	Verwaltung der Ergebnisse.

Tabelle 5.4: Identifikation & Beschreibung der Komponenten zur Generierung synthetischer Nachbarschaften.

Komponenten zur Generierung von synthetischen Fußgängerrouten

Als Grundlage zur Identfikation notwendiger Komponenten dient der Anwendungsfall, welcher in [GN22] konzipiert wird.

Komponente	Funktionalität
InputDataHandler	Speicherung der möglichen Aktionen eines Agenten in der Umgebung als CSV-Datei.
Controller	Kontrolle & Steuerung der Interaktion zwischen dem Agent und dem Environment; Schnittstelle für Input-Daten sowie Output-Daten.
Agent	Erstellung & Speicherung des Agenten inklusive der möglichen Aktionen.
Environment	Erstellung der Umgebung & Speicherung des Systemzustands, der Belohnung für den Agenten sowie Informationen über mögliche Aktionen für den Agenten .
OutputDataHandler	Speicherung der Belohnungen sowie der Folgezustände als CSV-Datei.

Tabelle 5.5: Identifikation & Beschreibung der Komponenten zur Generierung synthetischer Fußgängerrouten.

6 Modellierung der Architektur

Im folgenden Verlauf erfolgt eine detaillierte Betrachtung der Komponenten sowie ihrer gegenseitigen Interaktionen. Anschließend sind die Komponenten nahtlos in einem Architekturentwurf zu integrieren. Diese Integration wird durch die Nutzung unterschiedlicher Sichten genauer beschrieben, die in Kapitel 2.1.2 ausführlich erläutert werden.

6.1 Komponenten & deren Interaktionen

6.1.1 Regelbasierte Datengenerierung

Die Komponenten, die für die Umsetzung der regelbasierten Datengenerierung erforderlich sind, sind in Abbildung 6.1 dargestellt. Diese dienen dazu, sowohl eigene Spalten mit ausgewählten Datentypen, Verteilungen und Merkmalswerten zu definieren und Spalten mit bestimmten Merkmalen zu generieren ohne Zugriff auf bestehende Input-Daten. Zusätzlich können bestehende Datensätze um weitere Spalten erweitert werden.

Das **SchemaViewInterface** bietet Nutzer:innen die Möglichkeit, Spalten durch den **ColumnManager** und Funktionen zur Datengenerierung (**GeneratorEngine**) bereitzustellen. Sowohl die **GeneratorEngine** als auch der **ColumnManager** erhalten die erforderlichen Regeln für die Datengenerierung über den **RuleManager**. Der **ColumnManager** ermöglicht das Hinzufügen aller Tabellenspalten einer Tabelle für den **TableUpload**, das Hinzufügen einzelner Spalten für den **ColumnUpload** und den Empfang der generierten Spalten von der **GeneratorEngine**. Benutzer:innen können über das **TableViewInterface** Tabellen einsehen, die vom **FileManager** bereitgestellt werden. Der **FileManager** kann die benötigten Daten entweder vom **FileUpload** empfangen, vom **FileDownload** herunterladen oder von der **GeneratorEngine** erhalten [BgK23].

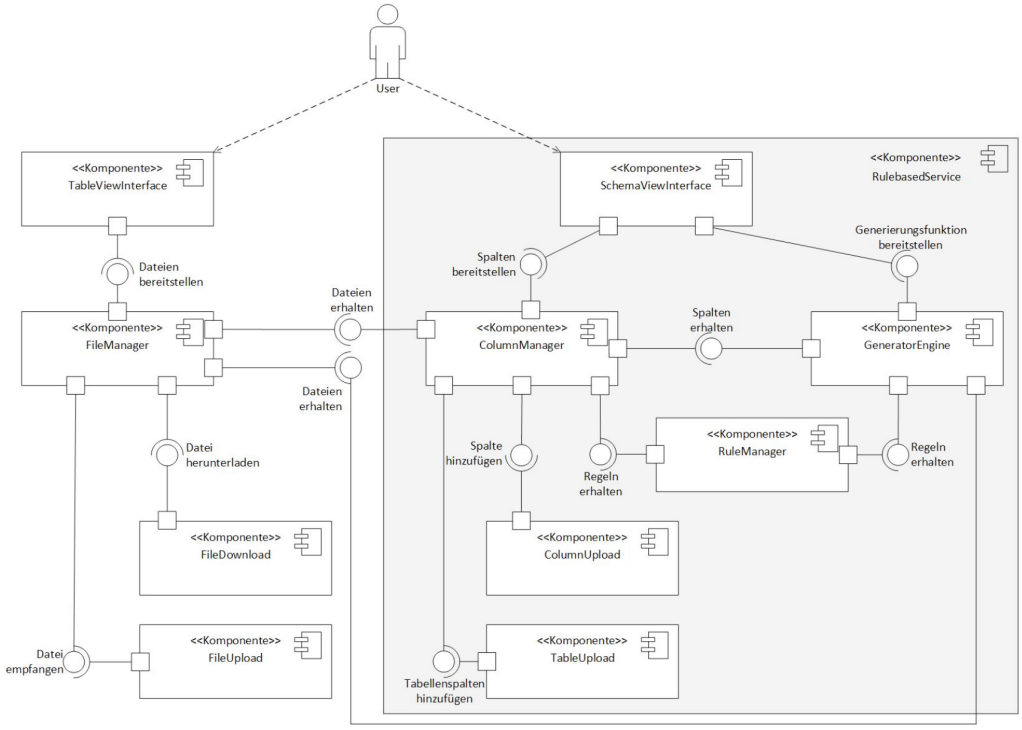


Abbildung 6.1: Komponentendiagramm für die regelbasierte Datengenerierung. Quelle: [BgK23]

6.1.2 Datenfusion

Die dargestellten Komponenten und ihre Interaktionen sind in Abbildung 6.2 veranschaulicht. Die **API** fungiert als zentrale Schnittstelle des Systems, die Endpunkte implementiert, um Anfragen zu akzeptieren und Ergebnisse sowie Anfragezustände bereitzustellen. Eng mit der **Application** verbunden, ist diese Komponente darauf angewiesen, die erforderlichen Funktionalitäten bereitzustellen. Verantwortlich für die Umsetzung der Fusion und die Koordination der Interaktionen zwischen den verschiedenen Komponenten ist die **Application**-Komponente. Der **State** übernimmt die Verwaltung der Zustände der Anfragebearbeitung sowie die Speicherung von Ergebnissen. Die **CSV-Processing**-Komponente implementiert das Lesen und Schreiben von CSV-Dateien, indem sie intern CSV-Dateien extrahiert, in ein internes Datenformat umwandelt und umgekehrt. Die zentrale Komponente, die **Fusion**, nutzt intern den JOIN-Algorithmus [BN09] zur Fusion von Datensätzen. Die **External Data**-Komponente integriert eine Schnittstelle, um externe Daten abzurufen und diese in das interne Datenformat zu konvertieren. Dadurch

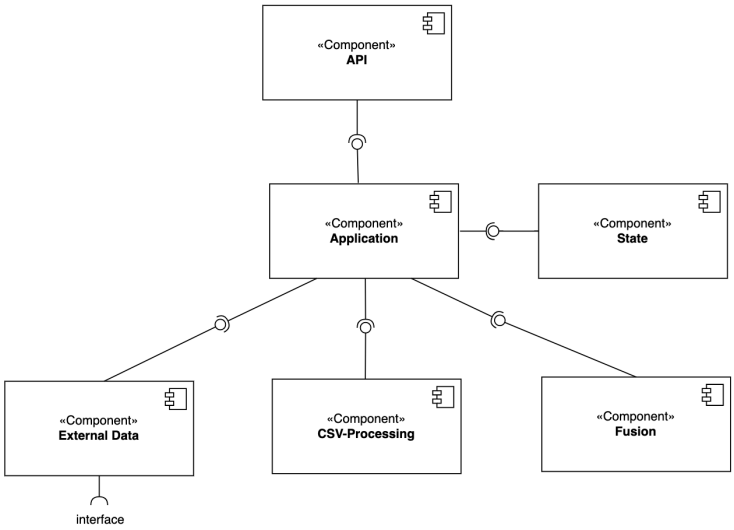


Abbildung 6.2: Komponentendiagramm für die Datenfusion. Quelle: In Anlehnung an [AM23]

wird ermöglicht, dass externe Daten (z.B. Brückendaten) nahtlos mit bereits vorhandenen Daten kombiniert werden können [AM23].

6.1.3 Reproduktion

Um dieses Anwendungsszenario umzusetzen, sind die in Abbildung 6.3 gezeigten Komponenten erforderlich. Die Benutzer:innen melden sich über die grafische Benutzeroberfläche (**GUI**) an, gelangen dann über das **API-Gateway** als Eintrittspunkt in das Backend der Reproduktion. Das **API-Gateway** dient dabei aus Sicherheitsgründen ebenfalls zur Autorisierung der Nutzer:innen. Für die Generierung von tabellarischen Reproduktionen erhält jede/-r Benutzer/-in dann einen eigenen **Generation Orchestrator** Docker-Container, um Skalierbarkeit und Selbstständigkeit zu gewährleisten. Diese Komponente orchestriert die Kommunikation mit anderen Diensten und die Ausführung des Trainingsalgorithmus. Um ein generatives ML-Modell verwenden zu können, fordert der **Generation Orchestrator** das Modell vom **Model Manager** an, welcher Docker-Images der Modelle (**Generative Model**) verwaltet und diese startet. Über Schnittstellen können Parameter und Gewichte des Modells abgefragt und definiert werden. Die Evaluationsmetriken (**Evaluation Method**) werden vom **Evaluation Manager** zur Verfügung

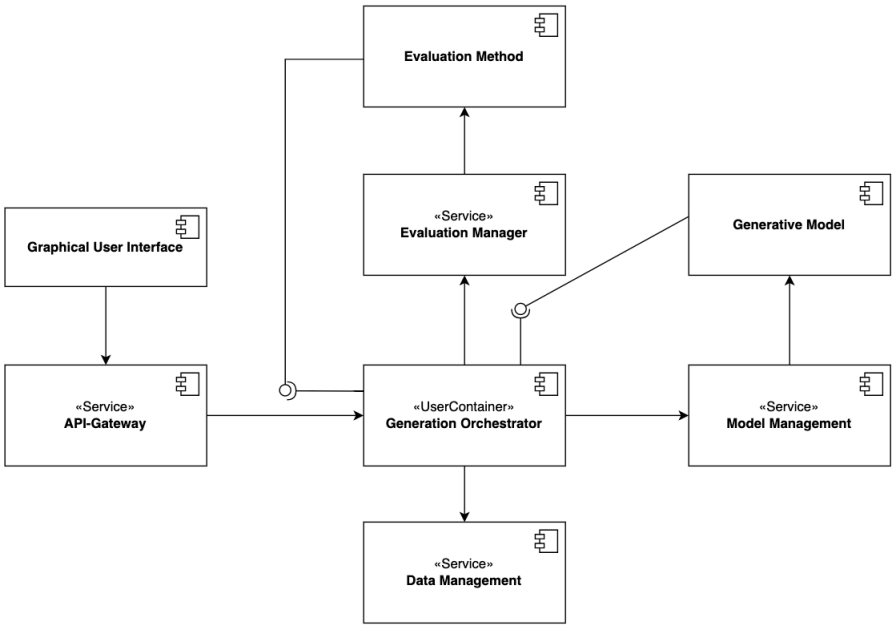


Abbildung 6.3: Komponentendiagramm für die Reproduktion. Quelle: In Anlehnung an [KKZS22]

gestellt. Folglich kann dadurch ein Modell für eine bestimmte Anzahl an Epochen trainiert werden, um anschließend die synthetischen Daten mit den realen Daten anhand von Metriken zu vergleichen. Diese Ergebnisse werden dann gespeichert. Dieser Vorgang wird mehrfach wiederholt, woraufhin der **Generation Orchestrator** die besten synthetischen Daten (basierend auf den Metriken) auswählt und den Benutzer:innen zur Verfügung stellt. Das beste Modell kann am Ende über das **API-Gateway** abgefragt werden.

