

Masterarbeit

Vergleich datenbasierter Signalgenerierung mittels Machine Learning und Time Series Decomposition

eingereicht von

Felix Fischer
geboren am 15-07-1998 in Wernigerode

Technische Universität Dresden

Fakultät Informatik
Institut für Angewandte Informatik
Lehrstuhl Mensch-Computer-Interaktion



**TECHNISCHE
UNIVERSITÄT
DRESDEN**

Betreuer:
M.Sc. Emma Franziska Etzold

Hochschullehrer:
Prof. Dr. rer. nat. habil. Gerhard Weber

Eingereicht am 28. April 2024

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit mit dem Titel *Vergleich datenbasierter Signalgenerierung mittels Machine Learning und Time Series Decomposition* selbstständig unter Angabe aller Zitate angefertigt und dabei ausschließlich die aufgeführte Literatur und genannten Hilfsmittel verwendet habe.

Dresden, 28. April 2024

Felix Fischer



Masterarbeit

Vergleich datenbasierter Signalgenerierung mittels Machine Learning und Time Series Decomposition

Name, Vorname: Fischer, Felix

Studiengang: Informatik

Matrikelnummer: 4682953

Durch die Simulation von Signalen, bspw. Sensorwerten, können Einsatzszenarien für zeitabhängige Werteänderungen modelliert und Experimente durchgeführt werden. Mit dem Ziel, blinden Menschen die Überwachung von Live-Daten zu ermöglichen, werden im MONITOR-Projekt Konzepte der Datenpräsentation entwickelt und in simulierten Anwendungsszenarien evaluiert. Für letzteres existiert ein Anwendungsprototyp, der basierend auf den Kernaspekten der Zeitreihenzerlegung (Time Series Decomposition) nutzerdefinierte Daten sendet, um Signale zu simulieren. Dabei werden u.a. die Zeitreihenkriterien Trend, Zyklus und Abweichungen von Nutzenden festgelegt und dementsprechend die zu sendenden Daten generiert. Zur Simulation von realen Daten ist eine manuelle Vorabanalyse notwendig, um die erforderlichen Eingaben in der Anwendung zu tätigen. Eine automatisierte Analyse existierender Daten ist daher wünschenswert, um den Prozess der Datengenerierung effizienter zu gestalten.

Die Verwendung künstlich generierter (synthetischer) Daten hat einige Vorteile gegenüber der Verwendung realer Daten. So ist beispielweise neben datenschutzrechtlichen Aspekten das Sammeln großer Datenmengen unter Umständen sehr aufwändig. Synthetische Daten basieren oft auf einem Beispieldatensatz und weisen die gleichen Charakteristiken auf, wie die realen Daten. In verwandten Arbeiten erscheint der Einsatz von Machine Learning Algorithmen sehr vielversprechend bei der Erzeugung synthetischer Daten. Zur Erstellung von Signalen wird hingegen häufig der Ansatz des Time Series Decomposition verwendet.

Ziel dieser Arbeit ist es herauszufinden, welcher Ansatz zur datenbasierten Generierung von Signalen geeigneter ist: Machine Learning oder Time Series Decomposition.

Im Rahmen dieser Arbeit werden folgende Schwerpunkte dieses Forschungsfeldes adressiert:

1. Recherche und Analyse bestehender Literatur und aktuellen Möglichkeiten, um Daten basierend auf existierenden Datensätzen zu erzeugen, mit besonderem Augenmerk auf
 - a. Time Series Decomposition und
 - b. Machine Learning Algorithmen
2. Ermittlung von Anforderungen an eine Software, die Daten auf Grundlage eines bestehenden Datensatzes simuliert, sowohl technisch als auch benutzendenzentriert
3. Konzeption einer Lösung, um Daten mittels Machine Learning zu generieren
4. Erweiterung des bestehenden Prototyps zur Simulation von Sensoren um einen datenbasierten Ansatz zum Erstellen der Berechnungsvorschrift für zu sendenden Daten basierend auf
 - a. Time Series Decomposition und
 - b. Machine Learning.
5. Vergleich der beiden Ansätze hinsichtlich ausgewählter Aspekte der Datengenerierung sowie Benutzungsfreundlichkeit.

Relevante Literatur:

- [1] Kothare, A., Chaube, S., Moharir, Y., Bajodia, G., & Dongre, S. (2021, November). SynGen: Synthetic Data Generation. In *2021 International Conference on Computational Intelligence and Computing Applications (ICCI-CA)* (pp. 1-4). IEEE.
- [2] Gujar, S., Shah, T., Honawale, D., Bhosale, V., Khan, F., Verma, D., & Ranjan, R. (2022, June). GenEthos: A Synthetic Data Generation System with Bias Detection And Mitigation. In *2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS)* (pp. 1-6). IEEE.
- [3] Fan, K., Xu, B., Zhang, M., Nan, M., & Huang, J. (2018, October). A New Method for Time Series Signal Decomposition. In *2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)* (pp. 1-9). IEEE.
- [4] Hendahewa, D. N., Perera, A. A. D. S. N. A., De Meraal, T. D. N., Indikadulle, I. K. P. H., Perera, S. D., & Rathnayake, H. M. S. C. (2021, October). Comparative Analysis between Supervised Machine Learning and Time Series Models for stock price prediction. In *2021 International Conference on Data Analytics for Business and Industry (ICDABI)* (pp. 354-358). IEEE.

Fachbetreuende:

Emma Franziska Etzold, M.Sc.

Verantwortlicher Hochschullehrender:

Prof. Dr. Gerhard Weber

Institut:

Institut für Angewandte Informatik, Prof. MCI

Beginn am:

31.07.2023

Einzureichen am:

01.01.2024

 Gerhard Weber

Digital unterschrieben von
Gerhard Weber
Datum: 2023.07.20 15:32:57
+02'00'

Unterschrift des verantwortlichen Hochschullehrenden

Abstrakt

Die vorliegende Masterarbeit erforscht die Generierung synthetischer Datenströme unter Einsatz von Machine Learning-Methoden und traditioneller Zeitreihenzerlegung, insbesondere im Kontext des MONITOR-Projekts. Angesichts der zunehmenden Bedeutung synthetischer Daten, wie durch frühe Entwicklungen autonomer Fahrzeuge und deren Einsatz von KI illustriert, widmet sich diese Arbeit der Fragestellung, wie numerische Datenströme effektiv generiert werden können. Dabei liegt der Fokus auf der Simulation von Sensordaten in verschiedenen Szenarien.

Zentraler Bestandteil dieser Arbeit ist der Vergleich zwischen Machine Learning-Ansätzen und traditionellen mathematischen Modellen zur Zeitreihenzerlegung. Beide Methoden werden hinsichtlich ihrer Effizienz, Genauigkeit und Anwendbarkeit in der Simulation von Datenströmen evaluiert. Es wird untersucht, inwieweit generative oder rekursive Modelle im Vergleich zu einfacheren, ressourcenschonenden mathematischen Modellen wie ARIMA oder Empirical Mode Decomposition zur Rekonstruktion von Zeitreihen eingesetzt werden können.

Ein wesentliches Ziel der Arbeit ist die Entwicklung und Implementierung eines Konzepts, das beide Ansätze in einer bestehenden Simulationsumgebung integriert. Dieses Konzept soll nicht nur die technischen Aspekte abdecken, sondern auch benutzerfreundlich gestaltet sein, um auch Nicht-Experten im Bereich des maschinellen Lernens den Zugang zu ermöglichen.

Abschließend zielt diese Arbeit darauf ab, einen wissenschaftlichen und praktischen Beitrag zu leisten, indem sie komplexe Methoden der Datengenerierung zugänglicher macht und tiefere Einblicke in die Dynamik und Potenziale beider Ansätze bietet. Dadurch sollen innovative Anwendungen und Forschungsmöglichkeiten in verschiedenen Bereichen eröffnet werden. Die Ergebnisse dieser Arbeit sollen Nutzern helfen, fundierte Entscheidungen bei der Wahl der geeigneten Methode zur Datenstromgenerierung zu treffen.

Inhaltsverzeichnis

1 Einleitung	1
1.1 Motivation	1
1.2 Zielsetzung	2
2 Ausgangslage und Stand der Technik	3
2.1 Ansätze aus dem Bereich des maschinellen Lernens	3
2.1.1 SynGen: Synthetic Data Generation	3
2.1.2 GenEthos	3
2.1.3 Synthetic Test Data Generation Using Recurrent Neural Networks: A Position Paper	4
2.1.4 Generation of Synthetic Continuous Numerical Data Using Generative Adversarial Networks	4
2.2 Zeitreihenanalyse- und Zerlegung	5
2.2.1 The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis	5
2.2.2 Singular Spectrum Decomposition: a New Method for Time Series Decomposition	6
2.2.3 Synthetic Data by Principal Component Analysis	6
2.3 Auswertung des aktuellen Standes der Technik	7
3 Techniken	9
3.1 Zeitreihenzerlegung	9
3.1.1 Empirical Mode Decomposition	10
3.1.2 Singular Specturm Analysis	11
3.2 Maschinelles Lernen	11
3.2.1 Rekursive Modelle	12
3.2.2 Generative Modelle	12
4 Personas	15
4.1 Fathima Olsson	15
4.2 Dieter Maibach	16
5 Anforderungsanalyse der Software	19
5.1 Rückblick auf die originale Anwendung	19
5.2 Anforderungen an die Software	21
5.2.1 Datenmanagement	21
5.2.2 Trainieren und Konfigurieren von Modellen	21
5.2.3 Benutzerschnittstelle und Benutzererfahrung	22
5.2.4 Sicherheit und Privatsphäre	23

5.2.5	Technische Anforderung	23
5.2.6	Anforderungen an die Software	24
5.3	Konzeption der Software	25
5.3.1	Datenmanagementkonzept	25
5.3.2	Konzeption des Modelltrainings	26
5.3.3	Konzeption der Benutzeroberfläche	26
5.3.4	Konzeption des Sicherheitskonzeptes	27
5.3.5	Konzeption der technischen Anforderungen	27
6	Aufbau	29
6.1	Überblick über die Infrastruktur	29
6.2	Microservice-Architektur	29
6.3	Technologie-Stack	30
6.4	Zusammenfassung	33
7	Implementation	35
7.1	Überblick	35
7.2	Java Spring Boot	36
7.2.1	Architekturentscheidung und Umstrukturierung	36
7.2.2	Fehlerbehandlung	43
7.2.3	Absicherung der Anwendung über Spring Security	43
7.2.4	Bidirektionale Kommunikation mittels Websockets	45
7.2.5	Kommunikation mit der Django API über REST-Clients	46
7.3	Django	49
7.3.1	Django API Architektur	49
7.3.2	Absicherung der Anwendung über Djangos Sicherheitskonzept	50
7.3.3	Bidirektionale Kommunikation über Django Channels	51
7.3.4	Fehlerbehandlung und Verarbeitung	52
7.3.5	Validierung der Daten und Vorverarbeitung	53
7.3.6	Generalisierung und Nutzung der Machine Learning Modelle	55
7.3.7	Generalisierung und Nutzung der Zeitreihenanalysealgorithmen	57
7.3.8	Das Datenbank Modell	58
7.4	React Implementation	60
7.4.1	Grundlagen der UI	60
7.4.2	Grundideen des UI-Designs	60
7.4.3	Kommunikation mit den APIs	60
7.4.4	Eingabenvvalidierung über JSON Schema	61
7.4.5	Bereitstellung von generischen Komponenten für Nutzerfeedback	63
7.4.6	Bereitstellung von Mehrsprachigkeit durch i18n	63
7.4.7	Websockets und Ihre Auswirkungen für den Nutzer	64
7.4.8	Konfiguration der Komponenten im Bereich maschinelles Lernen	64
7.4.9	Konfiguration der Komponenten im Bereich der Zeitreihenanalyse	66
7.4.10	Verwalten der hochgeladenen Daten	66

8 Aufbau der Testumgebung	73
8.1 Metriken und Methoden der Testumgebung	73
8.1.1 Nicht-Statistische Maße	73
8.1.2 Statistische Maße	74
8.1.3 Evaluation durch maschinelles Lernen	75
8.2 Bewertung der Methoden	76
8.2.1 Clustering	78
8.2.2 Getestete Modelle	80
8.2.3 Zeitreihenanalyse Algorithmen und Modelle	81
8.3 Deployment	82
8.3.1 Deployment der Anwendung	82
8.3.2 Automatisierung der Deployment Pipeline	82
9 Ereignisse und Auswertung der Modelle und Algorithmen	85
9.1 Trainingsdaten	85
9.2 Vergleich der Algorithmen und Modelle	87
9.2.1 Performance Metriken der ML Modelle	87
9.2.2 Performance Metriken der TSA Algorithmen	89
9.2.3 Auslastung der Hardware in Produktion	89
9.2.4 Ähnlichkeit der Zeitreihen der Machine Learning (ML) Modelle	90
9.2.5 Ähnlichkeit der Zeitreihen der TSA Modelle	96
9.3 Vorhersagefähigkeiten der rekursiven Modelle	98
9.4 Zusammenfassung	98
10 Nutzerstudie	103
10.1 Inhalte der Studie	103
10.2 Studienablauf	104
10.3 Auswertung der Studie	105
10.3.1 Ergebnisse der Thinking Aloud Studie	105
10.3.2 Auswertung der Fragebögen	106
11 Ausblick	109
11.1 Einführung neuer Modelle	109
11.2 Erweiterungsmöglichkeiten innerhalb des ML Bereiches	109
11.3 Erweiterungsmöglichkeiten außerhalb des TSA Bereiches	110
11.4 Erweiterungsmöglichkeiten innerhalb des Frontends	110
11.5 Testabdeckung	110
12 Fazit	111
12.1 Ergebnisse der Nutzererfahrung	111
12.2 Auswertung der umgesetzten Anforderungen	111
12.3 Ergebnisse der Evaluation	113
12.3.1 Schlusswort	114
A Appendix	i
A.1 Weitere Tabellen	i

List of Abbreviations	v
Abbildungsverzeichnis	vii
Tabellenverzeichnis	xi
Codeverzeichnis	xiii
Liste an Formeln	xv
Literatur	xvii

1 Einleitung

Die Bedeutung synthetischer Daten wurde bereits in den frühen Entwicklungsphasen autonomer Fahrzeuge in den 1980er Jahren deutlich. Ein markantes Beispiel hierfür ist der Navlab I”[Zee23], ein experimentelles autonomes Fahrzeug der Carnegie Mellon University, welches mit vier leistungsstarken Computern ausgestattet über den Campus fuhr und somit eine Schlüsselrolle in der Evolution der künstlichen Intelligenz spielte. Die Herausforderung, verschiedene Fahrszenarien allein mit einem Satz von Instruktionen abzudecken, die manuell gesammelt werden mussten, führte zu einem bedeutsamen Paradigmenwechsel hin zum Einsatz von maschinellem Lernen. Dean Pomerleau nutzte, anstatt alle Situationen selbst aufzubauen, ein Set an Daten, um weitere Situationen durch synthetische Daten zu generieren. Seine Pionierarbeit in der Entwicklung neuronaler Netze, die mit synthetischen Straßenbildern trainiert wurden, illustriert eindrucksvoll die wachsende Bedeutung von synthetischen Daten.

In der heutigen Ära, die durch einen erneuten AI-Boom und die Verfügbarkeit von leistungsfähigen Computerplattformen gekennzeichnet ist, gewinnen synthetische Daten weiter an Bedeutung. Vor diesem Hintergrund widmet sich diese Arbeit der Untersuchung des Mehrwerts von maschinellem Lernen bei der Generierung von numerischen Datenströmen und stellt diese Methodik der traditionellen Zeitreihenzerlegung gegenüber.

Diese Arbeit untersucht Machine Learning-Methoden im Vergleich zur traditionellen Zeitreihenzerlegung zur Generierung von numerischen Datenströmen, im Kontext des MONITOR-Projekts, das die Entwicklung und Evaluation von Datenpräsentationskonzepten in simulierten Anwendungsszenarien verfolgt. Der Fokus liegt auf der Simulation von Signalen, speziell Sensorwerten, die zeitabhängige Veränderungen in verschiedenen Szenarien modellieren. Ziel ist es, die Wirksamkeit und Effizienz von Machine Learning gegenüber der Time Series Decomposition zu bewerten, Methoden zu analysieren, Anforderungen an entsprechende Software zu formulieren und einen Lösungsansatz basierend auf Machine Learning zu entwickeln. Zusätzlich wird ein bestehender Prototyp zur Sensordatensimulation erweitert, um beide Ansätze zu implementieren und zu vergleichen.

1.1 Motivation

Die Simulation von Daten stellt eine komplexe Herausforderung dar, die sich durch verschiedene Ansätze charakterisiert und in Genauigkeit, Geschwindigkeit und Komplexität unterscheidet.

Aber zum ersten Thema. Die Frage, welcher Ansatz der richtige ist, hängt von zahlreichen Faktoren ab und ist von entscheidender Bedeutung in einer datengetriebenen Welt.

In den folgenden Kapiteln dieser Arbeit werden verschiedene Methoden zur Generierung synthetischer Daten aus bestehenden Datenquellen diskutiert. Insbesondere rücken generative Modelle, die

bereits in vielen Bereichen der Bild- und Tabellengenerierung eingesetzt werden, in den Vordergrund. Diese Modelle müssen nur einmal trainiert werden und können daraufhin Daten er-

zeugen, die den Originaldaten sehr nahekommen. Jedoch stellt dies auch hohe Anforderungen an die Rechenkapazität und ist in seiner Komplexität nicht zu unterschätzen. Besonders im Bereich der Zeitreihenanalyse scheinen diese Ansätze noch nicht umfassend erforscht zu sein, was den Einstieg in diesen speziellen Forschungsbereich spannend macht.

Eine Alternative bieten mathematische Modelle, die zwar weniger flexibel, aber deutlich ressourcenschonender sind. Solche Modelle, die bereits in verschiedenen Sektoren wie im Finanzwesen zur Vorhersage von Marktentwicklungen eingesetzt werden, könnten auch zur Rekonstruktion von Zeitreihen genutzt werden. Modelle wie ARIMA oder Facebooks Prophet, die darauf abzielen, Informationen aus Daten zu extrahieren, könnten somit verwendet werden, um eine den Originaldaten nahe kommende Zeitreihe zu generieren. Der Vergleich dieser einfacheren Ansätze mit den komplexeren generativen Modellen ist daher ein Kernaspekt dieser Arbeit.

In dieser Masterarbeit sollen die beiden Ansätze, Machine Learning und Zeitreihenzerlegung, gegenübergestellt werden. Beide haben ihre spezifischen Vor- und Nachteile, und die Wahl des geeigneten Ansatzes hängt von den jeweiligen Anforderungen ab. Das Ziel dieser Arbeit ist es daher, Erkenntnisse zu liefern, die die Entscheidungsfindung für die geeignete Methode erleichtern.

Darüber hinaus soll ein Konzept entwickelt und umgesetzt werden, das diese Ansätze praktikabel macht und sie in eine existierende Simulationsumgebung integriert. Dieses Konzept soll nicht nur die technischen Aspekte abdecken, sondern auch benutzerfreundlich gestaltet sein, sodass auch Personen ohne tiefgehende Kenntnisse im Bereich des maschinellen Lernens davon profitieren können. Der Vergleich der Ansätze wird aufzeigen, welcher Weg für bestimmte Anforderungen am besten geeignet ist, und Nutzern helfen, eine fundierte Entscheidung zu treffen.

Diese Arbeit soll nicht nur einen wissenschaftlichen Beitrag leisten, sondern auch praktische Anwendungen in verschiedenen Bereichen ermöglichen, indem sie komplexe Methoden zugänglicher macht. Die Motivation dahinter ist es, ein tieferes Verständnis für die Dynamik und Potenziale beider Ansätze zu schaffen und so die Tür für innovative Anwendungen und Forschung in der Zukunft zu öffnen.

1.2 Zielsetzung

Es handelt sich um eine zweiteilige Arbeit. Einerseits muss die bestehende Simulationsumgebung in vielen Punkten verändert und erweitert werden, da sie aktuell

nur die Generierung von Daten mittels Zeitreihenzerlegung beinhaltet und fast keine Flexibilität in der Art der gesendeten Daten bietet. Auch muss das Gesamtprojekt durch eine weitere Application Programmable Interface (API) zur Bereitstellung der Methoden des maschinellen Lernens und der Zeitreihenanalyse erweitert werden. Diese muss sich in die bestehende Architektur integrieren lassen und die Möglichkeit bieten, die Daten in derselben Form wie die Zeitreihenzerlegung zu erhalten. Auch steht eine nutzerfreundliche Oberfläche im Vordergrund, welche es gerade fachfremden Nutzern ermöglicht, diese Software zu verwenden.

Auf der anderen Seite steht eine Evaluation der bestehenden Konzepte und Methoden an. Es muss analysiert werden, wann und warum es sinnvoll ist, sich für eine der beiden Varianten zu entscheiden. Hierzu müssen die Vor- und Nachteile der beiden Ansätze gegenübergestellt werden und die Ergebnisse der beiden Ansätze verglichen werden.

2 Ausgangslage und Stand der Technik

2.1 Ansätze aus dem Bereich des maschinellen Lernens

2.1.1 SynGen: Synthetic Data Generation

SynGen, ein von Akash Kothare et al.[Kot+21] entwickeltes Instrument zur Synthese von Daten, wurde auf der ICCICA 2021 (International Conference on Computational Intelligence and Computing Applications) präsentiert. Das Tool folgt einem dreistufigen Prozess zur Generierung von Benutzerdaten, der Flexibilität und Anpassungsfähigkeit in der Datenproduktion bietet.

In der Initialisierungsphase bietet SynGen Benutzern die Wahl, entweder ein Beispielset zu importieren oder durch die Spezifikation von Feldnamen und der gewünschten Zeilenzahl einen neuen Datensatz zu generieren, wobei letzteres durch den Einsatz von Faker[Koh04] als Grundlage für die Datenerstellung dient.

Daraufhin wird im zweiten Schritt des Prozesses die maschinelle Lernstrategie gewählt. Das Tool stellt verschiedene Implementierungen des überwachten Lernens für Regression und Klassifikation sowie des unüberwachten Lernens zur Verfügung. Diese Auswahl ermöglicht es dem Benutzer, die Methode zu wählen, die am besten zu den Anforderungen der spezifischen Daten und des intendierten Anwendungsfalls passt.

Im abschließenden Verfahrensschritt ermöglicht der Similarity Index der Modelle eine Evaluation der Leistungsfähigkeit der verschiedenen Algorithmen. Um diese Bewertung zu vereinfachen, verwendet SynGen ein Effizienzvergleichsdiagramm sowie eine Konfusionsmatrix, die die Genauigkeit der Modelle illustriert und somit eine direkte und nutzerfreundliche Methode zur Bewertung der Modellqualität bietet.

2.1.2 GenEthos

Im Rahmen der fortschreitenden Entwicklung synthetischer Datengenerierungssysteme stellt das Werk von Shubham Gujar et al.[Guj+22], "GenEthos", ein innovatives GUI-basiertes Tool dar. Dieses System zeichnet sich durch seine Fähigkeit aus, nicht nur synthetische Daten zu erstellen, sondern auch eine kontinuierliche Überwachung und Bias-Detektion für die generierten Modelle zu ermöglichen. Zur Bewahrung logischer Konsistenzen innerhalb der erzeugten Datenmengen integriert GenEthos die Principal Component Analysis (PCA) und Learning Fair Representations (LFR). PCA dient hier als eine Technik zur Dimensionsreduzierung, die Daten aus einem hochdimensionalen in einen niedrigeren Raum transformiert, während die maximale Informationsmenge erhalten bleibt[Whi23]. LFR wird als Vorverarbeitungsmethode verwendet, um diskriminierende Merkmale aus den Daten zu entfernen[Ric13].

Darüber hinaus präsentieren die Autoren ein benutzerdefiniertes Konzept für die kategorische Erstellung von Benutzeroberflächen, das es dem Nutzer ermöglicht, die Datenbeziehungen durch vordefinierte Ausdrücke wie Zufallsfunktionen oder Bedingungen anzupassen oder sogar eigene

Python-Funktionen zu implementieren, um Verknüpfungen zwischen den Tabellenspalten zu modifizieren oder zu generieren.

Die Datenerstellung erfolgt unter Verwendung von Faker[Koh04] für den anfänglichen Input und fünf unterschiedlichen Modellen für die Datenmodellierung:

T-GAN bietet eine auf tabellarische Daten spezialisierte GAN-Variante.

Gretel verwendet LSTM-basierte Netzwerke und legt den Fokus auf Datenschutz.

CTGAN repräsentiert einen weiteren GAN-basierten Ansatz für die Modellierung tabellarischer Daten und die Generierung von Datenzeilen aus einer Verteilung.

Zur Bewertung von Fairness und zur Minderung ethischer Voreingenommenheit wurden zusätzlich Metriken wie Statistical Parity Difference und Disparate Impact sowie Algorithmen wie Prejudice Remover, Disparate Impact Remover und Learning Fair Representations verwendet. Diese Methoden liegen jedoch außerhalb des Umfangs dieser Arbeit.

In der Evaluation ihres Systems verwendeten die Forscher die Adult- und German Credit Datensätze, um Fairness und Bias zu untersuchen und führten Vergleiche mit synthetischen Daten aus den oben genannten Modellen durch. Alle drei Modelle replizierten die vorhandenen Daten mit geringfügigen Bias-Abweichungen effektiv. Die Hauptzielsetzung der Studie war es, den Beitrag von Algorithmen zur Bias-Entfernung zu bewerten. Dabei wurde festgestellt, dass zwar eine Reduktion von Bias möglich ist, diese jedoch ihre Grenzen hat.

2.1.3 Synthetic Test Data Generation Using Recurrent Neural Networks: A Position Paper

In der Untersuchung von Razieh Behjati et al.[Beh+19] wird der Einsatz von Long Short-Term Memory (LSTM) Netzwerken zur Generierung von synthetischen Testdaten für ereignisgesteuerte Systeme beleuchtet. Diese Forschung findet im Rahmen des Upgrades des norwegischen nationalen Registers statt, einem Kontext, der eine hohe Sensibilität für den Datenschutz erfordert. Synthetische Daten dienen hier als Alternative zu realen Produktionsdaten, insbesondere wenn letztere persönliche Informationen enthalten, die nicht für Tests freigegeben werden können.

Das Team hat einen Ansatz verfolgt, bei dem Eingabedaten in sogenannte Meta-Events segmentiert wurden, die jeweils eine Klassifizierung und assoziierte personenbezogene Daten enthielten. Diese methodische Aufteilung ermöglichte es dem LSTM-Modell, die Struktur und essentiellen Muster der Daten – wie die Zuordnung von Monaten zu deren maximaler Tagesanzahl – effektiv zu lernen.

Während das Modell Basisabhängigkeiten mit Erfolg reproduzierten konnte, zeigte sich bei der Abbildung komplexerer Regeln, wie der Berücksichtigung von Schaltjahren, eine gewisse Begrenzung. Dies lässt sich potenziell auf die Größe des Trainingsdatensatzes zurückführen und weist auf den Bedarf an umfangreicheren Daten für das Training fortgeschrittener Sequenzmodellierung hin.

2.1.4 Generation of Synthetic Continuous Numerical Data Using Generative Adversarial Networks

In ihrer Studie untersuchen A H Aziira und Kollegen[ASS20] die Anwendung von Generative Adversarial Networks (GANs) zur Erzeugung kontinuierlicher numerischer Daten für unüberwachte

maschinelle Lernprozesse (Unsupervised Machine Learning Tasks). Die Kernforschung evaluiert die Effizienz von GANs und Conditional GANs (CGANs) bei der Generierung authentischer Datensätze durch ein gezieltes Experiment mit begrenztem Umfang.

Für das Experiment bedienten sich die Autoren des öffentlichen Datensatzes WornBlade002¹, um die Performance eines GAN und eines CGAN zu trainieren und zu beurteilen. Die Authentizität der generierten Daten wurde durch zwei komplementäre Methoden geprüft: qualitative Bewertungen zur Beurteilung der Übereinstimmung der Datenverteilung mit dem Originaldatensatz und quantitative Maßnahmen zur Einschätzung der Realitätsnähe der synthetisierten Daten. Letztere basierten auf dem XGBoost-Algorithmus – einem Gradient Boosting Framework für baumbasierte Lernmodelle –, der auf einer Kombination von echten und generierten Daten trainiert wurde. Die Genauigkeit der Modelle wurde mittels einer Konfusionsmatrix evaluiert, mit dem Ziel, eine Genauigkeit (Accuracy) von 50% zu erreichen, was eine Ununterscheidbarkeit zwischen echten und generierten Daten suggerieren würde.

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} * 100\% \quad (2.1)$$

Zusätzlich wurde der Mann-Whitney-U-Test angewandt, um die statistische Signifikanz der Unterschiede zwischen den echten und generierten Datenstichproben zu ermitteln und Tendenzen der Abhängigkeit aufzudecken[Zür23].

Die Ergebnisse des Experiments zeigen, dass nach ungefähr 10.000 Trainingsschritten die Genauigkeitswerte der GANs und CGANs nur noch marginal verbessert werden konnten. Mit einer maximalen Genauigkeit von 55% für die CGANs demonstrieren die Ergebnisse eindrucksvoll, dass GANs prinzipiell fähig sind, kontinuierliche numerische Daten mit hoher Qualität zu generieren.

2.2 Zeitreihenanalyse- und Zerlegung

2.2.1 The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis

Huang u.w.[Hua+98] arbeiteten an einer neuen Methode um komplexe, nicht-lineare¹ und nicht-stationäre² Daten zu analysieren.

Dafür nutzten sie eine Empirical Mode Decomposition (EMD), eine Methode zum Zerlegen von komplexen Daten endliche, meist kleine Anzahl von Intrinsic Mode Functions (IMFs), welche zwei Regeln befolgen müssen.

1. Die Anzahl an Maxima und Nullpunkten sollte möglichst 0 sein.
2. An jedem Punkt sollte die Summe des lokalen Maximums und Minimums 0 sein.

Hierdurch werden Maxima und Minima mit Splines verbunden und daraufhin die Mittelwerte der Maxima- und Minimaanstiege aus dem Signal entfernt. Dieser Prozess ist iterativ und wiederholt sich, bis keine Maxima mehr vorhanden sind. Da dies teilweise zu Problemen führt, wurde

¹Dieser Datensatz stammt von Schneidmessern der Vega Schrumpffolienproduktion und dient der Überwachung der Maschinenperformance.

²Bei nicht-linearen Daten zeigt die Beziehung zwischen abhängigen und unabhängigen Variablen keine gerade Linie.

²Im Kontext von Time Series sind nicht-stationäre Daten Segmente oder Teile, die unabhängig von der Zeit Ihre Werte besitzen.

der Algorithmus 2003 von Huang Daji u.w.[DJ03] angepasst, indem eine **EEM!s (EEM!s)** eingeführt wurde, welche das Signal um erste und letzten lokale Extrema erweitert. Diese werden bei der späteren Berechnung besonders behandelt.

Aus den einzelnen IMFs, welche man mit oszillierenden Funktionen vergleichen kann, werden genutzt um durch eine Hilbert Spectral Analyses (HSAs) über einen Zeitparameter das Signal zu rekonstruieren. Die Summe der einzelnen Funktionen kann somit das originale Signal rekonstruieren.

$$X(t) = \sum_{i=1}^M IMF_i + r(t) \quad (2.2)$$

Während das Hilbert-Huang Transforms (HHTs), wie die Methode auch genannt wird, viele verschiedene Einsatzgebiete, von Biomedizin über Ozeanographie bis Seismographie, hat, ist gerade die Glättungsfunktion der EMD in vielen Bereichen besonders relevant.

2.2.2 Singular Spectrum Decomposition: a New Method for Time Series Decomposition

In [BON+14] stellten PIETRO BONIZZI Pietro Bonizzi u.w. eine neue Methode zur Aufspaltung von nichtlinearen, nichtstationären Zeitreihen vor.

Aufbauend auf Singular Specturm Analysis (SSA) (siehe 3.1.2), lieferten sie einen innovativen Ansatz, welcher das bestehende SSA Prinzip übernimmt, den dahinterliegenden Algorithmus aber überarbeitet. So wird beispielsweise die Trajektorienmatrix angepasst und fokussiert sich stärker auf die oszillierenden Komponenten des Signales. Im Zerlegungsprozess ist ein Fokus auf die Frequenzen somit einfacher und dies erlaubt wiederum aussagekräftige Komponenten zu isolieren. Eine weitere wichtige Neuerung ist die Auswahl von Hauptkomponenten zu automatisieren. Dieser in SSA noch manuelle Schritt

Es erlaubt somit einen Fokus auf die Frequenzen und kann somit besser die Frequenzen isolieren aber die Auswahl der Parameter wie window-length automatisiert und somit nicht für die jeweiligen Daten neu angepasst werden muss.

2.2.3 Synthetic Data by Principal Component Analysis

In dem Paper [San20] zur synthetischen Datengenerierung stellt Natsuki Sano zwei Methoden vor. Beide zielen darauf ab aus realen Daten (Daten aus von Decathlon organisierten Sport Events wie Weitsprung, Hochsprung und Sprint), die persönlichen Aspekte zu entfernen, aber die Eigenschaften der Daten an sich beizubehalten. Dies steht entgegen dem Konzept des Maskierens von Daten um Anonymität zu erreichen. Für die generierung wird ein linearer und ein nicht linearer Ansatz vorgesetzt.

Orthogonale Transformation ist eine Methode welche auf Principal Component Analysis (PCA) setzt. PCA ist eine statistisches Analyseverfahren, welches große Datenmengen in kleinere, den inhalt zusammenfassende Sets aufteilt. Diese sind dadurch leichter zu analysieren. Aber PCA ist lediglich in der Lage den linearen Zusammenhang zwischen Variablen zu analysieren und darzustellen. Interessant ist dieser Ansatz, da er eine direkte Bewertung des Verfahrens zulässt. Durch die im Prozess verworfenen Eigenwerte kann der Informationsverlust bestimmt werden.

Sandglass-Type Neural Networks Für nicht-lineare Zusammenhänge kann normale PCA nicht eingesetzt werden. Da aber nicht alle Zusammenhänge innerhalb der Daten linearer Natur sind, wird ein anderer Ansatz gebraucht, dier kommt in Form von Sandglass Neural Networks. Diese Sanduhr Förmigen Netzwerke besitzen eine stark eingeschränkte Zwischenschicht, ähnelt daher einer Sanduhr. Dies sorgt dafür, dass die originalen Daten in der Zwischenschicht stark reduziert werden um dann in der Ausgangsschicht aus der komprimierten Version die Daten wieder zu rekonstruieren. Somit kann Machine Learning im PCA Verfahren eingesetzt werden komplexe Zusammenhänge zu finden.

Zur Messung der Ergebnisse setzt Sano auf vier Methoden. Mean Absolute Error (MAE) kann über eine große Menge an Daten den durchschnittlichen Absoluten Fehler berechnen und somit einen Einblick in die Daten geben, Mean Absolute Error des Mittelwerts einzelner Variablen (MAEM) um zu schauen ob sich die Grundtendenz der synthetischen Daten sich noch an den originalen Daten richtet, Mean Absolute Error der Kovarianz zwischen Variablen (MEAC) um zu schauen wie gut die Beziehungen zwischen den Variablen erhalten bleiben. Zusätzlich zu diesen schlägt der Autor noch ein eigenes Methode vor um den Informationsverlust über verworfene Dimensionen, welche er aber nicht genauer erklärt.

Die Ergebnisse beider Methoden zeigen eine klare Überlegenheit des linearen Verfahrens, da hier der Informationsverlust geringer ist. Dies wird aber auf die geringe Datenmenge zurückgeführt.

2.3 Auswertung des aktuellen Standes der Technik

Die hier vorgestellten Ansätze und Werkzeuge spiegeln den aktuellen Stand der Forschung wider und bieten Einblicke in die fortschrittlichen Methoden, die in diesen Feldern angewendet werden.

Im Bereich der synthetischen Datengenerierung wurden unterschiedliche Tools und Methoden wie SynGen, GenEthos, und Ansätze unter Verwendung von LSTM-Netzwerken und GANs beleuchtet. Diese Technologien zeichnen sich durch ihre Fähigkeit aus, realistische, anpassungsfähige und datenschutzkonforme synthetische Daten zu erstellen. Besonders betont wird dabei die Bedeutung von Flexibilität, Fairness, Bias-Detektion und die Anpassung an verschiedene Anwendungsfälle, von der Bias-Minimierung bis hin zur Maschinenleistungsüberwachung.

Der zweite Schwerpunkt liegt auf der Zeitreihenzerlegung, wobei Methoden wie die Empirical Mode Decomposition (EMD), Hilbert-Huang-Transformation (HHT) und Singular Spectrum Analysis (SSA) hervorgehoben werden. Diese Methoden sind entscheidend für das Verständnis und die Analyse von nicht-linearen und nicht-stationären Zeitreihen. Sie ermöglichen eine detailliertere Untersuchung komplexer Datensätze durch ihre Zerlegung in fundamentale Komponenten.

Die Bewertung dieser Methoden und Technologien erfolgt sowohl qualitativ als auch quantitativ. Dabei stehen die Übereinstimmung der Datenverteilungen, die Genauigkeit der Modelle und die statistische Signifikanz im Vordergrund. Diese Bewertungskriterien sind entscheidend, um die Wirksamkeit und Zuverlässigkeit der Ansätze zu bestätigen.