6.1.4 Generierung synthetischer Nachbarschaften

Um die Generierung synthetischer Nachbarschaften zu realisieren, sind die Komponenten in Abbildung 6.4 erforderlich. Der **Neighbourhood Generation Orchestrator** startet die Neighbourhood Generation, wobei jede:r Benutzer:in einen eigenen **Neighbourhood Generation Orchestrator** erhält. Dies dient auch dazu, bereits trainierte Modelle vom **Graph Model Manager** zu erhalten. Um die Neighbourhood Generation zu ermöglichen, startet der Orchestrator den **Neighbourhood Generator**, der

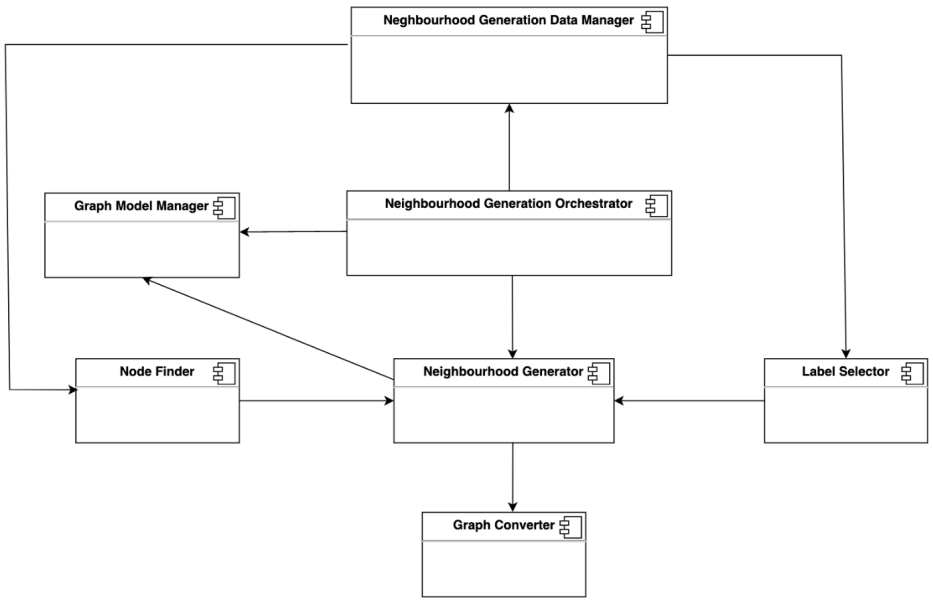


Abbildung 6.4: Komponentendiagramm für die Generierung synthetischer Nachbarschaften. Quelle: [Kra23]

anschließend den **Graph Converter** verwendet. Der **Graph Converter** ist dafür verantwortlich, Geodaten als Eingabe zu empfangen (zum Beispiel Städtenamen, Polygone oder den Radius von Adressen). Diese Eingabe wird dann in einen Adjazenzgraphen mit Labels umgewandelt. Die gelabelten Knoten in diesem Graphen entsprechen Gebäuden in einem referenzierten Bereich (z.B. Stadtteil). Der **Neighbourhood Generator** verwendet den gelabelten Graphen mit seiner Maske und dem erforderlichen Modell aus dem **Graph Model Manager** als Eingabe. Die Maske beschreibt eine Liste von Knoten, die vorhergesagt werden sollen. Dann generiert der **Neighbourhood Generator** intern eine Matrix der Vorhersagen der Labels für jeden Knoten im Graphen. Dies erzeugt einen Graphen, der eine Vorhersage darüber treffen sollte, wofür jeder Knoten/Gebäude verwendet werden sollte. Das Ergebnis wird als Eingabe vom **Label Selector** verwendet, um anschließend ein DataFrame sowie einen Graphen zu generieren. Der **Label Selector** wählt die höchste Punktzahl für jedes Label in einem Graphen aus, beispielsweise zur Platzierung von Fahrradwegen in Stadtvierteln. Der **Node Finder** geht ähnlich vor und nutzt den generierten Graphen des Neighbourhood Generators, um ein DataFrame und eine Graphenvisualisierung zu erstellen. Er wählt den Knoten mit der höchsten Punktzahl in einer bestimmten Kategorie aus. Der **Neighbourhood Generation Data Manager** verwaltet die Ergebnisse der Neighbourhood Generation und stellt sie über den **Node**

Finder und den **Label Selector** dem Benutzer zur Verfügung. Schließlich verwendet der **Neighbourhood Generation Orchestrator** am Ende der Neighbourhood Generation den **Neighbourhood Generation Data Manager**, um die abschließenden Ergebnisse bereitzustellen [Kra23].

6.1.5 Generierung synthetischer Fußgängerrouten

Um die Generierung synthetischer Fußgängerrouten umzusetzen, werden die Komponenten in Abbildung 6.5 benötigt. Der **Input Data Handler** verarbeitet die Input-Dateien, indem die möglichen Aktionen eines Agenten in einer CSV-Datei gespeichert werden. Die Schnittstelle, welche vom Input Data Handler angeboten wird, dient dem **Controller**, welcher die Interaktion zwischen der Komponente **Agent** sowie dem **Environment** steuert und kontrolliert. Der **Agent** speichert und erstellt den Agenten inklusive der möglichen Aktionen. Das **Environment** verwaltet die notwendige Umgebung sowie die Belohnungen für den Agenten. Um die notwendigen Interaktionen zwischen der Komponente **Agent** sowie dem **Environment** abzufragen, stellt der **Controller** eine Schnittstelle bereit, über die der **OutputDataHandler** die Daten abfragen kann.

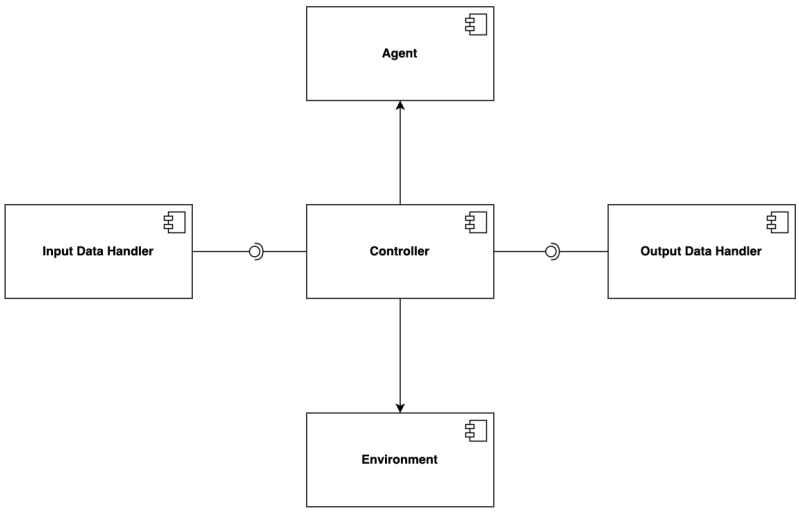


Abbildung 6.5: Komponentendiagramm für die Generierung synthetischer Fußgängerrouten.

6.2 Architekturentwurf

Die zuvor identifizierten Komponenten und ihre Interaktionen sind nun einheitlich in eine Gesamtarchitektur zu integrieren. Zur Realisierung dieser Integration werden in der Gesamtarchitektur erforderliche Anpassungen vorgenommen, beispielsweise durch das Hinzufügen oder Entfernen von Komponenten, welche im Anschluss begründet werden.

6.2.1 Bausteinsicht

Als Orientierung zur Modellierung dient die Bausteinsicht gemäß Kapitel 2.1.2, die eine statische Zerlegung des Systems in Bausteine, wie z.B. Komponenten zeigt. Der Schwerpunkt liegt zuerst darauf, die einzelnen Services darzustellen und ihre Abhängigkeiten innerhalb der Gesamtarchitektur zu verdeutlichen. Daraufhin wird eine detaillierte Visualisierung geliefert, bei der die einzelnen Komponenten innerhalb der Services betrachtet und deren Integration in die Plattform verdeutlicht werden. Die gestrichelten Pfeile beschreiben dabei klassische Abhängigkeitsbeziehungen und die durchgezogenen Pfeile Datenflüsse.

6.2.1.1 Services

Die Abbildung 6.6 zeigt die Bausteine des Gesamtentwurfs. Dabei handelt es sich um eine Microservices-Architektur bestehend aus einzelnen Services, die wiederum aus Komponenten bestehen. Die Services lassen sich kategorisieren. Eine Kategorie beschreibt die Services, welche die Kernfunktionalitäten, exemplarischen Anwendungsfälle und somit die reine Datengenerierung realisieren. Dies sind folgende Services: **Rule Based**, **Fusion**, **Neighbourhood Generation**, **Reproduction**, **Pedestrian Routes Generation**. Die zweite Kategorie, welche im Architekturentwurf berücksichtigt wird, identifiziert Hilfs-Services. Diese Hilfs-Services realisieren diverse Aufgaben, wie z.B. die Kommunikation zwischen den Services oder die Verwaltung von synthetisierten Daten. Folgende Services sind als Hilfs-Services zu behandeln: **Analyzer**, **Data Store Management**, **API-Gateway**, **Graphical User Interface**, **Logging**, **Metadata Management**. Die Hilfs-Services werden dabei hinsichtlich ihrer internen Komponentenstruktur nicht näher beschrieben.

Für die Interaktion zwischen den einzelnen Services werden einheitlich im gesamten Entwurf REST-Schnittstellen [Bie16] verwendet, um verschiedene Daten nahtlos zu übertragen und zu verarbeiten. Zwischen dem Graphical User Interface und dem API-Gateway können z.B. JSON-Dateien sowie HTML-Dateien übertragen werden. Um Daten zwischen Services der Datengenerierung (Reproduction, Rule Based, Pedestrian Routes Generation, Neighbourhood Generation, Fusion) und dem API-Gateway zu übertragen, werden JSON-Dateien benutzt. Der Datenaustausch zwischen den Generierungs-Services und dem Data Store Management basiert auf CSV-Dateien ebenso wie die Kommunikation zwischen dem Data Store Management und dem Analyzer. Das Metadata Management tauscht JSON-Dateien mit den Analyzer und dem Logging aus.

Das grundsätzliche Vorgehen, welches durch die Architektur in Abbildung 6.6 realisiert werden kann, ist folgendes: Nutzer:innen der Platttform werden über das API-Gateway autorisiert und anschließend zu dem jeweiligen Verfahren der Datensynthese geleitet. Folglich erfolgt die Kommunikation zwischen der grafischen Benutzeroberfläche und den Services zur Datengenerierung über das API-Gateway. Anschließend wird die jeweilige Methode zur Datengenerierung in den Services umgesetzt. Die Ergebnisse der Datengenerierung werden von den Services in einzelne Datenbanken geschrieben, welche vom Data Store Management verwaltet werden. Die Daten aus dem Data Store Management können dann über das Graphical User Interface von Nutzer:innen abgerufen werden. Des Weiteren ist ein Service im Entwurf integriert, welcher eine Analyse der synthetischen Daten mithilfe von statistischen Metriken ermöglicht (Analyzer). Dieser Service nutzt dazu die synthetischen Daten aus dem Data Store Management sowie die Metadaten aus dem Metadata Management, welches Logging-Daten der einzelnen Modelle verwaltet. Das Ergebnis dieser Analyse wird in einem PDF-Bericht zusammengefasst und kann ebenfalls über das Graphical User Interface betrachtet werden.