3 Techniken

3.1 Zeitreihenzerlegung

Time Series Decomposition ist ein statistisches Verfahren, welches eine Zeitreihe, wie beispielsweise ein Signal, in mehrere Komponenten zerlegt, die jeweils ein zugrunde liegendes Muster darstellen. Hauptsächlich wird von vier Komponenten gesprochen:

Trend oder auch Anstieg/Level. Der Trend einer Funktion gibt an, wie sich die Funktion auf lange Zeit verhält. Steigt eine Funktion über einen langen Zeitraum konstant an, liegt beispielsweise ein linearer Anstieg vor. Andere Anstiege wie exponentieller oder logarithmischer Anstieg sind auch möglich.

Season oder auch wiederkehrendes 'saisonales' Verhalten. Eine Funktion besitzt ein saisonales Verhalten, wenn sie in periodisch wiederkehrenden Abständen ein gleiches Verhalten aufweist. Besitzt ein Signal eine Sinus-komponente, so lässt sich diese gut beobachten.

Cyclic oder auch zyklische Komponente. Im Gegensatz zu seasonal-component misst die cyclic-component alle wiederkehrenden, aber nicht periodischen Schwankungen eines Signals.

Residual oder auch irreguläre Abweichung/Noise. Diese Komponente zeigt einen irregulären Einfluss auf das Signal zu jedem Zeitpunkt t und lässt sich durch die übergebliebenen Werte repräsentieren, welche nach der Entfernung der drei vorherigen Komponenten noch vorhanden sind.

Zur Zerlegung einer Zeitreihe kann zwischen zwei Techniken gewählt werden: der additiven und der multiplikativen Zerlegung. Die additive Zerlegung wird verwendet, wenn die Schwankungen in der Zeitreihe proportional zum Trend nicht variieren. Das bedeutet, dass die Werte der zyklischen oder saisonalen Schwankungen sich unabhängig vom Zeitpunkt nicht verändern [Rob18]. Die multiplikative Zerlegung wird angewendet, wenn die Schwankungen in der Zeitreihe proportional zum Niveau der Zeitreihe sind, also mit dem Trend ansteigen oder abfallen.

Die Zerlegung einer Zeitreihe in ihre Komponenten kann helfen, Muster in den Daten zu erkennen und Vorhersagen zu treffen.

Die additive Zerlegung kann wie folgt dargestellt werden:

$$y(t) = T(t) + S(t) + e(t) \quad (3.1)$$

wobei $y(t)$ die Zeitreihe ist, $T(t)$ der Trend, $S(t)$ die saisonale Komponente und $e(t)$ der Rest ist.

Die multiplikative Zerlegung kann wie folgt dargestellt werden:

$$y(t) = T(t) \times S(t) \times e(t) \quad (3.2)$$

wobei $y(t)$ die Zeitreihe ist, $T(t)$ der Trend, $S(t)$ die saisonale Komponente und $e(t)$ der Rest ist.

Die Zerlegung einer Zeitreihe in ihre Komponenten kann helfen, Muster in den Daten zu erkennen und Vorhersagen zu treffen.

3.1.1 Empirical Mode Decomposition

Neben der reinen Zerlegung von Signalen in die vier zuvor genannten Komponenten existieren auch Methoden, die sich stärker auf die Analyse des Signals selbst konzentrieren. Eine solche Methode ist die Empirical Mode Decomposition. Die EMD ist ein Algorithmus, der komplexe Daten analysiert und bei der Rekonstruktion das originale Signal wiederherstellt. Dabei zerlegt sie das ursprüngliche Signal in eine Reihe einfacherer Signale, die sogenannten Intrinsic Mode Function (IMF). In ihrer Summe bilden diese IMF wieder das Originalsignal ab. Eine der großen Stärken des EMD-Algorithmus ist seine Anpassungsfähigkeit an verschiedene Datensätze. Anstatt Annahmen über die Daten zu treffen, wie beispielsweise darüber, ob das Signal zufällig ist oder einer bestimmten Logik folgt, lernt EMD direkt aus den Daten. Diese Flexibilität macht ihn für vielfältige Anwendungen geeignet.

Der Algorithmus funktioniert wie folgt:

1. Identifiziere lokale Extrema (Minima und Maxima)
2. Interpoliere die Extrema und erzeuge zwei Hüllenkurvenfunktionen
3. Berechne den Mittelwert beider Funktionen: $m(t)$
4. Subtrahiere $m(t)$ von den originalen Daten r : $h(t) = r(t) - m(t)$
5. Teste ob $h(t)$ die Kriteria einer IMF erfüllt und gehe zu 1 wenn nicht
 - Die Anzahl an Extrema und Nullcrossing muss gleich sein oder sich um maximal eins unterscheiden.
 - An jedem Punkt sollte der Mittelwert zwischen Anstieg der Extrema fast null sein
6. Entferne $h(t)$ von den originalen Daten und wiederhole bis $r(t)$ konstant oder monoton ist.

Über diesen Algorithmus kann somit das Signal in der folgenden Form dargestellt werden:

$$X(t) = \sum_{i=1}^M h_i + r(t) \quad (3.3)$$

Leider ist dieser Algorithmus, sowie seine Erweiterung, die Hilbert-Huang Transform (HHT), nicht ohne Probleme. Es gibt einige Herausforderungen, an denen noch gearbeitet wird. Ein Beispiel hierfür ist, dass Signale die gleiche Frequenz teilen können. In solchen Fällen können die IMFs nicht immer die einzelnen Komponenten korrekt isolieren, was zu Fehlern führen kann. Ein ausführliches Beispiel hierzu liefern Raymond Ho und Kevin Hung in [HH22].

3.1.2 Singular Specturm Analysis

Singular Specturm Analysis ist eine weitere Methode um Time Series zu analysieren. Sie verbindet Elemente der klassischen Analyse, multivarianter Statistik und Geometrie, dynamischer Systeme und der Signalverarbeitung.[GNZ01] SSA findet Einsatz in der Meterorologie und Klimatologie, ist aber noch nicht so weit verbreitet wie andere Ansätze, da es, wie die Autoren von [GNZ01] sagen, eine exemplarisches Modellbautool ist und darauf abzielt kleinere, unabhängig interpretierbare Komponenten aus dem Original herauszulösen.

Um ein Signal in diese einzelnen Teile zu zerlegen, wird es in grob 4 Schritten zerlegt.

Embedding

Die Daten werden in n 'delayed vectors' zerlegt mit einer Länge L , sodass die Vektoren jeweils der Form

$$\{x_1, x_2, \dots, x_L\}, \{x_2, x_3, \dots, x_{L+1}\}, \dots, \{x_{n-L+1}, \dots, x_n\} \quad (3.4)$$

entsprechen. Daraus wird dann eine Matrix

$$X = LxK$$

mit $K = n - L + 1$ gebaut.

Singular Value Decomposition

Auf der Matrix X wird eine Singular Value Decompositions (SVDs) durchgeführt und es entstehen somit drei neue Matrizen, U , X und V . Die Matrix U , eine LxL Matrix, besteht aus den Eigenvektoren von XX^T . S , eine Singulärwertematrix, die die Wurzeln der Eigenwerte von XX^T enthält und V mit KxK , welches aus den Eigenvektoren von X^TX besteht.

Grouping

Dieser Schritt gruppiert die Spalten der Matrix U auf in r einzelne Eigenzeitreihen, welche jeweils separate Komponenten des ursprünglichen Signals repräsentieren sollen.

Diagonal Averaging

In vereinfachter Form kann dieser Schritt auch als Rekonstruktion beschrieben werden. Hier wird mit Hilfe eines inversen SSA versucht, das originale Signal wieder aufzubauen, indem man eine Matrix A baut, welche die Eigenzeitreihen mit ihren entsprechenden Eigenvektoren aus S und der Summe über der Diagonalen aus V multipliziert.

3.2 Machinelles Lernen

Machine Learning und Artifical Intelligence werden wohl zu den meist benutzten Begriffen gehören, die im Bereich Computer Science genutzt werden. Machine Learning im speziellen ist für viele Sektoren relevant, da es eine Möglichkeit bietet aus großen Datensätzen Muster und Zusammenhänge zu extrahieren, ohne diese im speziellen kennen zu müssen. Dies ist dem Umstand

geschuldet, dass ML Algorithmen zur Problemlösung nicht auf fester Regeln zurückgreifen, sondern diese versuchen selber zu definieren, zu erlernen. Um die Regeln aber erlernen zu können, müssen diese Algorithmen erst einmal trainiert werden. Hier kann grob zwischen drei verschiedenen Leveln unterschieden werden:

Supervised Learning oder Überwachtes Lernen. In diesem Teilbereich sind die Trainingsdaten bereits klassifiziert. D.h., dass jeder Eintrag bereits ein eigenes Label besitzt

Unsupervised Learning oder unüberwachtes Lernen. Hier bekommt der Algorithmus nur Trainingsdaten und muss selber versuchen Muster zu finden, ohne diesem eine Bedeutung zuzuordnen.

Reinforcement Learning oder auch bestärkendes Lernen. Der Algorithmus interagiert in diesem Gebiet mit der Außenwelt.

Grundbaustein von ML sind neuronale Netze, welche, ähnlich dem menschlichen Gehirn, Knotenpunkte sind, die sich mit anderen Knoten verknüpfen. Im Laufe des Trainings werden alte Verknüpfungen gelöscht und neue geknüpft, abhängig vom Input und ihrer Gewichtsfunktion. Die Gewichtsfunktion entscheidet, ob und wann neue Verknüpfungen erstellt werden. Im Deep Learning, der Methode des MLs, auf die sich diese Arbeit fokussieren wird, werden Artificial Neural Networks (ANNs) genutzt um die Komplexität zwischen den Input-Layer und dem Output-Layer aufzuspalten und damit in einfachere Entscheidungen aufzuteilen. Diese weiteren Layer werden als 'hidden layers' bezeichnet, während das Input- und Output Layer 'visible layer' sind.

3.2.1 Rekursive Modelle

Recurrent Neural Networks Recurrent Neural Networks (RNN)s sind eine Art von neuronalen Netzen, die für die Verarbeitung von Sequenzen verwendet werden. Im Gegensatz zu traditionellen neuronalen Netzen können RNNs Informationen aus der Vergangenheit speichern und verwenden, um Entscheidungen zu treffen oder Vorhersagen über zukünftige Ereignisse zu treffen.

Long Short Term Memory Wie im Buch 'Deep Learning'[GBC16] beschrieben, gehören Long Short-Term Memory (LSTM)s zu den sogenannten 'gated RNNs' und arbeiten mit 'self loops'. Anstatt mit hidden layers zu arbeiten, nutzen LSTMs Zellen, die Rückkopplungen/Rekursion erlauben. Durch das *input – gate* und ihren vorherigen *states* berechnet jede Zelle ihren aktuellen state. Die *output – gates* entscheiden daraufhin, ob in einer weiteren Iteration weitere Informationen akkumuliert werden oder die gesammelte Information 'geleaked' wird und der *state* zurückgesetzt wird, also deren Informationen 'vergessen' werden.

Die Fähigkeit, Informationen zu sammeln erlaubt es gated RNNs Abhängigkeiten über längere Zeiträume besser zu erlernen.

3.2.2 Generative Modelle

Generative Modelle, kurz Generative Adversarial Network (GAN), bilden eine Vielzahl von Algorithmen ab, welche für unsupervised learning genutzt werden. Sie bestehen aus einem Generator und einen Diskriminatoren sowie einer Zielfunktion. Während der Generator versucht neue Daten

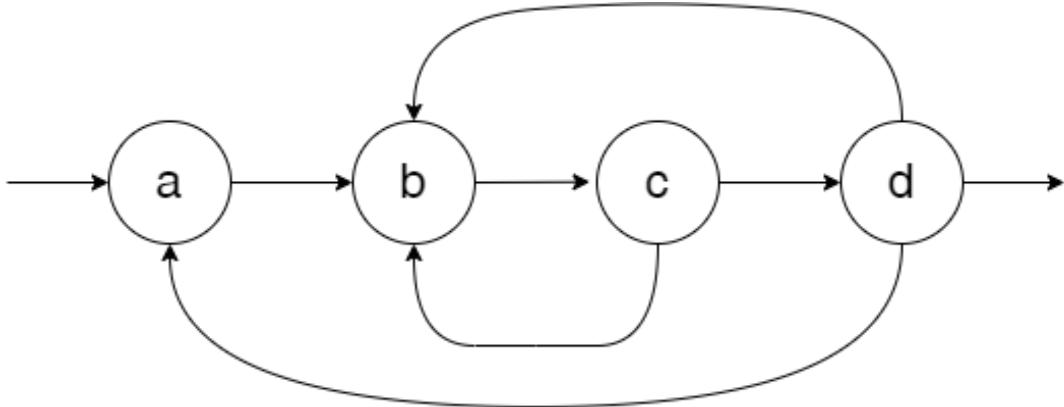


Abbildung 3.1 – Grundidee einer LSTM-Architektur

zu generieren und darauf trainiert wird, versucht der Diskriminatator zu entscheiden, ob die ihm gegebenen Daten echt sind, also aus der Trainingsmenge stammen, oder vom Generator generiert wurden. Werden die Daten als generiert klassifiziert, muss sich der Generator anpassen und das Training geht in die nächste Iteration.

Innerhalb der GAN-Familie gibt es spezialisierte Varianten wie temporale Generative Adversarial Networks (tGANs) und Conditional Generative Adversarial Networks (cGANs):

Temporale Generative Adversarial Networks (tGANs) Time Series Generative Adversarial Network (TGAN) sind zeitreihenoptimierte Modelle [GMB21]. Sie berücksichtigen die zeitliche Abfolge und Muster in Daten, um realistische Zeitreihen zu generieren oder Informationen aus diesen zu ziehen, was sie ideal für Sensordatenprognosen macht.

Conditional Generative Adversarial Networks (cGANs) Conditional Generative Adversarial Network (CGAN) fügen eine bedingte Komponente hinzu, die es ermöglicht, Daten basierend auf spezifischen Bedingungen oder Labels zu generieren [Nay19]. Sie sind daher interessant, wenn generierte Daten bestimmten Kriterien entsprechen müssen, wie beispielsweise bei der zielgerichteten Bildsynthese oder in spezialisierten medizinischen Anwendungen.

4 Personas

Personas sind fiktionale Repräsentationen von Zielgruppen oder Nutzern, die deren Charakteristika, Verhalten und Bedürfnisse aufzeigen. Im User Centered Design dienen sie dazu, benutzerorientierte Lösungen zu entwickeln. Da sich Personas an den realen Nutzern orientieren, helfen sie Designern und Entwicklern, einen tieferen Einblick in ihre Zielgruppe zu gewinnen. Der dadurch entstehende Designprozess ist nutzerorientiert und ermöglicht die Bereitstellung von Lösungen, die dem Kunden oder Nutzer einen echten Mehrwert bieten. Des Weiteren erleichtern Personas die Kommunikation innerhalb von Teams[Rik22], indem sie als gemeinsamer Bezugspunkt dienen, Verwirrungen beseitigen und die Bemühungen aller Beteiligten aufeinander abstimmen. Oft wird im Erstellungsprozess deutlich, welche Punkte für den Nutzer besonders wichtig sind, was zu einer Priorisierungsreihenfolge führen kann.

4.1 Fathima Olsson

Hintergrund

Fathima ist eine junge Medizintechnik-Studentin, die unter anderem im Bereich der Diabetes-Warngeräte forscht. Ihr Ziel ist es, die Fähigkeiten dieser Geräte zu erweitern. Da diese Geräte den Blutzuckerspiegel messen und den Nutzer warnen müssen, wenn dieser zu niedrig ist, muss das neue Gerät fehlerfrei funktionieren. Für die Validierung der Geräte sind umfangreiche Tests erforderlich, für die sie anonymisierte Testdaten mit Blutzuckerverläufen benötigt.

Demographie

Geschlecht:	weiblich
höchste Ausbildung:	Abitur
Einkommen:	-
Familienstand:	alleinstehend
Sprachkenntnisse:	Deutsch, Englisch, Schwedisch

Ziele

- Aufbau einer einfachen Testumgebung für schnelle und sichere Geräteentwicklung.
- Erweiterung der Fachkenntnisse im Bereich Qualitätskontrolle medizinischer Geräte.

Herausforderungen

- Beschaffung kostengünstiger und präziser Daten, die bedarfsgerecht angepasst oder erstellt werden können.

- Aufbau von Testszenarien, die auch kritische Werte simulieren können.
- Entwicklung eines sicheren und prüfbar funktionalen Konzepts zum Testen medizinischer Technologien.

Verhalten

- Forschungsorientierung: Fathima ist neugierig und motiviert, in ihrem Fachgebiet zu forschen und neue Erkenntnisse zu gewinnen.
- Präzision: Sie arbeitet genau und sorgfältig, insbesondere bei der Durchführung von Tests und Validierungen.
- Problemlösungskompetenz: Sie hat die Fähigkeit, komplexe Probleme zu analysieren und kreative Lösungen zu entwickeln.
- Zielorientierung: Ihr Ziel ist es, Menschen zu helfen, und sie ist bereit, die notwendigen Schritte zu unternehmen, um dies zu erreichen.

Bedürfnisse

- Einfache Nutzung: Das System muss einfach und flexibel einsetzbar sein.
- Nachnutzung eigener Daten: Die Software muss Möglichkeiten bieten, mit eigenen Daten zu arbeiten.

Zitat

“Wenn es darum geht, Menschen in schwierigen Situationen zu helfen, darf man sich keine groben Fehler erlauben.“

4.2 Dieter Maibach

Hintergrund

Dieter Maibach ist ein 50-jähriger Maschinenbau-Ingenieur, der vor etwa 30 Jahren studiert hat und seitdem in derselben Firma arbeitet. Er und seine Kollegen sind hauptsächlich für die Entwicklung neuer Mikrocontroller zuständig. Diese sind hochspezialisiert und müssen daher sehr stabil und zuverlässig laufen, weshalb sie eine umfangreiche Testsuite benötigen. Die Testsuite muss in der Lage sein, Signale, die von Sensoren an den Mikrocontroller gesendet werden, zu simulieren, um zu überprüfen, ob er richtig reagiert. Da das Ganze in einem großen Unternehmen stattfinden soll und frei zugänglich sein muss, muss die Anwendung frei zugänglich sein.

Demographie

Geschlecht:	männlich
höchste Ausbildung:	Hochschulabschluss
Einkommen:	60.000

Familienstand:	verheiratet
Sprachkenntnisse:	Deutsch, Englisch (A1), Russisch

Ziele

- Einführung von Qualitätsverbesserungsmethoden in den Entwicklungsprozess, um die Stabilität und Zuverlässigkeit der entwickelten Mikrocontroller zu steigern.
- Erweiterung der Fachkenntnisse im Bereich der Mikrocontroller-Entwicklung, um mit neuen Technologien und Designkonzepten auf dem neuesten Stand zu sein.

Herausforderungen

- Generierung von realistischen Testdaten und deren Konfiguration.
- Aufbau von Testszenarien, die auch kritische Werte simulieren können.
- Einführung von neuen Methoden/Tools in einen alten, lang bestehenden Entwicklungsprozess.

Verhalten

- Gewissenhaftigkeit: Dieter arbeitet präzise, genau und sorgfältig.
- Analytisches Denken: Er hat eine ausgeprägte Fähigkeit, komplexe Probleme zu analysieren und logische Lösungen zu finden.
- Innovationsfähigkeit: Dieter ist offen für neue Ideen und hat die Fähigkeit, kreative Lösungsansätze zu entwickeln.
- Beharrlichkeit: Er gibt nicht schnell auf und ist bereit, Herausforderungen anzunehmen und hartnäckig an Lösungen zu arbeiten.
- Teamorientierung: Dieter arbeitet gerne im Team und bringt einen kooperativen Ansatz ein.

Bedürfnisse

- Einfaches UI/UX Design: Die Software muss einfach zu bedienen und möglichst selbsterklärend sein.
- Einfache Bedienung: Die Software muss einfach in bestehende Umgebungen integrierbar sein.
- Flexible Nachnutzung: Testszenarien müssen im Team einfach nachnutzbar sein.

Zitat

“Solange es zuverlässig und einfach ist, kann ich mich gerne mit neuen Sachen anfreunden.“

5 Anforderungsanalyse der Software

5.1 Rückblick auf die originale Anwendung

Um die Anforderungen an die neue und erweiterte Software zu verstehen, bietet es sich an erst einmal sich den Ausgangspunkt der originalen Software anzuschauen. Diese wurde als Single-Page-Applikation (SPA) entwickelt. Sie besaß eine recht flache Hierarchie und bildete die Datenstruktur des Backends ab. Das bedeutet, dass der Nutzer anstatt in verschiedenen Untermenüs zu navigieren, nur tiefer in die Beschreibung des Projects eintaucht. Dieses Vorgehen stellte sich als eine intuitive Lösung für den Nutzer heraus.

Zu Beginn bekam der Nutzer direkt auf der Hauptseite eine Übersicht über alle Projects, die er angelegt hat und die Möglichkeit neue Projekte anzulegen, zu bearbeiten, zu löschen oder hochzuladen (siehe Abbildung 5.1). Ein Nutzeraccount an sich gab es nicht. Somit teilten sich alle Nutzer im Netzwerk die gleichen Projects.

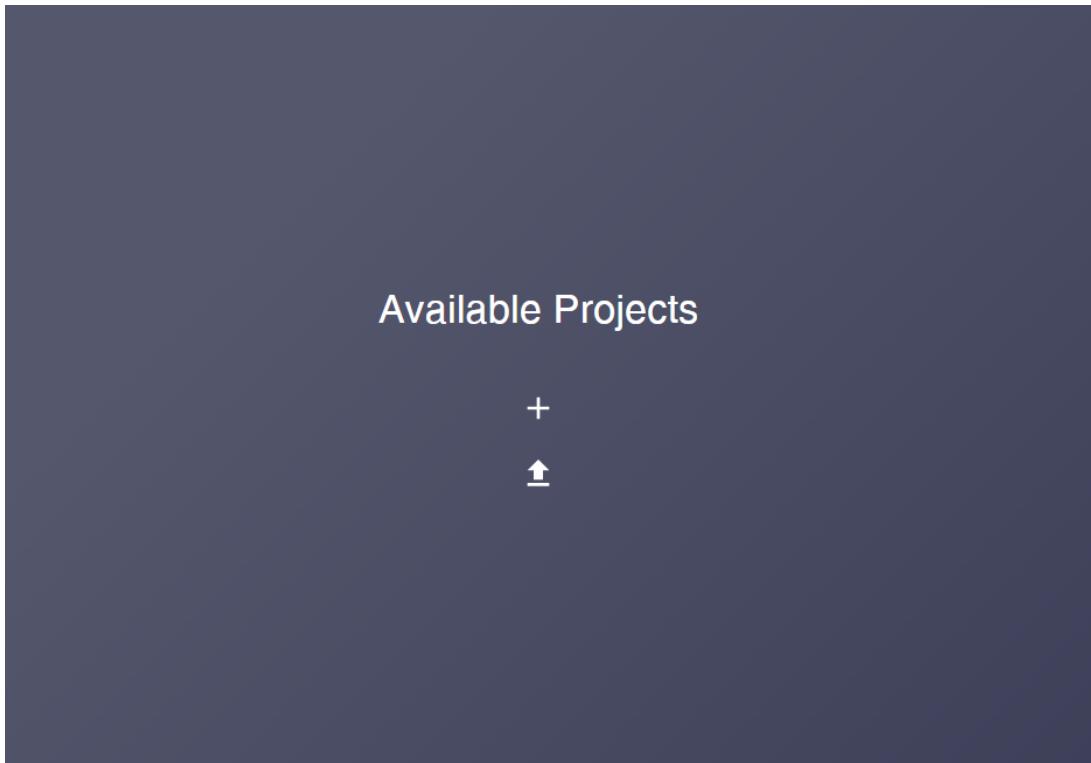


Abbildung 5.1 – Hauptansicht der originalen Anwendung, hier sind alle Projekte aufgelistet

Um seine Projects zu bearbeiten, muss der Nutzer auf den entsprechenden Button klicken und wird auf die Projektseite weitergeleitet. Dort findet sich eine Übersicht über alle vorhandenen Tracks, Optionen diese zu bearbeiten und neue Tracks anlegen. Auch finden sich innerhalb des

Tracks die einzelnen DataSets, welche frei erstellt, bearbeitet, verschoben und gelöscht werden können (siehe Abbildung 5.2). Dies spiegelt die originale Datenstruktur wieder. Ein Project besteht aus mehreren Tracks, welche wiederum aus mehreren DataSets bestehen. Die Abfolge wurde gewählt, damit Nutzer mehrere Track parallel senden, und über die sukzessiv abgearbeiteten DataSets die Struktur des im jeweiligen Tracks definieren können.

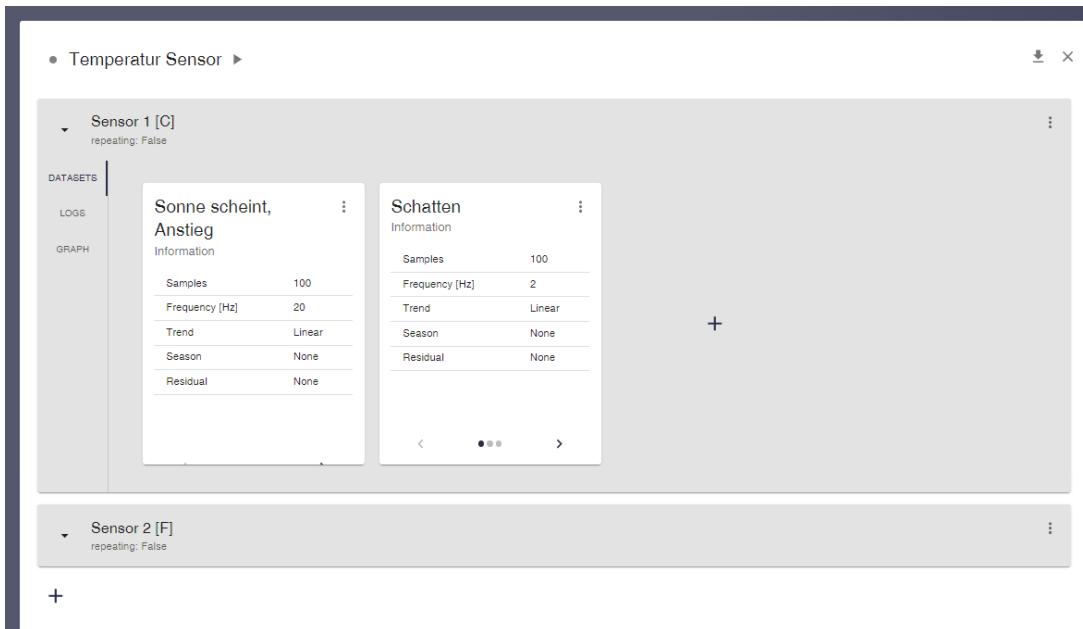


Abbildung 5.2 – DataSet Ansicht der originalen Anwendung

Das Senden der Daten erfolgt über einen entsprechenden Button, welcher, während gesendet wird, entsprechend der Abbildung 5.3 innerhalb der Projects-Seite und der Übersicht rot leuchtet und somit den Sendevorgang auch visuell bestätigt.

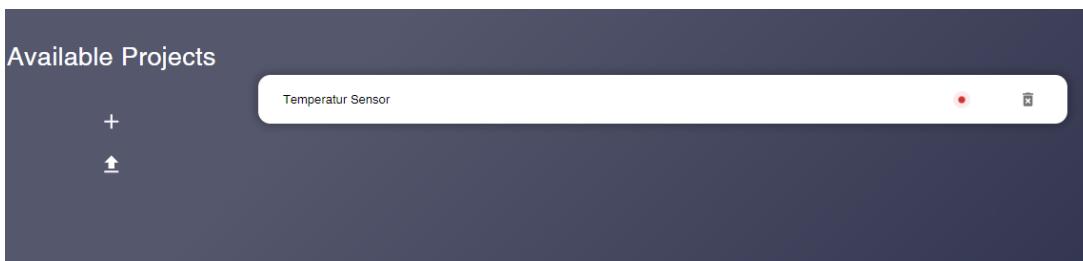


Abbildung 5.3 – DataSet Ansicht der originalen Anwendung

Themen wie Eingabeanvalidierung und Fehlerbehandlung wurden in der originalen Version nur durch JsonSchema vorgenommen und nicht tiefer konzeptioniert. Flexibilität der zu sendenden Datentypen gab es auch nicht. Aufbauend auf dem gegebenen Schema wurden Fließkommazahlen generiert und versendet. Auch wurde angenommen, dass der Nutzer der englischen Sprache mächtig ist und die Eingabeanforderungen und Beschreibungen versteht.

Eine Software, welche sowohl Daten generieren, als auch anhand von Daten trainieren soll kommt mit einer vielzahl an Anforderungen. Grundsätzlich ist Datengenerierung bereits komplexes Thema, aber hierfür wurde im Ursprünglichen Projekt bereits ein Konzept erstellt. Dieses

muss nur um neue Funktionalität erweitert/vorhandene Konzepte angepasst werden. Trainieren anhand von Daten benötigt aber ein eigenes Konzept. Um beide Systeme zusammenzuführen, muss grob auf 5 Punkte geachtet werden.

- Datenmanagement
- Trainieren der Modelle
- Benutzerschnittstelle und Benutzererfahrung
- Sicherheit und Privatsphäre
- Technische Anforderungen

Neben diesen 4 Punkten müssen auch ein paar technische Anforderungen erfüllt werden.

5.2 Anforderungen an die Software

5.2.1 Datenmanagement

Ein großer und elementarer Teil der zu erstellenden Software wird es sein, Daten zu verwalten und in verschiedensten Elementen der Software zu verarbeiten. Hierfür bedarf es eines sinnvollen Konzepts, um die Anforderungen der Software formulieren zu können.

Das spätere System muss in der Lage sein, Daten aus unterschiedlichen Quellen mit verschiedenen Datenformaten effizient zu verarbeiten. Da es Ziel ist, Daten aus fremden Quellen im System zu verarbeiten, müssen vom System gängige Formate verarbeitet werden. Eine solche Flexibilität ermöglicht dem System eine einfache Benutzung und eine breite Anwendbarkeit in verschiedenen Szenarien und erleichtert dem Nutzer die Handhabung der Software. Für die Verarbeitung verschiedener Datenformate muss eine universelle Verarbeitungslogik entwickelt werden, die diese in ein für das System nutzbares Format konvertiert und, da Formate wie JSON und CSV flexibel sind, diese auf valide Inhalte überprüft. Effizienter Umgang mit großen Datenmengen ist ebenfalls wichtig, um die Ressourcen schwächerer Systeme nicht zu überlasten. Entsprechende Vorverarbeitungsschritte müssen vom System durchgeführt werden. Die Umsetzung dieser Funktionen erfordert ein visuelles Bestätigungsgerüst, das die Auswirkungen der Datenmanipulation in Echtzeit anzeigt und so eine transparente und zugängliche Benutzererfahrung schafft. Zudem muss sichergestellt werden, dass ausgewählte Vorverarbeitungsschritte reversibel sind, sodass Nutzer die Möglichkeit haben, die Vorverarbeitung und ihre spezifischen Parameter zu speichern und zu einem späteren Zeitpunkt zu nutzen oder zu verändern.

Das gesamte Konzept des Datenmanagements muss zudem erweiterbar sein, um neue Funktionalitäten und Abhängigkeiten in Zukunft zu ermöglichen. Das Datenmodell muss daher Möglichkeiten bieten, um neue Datenformate und Vorverarbeitungsschritte zu integrieren. Während dies die Komplexität und den initialen Aufwand erhöht, wird es die Wartung und Erweiterung der Software in Zukunft erleichtern.

5.2.2 Trainieren und Konfigurieren von Modellen

In der Anforderungsanalyse für das Training von ML Modellen sowie Zeitreihenzerlegungs- und Analysealgorithmen liegt der Fokus auf der Entwicklung einer benutzerfreundlichen, effizienten

und interaktiven Trainingsumgebung. Da es nicht möglich ist, den richtigen Algorithmus für alle Anwendungsfälle zu finden, muss die Software eine Vielzahl von Algorithmen bereitstellen, die jeweils detailliert in Bezug auf ihre Eigenschaften und Anwendungsbereiche beschrieben werden. Wesentlich ist hierbei eine intuitive Konfiguration aller Algorithmen, die auch für Benutzer ohne spezialisiertes Wissen in diesen Bereichen zugänglich ist. Gleichzeitig sollte eine Balance zwischen einfacher Bedienbarkeit und der Möglichkeit zur Feinabstimmung der Modelle gefunden werden, um eine breite Nutzerbasis zu bedienen.

Da das Training von Modellen eine rechenintensive Aufgabe darstellt, muss die Software eine effiziente Nutzung der Systemressourcen sicherstellen, um auch auf weniger leistungsfähigen Systemen eine reibungslose Funktion zu garantieren. Dies ist wichtig, da die Infrastruktur der Software nicht nur auf leistungsstarken Servern, sondern auch auf lokalen Systemen mit begrenzten Ressourcen betrieben werden soll.

Abschließend ist die transparente Darstellung der Trainings- und Validierungsergebnisse entscheidend. Die Ergebnisse sollten schnell und effizient präsentiert werden und sich an unterschiedliche Benutzeranforderungen anpassen lassen. Dies gewährleistet, dass die Trainingskomponente der Software sowohl für Experten als auch für Laien geeignet ist, was die Zugänglichkeit und Effektivität der Software insgesamt steigert.

5.2.3 Benutzerschnittstelle und Benutzererfahrung

Die Gestaltung der Benutzeroberfläche der Software erfordert ein tiefgreifendes Verständnis für die Bedürfnisse und Erwartungen der Nutzer. Der Nutzer muss im Zentrum aller Designentscheidungen stehen, wobei das Ziel ist, eine Umgebung zu schaffen, in der sich die Benutzer sicher und kompetent in ihren Interaktionen fühlen und stets die Kontrolle über die Software behalten. Um dies zu erreichen, ist eine effiziente und benutzerfreundliche Oberfläche unerlässlich.

Gemäß der ISO 9241-110¹ lassen sich verschiedene grundlegende Anforderungen identifizieren, die für die Gestaltung einer effektiven Benutzeroberfläche wesentlich sind. Nachfolgend werden diese Anforderungen, basierend auf der Interpretation aus [Kri23], dargestellt:

Aufgabenangemessenheit Die Benutzeroberfläche sollte so gestaltet sein, dass Voreinstellungen von Standardwerten und die Struktur der Dialogwege die Aufgaben des Benutzers widerspiegeln und unterstützen.

Steuerbarkeit Die Benutzer müssen die Möglichkeit haben, die Abfolge und Tiefe der angebotenen Dialoge nach ihren Bedürfnissen zu steuern.

Erwartungskonformität Alle Aktionen und Steuerelemente in der Software sollten nachvollziehbar und konsistent mit den Erwartungen des Benutzers sein.

Selbstbeschreibungsfähigkeit Die Software sollte Wartezeiten visualisieren, klare Links bereitstellen und bei Bedarf Hilfestellungen anbieten, um dem Benutzer stets Orientierung zu geben.

Lernförderlichkeit Die Software sollte eine klare, logische und sinnvolle Strukturierung komplexer Aufgaben bieten, um eine einfache Erlernbarkeit zu gewährleisten.

¹Die ISO Norm 9241-110, auch als Grundsätze der Dialoggestaltung bekannt, behandelt die ergonomische Gestaltung von interaktiven Systemen und unterteilt diese in sieben Grundkonzepte.

Fehlertoleranz Eingabefehler sollten angemessen behandelt werden, ohne dass die Software abstürzt, und der Benutzer sollte weiterhin in der Lage sein, seine Ziele zu erreichen.

Individualisierbarkeit Die Software sollte es dem Benutzer ermöglichen, die Oberfläche und Funktionalitäten an seine individuellen Bedürfnisse anzupassen.

Eine weitere wichtige Anforderung ist die Bereitstellung von mehrsprachigkeit. Software, gerade wenn sie komplexe Themen aufgreift, sollte die mentale Leistung seiner Nutzer nicht unnötig belasten, indem man Fachthemen in einer Fremdsprache behandelt. Dies kann zu Missverständnissen und Fehlern führen, die die Effektivität der Software beeinträchtigen. Die Berücksichtigung dieser Prinzipien stellt sicher, dass die Benutzeroberfläche nicht nur funktional und effizient ist, sondern auch ein angenehmes und effektives Benutzererlebnis bietet, das auf die Bedürfnisse und Fähigkeiten verschiedener Nutzergruppen zugeschnitten ist.

5.2.4 Sicherheit und Privatsphäre

Da es ein integraler Teil der Software sein wird, eigene Daten hochzuladen, diese zu speichern und zu verarbeiten, ist die Anforderung an einen vernünftigen Datenschutz und Datensicherheit notwendig. Um den Schutz dieser sensiblen Benutzerdaten zu gewährleisten, ergeben sich aus den folgenden rechtlichen Rahmenbedingungen spezifische Anforderungen.