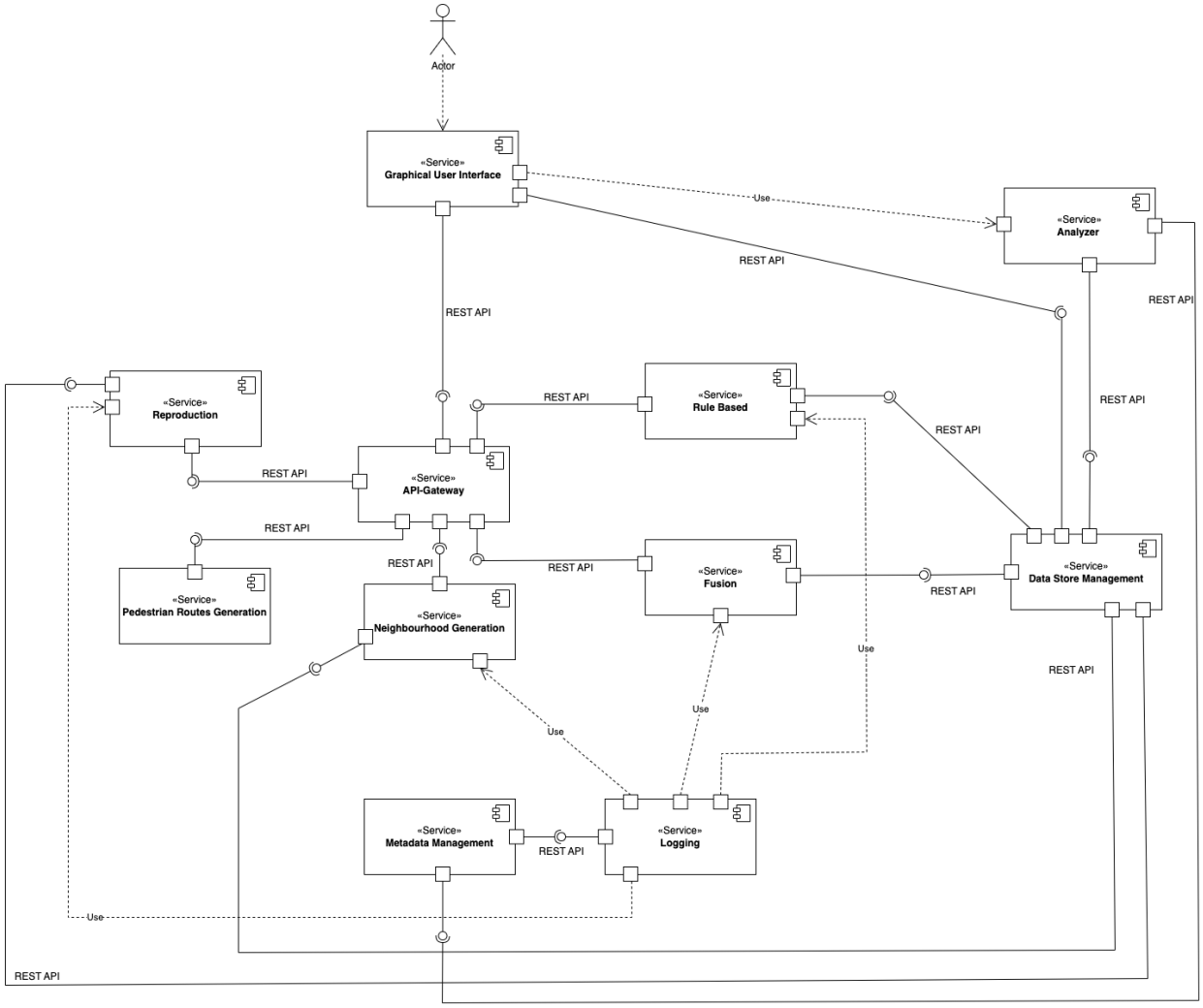


Abbildung 6.6: Architekturentwurf für eine Plattform zur Datensynthese.

6.2.1.2 Komponenten

Die Abbildungen, welche die Integration der Komponenten zeigen, sind im Anhang A.1 zu finden.

Die Integration der regelbasierten Datengenerierung ist in Abbildung A.1 dargestellt. Um die Kommunikation mit dem API-Gateway zu realisieren, wird eine Komponente (Controller) integriert. Die zuvor integrierten grafischen Schnittstellen zur Datenansicht (TableViewInterface, SchemaViewinterface) werden ausgelagert und durch eine einzelne Benutzeroberfläche (Graphical User Interface) realisiert. Dadurch soll eine einheitliche

Darstellung der synthetischen Daten ermöglicht werden. Der File Manager integriert nun eine Schnittstelle, um die Daten im Data Store Management zu speichern.

Die integrierte Datenfusion und deren Komponenten ist in Abbildung A.2 dargestellt. Die Komponente API ist nun in der Lage, die Kommunikation mit dem API-Gateway zu realisieren. Zusätzlich stellt die Komponente der Generierung (Fusion) eine Schnittstelle für das Data Store Management zur Verfügung, um die generierten Daten im weiteren Verlauf entweder zu betrachten (Graphical User Interface) oder zu analysieren (Analyzer).

Für die Reproduktion zeigt die Abbildung A.3 die Integration in die Gesamtarchitektur. Die einheitliche Strategie, eine Komponente zur Kommunikation mit dem API Gateway zu integrieren, wird hier fortgesetzt. Bei der Reproduktion (Reproduction) werden zusätzlich zwei Komponenten ausgelagert und als Services umgesetzt. Dabei handelt es sich um die Benutzeroberfläche (Graphical User Interface) und um die Evaluation (Evaluation Manager). Die Benutzeroberfläche wird in dem Entwurf so integriert, dass es eine gesamte Benutzeroberfläche für alle Services gibt. Der Service zur Evaluation (Evaluation Manager) wird als Analyse-Service (Analyzer) für alle Services berücksichtigt. Des Weiteren bietet das Data Management nun eine Schnittstelle an, um die Datenverwaltung durch das Data Store Management zu ermöglichen.

Die Generierung synthetischer Nachbarschaften und deren Integration in die Gesamtarchitektur ist in Abbildung A.4 visualisiert. Ein Controller für das API-Gateway wird ebenfalls integriert. Der Neighbourhood Generation Data Manager integriert nun ebenfalls eine Schnitstelle, um am Ende der Datengenerierung die Daten im Data Store Management der Plattform zu verwalten.

Für die Generierung synthetischer Fußgängerrouten wird das Konzept zur Integration in Abbildung A.5 dargestellt. Ein Controller wird für das API-Gateway zusätzlich integriert. Um den Vorgang der Datengenerierung zu steuern, wird zusätzlich ein Orchestrator integriert.Für diesen Anwendungsfall wird lediglich ein Grundgerüst für die Plattform berücksichtigt, um im weiteren Verlauf die Datengenerierung ebenfalls realisieren zu können. Folglich sind in diesem Fall keine Interaktionen mit Hilfs-Services (Analyzer etc.) zum jetzigen Zeitpunkt vorgesehen.

6.2.2 Entwurfsentscheidungen

Die Softwarearchitektur wird durch Microservices realisiert, um die zuvor beschriebenen Anforderungen durch einzelne Services zu erreichen. Durch das Modularisierungskonzept sollen die folgenden Vorteile der Software gewährleistet werden: Skalierbarkeit, Wartbarkeit, Flexibilität und Erweiterbarkeit.

Eine wesentliche Entwurfsentscheidung besteht darin, eine einheitliche Integrationsstrategie weiterer Services durchzuführen, um nahtlos weitere Funktionalitäten hinzufügen zu können. Diese einheitliche Strategie wird durch die Verwendung des API-Gateway gewährleistet, welches die Kommunikation zwischen den Services abbildet. Dadurch ist es möglich, weitere Backends und somit auch weitere Funktionalitäten hinzuzufügen.

Des Weiteren wird sich dazu entschieden zusätzlich zu den Funktionalitäten und Anwendungsfällen, welche durch einzelne Services realisiert werden, Hilfs-Services zu implementieren, um die Anforderungen aus Kapitel 5.1 zu erfüllen. Einer dieser Hilfs-Services ist z.B. das API-Gateway, welches die zuvor erwähnte Erweiterbarkeit der Architektur garantiert. Dadurch können zukünftig weitere Services (Funktionalitäten/Anwendungsfälle) hinzugefügt werden. Zusätzlich wird ein Metadatenmanagement (Metadata Management) integriert, um Metadaten über die einzelnen Modelle zur Datengenerierung zu erhalten. Der Analyse-Service (Analyzer) verschafft zusätzliche Transparenz über die generierten Daten.

Um die Skalierbarkeit der Plattform zu gewährleisten, wird pro Service ein Datenspeicher berücksichtigt. Dadurch kann die Plattform zukünftig weiterhin skaliert und Daten leichter wiederhergestellt werden.

Für die konsistente Kommunikation zwischen Services sollten einheitliche Schnittstellen entwickelt werden. Diese Beschränkungen der API gewährleisten einen einfachen und klaren Zugang zu den einzelnen Diensten. REST-Schnittstellen sind dazu geeignet [Bie16], weshalb die Entscheidung auf die Verwendung von REST-Schnittstellen fällt, über die das API-Gateway kommuniziert. Dazu wird in den jeweiligen Services eine Komponente integriert, welche diese Schnittstelle zur Kommunikation mit dem API-Gateway anbietet.

Des Weiteren wird sich dazu entschieden, beim Service zur Generierung synthetischer Fußgängerrouten (Pedestrian Routes Generation) den Vorgang zur Datengenerierung vorerst nicht zu berücksichtigen, da dieser Anwendungsfall im Kern verstärktes Lernen

durchführt, wofür noch klare Anforderungen zu erheben sind. Dies ist auch der Grund dafür, weshalb dieser Service keine Ergebnisdaten in eine Datenbank schreibt. Das Grundgerüst für eine spätere komplette Integration wird jedoch geschaffen, indem ein Service in der Architektur dafür berücksichtigt wird. Durch die fehlende Datensynthese wird dort ebenfalls vorerst kein Modellmanagament integriert.

6.2.3 Laufzeitsicht

Um das mögliche Verhalten der konzipierten Plattform zur Laufzeit abzubilden, wird ein Sequenzdiagramm entwickelt. Dabei handelt es sich um einen exemplarischen Anwendungsfall, der zur Datensynthese durchgeführt werden. Als Beispiel dient die Generierung synthetischer Nachbarschaften (Neighbourhood Generation).

Die Abbildung 6.7 zeigt das Laufzeitverhalten der Plattform zur Generierung synthetischer Nachbarschaften. Nutzer:innen der Plattform entscheiden sich über die grafische Benutzeroberfläche Daten mittels der Generierung synthetischer Nachbarschaften zu generieren. Um dies zu realisieren, kommuniziert die grafische Benutzeroberfläche (Graphical User Interface) über das API-Gateway mit dem Controller, welcher die Anfrage im Service erhält. Der Controller ruft im nächsten Schritt den Neighbourhood Generation Orchestrator auf, um den Vorgang zur Generierung synthetischer Nachbarschaften zu starten. Der Neighbourhood Generation Orchestrator startet dafür den Generierungsprozess und ruft zu diesem Zweck den Neighbourhood Generator auf. Der Neighbourhood Generator verwendet den Graph Converter, um öffentliche Geodaten zu konvertieren. Als Ergebnis erhält der Neighbourhood Generator ein Array mit dem Graphen und seinen Labels. Im nächsten Schritt fragt der Nachbarschaftsgenerator das Modell vom Graph Model Manager ab, um dann die Nachbarschaften zu trainieren und zu generieren. Während der Generierung erfragt die Logging-Komponente die Daten vom Graph Model Manager. Daraufhin werden Protokolle generiert, welche anschließend vom Metadata Management genutzt werden. Wenn der Generierungsprozess abgeschlossen ist, verwendet der Node Finder das Objekt des Neighbourhood Generator, um die Erzeugungsausgabe zu erhalten (ein Array mit dem beschrifteten Graphen und einer Matrix der Vorhersagen des Nachbarschaftsgenerators). Dann verwendet der Node Finder eine interne Methode, um den Graphenknoten mit der höchsten Punktzahl in jeder Kategorie zu bestimmen. Basierend auf einem Array von Vorhersagen des Neighbourhood Generator und dem Graphen bestimmt der Label Selector den Graphenknoten mit der höchsten Punktzahl pro Label. Die Ergebnisse werden am Ende der Generierung vom Data Store Management abgefragt

und den Benutzer:innen über die grafische Benutzeroberfläche zur Verfügung gestellt. Des Weiteren haben Anwender:innen die Möglichkeit die synthetischen Daten zu evaluieren. Dazu kann über die grafische Benutzeroberfläche der Analyse-Service (Analyzer) aufgerufen werden, der nach dem Aufruf die synthetischen Daten vom Data Store Management abfragt sowie Metadaten des Modells vom Metadata Management. Anschließend wird ein Generierungsbericht im PDF-Format erzeugt, welcher über die Benutzeroberfläche betrachtet werden kann.

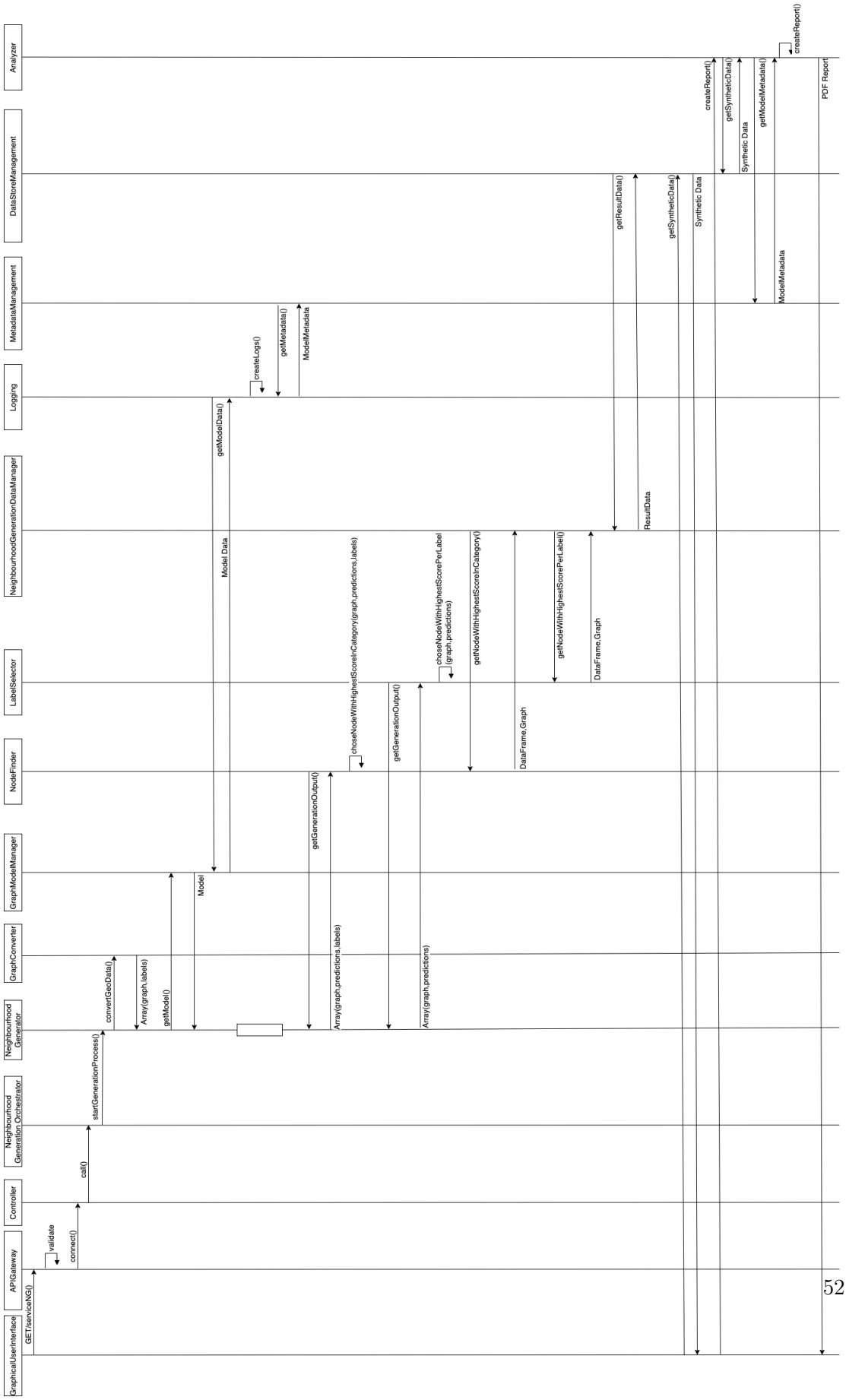


Abbildung 6.7: Laufzeitverhalten der Plattform zur Generierung synhetischer Nachbarschaften.

7 Analyse

Im folgenden Kapitel wird der Architekturentwurf für eine Plattform zur Datensynthese (Kapitel 6.2) analysiert. Als Grundlage dazu dienen die Anforderungen aus Kapitel 5.1.

Anforderung	**Erfüllt: ja/nein**
1.) Skalierbarkeit	ja
2.) Erweiterbarkeit	ja
3.) Flexibilität	(ja)
4.) Überwachung	ja
5.) Sicherheit	ja
6.) Wartbarkeit	ja
7.) Vermeidung von Engpässen	nein
8.) Integration von ML-Komponenten	ja
9.) Trainingszustand der Modelle	ja
10.) Modellmanagement	(ja)
11.) Wiederverwendung	(ja)
12.) Datenvielfalt	ja
13.) Datenqualität	(ja)
14.) Generierung eines Berichts	ja
15.) Metadatenschema	(ja)

Tabelle 7.1: Architekturanalyse.

1.) Skalierbarkeit:

Durch den Microservice-Entwurf ist diese Plattform skalierbar. Diese Skalierbarkeit gilt ebenfalls für die Speicherung der generierten Daten, da für jeden Service zur Datengenerierung ein Speicher verwendet wird.

2.) Erweiterbarkeit

Durch die Integration eines API-Gateway, welches einheitlich mit den Services kommuniziert, ist die nathlose einheitliche Integration weiterer Algorithmen zur Datengenerierung durch Microservices möglich.

3.) Flexibilität

Die Flexbilität der Architektur ist teilweise zu bestätigen. Da es sich um eine modulare Architektur handelt, welche im Kern aus Komponenten besteht, können z.B. diverse ML-Bibliotheken implementiert werden. Des Weiteren ist die Plattform dazu geeignet, verschiedenen Anwender:innen ein Nutzen zu bieten, indem verschiedene Funktionalitäten und Anwendungsfälle realisiert werden. Die Flexiblität hinsichtlich unterschiedlicher ML-Kenntnisse der einzelnen Nutzer:innen wird bei dem Architekturentwurf nicht priorisiert, kann aber durch das Anpassen der Services in Zukunft integriert werden. Ein möglicher Ansatz dazu wäre es, einen individuellen Service zur Autorisierung und Personalisierung zu integrieren, um persönliche Nutzer-Bereiche innerhalb der Plattform zu berücksichtigen, in dem Nutzer:innen basierend auf ihren ML-Kenntnissen agieren können.

4.) Überwachung

Die Überwachung der Datensynthese wird durch ein Logging-Service berücksichtigt. Dadurch sind Nutzer:innen dazu in der Lage, Informationen über die ML-Modelle und deren Ergebnisse zu erhalten. Diese Protokolle werden zusätzlich in dem Analyse-Service zur Analyse der generierten Daten genutzt.

5.) Sicherheit

Die Sicherheit wird bei dem Architektur-Entwurf berücksichtigt. Der Direktzugriff von außen auf die Services wird durch das API-Gateway vermieden, wodurch dieser Ansatz zur Sicherheit an einer zentralen Stelle integriert wird. Folglich werden die Sicherheit und Autorisierung für die Plattform durch das API-Gateway realisiert.

6.) Wartbarkeit

Der vorliegende Architekturentwurf ist als wartbar zu bewerten. Dies ist der Fall, da es sich um eine Microservice-Architektur handelt, welche im Kern eine einheitliche Integration der Services verfolgt. Zusätzlich sorgt die einheitliche Kommunikation der Services über klar definierte REST-Schnittstellen für Wartbarkeit.

7.) Vermeidung von Engpässen

Der Engpass bezüglich der Datenverarbeitung ist beim gegenwärtigen Plattformentwurf nicht berücksichtigt. Lediglich die Skalierbarkeit der einzelnen Services durch die Nutzung einzelner Datenbanken ist gegeben.

8.) Integration von ML-Komponenten

Die Komponenten für Maschinelles Lernen sind innerhalb der einzelnen Services integriert. Durch die Bündelung in Services, welche über einheitliche Schnittstellen angesprochen werden, ist eine konsequente Integration nachzuweisen.

9.) Trainingszustand der Modelle

Durch Hilfs-Services, welche das Trainingsverhalten der Modelle protokollieren sowie die Metadaten dazu, wird die Transparenz durch den Plattform-Entwurf erfüllt. Bei den protokollierten Daten handelt es sich um dynamische Informationen der Modelle zur Trainingszeit um Nutzer:innen Informationen darüber zu liefern, welches Modell die jeweiligen Ergebnisse produziert.

10.) Modellmanagement

Das Modellmanagement ist nur teilweise im Entwurf integriert. Bei einigen Services sind Komponenten konzipiert, die das Modellmanagement abdecken. Bei folgenden Services ist bisher kein Modellmanagement integriert: Pedestrian Routes Generation, Fusion und Rule Based.

11.) Wiederverwendung

Diese Anforderung baut auf dem Modellmanagement auf und wird folglich ebenfalls nur teilweise erfüllt. Bei den Services, in denen ein Modellmanagement integriert ist, wird die Wiederverwendung von Modellen ermöglicht.

12.) Datenvielfalt

Diese Anforderung wird erfüllt. Die Plattform ist durch den hier konzipierten Entwurf in der Lage tabellarische Daten durch verschiedene Generierungsverfahren zu erstellen. Die Generierung von unstrukturierten Datentypen ist bisher nicht berücksichtigt.

13.) Datenqualität

Die Datenqualität wird teilweise erfüllt. Der Plattformentwurf integriert keine Services, welche Input-Daten hinsichtlich der Qualität untersuchen. Die Qualität der synthetischen Daten wird jedoch durch einen Service zur Analyse (Analyzer) basierend auf statistischen Metriken evaluiert.

14.) Generierung eines Berichts

Der zuvor erwähnte Analyse-Service erstellt einen Bericht zur Datengenerierung im PDF-Format, um Informationen über statistische Kennzahlen, wie z.B. Datenverteilung, zu erhalten.