Unter Berücksichtigung der EU-Datenschutz-Grundverordnung (DSGVO) müssen technische und organisatorische Maßnahmen implementiert werden, um ein angemessenes Schutzniveau für die Verarbeitung personenbezogener Daten sicherzustellen. Insbesondere Artikel 32 DSGVO fordert die Verschlüsselung und sichere Verarbeitung von Daten, um sie vor unbefugtem Zugriff zu schützen. Dies beinhaltet in diesem Fall auch den Schutz vor anderen Nutzern. Während die hochgeladenen Daten nicht zwangsläufig die Identifikation einer Person ermöglichen, können sie dennoch Daten enthalten, welche Rückschlüsse auf eine Person zulassen könnten. Beispielsweise können Sensordaten der Haussteuerung Rückschlüsse auf die Gewohnheiten und das Verhalten einer Person zulassen. Dies muss durch entsprechende Maßnahmen abgesichert werden. Hieraus ergibt sich aber auch eine technische Anforderung. Da es sich bei der Anwendung um eine Webanwendung handelt, müssen die gesamten Strukturen, die ein Nutzer pflegt, separiert werden. Hierfür ist ein entsprechendes Konzept zu entwickeln.

Zusammenfassend ist es von höchster Wichtigkeit, dass die Anwendung ein robustes Sicherheitssystem implementiert, welches den rechtlichen Anforderungen entsprechend die Privatsphäre sowie die Sicherheit der Benutzerdaten effektiv schützt.

5.2.5 Technische Anforderung

Da sich in eine Vielzahl von Komponenten in der Entwicklung der Software abzeichnet, ist eine präzise und effektive Kommunikation zwischen diesen Komponenten unerlässlich, um die Gesamtfunktionalität der Anwendung zu gewährleisten. Daher sind umfassende Dokumentationen für jede Komponente entscheidend. Diese sollten nicht nur die Funktionsweise der einzelnen Komponenten detailliert beschreiben, sondern auch die Kommunikationsprotokolle und Schnittstellen zwischen ihnen klar darlegen. Eine solche Dokumentation ermöglicht die Entwicklung der Systemarchitektur vollständig zu verstehen und effizient damit zu arbeiten.

Ein weiterer wichtiger Aspekt ist die einheitliche Fehlerbehandlung über die APIs. Ein konsistenter Ansatz in diesem Bereich erleichtert es dem Frontend, Fehler schnell zu identifizieren,

zu verarbeiten und in einer für die Nutzer verständlichen Weise zu kommunizieren. Diese Konsistenz in der Fehlerbehandlung ist entscheidend für die Benutzerfreundlichkeit und die Stabilität der Anwendung.

Zusätzlich ist die Implementierung von Überwachungssystemen von großer Bedeutung, um den Zustand der Anwendung kontinuierlich zu überprüfen und mögliche Probleme frühzeitig zu erkennen. Solche Systeme sollten in der Lage sein, schnell auf erkannte Fehler zu reagieren und geeignete Maßnahmen zur Fehlerbehebung einzuleiten oder entsprechende Stellen darüber zu informieren.

Abschließend spielt auch die Einfachheit des Deployments eine wesentliche Rolle. Die Bereitstellung und Inbetriebnahme der Anwendung sollte so unkompliziert wie möglich sein, um eine effiziente und reibungslose Inbetriebnahme zu gewährleisten. Dies umfasst sowohl die Erstinstallation als auch die Durchführung fortlaufender Updates. Durch die Berücksichtigung dieser Anforderungen wird sichergestellt, dass die Software nicht nur ihre Kernfunktionen erfüllt, sondern auch hohe Nutzerzufriedenheit und geringe Wartungsaufwände bietet.

5.2.6 Anforderungen an die Software

Fasst man die in den vorherigen Kapitel gesammelten Anforderungen zusammen, ergibt sich folgende Liste an Anforderungen an die Software:

1. **Vielseitige Datenintegration und Unterstützung mehrerer Datenformate:** Unterstützt verschiedene Datenquellen und Formate, um eine flexible Datenhandhabung zu gewährleisten.
2. **Effiziente Datenübertragung, -validierung und Systemressourcennutzung:** Sorgt für schnelle und korrekte Datenverarbeitung sowie optimale Nutzung der Systemressourcen.
3. **Benutzerorientierte Datenauswahl und -vorverarbeitung sowie Verwaltung:** Ermöglicht es Nutzern, Daten auszuwählen und vorzubereiten, einschließlich Qualitätsoptimierung und reversibler Vorverarbeitungsschritte.
4. **Visuelles Feedback, Datenexploration und transparente Ergebnisdarstellung:** Bietet klare visuelle Darstellungen und Werkzeuge zur Datenexploration, um Benutzern das Verständnis der Daten und Ergebnisse zu erleichtern.
5. **Sicherheit, Privatsphäre und sicherer Umgang mit Benutzerdaten:** Gewährleistet Datenschutz und sichere Datenverarbeitung, einschließlich Zugriffskontrollen und Datentrennung.
6. **Trainieren der Modelle und Bereitstellung einer Vielzahl von Algorithmen:** Ermöglicht das Trainieren verschiedener Modelle und bietet eine breite Palette an Algorithmen mit Beschreibungen und Anwendungsbereichen.
7. **Benutzerschnittstelle, Benutzererfahrung und intuitive Konfiguration der Algorithmen:** Stellt eine benutzerfreundliche Oberfläche und einfache Konfigurationsmöglichkeiten der Algorithmen bereit.
8. **Dokumentation, Fehlerbehandlung, Überwachung und Deployment:** Umfasst eine umfassende Dokumentation, einen einheitlichen Ansatz zur Fehlerbehandlung, Überwachung der Anwendungsperformance und vereinfachtes Deployment.

Auf Grundlage der hier definierten Anforderungen lässt sich direkt eine grobe Struktur des Projektes ableiten, unabhängig von den genutzten Technologien. Dieses wird im folgenden Abschnitt erläutert. Die gewählten Technologien und deren Umsetzung wird in den Kapiteln 7 und 6 erläutert.

5.3 Konzeption der Software

5.3.1 Datenmanagementkonzept

Um die Anforderungen an das Datenmanagement zu erfüllen, muss ein Konzept entwickelt werden, das die Integration, Validierung und Verarbeitung von Daten ermöglicht. Da das System eine Vielzahl an Daten und Formaten unterstützen soll, ist die Entwicklung einer Datenstruktur sinnvoll, die alle Formate abbilden kann. Eine geeignete Lösung wäre die Konvertierung der Originaldaten in ein systemkompatibles Array-Format, wie beispielsweise Listen, die in den meisten Programmiersprachen einfach und effizient verarbeitet werden können.

Da es sich um Dokumente handelt, ist eine angemessene Persistenzmethode erforderlich. Um dies mit der relationalen Struktur des ursprünglichen Projekts zu verbinden, bietet sich eine Separation oder Kombination an. PostgreSQL, eine relationale Datenbank, die JSON-Dokumente speichern kann, wäre hierfür eine geeignete Wahl. Formate wie JavaScript Object Notation (JSON), Comma Separated Values (CSV) oder binäre Array-Formate wie NumPy-Arrays unterstützen solche Strukturen. Werden diese durch eine universelle Verarbeitungslogik in ein nutzbares Format konvertiert, muss die Validierung dem Nutzer aussagekräftiges Feedback zu möglichen Fehlern geben.

Im Rahmen der Vorverarbeitung muss es Nutzern möglich sein, hochgeladene Daten durch eine Reihe von Transformationsschritten zu leiten, die zur Optimierung der Datenqualität und zur Erleichterung des Lernprozesses der Algorithmen unerlässlich sind. Zu diesen Schritten gehören die Normalisierung der Daten, um eine einheitliche Skala zu erreichen, die optionale Entfernung linearer Trends zur Hervorhebung stationärer Komponenten in Zeitreihen und die frei wählbare Reduktion des Datenvolumens durch gezielte Verfahren. Bei mit Zeitstempeln versehenen Daten müssen eventuelle Lücken adäquat behandelt werden.

Angesichts der Komplexität dieser Schritte ist es wichtig, dass Nutzer Rückmeldungen zu ihren Änderungen oder Konfigurationen erhalten. Visualisierungswerzeuge wie dynamische Graphen oder interaktive Datenexplorationsschnittstellen spielen dabei eine Schlüsselrolle. Sie ermöglichen es Nutzern, komplexe Datentransformationsprozesse zu visualisieren und deren Einfluss auf die resultierenden Daten und deren Qualität zu verstehen. Werden die Konfigurationen getrennt von den Originaldaten gespeichert, stellt die Reversibilität der Aktionen kein Problem dar.

Neben den Trainingsdaten müssen auch andere Datenstrukturen erstellt werden, die alle Notwendigkeiten rund um die neuen und alten API-Bereiche abdecken, da das bestehende Datenmodell nicht in der Lage ist, diese neuen Abhängigkeiten abzubilden. Ziel des Datenmanagementmoduls ist es, eine robuste, nutzerzentrierte Plattform für die Datenaufbereitung zu schaffen, die den Weg für präzises und effektives maschinelles Lernen ebnet.

5.3.2 Konzeption des Modelltrainings

Wie in der Sektion 5.2.2 erörtert, müssen verschiedene Algorithmen mit spezifischen Eigenschaften und Fähigkeiten zur Verfügung gestellt werden. Die Benutzeroberfläche der Webanwendung muss daher eine Auflistung dieser Algorithmen bereitstellen, inklusive einer detaillierten Beschreibung ihrer Anwendungsbereiche und Leistungsmerkmale, um Benutzern eine informierte Auswahl zu ermöglichen, die ihren spezifischen Anforderungen und dem Kontext ihrer Daten gerecht wird. Da sich die Software nicht nur an Informatiker richtet, ist dies extrem wichtig.

Die individuelle Konfiguration der Modelle stellt eine Herausforderung dar, da sie eine Balance zwischen Benutzerfreundlichkeit und der Flexibilität der Anpassung erfordert. Feinabstimmungen der Hyperparameter im Konfigurationsprozess können die Komplexität des Modells erhöhen, die Verarbeitungszeit verändern und potenziell signifikante Auswirkungen auf das Ergebnis des trainierten Modells haben. Daher ist es notwendig, die Konfiguration der Modelle einfach zu gestalten, den Nutzer während des Prozesses über die Auswirkungen zu informieren und die Möglichkeit zu bieten, die Konfigurationen zu speichern und zu laden, um sie später wiederzuverwenden.

Um einen effizienten Betrieb der Software zu gewährleisten, muss das Training die Systemressourcen effizient nutzen und sollte möglichst unabhängig vom Haupt-Thread operieren. Um die Systemauslastung zu minimieren, kann eine eigene Implementierung der Modelle teilweise die bessere Lösung sein, da hier die Anzahl der Parameter auf niedrige Systemressourcen optimiert werden kann.

Der letzte wichtige Punkt ist die Echtzeitinformation für das Modelltraining sowie das transparente Darstellen von Trainingsergebnissen. Um den ersten Punkt umzusetzen, muss die Software eine Möglichkeit bieten, den Trainingsprozess zu überwachen und die Ergebnisse in Echtzeit zu visualisieren. Hierfür wären Websockets eine geeignete Lösung, da sie eine bidirektionale Kommunikation zwischen Client und Server ermöglichen. Die Umsetzung des zweiten Punktes ist etwas einfacher. Während des Trainings Daten über Laufzeit, Loss und weitere Metriken zu sammeln und zu speichern, bedarf lediglich einer hierfür passenden Datenstruktur. Die Visualisierung der Ergebnisse kann durch einen Auszug aus den generierten Daten erreicht werden.

5.3.3 Konzeption der Benutzeroberfläche

Die Gestaltung der bereits bestehenden Benutzeroberfläche basierte auf den etablierten Prinzipien von Ben Shneidermans 8 Golden Rules of Interface Design², wobei eine starke Übereinstimmung mit den zuvor genannten Anforderungen besteht. Daher wird in dieser Konzeption eine erneute Aufzählung dieser Richtlinien vermieden.

Die bestehende Benutzeroberfläche wurde unter Verwendung von React und Material UI entwickelt, was ein kohärentes und ästhetisch ansprechendes Erscheinungsbild sicherstellt. Dieses Framework bietet bereits eine umfangreiche Palette an Komponenten und Funktionen, die den Grundprinzipien des User Interface Designs entsprechen und somit viele der erforderlichen Anforderungen abdecken.

Ein wichtiger Aspekt in der Gestaltung ist die Nachbildung der Datenstruktur der API, die den Nutzer durch die entsprechenden Interaktionsschritte leitet. Diese strukturelle Herangehensweise soll in der neuen Konzeption beibehalten werden. Eine Möglichkeit, dies zu erreichen, ist

²Die 8 Golden Rules of Interface Design sind eine Sammlung von Richtlinien für die Gestaltung von Benutzeroberflächen, die von Ben Shneiderman entwickelt wurden.

die Generalisierung der bereits verwendeten Komponenten, sodass diese die benötigten Datenstrukturen als Parameter erhalten und somit flexibel an neue Anforderungen angepasst werden können. Um den Nutzer sicher durch die Anwendung zu leiten, muss dafür nur noch eine entsprechende Navigationskomponente entworfen werden, welche den Nutzer verständlich durch die verschiedenen Bereiche der Anwendung führt, hierbei aber eine klare, strukturelle Trennung der einzelnen Gebiete gewährleistet.

In Bezug auf die Individualisierbarkeit der Software sind derzeit nur begrenzte Anpassungen geplant. Funktionen wie Sprachänderungen sind vorgesehen, um das Benutzererlebnis zu verbessern. Hier sollte daher eine sinnvolle Lösung gefunden werden, eine I18n Integration in das gesamtkonzept einzuarbeiten. Allerdings ist eine umfassende Personalisierung von Dialogen und Abläufen auf Nutzerebene über die grundlegenden Anforderungen der Software hinaus nicht vorgesehen. Diese Entscheidung basiert auf der Abwägung von Nutzen, Notwendigkeit und Zeitaufwand für die spezifischen Anforderungen der Software.

5.3.4 Konzeption des Sicherheitskonzeptes

Um den Datenschutz und die Datensicherheit gemäß den Anforderungen zu gewährleisten, ist die Integration eines Nutzerkontosystems in die Anwendung von grundlegender Bedeutung. Durch eine angemessene Zugriffskontrolle, die über das Nutzerkonto realisiert wird, lässt sich ein unbefugter Zugriff auf sensible Daten effektiv verhindern.

Die Implementierung einer Verschlüsselung für entsprechende Datenfelder bildet eine zusätzliche Schutzschicht. Dabei muss evaluiert werden, inwieweit diese Sicherheitsmaßnahme über mehrere APIs hinweg und innerhalb einer gemeinsam genutzten Datenbank umsetzbar ist. Aufgrund der Herausforderungen, die eine direkte Separierung oder Isolation innerhalb einer relationalen Datenbank mit sich bringt, ist eine umfassende Zugriffskontrolle über die APIs unerlässlich. Dies erfordert ein Sicherheitskonzept, in dem die Datenbank sicher hinter den Strukturen der API positioniert und geschützt wird.

Dieses Konzept stellt sicher, dass der Zugang zu sensiblen Benutzerdaten strikt kontrolliert und gesichert wird, um sowohl die Privatsphäre der Nutzer als auch die Integrität und Vertraulichkeit der Daten zu gewährleisten.

5.3.5 Konzeption der technischen Anforderungen

Das Projekt wird mehrere APIs sowie ein Frontend umfassen, welches diese APIs integrieren muss. Für eine effiziente Zusammenarbeit und nahtlose Integration aller Komponenten sind bestimmte technische Anforderungen und die Umsetzung von Best Practices notwendig. Eine dieser Anforderungen ist die Erstellung einer OpenAPI Spezifikation³, die für die APIs entwickelt werden muss. Diese Spezifikation ermöglicht es Dritten, die APIs zu verstehen und zu nutzen, und bietet einen umfassenden Einblick in die vorhandenen REST-Endpunkte. Sie liefert präzise Beschreibungen der erforderlichen Parameter und illustriert die erwarteten Antwortstrukturen.

Zur Vereinheitlichung der Fehlerstruktur der APIs ist ein einheitlicher Ansatz für die Fehlerbehandlung erforderlich. Dies umfasst nicht nur ein konsistentes Verhalten beim Auftreten von Fehlern, sondern auch ein einheitliches Format für das Versenden von Fehlercodes. Eine solche

³Die OpenAPI Specification ist ein Format zur Beschreibung von REST APIs.

Standardisierung ermöglicht eine effiziente Fehlerverarbeitung und gewährleistet, dass Fehler dem Nutzer auf verständliche Weise kommuniziert werden.

Ein umfassendes Überwachungssystem ist für die Gewährleistung hoher Stabilität der Anwendung unerlässlich. Dieses System sollte in der Lage sein, den Zustand der Anwendung kontinuierlich zu überwachen und mögliche Probleme frühzeitig zu erkennen. Die Implementierung sollte sowohl die Überwachung und Zentralisierung als auch die Auswertung der Logs der einzelnen Komponenten beinhalten, ergänzt durch die Überwachung der Metriken jeder Komponente.

Um den Deploymentprozess der Anwendung zu optimieren, ist es notwendig, diesen so weit wie möglich zu automatisieren. Idealerweise sollte die Anwendung mit einem einzigen Befehl aufgesetzt werden können. In diesem Kontext bietet es sich an, das in der ursprünglichen Anwendung genutzte System, bestehend aus einer Pipeline und einem Docker Compose Skript, zu übernehmen und um alle notwendigen Schritte zu erweitern.

6 Aufbau

6.1 Überblick über die Infrastruktur

Die ursprüngliche Anwendung setzte sich aus einer RESTful-API und einer Single-Page-Applikation für das Frontend zusammen. Für die Entwicklung des Frontends kam React zum Einsatz, ein Framework zur Gestaltung von Benutzeroberflächen, das 2013 von Meta veröffentlicht wurde [Tea20]. Dieses Framework ermöglicht insbesondere die Wiederverwendung von Komponenten, den fundamentalen Bausteinen der Benutzerschnittstelle. Die Wiederverwendbarkeit und Erweiterbarkeit der Komponenten sind entscheidende Prinzipien in React. Ergänzend dazu bieten Bibliotheken wie Material-UI, die in diesem Projekt verwendet wurden, eine Sammlung vorgefertigter Komponenten an.

Das Backend wurde unter Verwendung von Spring Boot konzipiert, einem Tool, das auf dem Spring-Framework basiert und 2014 eingeführt wurde [14]. Spring Boot erleichtert die Erstellung autonomer Microservices, die in der Java Virtual Machine (JVM) laufen und mit einem integrierten Tomcat-Webserver sowie einer Konfiguration ausgestattet sind, die zahlreiche Komponenten des Spring-Ökosystems zusammenführt.

Für die Datenpersistenz kam PostgreSQL zum Einsatz, ein relationales Datenbanksystem, das eine verlässliche Datenspeicherung bietet. Um Signale zu versenden, wurde Kafka, eine verteilte Event-Streaming-Plattform, verwendet.

Die bestehende Infrastruktur, basierend auf einem Technologie-Stack aus React, Spring und PostgreSQL erfordert Erweiterungen, um die Integration von Machine Learning und Zeitreihenanalyse zu ermöglichen.