15.) Metadatenschema

Diese Anforderung wird erfüllt, da den Nutzer:innen der Plattform Metadaten über die einzelnen ML-Modelle über die Benutzeroberfläche zur Verfügung gestellt werden können. Metadaten über die generierten Daten sowie über vorhandene Input-Daten sind bisher nicht berücksichtigt.

Analyseergebnis:

Die Analyse zeigt, dass die entwickelte Architektur die Mehrheit der definierten Anforderungen erfüllen kann. Dadurch lässt sich auch die zweite Forschungsfrage der Arbeit beantworten, wie ein Architekturentwurf aussehen kann, der Funktionalitäten und Anwendungsfälle nahtlos integrieren kann. Der im Rahmen dieser Arbeit entwickelte Architekturentwurf schafft die Grundlage dafür. Es handelt sich dabei im Kern um einen Microservice-Entwurf, welcher ein API-Gateway zur einheitlichen Kommunikation zwischen den Services integriert. Dadurch kann auch eine zukünftige Erweiterbarkeit gewährleistet werden.

Des Weiteren bestätigt die Analyse aber auch, dass der vorliegende Architekturentwurf weiterhin kritisch betrachtet werden muss. Zwar kann die Architektur verschiedene Funktionalitäten integrieren, jedoch sind dafür Anpassungen durch das Hinzufügen und Entfernen von Services/Komponenten notwendig. Folglich ist auch der hier entwickelte Architekturentwurf nicht komplett dazu geeignet, Funktionalitäten ohne Änderungen zu integrieren. Zusätzlich können nicht alle Anforderungen erfüllt werden. Für einen zukünftigen Entwurf könnten z.B. die unterschiedlichen Vorkenntnisse der Anwender:innen berücksichtigt werden, indem eine Personalisierung innerhalb der Plattform stattfindet.

Dadurch könnten Nutzer:innen einen eigenen Nutzerbereich, welcher auf den individuellen Kenntnissen basiert, erhalten. Eine Möglichkeit zur Umsetzung könnte darin bestehen, einen Service zur Personalisierung im Entwurf zu integrieren. Bei genauerer Betrachtung des Entwurfs lässt sich feststellen, dass ein Modellmanagement konsequent integriert werden könnte und somit auch für alle Verfahren zur Synthese eine Wiederverwendung der ML-Modelle sichergestellt wird. Für einen zukünftigen Entwurf kann es zusätzlich interessant sein, weitere Datentypen (z.B. unstrukturierte Datentypen) zu verarbeiten, um eine größere Datenvielfalt innerhalb der Plattform zu gewährleisten. Um eine einheitliche Vorverarbeitung diverser Datentypen zu ermöglichen und dadurch eine Datenqualität für Input-Daten zu garantieren, kann dafür ein Service in der Architektur integriert werden. Mit Blick auf das Metadatenschema sollten zukünftig auch die Metadaten der vorhandenen Input-Daten berücksichtigt werden.

8 Fazit und Ausblick

8.1 Fazit

Die vorliegende Arbeit untersuchte zwei zentrale Forschungsfragen im Kontext der Entwicklung einer Plattform zur Datensynthese. Die erste Frage fokussierte sich auf die Eignung bestehender Architekturmuster für Maschinelles Lernen für dieses Vorhaben. Darauf aufbauend wurde, im Rahmen der zweiten Forschungsfrage, ein möglicher Architekturentwurf vorgestellt, welcher diverse Anwendungsfälle zur Datensynthese integrieren kann.

Um diese Fragen zu beantworten, erfolgte zunächst im zweiten Kapitel eine ausführliche Darstellung des aktuellen Forschungsstands. Dabei wurden relevante Themenfelder wie allgemeine Softwarearchitektur, Microservices, Softwarearchitektur für Maschinelles Lernen und Datensynthese behandelt.

In Kapitel 3 wurden Leser:innen in verwandte Arbeiten eingeführt, die bereits ML-Plattformen entworfen und getestet haben. Kapitel 4 charakterisierte die zu entwickelnde Plattform durch die Beschreibung ihrer Funktionalitäten, exemplarischer Anwendungsfälle und potenzieller Nutzerrollen. Die Charakterisierung der Plattform hat die Notwendigkeit gezeigt, eine Architektur zu konzipieren, welche sowohl den Anforderungen verschiedener Nutzer:innen gerecht werden kann, als auch durch einen integrativen Charakter verschiedene Funktionalitäten abbilden kann.

Die Erkenntnisse der Plattform-Charakterisierung bildeten die Grundlage für die Konzeption der Architektur in Kapitel 5. Hier wurden Plattform-Anforderungen identifiziert, Architekturmuster für Maschinelles Lernen vorgestellt und hinsichtlich der Anforderungen analysiert. Diese Analyse legte den Grundstein für die Beantwortung der ersten Forschungsfrage. Die in Kapitel 5 durchgeführen Musteranalysen zeigen, dass zunächst verschiedene Ansätze für ML-Architekturmuster existieren. Die erste Forschungsfrage dieser Arbeit kann dabei wie folgt beantwortet werden: Die ausgewählten ML-

Architekturmuster können durch ihre individuellen Vorteile eine Grundlage dafür schaffen, eine Plattform zur Datensynthese zu entwickeln. Dabei können gemeinsame Vorteile und Schwachstellen identifiziert werden. Die Literatur zeigt, dass durch eine Microservice-Architektur mit einem integrierten Gateway die meisten Anforderungen aus Kapitel 5.1 erfüllt werden können.

Die Anwendung der Erkenntnisse aus Kapitel 5 und der zugehörigen Integration einzelner Services resultieren in einem Architekturentwurf in Kapitel 6. Damit wird ein Vorschlag geliefert einen Entwurf zu konzipieren, welcher diverse Anwendungsfälle nahtlos integrieren kann. Der im Rahmen dieser Arbeit entwickelte Entwurf ist dabei größtenteils in der Lage, die Anforderungen aus Kapitel 5 umzusetzen, was die Analyse in Kapitel 7 belegt. Dabei handelt es sich um einen Architekturentwurf für eine Datensynthese-Plattform, bestehend aus Microservices mit einem integrierten API-Gateway zur einheitlichen Kommunikation zwischen den Services. Dadurch ist die zweite Forschungsfrage beantwortet. Die Analyse des Entwurfs weist jedoch auch auf Schwachstellen hin, die in weiteren Arbeiten berücksichtigt werden sollten. Diesbezüglich könnten zukünftige Entwürfe von einer Personalisierung der Plattform, einem konsequenten Modellmanagement und der Erweiterung zur Verarbeitung verschiedener Datentypen profitieren. Eine angestrebte Datenvielfalt ist durch die zusätzliche Integration einer Datenvorverarbeitung sowie einer einheitlichen Verwaltung der zugehörigen Metadaten zu unterstützen.

8.2 Ausblick

Die vorliegende Arbeit belegt, dass der Entwurf von ML-Anwendungen, wie z.B. Plattformen, diverse Herausforderungen impliziert. Dies ist ein Indikator dafür, dass der Bedarf an weiteren Architekturmustern für ML-Anwendungen steigt. Diese ML-Architekturmuster können dann gebündelt zusammengefasst werden für wiederkehrende Herausforderungen bei ML-Anwendungen. Dies kann als Grundlage dazu dienen, individuelle Anwendungen für Maschinelles Lernen mit bereits bekannten Problemen einfacher umzusetzen.

Die entwickelte Softwarearchitektur bildet die Grundlage dafür, diverse Funktionalitäten und Anwendungsfälle in eine Plattform zur Datensynthese nahtlos zu integrieren. Diese Softwarearchitektur könnte im Rahmen weiterer Arbeiten um zusätzliche Funktionalitäten oder Anwendungsfälle erweitert werden. Diese Erweiterung kann dann erneut, basierend auf Anforderungen, untersucht werden.

Mit Blick auf den Entwurf kann dieser im Rahmen weiterer Arbeiten aus einer anderen Perspektive betrachtet werden. Ein Vorschlag dazu wäre den Fokus auf die Infrastruktur der Plattform zu richten, welche in dieser Arbeit nicht priorisiert wird. Dafür könnte man z.B. eine Verteilungssicht modellieren und den Entwurf anhand von Infrastruktur-Anforderungen analysieren. Ein weiterer Vorschlag könnte der Fokus auf die reine Datenverarbeitung in der Plattform sein, da die reine Datenverarbeitung in dem hier entwickelten Architekturentwurf weniger priorisiert wird. Zusätzlich könnte eine Integrationssicht untersucht werden. Dadurch könnten die Interaktionen der Services genauer betrachtet werden, indem z.B. Protokolle/Datenformate, Abhängigkeiten, Schnittstellen und Nachrichtenflüsse genauer untersucht werden. Auch dafür sind gegebenenfalls andere Anforderungen zu erfüllen.

Ein finaler Schritt könnte darin bestehen, die einzelnen Services in der Praxis zu implementieren und eine Integration in die Plattform durchzuführen. Dabei könnten Schwachstellen des Entwurfs in der Praxis identifiziert und behoben werden.

Es lässt sich daher abschließend festhalten, dass zusätzliche Forschungen, bestehend aus Anpassungen und Umsetzung der Softwarearchitektur, weitere Fortschritte für die Entwicklungen von Plattformen des Maschinellen Lernens implizieren würden.

Literaturverzeichnis

[AAB+19] Pulkit Agrawal, Rajat Arya, Aanchal Bindal, Sandeep Bhatia, Anupriya Gagneja, Joseph Godlewski, Yucheng Low, Timothy Muss, Mudit Manu Paliwal, Sethu Raman, et al. Data platform for machine learning. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1803–1816, 2019.

[ABB+19] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300. IEEE, 2019.

[ABCFB18] Anders Arpteg, Björn Brinne, Luka Crnkovic-Friis, and Jan Bosch. Software engineering challenges of deep learning. In *2018 44th euromicro conference on software engineering and advanced applications (SEAA)*, pages 50–59. IEEE, 2018.

[ADM+20] Samuel A Assefa, Danial Dervovic, Mahmoud Mahfouz, Robert E Tillman, Prashant Reddy, and Manuela Veloso. Generating synthetic data in finance: opportunities, challenges and pitfalls. In *Proceedings of the First ACM International Conference on AI in Finance*, pages 1–8, 2020.

[AM23] Abdullah Al Mohammad. Konzeption und Entwicklung eines generischen Services zur Datenfusion. *Technical report*, 2023.

[AZM15] Theodoros Anagnostopoulos, Arkady Zaslavsky, and Alexey Medvedev. Robust waste collection exploiting cost efficiency of iot potentiality in smart cities. In *2015 International conference on recent advances in internet of things (RIoT)*, pages 1–6. IEEE, 2015.

[BBC+17] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, et al. Tfx: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1387–1395, 2017.

[BgK23] Mareile Beernink genannt Konjer. Protypische Umsetzung einer regelbasierten Datengenerierung. *Technical report*, 2023.

[Bie16] Matthias Biehl. *RESTful Api Design*, volume 3. API-University Press, 2016.

[BJS19] Lucas Baier, Fabian Jöhren, and Stefan Seebacher. Challenges in the deployment and operation of machine learning in practice. In *ECIS*, volume 1, 2019.

[BN09] Jens Bleiholder and Felix Naumann. Data fusion. *ACM computing surveys (CSUR)*, 41(1):1–41, 2009.

[BSD+17] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3722–3731, 2017.

[BVC19] Hrvoje Belani, Marin Vukovic, and Željka Car. Requirements engineering challenges in building ai-based complex systems. In *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, pages 252–255. IEEE, 2019.

[CHBJ+18] Travers Ching, Daniel S Himmelstein, Brett K Beaulieu-Jones, Alexandr A Kalinin, Brian T Do, Gregory P Way, Enrico Ferrero, Paul-Michael Agapow, Michael Zietz, Michael M Hoffman, et al. Opportunities and obstacles for deep learning in biology and medicine. *Journal of The Royal Society Interface*, 15(141):20170387, 2018.

[CPM+20] João Lucas Correia, Juliana Alves Pereira, Rafael Mello, Alessandro Garcia, Baldoino Fonseca, Márcio Ribeiro, Rohit Gheyi, Marcos Kalinowski, Renato Cerqueira, and Willy Tiengo. Brazilian data scientists:

Revealing their challenges and practices on machine learning model development. In *Proceedings of the XIX Brazilian Symposium on Software Quality*, pages 1–10, 2020.

[CWD+18] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018.

[DL20] Ali Davoudian and Mengchi Liu. Big data systems: A software engineering perspective. *ACM Computing Surveys (CSUR)*, 53(5):1–39, 2020.

[EEMH20] Khaled El Emam, Lucy Mosquera, and Richard Hoptroff. *Practical synthetic data generation: balancing privacy and the broad availability of data*. O'Reilly Media, 2020.

[ESHEB11] Shaker H Ali El-Sappagh, Abdeltawab M Ahmed Hendawi, and Ali Hamed El Bastawissy. A proposed model for data warehouse etl processes. *Journal of King Saud University-Computer and Information Sciences*, 23(2):91–104, 2011.

[FHI+20] Gaku Fujii, Koichi Hamada, Fuyuki Ishikawa, Satoshi Masuda, Mineo Matsuya, Tomoyuki Myojin, Yasuharu Nishi, Hideto Ogawa, Takahiro Toku, Susumu Tokumoto, et al. Guidelines for quality assurance of machine learning-based artificial intelligence. *International journal of software engineering and knowledge engineering*, 30(11n12):1589–1606, 2020.

[FM] Daniel Fasel and Andreas Meier. *Big Data: Grundlagen, Systeme und Nutzungspotenziale*. Edition HMD. URL: https://books.google.de/books?id=etR6DAAAQBAJ.

[FMBO20] Teodor Fredriksson, David Issa Mattos, Jan Bosch, and Helena Holmström Olsson. Data labeling: An empirical investigation into industrial challenges and mitigation strategies. In *International Conference on Product-Focused Software Process Improvement*, pages 202–216. Springer, 2020.

[Foua] The Apache Software Foundation. Apache hadoop. URL: https://hadoop.apache.org/?uclick_id=f1ac0d11-3b7f-4b44-9a04-15b953faec79,note={Zugriffsdatum:17.08.23},.

[Foub] The Apache Software Foundation. Apache samza. URL: https://samza.apache.org/?uclick_id=f1ac0d11-3b7f-4b44-9a04-15b953faec79,note={Zugriffsdatum:17.08.23},.

[Fouc] The Apache Software Foundation. Apache spark. URL: https://spark.apache.org/?uclick_id=f1ac0d11-3b7f-4b44-9a04-15b953faec79,note={Zugriffsdatum:17.08.23},.

[Foud] The Apache Software Foundation. Apache spark: Mllib. URL: https://spark.apache.org/mllib/?uclick_id=f1ac0d11-3b7f-4b44-9a04-15b953faec79,note={Zugriffsdatum:17.08.23},.

[GN22] Ayşe Glass and Jörg Rainer Noennig. Synthetic pedestrian routes generation: exploring mobility behavior of citizens through multi-agent reinforcement learning. *Procedia Computer Science*, 207:3367–3375, 2022.

[Gol16] Sunila Gollapudi. *Practical machine learning*. Packt Publishing Ltd, 2016.

[GPAM+20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[GWRL19] Gharib Gharibi, Vijay Walunj, Sirisha Rella, and Yugyung Lee. Modelkb: towards automated management of the modeling lifecycle in deep learning. In *2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, pages 28–34. IEEE, 2019.

[HBEB16] Charles Hill, Rachel Bellamy, Thomas Erickson, and Margaret Burnett. Trials and tribulations of developers of intelligent systems: A field study. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 162–170. IEEE, 2016.

[HDB] Jeremy Hermann and Mike Del Baslo. Meet michelangelo: Uber's machine learning platform. Zugriffsdatum: 17.08.23. URL: https://www.uber.com/en-UY/blog/michelangelo-machine-learning-platform/.

[Inca] Docker Inc. Docker. Zugriffsdatum: 15.07.23. URL: https://www.docker.com/.

[Incb] GitHub Inc. Github. URL: https://github.com/dmlc/xgboost?uclick_id=f1ac0d11-3b7f-4b44-9a04-15b953faec79, note={Zugriffsdatum:17.08.23},.

[Incc] Google Inc. Google.

[Incd] Google Inc. Tensorflow.

[Ince] Uber Technologies Inc. Uber. Zugriffsdatum: 17.08.23. URL: https://www.uber.com/de/de/.

[JM17] T. John and P. Misra. *Data Lake for Enterprises*. Packt Publishing, 2017. URL: https://books.google.de/books?id=nHc5DwAAQBAJ.

[JYVDS18] James Jordon, Jinsung Yoon, and Mihaela Van Der Schaar. Pate-gan: Generating synthetic data with differential privacy guarantees. In *International conference on learning representations*, 2018.

[KAK+17] Artur Kadurin, Alexander Aliper, Andrey Kazennov, Polina Mamoshina, Quentin Vanhaelen, Kuzma Khrabrov, and Alex Zhavoronkov. The cornucopia of meaningful leads: Applying deep adversarial autoencoders for new molecule development in oncology. *Oncotarget*, 8(7):10883, 2017.

[KG] FOUTSE KHOMH and YANN-GAËL GUÉHÉNEUC. Software engineering patterns for machine learning applications (sep4mla).

[KKZS22] Pamela Kunert, Tom Krause, Olaf Zukunft, and Ulrike Steffens. A platform providing machine learning algorithms for data generation and fusion - an architectural approach. *Technical report*, 2022.

[Kra23] Tom Krause. A platform providing machine learning algorithms for neighbourhood generation - an architectural approach. *Technical report*, 2023.

[KTC+18] Minsuk Kahng, Nikhil Thorat, Duen Horng Chau, Fernanda B Viégas, and Martin Wattenberg. Gan lab: Understanding complex deep generative models using interactive visual experimentation. *IEEE transactions on visualization and computer graphics*, 25(1):310–320, 2018.

[KZDB17] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. Data scientists in software teams: State of the art and challenges. *IEEE Transactions on Software Engineering*, 44(11):1024–1038, 2017.

[Lar12] Craig Larman. *Applying UML and patterns: an introduction to object oriented analysis and design and interative development*. Pearson Education India, 2012.

[Lip23] B. Lipp. *Modern Data Architectures with Python: A practical guide to building and deploying data pipelines, data warehouses, and data lakes with Python*. Packt Publishing, 2023. URL: https://books.google.de/books?id=MJrVEAAAQBAJ.

[LMNVGDB20] Fernando López-Martínez, Edward Rolando Núñez-Valdez, Vicente García-Díaz, and Zoran Bursac. A case study for a big data and machine learning platform to improve medical decision support in population health management. *Algorithms*, 13(4):102, 2020.

[LRB+19] Lucy Ellen Lwakatare, Aiswarya Raj, Jan Bosch, Helena Holmström Olsson, and Ivica Crnkovic. A taxonomy of software engineering challenges for machine learning systems: An empirical investigation. In *Agile Processes in Software Engineering and Extreme Programming: 20th International Conference, XP 2019, Montréal, QC, Canada, May 21–25, 2019, Proceedings 20*, pages 227–243. Springer International Publishing, 2019.

[MAG+23] Igor L Markov, Pavlos A Apostolopoulos, Mia Garrard, Yin Huang, Tanvi Gupta, Anika Li, Cesar Cardoso, George Han, Ryan Maghsoudian, Norm Zhou, et al. Scalable end-to-end ml platforms: from automl to self-serve. *arXiv preprint arXiv:2302.14139*, 2023.

[Mat17] Christian Mathis. Data lakes. *Datenbank-Spektrum*, 17(3):289–293, 2017.

[MB00] Harry Mucksch and Wolfgang Behme. Das data warehouse-konzept. *Aufl., Wiesbaden*, 2000.

[MWK+22] Igor L Markov, Hanson Wang, Nitya S Kasturi, Shaun Singh, Mia R Garrard, Yin Huang, Sze Wai Celeste Yuen, Sarah Tran, Zehui Wang, Igor Glotov, et al. Looper: An end-to-end ml platform for product

decisions. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3513–3523, 2022.

[NZM+19] Fatemeh Nargesian, Erkang Zhu, Renée J Miller, Ken Q Pu, and Patricia C Arocena. Data lake management: challenges and opportunities. *Proceedings of the VLDB Endowment*, 12(12):1986–1989, 2019.

[PKG] SIEN REEVE ORDONEZ PERALTA, FOUTSE KHOMH, and YANN-GAËL GUÉHÉNEUC. Software engineering patterns for machine learning applications (sep4mla)-part 3-data processing architectures.

[Pla09] Hasso Plattner. A common database approach for oltp and olap using an in-memory column database. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 1–2, 2009.

[PQW+20] Hung Viet Pham, Shangshu Qian, Jiannan Wang, Thibaud Lutellier, Jonathan Rosenthal, Lin Tan, Yaoliang Yu, and Nachiappan Nagappan. Problems and opportunities in training deep learning software systems: An analysis of variance. In *Proceedings of the 35th IEEE/ACM international conference on automated software engineering*, pages 771–783, 2020.

[RMC15] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[RNGS19] Christopher Ré, Feng Niu, Pallavi Gudipati, and Charles Srisuwananukorn. Overton: A data system for monitoring and improving machine-learned products. *arXiv preprint arXiv:1909.05372*, 2019.

[RRK+19] Md Saidur Rahman, Emilio Rivera, Foutse Khomh, Yann-Gaël Guéhéneuc, and Bernd Lehnert. Machine learning software engineering in practice: An industrial case study. *arXiv preprint arXiv:1906.07154*, 2019.

[SAM+17] Anush Sankaran, Rahul Aralikatte, Senthil Mani, Shreya Khare, Naveen Panwar, and Neelamadhav Gantayat. Darviz: deep abstract representation, visualization, and verification of deep learning models. In *2017 IEEE/ACM 39th International Conference on Software Engineering: New*

Ideas and Emerging Technologies Results Track (ICSE-NIER), pages 47–50. IEEE, 2017.

[SHa] Gernot Starke and Peter Hruschka. arc42 dokumentation. Zugriffsdatum: 07.07.23. URL: https://docs.arc42.org/home/.

[SHb] Gernot Starke and Peter Hruschka. arc42 dokumentation: Bausteinsicht. Zugriffsdatum: 07.07.23. URL: https://docs.arc42.org/section-5/.

[SHc] Gernot Starke and Peter Hruschka. arc42 dokumentation: Laufzeitsicht. Zugriffsdatum: 07.07.23. URL: https://docs.arc42.org/section-6/.

[SHB16] Rico Sennrich, Barry Haddow, and Alexandra Birch. Edinburgh neural machine translation systems for wmt 16. *arXiv preprint arXiv:1606.02891*, 2016.

[SHG+14] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. Machine learning: The high interest credit card of technical debt. 2014.

[SHG+15] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, 28, 2015.

[Sta15] Gernot Starke. *Effektive Softwarearchitekturen: Ein praktischer Leitfaden*. Carl Hanser Verlag GmbH Co KG, 2015.