Wie in Abbildung 6.1 schematisch dargestellt, erfordert die Erweiterung des Projekts ein zusätzliches REST-Framework. Django, ein auf Python basierendes Webframework, das 2005 veröffentlicht wurde, soll in das System integriert werden und für das Training der Machine-Learning-Modelle verantwortlich sein. Graylog wird erst durch die Einführung von Microservices notwendig und dient dazu, die Protokolle der verteilten Systeme zu erfassen. Prometheus wird als Monitoring-Tool eingesetzt, um die Überwachung der jeweiligen Container zu gewährleisten. Redis, ein Key-Value-Datenspeicher, ermöglicht Caching und den Betrieb von Multithreading-Operationen über Websockets. Traefik fungiert als Reverse-Proxy und regelt die Kommunikation zwischen den verschiedenen Diensten bzw. Containern.

Das Ziel dieser neuen Container und Werkzeuge ist es, ein stabil laufendes und wartbares System zu schaffen.

6.2 Microservice-Architektur

Die Microservice-Architektur ist eine methodische Innovation in der Softwareentwicklung, die eine Anwendung in eine Kollektion von kleineren, unabhängigen Diensten aufspaltet. Diese Dienste, bekannt als Microservices, sind für spezifische Funktionen oder Geschäftslogiken verantwortlich

und können unabhängig voneinander entwickelt, bereitgestellt und skaliert werden. Die Kommunikation zwischen diesen Diensten erfolgt über wohldefinierte Schnittstellen, meist RESTful APIs, die eine hohe Interoperabilität gewährleisten. Auch erlauben Microservices polyglotte Programmierung, dies bedeutet, dass Microservices in unterschiedlichen Sprachen und mit unterschiedlichen Technologien implementiert werden können.

Diese Art der Architektur steht im Kontrast zu monolithischen Systemen, in denen alle Komponenten einer Anwendung eng miteinander in einer einzigen Codebasis integriert sind. Monolithen bieten zwar Vorteile wie die Wiederverwendung von Code und eine einheitliche Entwicklungs- und Deployment-Umgebung, sie sind jedoch in Bezug auf Skalierbarkeit und Flexibilität limitiert. Skalierung ist bei Monolithen oft nur vertikal möglich, was bedeutet, dass man die Ressourcen eines einzelnen Servers erhöht, im Gegensatz zum horizontalen Skalieren bei Microservices, wo man die Last auf mehrere Server verteilen kann.

In diesem spezifischen Anwendungsfall ermöglicht die Microservice-Architektur eine sinnvolle Skalierbarkeit, indem beispielsweise Dienste für das Training von Machine Learning Modellen auf leistungsstarken Servern betrieben werden, während einfache CRUD-APIs¹ auf weniger leistungsfähigen Systemen laufen können. So könnte ein Django-Service für Machine Learning und Zeitreihenzerlegung zuständig sein, während ein Spring-Service sich auf das Erstellen und Versenden von Datenstreams konzentriert.

Die Flexibilität der Microservice-Architektur erleichtert zudem die Integration neuer Technologien und Ansätze. Da jeder Service unabhängig ist, können Innovationen in einem Service implementiert werden, ohne von bestehenden Systemen oder Technologiestacks beeinträchtigt zu sein. Dies fördert eine kontinuierliche Weiterentwicklung und Anpassung an neue Anforderungen.

Zusammenfassend bietet die Microservice-Architektur eine starke Grundlage für moderne, skalierbare und flexible Softwareentwicklungsprojekte. Sie unterstützt eine dezentralisierte Entwicklungsstrategie, die Anpassungen und Skalierungen erleichtert und dabei hilft, die technische Schuld zu minimieren, indem sie die Unabhängigkeit von Diensten gewährleistet.

6.3 Technologie-Stack

Apache Kafka Apache Kafka, ein leistungsstarkes Open-Source-Stream-Processing-System, bildet das Herzstück dieses Projekts für die Verarbeitung von Datenströmen in Echtzeit. Es fungiert als zentrale Austauschplattform, die eine effiziente und zuverlässige Datenübertragung zwischen den Datenproduzenten (Producers) und den Datenkonsumenten (Consumers) ermöglicht. Somit kann die Datenübertragung der Signale, die zeitrelevant und kontinuierlich erfolgen muss, abgesichert und kann durch neue Consumer unabhängig auf verschiedenen Plattformen genutzt werden.

Docker Docker hat sich als eine transformative Technologie in der Welt der Softwareentwicklung etabliert, die es ermöglicht, Anwendungen in Containern zu verpacken. Diese Container sind leichtgewichtige, eigenständige Pakete, die alles enthalten, was eine Anwendung zum Laufen benötigt, von Umgebungsvariablen und Konfigurationsdateien bis hin zu Code, Laufzeitumgebung und den genutzten Bibliotheken. Docker vereinfacht damit die Bereitstellung und den Betrieb

¹CRUD-API bezieht sich auf eine API, welche hauptsächlich für Datenbankoperationen zuständig ist

von Anwendungen, indem es für Konsistenz über verschiedene Entwicklung, Release-Zyklen und Cloud-Umgebungen hinweg sorgt.

Der Einsatz von Docker im Projekt bietet entscheidende Vorteile: Es ermöglicht, dass das Projekt praktisch überall ausgeführt werden kann, unabhängig von spezifischen Hardwarekonfigurationen. Dies ist besonders nützlich in heterogenen Umgebungen, wo die Unterstützung verschiedener Betriebssysteme und Plattformen erforderlich ist. Docker ermöglicht auch eine erhebliche Skalierbarkeit, vor allem in Verbindung mit einer Microservice-Architektur. In solch einem Ökosystem kann Docker dazu beitragen, die Anwendungen leicht skalierbar und wartbar zu machen. Mit Docker können auch Aktualisierungszyklen beschleunigt werden, da neue Versionen schnell und automatisiert bereitgestellt werden können, was für kontinuierliche Integration und kontinuierliche Bereitstellung (CI/CD) von entscheidender Bedeutung ist. Auch ermöglichen Techniken wie Blue-Green-Deployment eine nahtlose Umschaltung zwischen verschiedenen Versionen der Anwendung, was das Risiko bei der Bereitstellung neuer Versionen minimiert.

Docker Compose ist ein Tool, das das Management von Multi-Container-Anwendungen vereinfacht. Mit einer einzigen Konfigurationsdatei können Entwickler die Dienste, Netzwerke und Volumes definieren, die für ihre Anwendung erforderlich sind. Dies ist ideal für Single-Host-Deployments, welches für die Anwendung vorerst vorgesehen ist, und bietet die Flexibilität, unterschiedliche Konfigurationen für Entwicklung und Produktion zu definieren. Docker Compose erleichtert auch das Starten, Stoppen und Neubauen von Diensten und die Skalierung von Containern.

Für größere Deployments bietet Docker Swarm eine native Cluster-Verwaltungsfunktionalität, die Docker-Hosts zu einem virtuellen Single-Host macht. Swarm nutzt die Docker-API, was bedeutet, dass jede Software, die bereits mit Docker funktioniert, ohne Anpassungen mit Docker Swarm verwendet werden kann. Es orchestriert die Container, die auf einer Gruppe von Hosts ausgeführt werden, und enthält Dienste wie Load Balancing, die Anfragen über die Knoten hinweg verteilen.

Insgesamt bietet Docker eine effiziente, skalierbare und sichere Lösung für die Verpackung und Ausführung von Anwendungen, die die Art und Weise, wie Software entwickelt und betrieben wird, revolutioniert hat. Mit seinen Tools und Ökosystemkomponenten ist Docker eine ausgewogene Wahl für moderne Softwareprojekte, die schnelle Iterationen und eine hohe Verfügbarkeit erfordern.

Graylog Graylog, ein zentrales Log-Management-Tool, dient als eine entscheidendes Tool in der Systemarchitektur. Als Open-Source-Plattform konzipiert, ermöglicht Graylog die automatisierte Zentralisierung, Sammlung und Analyse von Log-Daten. Spezifisch im Projekt spielt Graylog eine kritische Rolle bei der Überwachung der Kommunikation zwischen verschiedenen APIs, dem Frontend und der Datenbank. Durch die zentrale Erfassung von Logs bietet Graylog Einblicke in Systemereignisse und unterstützt so die Fehlerdiagnose und Optimierung des Systembetriebs.

Graylog baut auf dem ELK-Stack auf, einer Kombination aus Elasticsearch, Logstash und Kibana, die eine effiziente Log-Aggregation, -Analyse und -Visualisierung auf Systemebene ermöglicht. Der ELK-Stack wird häufig für umfassende Logging-Lösungen verwendet, wobei Graylog eine alternative Schnittstelle bietet, die auf ähnlichen Technologien basiert und darauf ausgerichtet ist, die Handhabung und Verarbeitung von Log-Daten zu vereinfachen.

Die Einrichtung eines zentralen Log-Management-Tools wie Graylog ist entscheidend für die Aufrechterhaltung der Systemintegrität, besonders in einer komplexen Microservice-Architektur.

Durch die Konsolidierung der Logs an einem zentralen Punkt erleichtert Graylog das Monitoring und die Analyse von verteilten Systemen, was bei der Fehlersuche, Leistungsoptimierung und Sicherheitsüberwachung unerlässlich ist. Mit seinen leistungsstarken Such- und Analysefähigkeiten hilft Graylog Entwicklern und Systemadministratoren, schnell auf Ereignisse zu reagieren und die Systemstabilität zu gewährleisten.

Prometheus Im Jahre 2012 stellte Soundcloud mit Prometheus ein bedeutendes Werkzeug für das Monitoring und Alerting vor, das seither insbesondere in großen und verteilten Systemen breite Anwendung findet. Das Tool ist spezialisiert auf die Sammlung, Speicherung und das Management von Metriken in Zeitreihenform. Diese Fähigkeit ist entscheidend, um einen Überblick über die Gesundheit und Leistung der überwachten Systeme zu erhalten. Bei auftretenden Anomalien ist Prometheus so konzipiert, dass es Warnmeldungen auslöst, um auf potenzielle Probleme hinzuweisen[Aut].

In Bezug auf dein aktuelles Projekt ermöglicht Prometheus eine zentralisierte Überwachung der einzelnen Systemkomponenten, wodurch eine sofortige und präzise Diagnose bei Performance-Einbußen oder Ausfällen einzelner Teile des Systems ermöglicht wird. Durch die Analyse der erfassten Daten lassen sich nicht nur kurzfristige Zustände und Fehlfunktionen erkennen, sondern auch langfristige Muster und Trends, die Rückschlüsse auf die Effizienz und Angemessenheit der zugrunde liegenden Infrastruktur zulassen.

Traefik Traefik, ein moderner HTTP-Reverse-Proxy und Load Balancer, wurde speziell für Microservice-Architekturen und deren dynamische Anforderungen entwickelt. Er zeichnet sich durch eine automatische Systemerkennung und Konfiguration aus, was ihn zu einem unverzichtbaren Werkzeug für die reibungslose Verwaltung von Netzwerkanfragen in komplexen Systemen macht. Traefik integriert sich nahtlos mit etablierten Infrastrukturkomponenten wie Docker, Docker Compose und Docker Swarm, wodurch die Verteilung von Anfragen auf die entsprechenden Services vereinfacht wird.

Durch seine intuitive Konfiguration und automatische Serviceerkennung entlastet Traefik Entwickler von manuellen Setup-Prozessen und fördert eine effiziente Bereitstellung von Services. Darüber hinaus bietet Traefik Echtzeit-Einblicke in das Netzwerkverhalten und unterstützt Let's Encrypt für die Automatisierung von SSL/TLS-Zertifikaten, was die Sicherheit erhöht. Durch seine Fähigkeit, dynamisch auf Änderungen in der Service-Landschaft zu reagieren, ist Traefik ein essentielles Element in der Toolchain für die Implementierung und den Betrieb von Microservices, das Leistungsfähigkeit mit Benutzerfreundlichkeit vereint.

PostgreSQL PostgreSQL ist ein fortschrittliches Open-Source-Datenbanksystem, das für seine Robustheit, Flexibilität und Compliance mit SQL-Standards bekannt ist. Es bietet erweiterte Funktionen, wie komplexe Abfragen, Fremdschlüssel, Trigger, updatable Views und Transaktionen mit hoher Integrität. PostgreSQL eignet sich besonders gut für Aufgaben, die komplexe Datenverarbeitungen erfordern und bei denen die Datenkonsistenz von höchster Wichtigkeit ist. Die Architektur unterstützt sowohl relationale als auch JSON-Datentypen, was es zu einer vielseitigen Wahl für vielfältige Anwendungsfälle macht.

Redis Redis ist ein Open-Source-Key-Value-Datenspeicher, der für seine Geschwindigkeit, Flexibilität und Vielseitigkeit bekannt ist. Es ist ein NoSQL-Datenspeicher, der Daten in einem Schlüssel-

Wert-Format speichert und eine Vielzahl von Datenstrukturen unterstützt, darunter Strings, Hashes, Listen, Sets und geordnete Sets. Redis ist besonders gut für Anwendungen geeignet, die eine hohe Leistung erfordern, da es die Daten im Arbeitsspeicher speichert und so eine schnelle Datenverarbeitung ermöglicht. Im aktuellen Projekt wird Redis für Websockets eingesetzt.

Gitlab CI und Gitlab Runner Gitlab bietet über eigene, sogenannte Pipelines, die Möglichkeit kontinuierlich Software zu testen und auszurollen. Dank Continuous Integration Continuous Delivery (CICD) kann der auf Gitlab hochgeladene Code gebaut, getestet und, je nach Konfiguration des dafür eigens angelegten .gitlab-ci Files, deployed werden. Es ist ein hilfreiches Tool um die Qualität der Software aufrecht zu erhalten und darüber hinaus ständig eine funktionierende Version aufrufbar zu haben.

Hugo Hugo ist ein Open-Source-Static-Site-Generator, der 2013 von Steve Francia veröffentlicht wurde. Er erlaubt das Erstellen von statischen Webseiten, die aus Markdown-Dateien generiert werden. Somit lässt sich eine einfach und schnelle Dokumentation erstellen, die zudem noch leicht zu pflegen ist. In Kombination mit einem Nginx² kann die Dokumentation auf einem Server gehostet werden und somit direkt an das Gesamtprojekt angebunden werden.

6.4 Zusammenfassung

In diesem Kapitel wurde kurz die Architektur des Projekts präsentiert um dieses in einem produktiven Betrieb effizient nutzen zu können. Hierbei wurde auf die Wahl einer Microservice-Architektur eingegangen und die einzelnen Komponenten vorgestellt. Es existieren somit neben den normalen Komponenten wie Front- und Backend auch Elementen, die sich um das Datenmanagement kümmern, wie PostgreSQL, Kafka und Redis. Um die Anwendung zu überwachen und zu warten, werden Prometheus und Graylog eingesetzt. Wichtig für die flexible Inbetriebnahme sind die Komponenten aus dem DevOps Bereich, wie Docker, Docker Compose und Traefik. Damit diese Schritte automatisiert werden können, wird Gitlab CI und Gitlab Runner eingesetzt. Zur Darstellung der Dokumentation dient Hugo, neben der OpenAPI und Swagger Dokumentation.

All diese Elemente sollten helfen, die Anwendung zuverlässig und stabil zu betreiben und die Wartungskosten zu minimieren.

²Nginx ist ein Webserver, der sich durch seine hohe Performance auszeichnet. Er dient hier als static file Server.

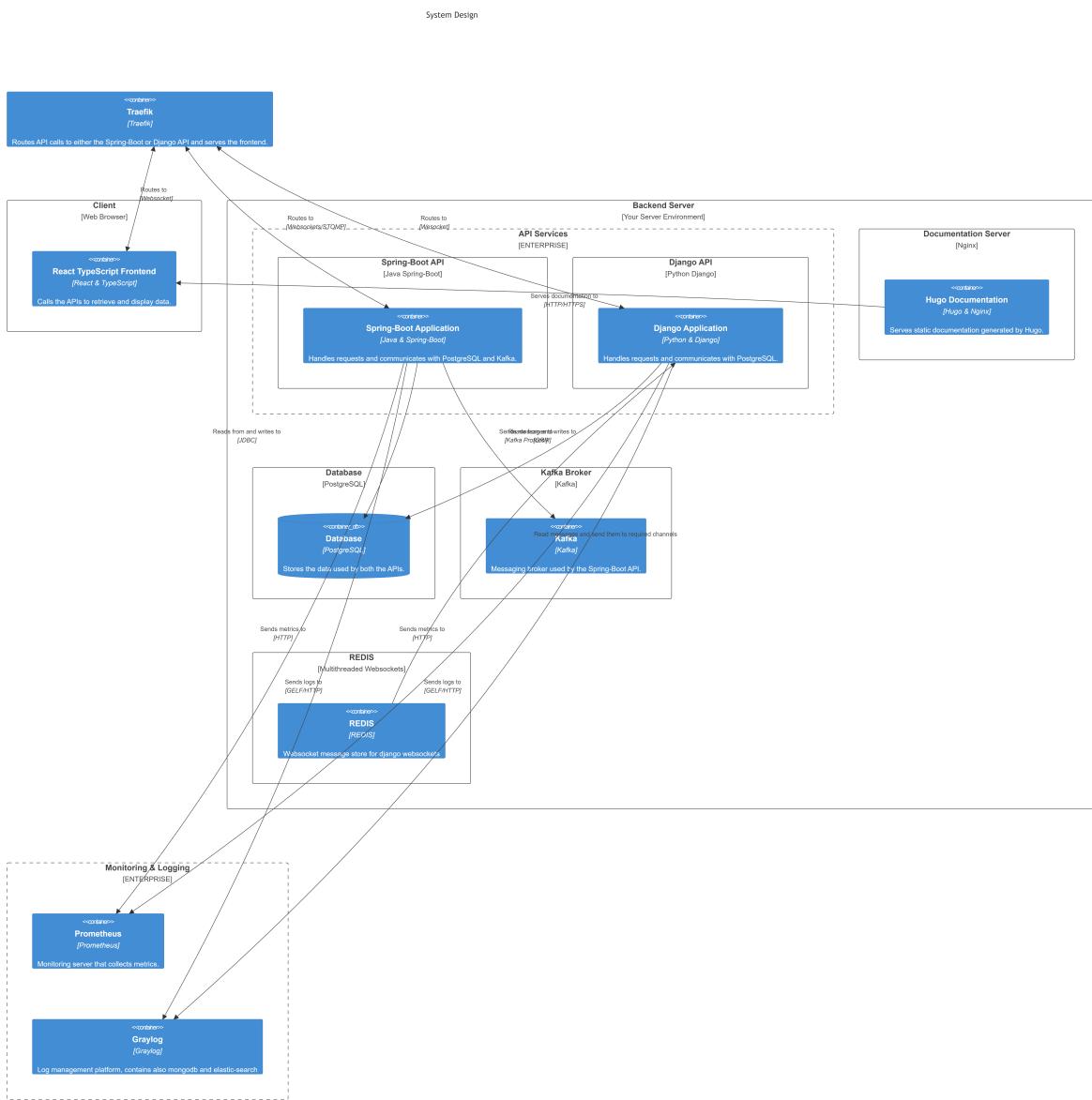


Abbildung 6.1 – Aktuelle Infrastruktur, sie verbindet das Frontend mit beiden APIs, welche Zugriff auf die PostgreSQL Datenbank haben. Überwacht werden sie von Prometheus und Graylog. Ein Kafka Producer wird genutzt um die generierten Daten als Streams an entsprechende Consumer zu senden. Redis dient als Cache für die Websockets des Django-Channel Frameworks. Hugo wird genutzt um die Dokumentation zu generieren und über Nginx verfügbar zu machen. Dies alles wird in Docker Containern ausgeführt.

7 Implementation

7.1 Überblick

Wie bereits in den vorherigen Kapiteln kurz angeschnitten muss die Software an vielen verschiedenen Punkten erweitert und, gerade im Bereich des maschinellen Lernens und der Zeit Serien Analyse, neu implementiert werden. Da das Projekt aus 4 Teilen besteht, wird jedes Thema in einem der folgenden Kapitel separat erläutert.

In den jeweiligen Kapiteln werden sehr grob die Schritte gezeigt, welche für die jeweilige Implementierung notwendig waren. Auch wird auf spezifische Designentscheidungen eingegangen, welche für die jeweilige Implementierung notwendig waren und warum sich für gewisse Technologien entschieden wurde.

7.2 Java Spring Boot

Das Backend in Spring-Boot bildet das Herzstück der ursprünglichen Software. Es stellt eine vollständige API für das Frontend bereit, bindet die PostgreSQL-Datenbank ein und kommuniziert mit den Kafka-Brokern. Ursprünglich handelte es sich um eine CRUD-ähnliche API, die neben den grundlegenden CRUD-Operationen auch das Starten und Stoppen von Projekten ermöglichte.

Da die Software jedoch noch auf Spring Boot 2.7.5 lief und daher den Übergang von JavaX zu Jatkarta nicht vollzogen hatte, mussten zuerst alle Maven-Abhängigkeiten aktualisiert werden. Zudem war das Frontend noch Teil der Maven-Ziele. Da die Anwendungen jedoch separat voneinander ausgeführt werden müssen, musste dies ebenfalls entfernt werden.

Basierend auf der Anforderungsanalyse aus Kapitel 5.1 werden hier drei der fünf vorgestellten Punkte behandelt. Da es sich um die erste API handelt, die lediglich erweitert wurde, waren viele der technischen Anforderungen bereits erfüllt. Die Einführung von Metriken für Prometheus und ein zentralisiertes Logging wurde mithilfe entsprechender Bibliotheken, Konfigurationsdateien und Annotationen schnell umgesetzt.

7.2.1 Architekturentscheidung und Umstrukturierung

Um die Architektur der Software flexibler zu gestalten, war eine Umstrukturierung in Richtung einer hexagonalen Architektur wichtig. Diese Architektur stellt eine Erweiterung der von Eric Evans in "Domain-Driven Design: Tackling Complexity in the Heart of Software" [Eva04] beschriebenen Domain-Driven-Architektur dar. Diese vierstufige Architektur unterscheidet sich in einigen wesentlichen Aspekten von den zuvor genutzten Strukturen, in denen Pakete streng nach Entitäten strukturiert wurden und sowohl Datenbank-, Service- als auch Kommunikationsschichten integriert waren. Dies reflektierte das übliche Domain-Driven Design, bei dem Anwendungen in Präsentations-,

Logik- und Datenschichten unterteilt werden, um Concerns", also Aufgaben, getrennt voneinander zu behandeln. Beispielsweise darf nur die Logik-Schicht Änderungen an den Daten vornehmen, während die Datenbankschicht lediglich zum Schreiben und Lesen genutzt wird. In der ursprünglichen Architektur war dies jedoch nicht konsequent umgesetzt, und es gab viele Überschneidungen.

Um die Trennung der Zuständigkeiten konsequent umzusetzen, war eine Umstrukturierung erforderlich. Da bereits zu Beginn klar war, dass die Spring-API auch mit anderen Services kommunizieren muss, bot sich eine hexagonale Architektur an. Das Ziel der hexagonalen Architektur, auch bekannt als Ports-und-Adapter-Pattern", besteht darin, die Trennung zwischen der Logikschicht und den externen Schichten zu verstärken und zu konkretisieren. Dies geschieht, indem Akteure über technologieagnostische Schnittstellen wie Interfaces kommunizieren. Diese Ports dienen daher als Schnittstellen zwischen den Schichten. Die Adapter sind die Initiatoren der Interaktionen, beispielsweise RestController, JPRepositories oder WebSocketController, die als Schnittstellen nach außen agieren.

Die Vorteile dieser Architektur sind vielfältig: Sie fördert die Entkopplung von Komponenten, vereinfacht Tests, ermöglicht eine flexiblere Entwicklung durch die ausschließliche Implementierung gegen Interfaces und zeigt eine hohe Anpassungsfähigkeit gegenüber Änderungen. Ein potenzieller Nachteil könnte in der erhöhten Komplexität liegen, die jedoch durch das Weglassen der Domain-Modelle stark reduziert wurde.

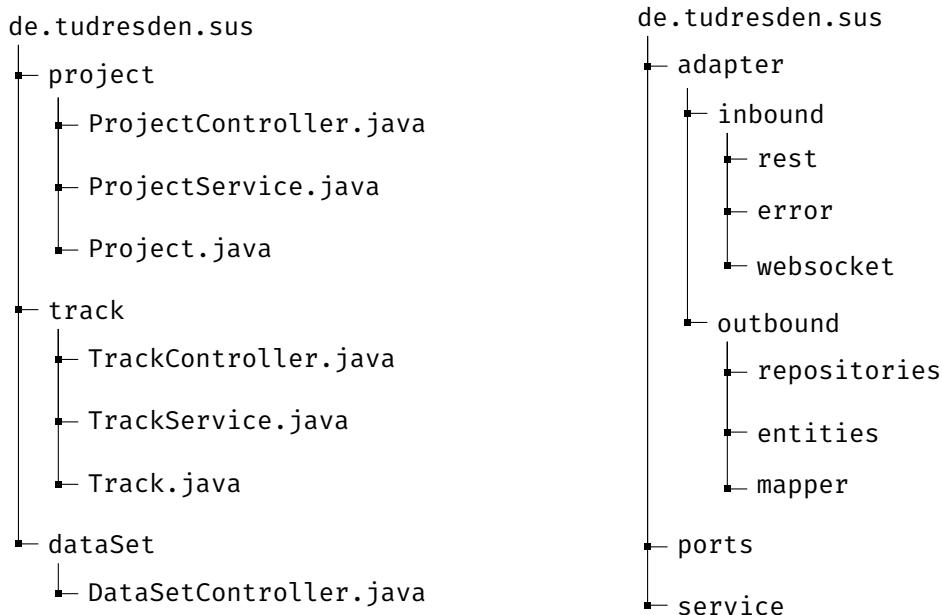


Abbildung 7.1 – Alte Datenstruktur (links) und neue (rechts)

Die neu definierten Schichten haben klare und abgegrenzte Verantwortlichkeiten. Das Adapter-Layer fungiert als oberste Schicht und beinhaltet alle externen Funktionalitäten. Es wird in einen Inbound- und einen Outbound-Teil unterteilt: Der Inbound-Teil ist verantwortlich für alle eingehenden Anfragen wie REST-Requests oder WebSocket-Verbindungen. Der Outbound-Teil umfasst alle nach außen gerichteten Funktionen, z.B. die Datenbankkommunikation oder den Rest-Client. Die Unterscheidung zwischen Inbound- und Outbound-Adaptoren wird vom Datenfluss innerhalb des Service Layers gesteuert. Adapter, die Daten in das Layer eingeben und somit steuern, gehören zu den Inbound-Objekten. Adapter, die nach außen gehen, also vom Service Layer angesprochen werden, gehören in den Outbound-Bereich.

Im Port-Layer erfolgt die Kommunikation zwischen den Schichten. Anstatt eines direkten Zugriffs der RestController auf die Service-Beans wird ein Interface bereitgestellt, wobei Spring die korrekte Bean injiziert.

Dies mag zunächst überflüssig erscheinen, entspricht jedoch dem Prinzip der Dependency Inversion: Höherrangige Komponenten sollten nicht direkt auf niederrangigere zugreifen, sondern lediglich auf deren Abstraktionen. Dies ermöglicht die flexible Austauschbarkeit verschiedener Implementierungen des Interfaces und erleichtert das Testen. Auch können Komponenten einfach ausgetauscht werden, ohne Einfluss auf die darunterliegende Struktur zu nehmen. So kann beispielsweise die REST-Schnittstelle komplett durch Websockets ersetzt werden, wobei nur die jeweiligen Endpunkte implementiert werden müssen. Die Geschäftslogik bleibt davon unberührt. Ebenso kann die verwendete Datenbank problemlos ausgetauscht werden, da ihre Schnittstelle durch das jeweilige Datenbank-Interface abstrahiert wird.

Obwohl viele hexagonale Architekturen ein Domain-Layer enthalten, der Domain-Modelle mit erweiterter Funktionalität gegenüber den Datenbankmodellen enthält, wurde in diesem Projekt auf dieses Layer verzichtet. Dies wurde gemacht, da es eine unnötige Komplexität eingeführt hätte und aktuell nicht notwendig erscheint. Die erforderlichen Funktionalitäten werden bereits durch

bestehende Mapper-Klassen abgedeckt.

Anpassung der Datenstruktur

Ein Umbau der Datenstruktur, wie sie bereits in 5.2.1 angedeutet wurde, lässt sich in zwei wesentliche Teile aufteilen.

Datensätze und Vererbung

Ursprünglich verfügte die Anwendung über eine recht einfache und sehr lineare Datenstruktur, welche im folgenden ER-Diagramm in Abbildung 7.2 gezeigt wird.

Diese Struktur bestand aus vier Komponenten: Ein Project ist in einer 1:n-Relation zu einem Track, der wiederum eine 1:n-Relation zu einem Datensatz hat. Diese Datenstruktur wurde ausgewählt, um komplexe Systeme modellieren zu können. Ein Project dient hierbei als Komponente, die mehrere datensendende Einheiten, sogenannte Tracks, umfasst. Da diese Tracks in unterschiedlichen Intervallen und insbesondere parallel senden können, war eine solche Unterscheidung notwendig. DataSets, aus denen ein Track besteht, fungieren als Abstraktionsebene der verschiedenen Phasen, in denen sich ein Track befinden könnte. So lassen sich durch aufeinanderfolgende DataSets unterschiedliche Situationen abbilden, beispielsweise die Simulation eines Gewächshauses (Project) mit drei Sensoren (Tracks) für Luftfeuchtigkeit, Temperatur und Lichtintensität. Die varierenden Bedingungen im Tagesverlauf, wie steigende und fallende Temperaturen, werden durch aufeinanderfolgende DataSets dargestellt. Jeder Track kann dabei eine eigene Frequenz besitzen.

Zur Nachverfolgung gesendeter Daten wird zudem ein Logverzeichnis angelegt.

Diese Herangehensweise erwies sich zwar als robust, jedoch auch als wenig flexibel. Die Art der zu sendenden Daten ließ sich nicht einfach anpassen, und die Zuweisung spezieller Eigenschaften zu unterschiedlichen Datentypen war nicht möglich. Die Beschränkung auf die in Trend, Residual und Season unterteilte Berechnungsvorschrift stellte eine weitere Limitierung dar.

Die erforderliche Flexibilität für die Erstellung komplexer Simulationsumgebungen und die Integration externer Konfigurationen, wie sie die Django-API ermöglicht, machte eine methodische Änderung notwendig.

Hierfür wurden zwei Ansätze verfolgt:

Vererbung Gemäß der in Listing 7.1 dargestellten Methode können neue Datensätze einfach von der Basisklasse erben. Neue Felder werden im selben Table gespeichert, wobei bei vielen neuen Feldern idealerweise eine neue Entität oder Entitätenstruktur erstellt werden sollte. Beim Auslesen aus der Datenbank wird dann über Casting die gewünschte Klasse rekonstruiert.

```
1 @Entity
2 @Inheritance(strategy = InheritanceType.SINGLE_TABLE)
3 @Accessors(chain = true)
4 @Data
5 @JsonTypeInfo(
6     use = JsonTypeInfo.Id.NAME,
7     include = JsonTypeInfo.As.PROPERTY,
8     property = "type")
9 @JsonSubTypes({
10     @JsonSubTypes.Type(value = CharDataSet.class, name = "char"),
```

```

11     ...
12 })
13 @JsonAutoDetect(fieldVisibility = JsonAutoDetect.Visibility.ANY)
14 public abstract class PlainData implements DataTypeOption {
15
16     @Id
17     @Column(name = "id")
18     @GeneratedValue(strategy = GenerationType.AUTO)
19     private Long id;
20     @Column(name = "position")
21     private int position;
22     @Column(name = "numSamples")
23     private int numSamples;
24     @Column(name = "frequency")
25     private float frequency;
26     @Column(name = "name")
27     private String name;
28     @Column(name = "dataType")
29     private DataTypes dataType;
30     ....
31 }

```

Listing 7.1 – Abstrakte Basisklasse der DataSets

Polymorphe Deserialisierung Um eine flexible Kommunikation zu gewährleisten, wird Jacksons polymorphe Deserialisierung eingesetzt¹. Diese nutzt die in den @JsonSubTypes definierten Optionen für das zielgerichtete Casting in die korrekten Objekte. Somit können die Objekte unabhängig vom Datentypen im Business-Layer verarbeitet werden und typgerecht in der Datenbank gespeichert werden.

```

1 @JsonTypeInfo(
2     use = JsonTypeInfo.Id.NAME,
3     include = JsonTypeInfo.As.PROPERTY,
4     property = "type")
5 @JsonSubTypes({
6     @JsonSubTypes.Type(value = CharDataSetDTO.class, name = "char"),
7     ...
8 })
9 @Data
10 @Accessors(chain = true)
11 @JsonAutoDetect(fieldVisibility = JsonAutoDetect.Visibility.ANY)
12 public class PlainDataDTO {
13
14     @Schema(name = "id", nullable = true, minimum = "1")
15     private Long id;
16     ...
17
18 }

```

Listing 7.2 – Basisklasse der DTO Objekte

¹Jackson bietet eine umfangreiche Java API zum Arbeiten mit JSON

Jede Klasse muss außerdem eine eigene Mapper-Klasse bereitstellen, um die Datenstrukturen adäquat in die gewünschten DTOs zu konvertieren.

Die Implementierung dieser drei Klassen und ihrer Konfigurationen in den Basisklassen ermöglicht eine schnelle Erweiterung der bestehenden Datenstruktur. Zusätzlich implementieren alle Klassen ein Interface, das ihr jeweiliges Verhalten definiert. Dadurch kann eine neue Datenstruktur implementiert werden, ohne das Business-Layer zu beeinflussen. Wichtig ist, dass dieses Vorgehen den Grundsatz der Separation of Concerns nicht verletzt, da die neuen Datentypen lediglich eigene Datengenerierungsstrategien implementieren, ohne das Verhalten außerhalb zu beeinflussen. Eine Ausnahme bilden die MLDataSet und TSADataset, die ihre Daten extern (von der Django-API) beziehen. Hier wäre es theoretisch möglich, innerhalb der Entitäten einen REST-Client zu implementieren, was jedoch einen ungünstigen Datenfluss zur Folge hätte. Diese Aufgabe gehört eher zur Business-Logik und wird daher dort verarbeitet. Eine alternative Lösung wäre das Hinzufügen einer Domain-Schicht, die DataSets auf ein DomainModel projiziert. Da dies jedoch den Code unnötig aufblähen und bei den meisten DataSets diese Komplexität oder Abstraktion nicht benötigt wird, übernimmt das Business-Layer direkt diese Aufgabe.

```

1 private final CharDataSetMapper stringMapper;
2 private final FloatDataSetMapper floatMapper;
3 private final IntegerDataSetMapper intMapper;
4 private final MLDataSetMapper mlMapper;
5 private final TSADatasetMapper tsaMapper;
6 private final CustomValueConverter customValueConverter;
7 private final SleepDataSetMapper sleepMapper;
8
9 public PlainData toEO(PlainDataDTO data) {
10     var eo = switch (data.getDataType()) {
11         case FLOAT:
12             yield floatMapper.toEO((FloatDataSetDTO) data);
13         case CHAR:
14             yield stringMapper.toEO((CharDataSetDTO) data);
15         case INTEGER:
16             yield intMapper.toEO((IntegerDataSetDTO) data);
17         case ML:
18             yield mlMapper.toEO((MLDataSetDTO) data);
19         case TSA:
20             yield tsaMapper.toEO((TSADatasetDTO) data);
21         case SLEEP:
22             yield sleepMapper.toEO((SleepDataSetDTO) data);
23     };
24     if (eo == null) {
25         throw new DataConversionException("cannot convert dto: %s".formatted(data));
26     }
27     return eo.setId(data.getId())
28             .setFrequency(data.getFrequency())
29             .setName(data.getName())
30             .setNumSamples(data.getNumSamples())
31             .setPosition(data.getPosition());
32 }
```

Listing 7.3 – Mapper Klasse für DataSets, basierend auf den dataType, ein separates enum, fügen die Datentype eigenen mapper ihre Felder hinzu, bevor die global gültigen Felder gefüllt werden.

Über dieses System konnten problemlos der existierende Datentyp, welcher nur Fließkomma-zahlen unterstützte, durch neue Datentypen erweitert werden. Beispielsweise konnte die grund-sätzliche Funktionalität dieses erhalten bleiben und durch eine Integer Konvertierung innerhalb der Logik zu einem IntegerDataSet erweitert werden. Um Charactere zu versenden, musste die Funktion zur Generierung der Daten nur umgeschrieben werden, um die entsprechenden Zei-chen aus dem gegebenen Alphabet zu generieren. Zum erstellen einer Pause sind weniger Daten relevant. Hier sind weniger Daten relevant. Da dieses Element aber die Verhaltensweise des Codes verändert, sind hier auch Anpassungen im Business-Layer notwendig. Dieses muss nun die Pau-sen in den Stream einfügen. Um die MI-TSA bzw TSA-DataSets zu erstellen gilt grundsätzlich das gleiche Prinzip, nur sind hier weit mehr Daten relevant. Diese Komplexität wird auf die jeweiligen Mapper ausgelagert um die Separierung der Zuständigkeiten zu gewährleisten.

Eine Anforderung des Frontend sorgt nur für einen Bruch von Liskov Substitution Principle², da sich die neuen Datentypen nicht ganz wie das original Verhalten, bzw eine Implementation dieser Funktion schwer an die Anforderungen des Frontend anzupassen ist. Während im original das FloatingPointDataSet die Möglichkeit anbot, einen Vorschau der zu sendenden Daten zu erhalten und diese als Graph zu visualieren, ist dies über Chars oder Schlafzyklen schwer möglich. Hierfür müsste man die DataSets weiter über neue Interfaces aufspalten oder im Frontend eine neue Möglichkeit der Darstellung für solche Fälle finden.