[SZC+17] Derrick C Spell, Xiao-Han T Zeng, Jae Young Chung, Bahador Nooraei, Richard T Shomer, Ling-Yong Wang, James C Gibson, and Daniel Kirsche. Flux: Groupon's automated, scalable, extensible machine learning platform. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1554–1559. IEEE, 2017.

[TLT23] M. Tranquillin, V. Lakshmanan, and F. Tekiner. *Architecting Data and Machine Learning Platforms*. O'Reilly Media, 2023. URL: https://books.google.de/books?id=TYjcEAAAQBAJ.

[Tri18] Stavros Tripakis. Data-driven and model-based design. In *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pages 103–108. IEEE, 2018.

[Vas] Yales Stéfano Rios Vasconcelos. Neighbourhood generation. *PowerPoint slides, year=2022*.

[Wol16] Eberhard Wolff. *Continuous delivery: der pragmatische Einstieg*. dpunkt. verlag, 2016.

[Wol18] Eberhard Wolff. *Microservices: Grundlagen flexibler Softwarearchitekturen*. dpunkt. verlag, 2018.

[WUKG19] Hironori Washizaki, Hiromu Uchida, Foutse Khomh, and Yann-Gaël Guéhéneuc. Studying software engineering patterns for designing machine learning systems. In *2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, pages 49–495. IEEE, 2019.

[WUKG20] Hironori Washizaki, Hiromu Uchida, Foutse Khomh, and Yann-Gaël Guéhéneuc. Machine learning architecture and design patterns. *IEEE Software*, 8, 2020.

[XQMZ19] Chao Xie, Hua Qi, Lei Ma, and Jianjun Zhao. Deepvisual: a visual programming tool for deep learning systems. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, pages 130–134. IEEE, 2019.

[XWL+17] Ziang Xie, Sida I Wang, Jiwei Li, Daniel Lévy, Aiming Nie, Dan Jurafsky, and Andrew Y Ng. Data noising as smoothing in neural network language models. *arXiv preprint arXiv:1703.02573*, 2017.

[Yok19] Haruki Yokoyama. Machine learning system architectural pattern for improving operational stability. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 267–274. IEEE, 2019.

[ZGM+19] Tianyi Zhang, Cuiyun Gao, Lei Ma, Michael Lyu, and Miryung Kim. An empirical study of common challenges in developing deep learning applications. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pages 104–115. IEEE, 2019.

[ZKSE16] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. Generative visual manipulation on the natural image manifold. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The*

Netherlands, October 11-14, 2016, Proceedings, Part V 14, pages 597–613. Springer, 2016.

[Zör15] Stefan Zörner. *Software-Architekturen dokumentieren und kommunizieren*. Carl Hanser Verlag GmbH Co KG, 2015.

[ZXZ+20] Ru Zhang, Wencong Xiao, Hongyu Zhang, Yu Liu, Haoxiang Lin, and Mao Yang. An empirical study on program failures of deep learning jobs. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 1159–1170, 2020.

[ZYF+19] Xufan Zhang, Ziyue Yin, Yang Feng, Qingkai Shi, Jia Liu, and Zhenyu Chen. Neuralvis: visualizing and interpreting deep learning models. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1106–1109. IEEE, 2019.

A Anhang

A.1 Bausteinsicht der Komponenten-Integration

Im folgenden Abschnitt wird die Integration der Komponenten aus den einzelnen Services visualisiert, welche in Kapitel 6.2.1.2 nicht gezeigt wurden.

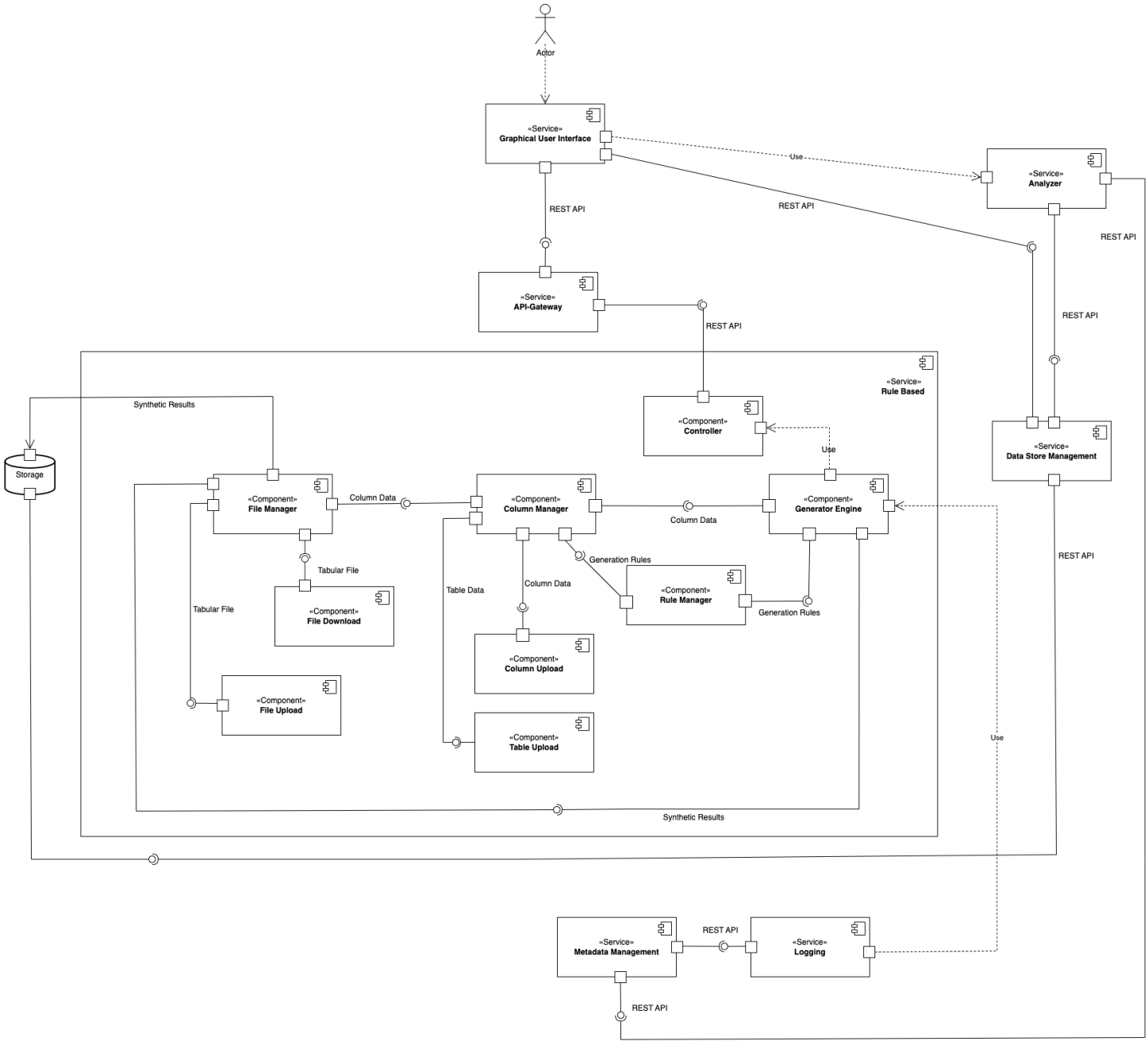


Abbildung A.1: Integration der regelbasierten Datengenerierung in die Gesamtarchitektur (aus Kapitel 6.2.1.2).

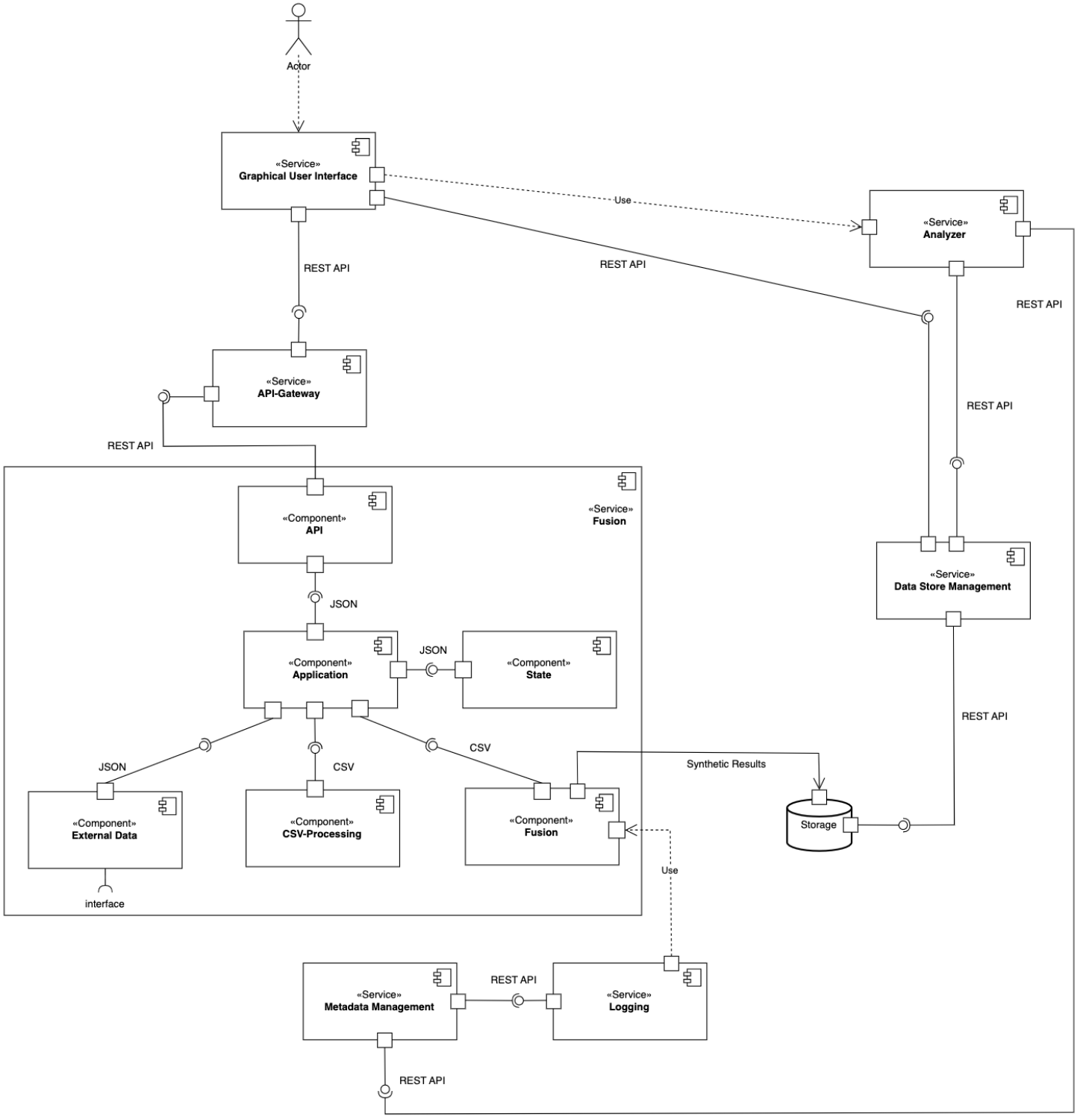


Abbildung A.2: Integration der Fusion in die Gesamtarchitektur (aus Kapitel 6.2.1.2).

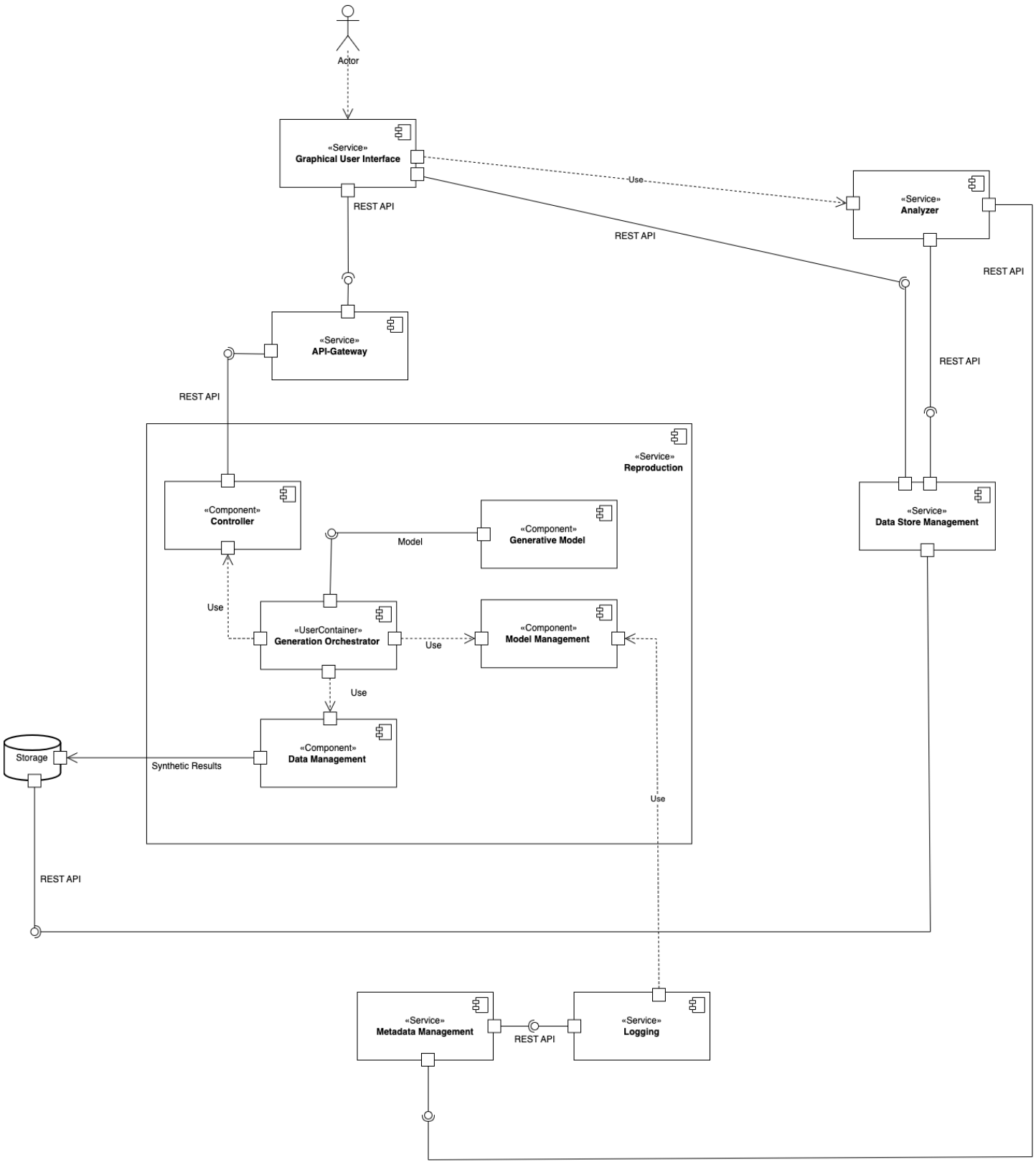


Abbildung A.3: Integration der Reproduktion in die Gesamtarchitektur (aus Kapitel 6.2.1.2).

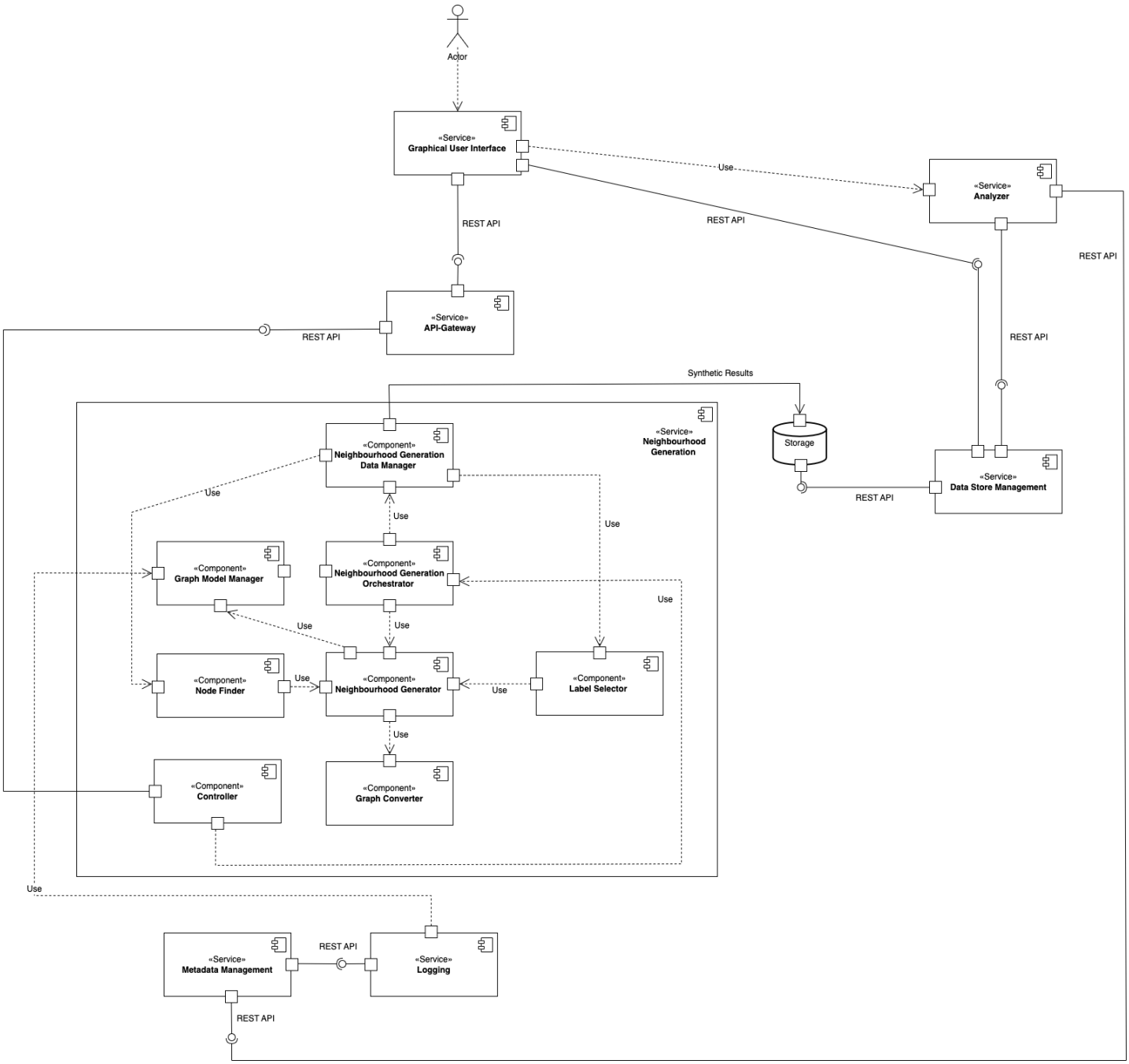


Abbildung A.4: Integration der Generierung synthetischer Nachbarschaften in die Gesamtarchitektur (aus Kapitel 6.2.1.2).

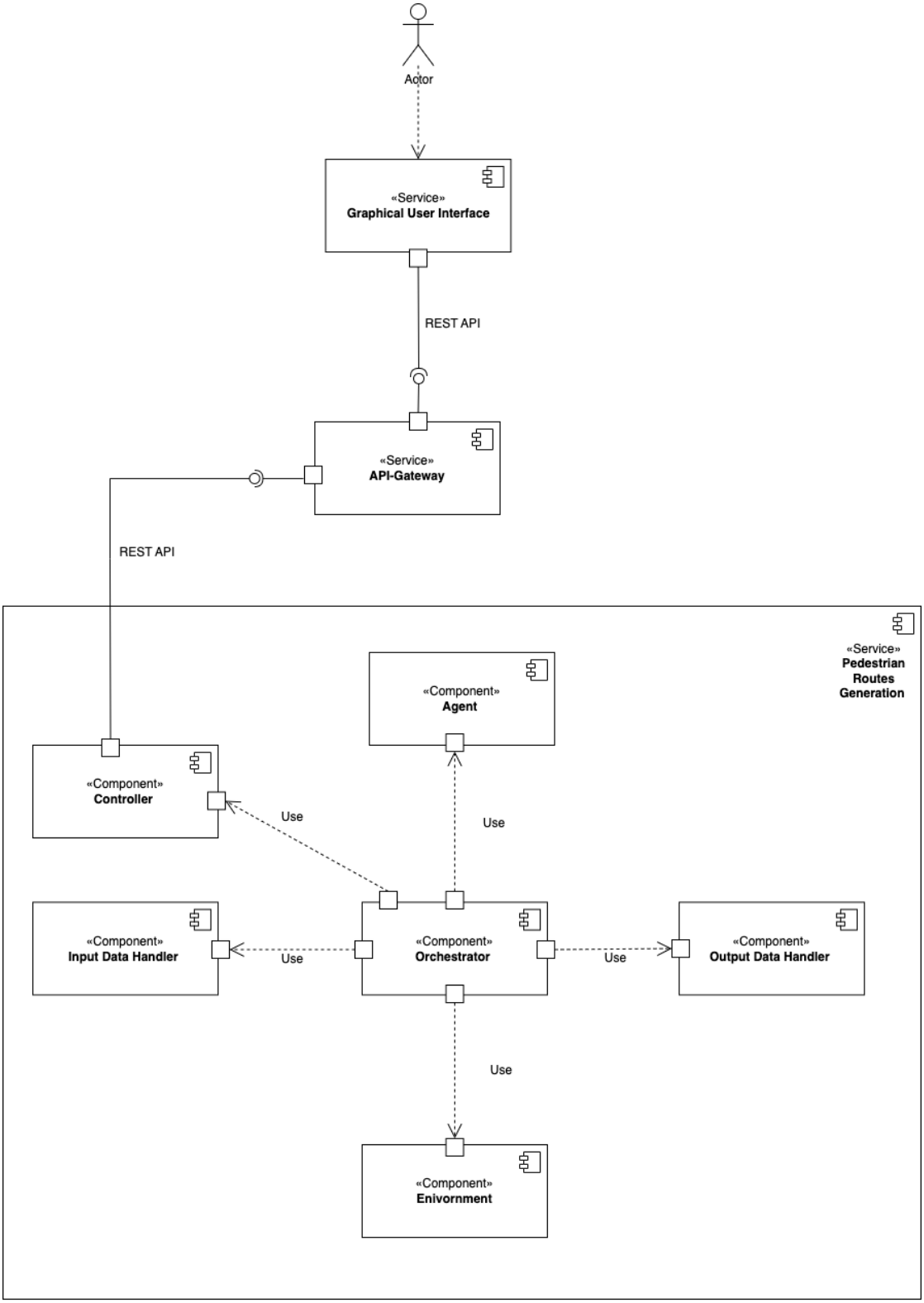


Abbildung A.5: Integration der Generierung synthetischer Fußgängerrouten in die Gesamtarchitektur (aus Kapitel 6.2.1.2).

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Masterarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Entwurf einer Softwarearchitektur zur Integration von Anwendungsfällen in eine Plattform zur Datensynthese

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original