Eingabeanvalidierung

Ein wichtiger Aspekt bei der Entwicklung einer API ist die Validierung der Nutzereingaben. Im ursprünglichen Projekt wurde sowohl auf Frontend-Validierung als auch auf Backend-Validierung Wert gelegt. Die Frontend-Validierung erfolgte über im React Code integrierte JSON-Schemata³, während im Backend Jackson verwendet wurde, um die JSON-Keys den entsprechenden Feldern zuzuordnen. Diese Methode funktionierte grundsätzlich, erwies sich jedoch als zu ungenau und unflexibel und bedurfte vor allem einer manuellen Validierung.

Anpassung der JSON-Schemata Um den neuen, komplexeren Anforderungen gerecht zu wer-den, war es notwendig, die JSON-Schemata aus dem Frontend zu entfernen und ins Backend bzw. in die Datenbank zu überführen. Die Schemata werden nun in der Datenbank gespeichert und von der API bereitgestellt. Dies bietet den Vorteil, dass Anpassungen an den Schemata vorgenom-men werden können, ohne das Frontend neu bauen zu müssen. Zudem wird die Komplexität des Frontends reduziert, und Anpassungen an der Datenstruktur im Backend führen nicht mehr zu Komplikationen in der Kommunikation mit dem Frontend. Änderungen können direkt eingespielt werden, und durch entsprechendes Routing des Traefik kann die Datenstruktur überarbeitet wer-den, während eine alte Version noch läuft. Es muss lediglich nach dem Start des neuen Backend-Containers die Requests an ihn statt an den alten Container geleitet werden. Ein Nachteil ist die zusätzliche Serverbelastung, da die Schemata erst geladen und dann, abhängig von der Konfigu-rierbarkeit des DataSets, angepasst werden müssen, was zu Verzögerungen beim Nutzer führen kann. Angesichts der relativ geringen Datenmenge und den zielgerecht angepassten Schemata sowie passenderen Beschreibungen und Fehlermeldungen stellt dies jedoch einen akzeptablen Kompromiss dar. Darüber hinaus kann hier Caching zielgerichtet eingesetzt werden.

²Dieses Prinzip besagt, dass eine Superklasse

³JSON-Schemata sind ein IETF-Standard, der das Format und die Struktur eines JSON-Objekts definiert

Ein wichtiger Schritt bei der Überführung der JSON-Schemata ins Backend war der Umstieg auf Hibernate ORM 6, das nun auch JSONB-Support für Postgres in Java bietet, ohne dass eigene Converter-Klassen erforderlich sind. Dies vereinfacht den Umgang mit JSON-Objekten im Java-Code erheblich.

```
1 @Column(columnDefinition = "jsonb", name = "schema", nullable = true)
2 @JdbcTypeCode(SqlTypes.JSON)
3 private Map<String, Object> schema = new HashMap<>();
```

Listing 7.4 – Neuer Umgang mit JSONB-Elementen in Java

Über einen eigenen Service werden nun durch die Angabe des gewünschten DataTypes die JSON-Schemata dynamisch zusammengestellt. Dies beinhaltet das Hinzufügen und die Integration von Optionen, die zuvor über separate Anfragen gehandhabt wurden.

Beispielsweise haben nicht alle Machine-Learning-Modelle die Fähigkeit, Zukunftsprognosen zu erstellen. Da die Modelle jedoch alle derselben Kategorie angehören, wird für sie das gleiche Schema im Datenbankmodell gehängt. Hier muss das JSON-Schema dynamisch an die neuen Anforderungen angepasst werden. Normalerweise ist diese Option nicht vorhanden, daher muss der entsprechende Service sie nachträglich hinzufügen. Es gäbe auch die Möglichkeit, viele verschiedene Versionen der Schemata in der Datenbank zu speichern und zielgerichtet zu laden. Der Vorteil hier liegt in der Einfachheit und Versionierbarkeit. Darin liegt aber auch der Nachteil: Unterschiedliche Versionen müssen dennoch mit den zugrundeliegenden Datenstrukturen übereinstimmen. Eine dynamische Generierung über den Code hat dieses Problem nicht. Hier werden die Anforderungen der jeweiligen Version bereitgestellt. Es muss nur darauf geachtet werden, den Schema-Generator konsistent mit den Anforderungen des DataSets zu halten.

Ein weiterer Punkt, der grundlegend verändert wurde, ist der Umgang mit separaten Funktionen innerhalb der ursprünglichen DataSets. Im originalen Projekt gab es die Möglichkeit, diese durch die Optionen Residual, Season und Trend zu erweitern und zu konfigurieren. Dafür wurden mehrere Anfragen an das Backend gesendet, um die verfügbaren Optionen zu erhalten und anschließend je nach Auswahl der Option einen weiteren Request an das Backend gesendet, um das dafür vorgesehene Schema zu laden. Dies führte insgesamt zu einer geringeren übertragenen Datenmenge, die das Frontend verarbeiten musste, erhöhte jedoch den Aufwand für das Backend, da es mindestens sechs Anfragen bewältigen musste, um jeweils ein komplett konfiguriertes DataSet-Schema zu erstellen. Daher wurde auch dies geändert. Die jeweiligen Optionen sind nun Teil des initialen JSON-Schemas und werden durch die von JSON-Schema bereitgestellte Abhängigkeitsmethode dynamisch angezeigt.

Anpassung der Eingabeanvalidierung im Backend Die Eingabeanvalidierung im Backend erfolgte ursprünglich allein durch Jackson und gelegentliche Null-Checks. Dies ist ausreichend, wenn ausschließlich über ein Frontend kommuniziert wird, das jegliche Validierung übernimmt. Für eine komplexere API ohne angemessene Eingabeanvalidierung ist dieser Ansatz jedoch sehr unsicher und kann zu vielen, insbesondere für den Nutzer unklaren, Fehlern führen.

Jakarta bietet jedoch auch hierfür eine Lösung in Form der Bean-Validation, welche die JSR-380-Spezifikation⁴ umsetzt. Über das spring-boot-starter-validation-Artefakt erhält man Zugriff auf Annotationen wie @NotNull, @Positive, @Size und weitere, die eine direkte Validierung der

⁴JSR 380 ist eine Java-API-Spezifikation zur Validierung von Beans, die Teil von Jakarta EE und JavaSE ist. Sie stellt sicher, dass Beans bestimmte Anforderungen erfüllen.

DTOs ermöglichen. Zudem werden Fehlermeldungen an die jeweiligen Exceptions weitergeleitet. Somit greift die in Sektion 7.2.2 beschriebene Fehlerbehandlungsstrategie, und der Nutzer erhält direktes Feedback zu seinen Eingabefehlern.

7.2.2 Fehlerbehandlung

Ein wichtiger Aspekt ist der Umgang mit Fehlern. Fehler müssen, sofern möglich, dem Nutzer klar und strukturiert Mitgeteilt werden. In einer REST API heißt dies, dass im Fehlerfalle dem Nutzer eine ErrorMessage übergeben wird, warum die Eingabe oder Anfrage nicht akzeptiert oder bearbeitet werden kann.

Dafür mussten im ursprünglichen Projekt auch viele Änderungen vorgenommen werden. Einseitig wurde wie in Paragraph 7.2.1 beschriebene eine Bean Validation eingefügt. Diese kontrolliert bereits beim Eingang der Objekte, ob alle Kriterien erfüllt sind, wie beispielsweise das richtige Format, ob Zahlen positiv sind oder ob Namen einem gewissen Regex Muster folgen. Sollte dies nicht der Fall sein, wird eine Exception geworfen, welche von einem eigenen ExceptionHandler abgefangen wird.

ExceptionHandler sind ein zentrales Element in der Fehlerverarbeitung. Sie helfen enorm bei der Umsetzung des Fail Fast Prinzips[Fow04], welches prinzipiell besagt, dass Fehler möglichst schnell behandelt und die Bearbeitung abgebrochen werden soll.

Um die geworfenen Exceptions in verständliche ErrorMessage zu konvertieren und dem Nutzer zu senden bietet Spring die besagten ExceptionHandler. Diese fangen die geworfenen Exceptions ab und erstellen daraus passende Antworten.

```

1  @ControllerAdvice
2  @Slf4j
3  public class EntityNotFoundExceptionHandler {
4      @ExceptionHandler(EntityNotFoundException.class)
5      public ResponseEntity<RestError> handleEntityNotFoundException(
6          final EntityNotFoundException ex) {
7          log.error("Entity not found: {}", ex.getMessage());
8          return new ResponseEntity<>(new RestError()
9              .setMessage(ex.getMessage())
10             .setError("INVALID_ID")
11             .setI18nKey("INVALID_ID"),
12             HttpStatus.NOT_FOUND);
13      }
14  }
```

Listing 7.5 – Error Handling für den Fall, dass Ids nicht gefunden werden

In Listing 7.5 ist ein Beispiel für einen solchen ExceptionHandler zu sehen. Dieser fängt die EntityNotFoundException ab, welche geworfen wird, wenn eine Id nicht gefunden werden kann. Um diesen Fehler zu internationalisieren, wird ein i18nKey mitgegeben, welcher in der entsprechenden Sprachdatei nachgeschlagen wird und dem Nutzer die entsprechende Fehlermeldung in seiner Sprache anzeigt.

7.2.3 Absicherung der Anwendung über Spring Security

Nutzerverwaltung und Anmeldung Ein entscheidender Aspekt bei der Entwicklung einer Webanwendung ist die Nutzerverwaltung und die damit verbundenen Sicherheitsanforderungen. Das

ursprüngliche Projekt, obwohl als Webanwendung konzipiert, hatte keine Nutzerverwaltung implementiert. Dies führte dazu, dass jeder Nutzer Zugriff auf alle Projects, auch die von anderen Nutzern erstellten, hatte, ohne eine valide Möglichkeit, anderen Nutzern den Zugriff auf eigene Projekte zu verwehren. Es bot die Möglichkeit Projects hoch- und runter zu laden, eine etwas aufwändige Möglichkeit den Zugriff einzuschränken.

Die Integration eines User-Elements in das Datenbankmodell war daher unumgänglich. User-Objekte nehmen nun die zentrale Rolle ein, da Projects, und somit auch die darunter liegenden Entitäten, an das Objekt gebunden sind. Um sicherzustellen, dass jeder Nutzer nur Zugriff auf seine eigenen Projekte hat, müssen daher Nutzerdaten an den Requests hängen. Es bietet sich daher an, ein grundsätzliches Nutzerauthentifizierungskonzept zu implementieren. Hierfür wird Spring Security eingesetzt, ein Framework für Authentifizierung und Autorisierung, das auch Schutz vor gängigen Angriffen bietet. Es wurde konfiguriert, um JSON Web Tokens (JWT) zu erstellen und zu validieren, die nach erfolgreicher Anmeldung mittels E-Mail und Passwort dem Nutzer bereitgestellt werden.

Die Nutzung von JWT bietet mehrere Vorteile: Sie sind stateless, was bedeutet, dass der Server keine Session-Informationen speichern muss. Da JWT signiert sind, können sie als valide angesehen werden, solange das Secret sicher ist. Zudem sind sie unabhängig von der Anwendung einsetzbar, was besonders für das Zusammenspiel mit der Django API wichtig ist. JWTs bieten jedoch auch Nachteile: Sie sind nicht widerrufbar, was bedeutet, dass ein einmal gestohlener JWT weiterhin gültig ist und von dritten Nachgenutzt werden kann. Zur minderung des Risikos besitzen sie daher nur eine kurze Lebensdauer (time to live, TTL). Refresh-Tokens können hierbei helfen, da sie eine längere Lebensdauer besitzen und somit die Anzahl der notwendigen Anmeldungen reduzieren. Diese werden nur gesendet, sofern der JWT-Token abgelaufen ist.

Im Vergleich zu anderen Authentifizierungsmethoden wie LDAP ist die Integration von JWT einfacher und weniger aufwändig in der Konfiguration. Da JWT eine begrenzte Lebensdauer (time to live, TTL) haben, sind sie sicherer als API-Schlüssel, die regelmäßig erneuert werden müssen. Diese Notwendigkeit zur regelmäßigen Erneuerung war ein wichtiger Faktor bei der Entscheidung, JWT auch für die Django-API zu implementieren und zu konfigurieren.

Eine weitere Option wäre OAuth2.0 gewesen, ein weit verbreitetes Autorisierungsframework, das den Zugriff für Drittanbieteranwendungen ermöglicht. Trotz seiner Verbreitung wurde es aufgrund der hinzugefügten Komplexität nicht für die Integration in das Projekt ausgewählt.

Konfiguration von Spring Security Cross-Origin Resource Sharing (CORS) und Cross-Site Request Forgery (CSRF) stellten viele Herausforderungen während des Entwicklungsprozesses dar. Nachdem das ursprüngliche Projekt aufgespalten und Axios im Frontend integriert wurde (siehe 7.4.3), traten erstmals CORS-Probleme auf, da das Frontend und das Backend nun über unterschiedliche IP-Adressen im Docker-Netzwerk kommunizierten.

Anfangs wurde das Problem durch die Verwendung der Annotation @CrossOrigin behoben, was jedoch nur eine temporäre Lösung war. Spätere Probleme bei der Konfiguration mit NginX wurden schließlich durch Spring Security gelöst. Mit der Implementierung von Spring Security 6 kam auch die überarbeitete Version der SecurityFilterChain, die das Injizieren von frei konfigurierbaren CorsConfigurationSource Beans ermöglicht. Ihre erlaubten Origins können über die application.properties und damit über Umgebungsvariablen gesteuert werden.

Die Security-Chain kümmert sich auch um die Nutzervalidierung. Hierfür muss lediglich der eigene AuthenticationProvider als Bean injiziert werden. Dieser implementiert den OncePerRe-

questFilter und prüft den anhängenden JWT auf sein Ablaufdatum und validiert ihn. Refresh-Tokens werden über einen eigenen Endpunkt validiert und erneuert.

Anbindung des Nutzers an die Datenstruktur Die Anbindung des Nutzermodells an die bestehende Datenstruktur erfolgte, ohne den bestehenden Code groß anzupassen, aus zwei Gründen: Einerseits erleichtert es die Entwicklung, wenn der Fokus nicht auf Sicherheit liegt, andererseits ist es auch nicht die Aufgabe des Business Layers, sich um die Sicherheitsaspekte zu kümmern. Da die Nutzervalidierung über JWT bereits im Paragraph ?? stattgefunden hat, bevor die Requests an den jeweiligen RestController gehen, filtert Spring Security die Anfragen vorab.

Um den Nutzer dennoch an die Projekte zu binden, wurde ein aspektorientierter Programmieransatz gewählt. Dieser Ansatz, ein Paradigma innerhalb der objektorientierten Programmierung, versucht, Funktionalitäten über mehrere Klassen hinweg zu verwenden. Dies geschieht über eine Annotation, die an die gewünschte Methode angehängt werden muss. Da die Kommunikation mit der Datenbank über JpaRepository-Interfaces stattfindet, kann das Filtern nach Nutzern über das Ausnutzen von Default-Methoden in den Interfaces implementiert werden, ohne den restlichen Code anzutasten.

Die in Listing 7.6 dargestellte Annotation @AttachUser zeigt einen Spring-Interceptor, der Spring signalisiert, dass hier eine Aktion eingewoben werden muss. In der Terminologie der aspektorientierten Programmierung handelt es sich bei der Methode findById(), sowie jeder anderen Methode, um einen Joinpoint. Durch den Pointcut, welchen die @AttachUser-Annotation liefert, wird ein Aspekt (der UserAspect) auf eine Advice gemapped, die von der Annotation selbst bereitgestellt wird.

Da Tracks und deren DataSets an das jeweilige Projekt gebunden sind, ein Zugriff ohne JWT nicht möglich ist und das User-Objekt an die Datenbankabfragen gebunden wird, ist somit ein Zugriff auf fremde Projekte ausgeschlossen.

```

1 @Retention(RetentionPolicy.RUNTIME)
2 @Target(ElementType.METHOD)
3 public @interface AttachUser {
4 }
```

Listing 7.6 – Annotation um einen Advice Annotation zu signalisieren

7.2.4 Bidirektionale Kommunikation mittels Websockets

Nutzer mögen es visuelles Feedback zu bekommen, gerade bei Aktionen, deren direkten Verlauf sie nicht sehen können. Dieser Fall tritt ein, wenn der Nutzer die Daten an Kafka sendet, da dies über einen längeren Zeitraum stattfindet und, abhängig von den Konfigurationen, veränderlich ist. In der alten Version wurde das Senden von Inhalten über eine Status LED angedeutet, welche blinnte, sobald Daten gesendet wurden. Dies ist aber nur eine sehr grobe Anzeige, welche dem Nutzer nur bedingt weiterhilft. Da sie über einen Status in der ProjektResource angesteuert wurde, konnte sie auch nicht anzeigen ob der Sendevorgang bereits abgeschlossen ist.

Um solche Funktionalitäten zu ermöglichen bedarf es einer bidirektionale Kommunikationschicht. Im Gegensatz zu REST, welches auf dem Request-Response Prinzip basiert, ist Websocket eine bidirektionale Kommunikationsschicht, welche auf dem Publish-Subscribe Prinzip basiert und somit die Kommunikation in beide Richtungen ermöglicht. Somit kann der Server den Client über den aktuellen Status in real time informieren.

Der Einsatz von Websockets hat zu ein paar Problemen geführt, Websockets unterschützen keine AuthorizationHeader, weshalb der in Sektion 7.2.3 erläuterte Validierungsprozess nicht mit Websockets funktionierte, da normale Websocket keinerlei Header benutzen. Somit musste der Loginprozess, für die Websocket Controller, neu definiert werden. Um die Validierung ähnlich früh abzufangen wie im JWT Prozess musste daher ein eigener WebsocketJWTHandler implementiert werden, welcher den initialen ServerHttpRequest nimmt und über auf einen token Query-Parameter prüft. Für diesen QueryParameter wird der JWT genutzt. Es wird nur beim ersten connection Aufbau geschaut ob der JWT gültig ist, websockets führen keine weitere Authentifizierung weiter durch. Dies ist aber im Bezug zur Aufgabe, welche die Websockets übernehmen müssen, auch sehr passend. Die langen Tasks sollten nicht durch ablaufende Authorisierungskeys behindert werden.

Ist eine Websocket Connection erstellt, wird das senden vorbereitet. Dies bedeutet, dass alle DataSets aus ihrer Konfigurationen Daten erstellen und diese in einem speziellen Objekt ablegen. Sollte es sich bei den DataSets um MLDataSets oder TSADatasets handeln, muss hierfür ein RESTCall zur Django API getätigt werden (siehe Sektion 7.2.5).

Da die Elemente auf verschiedenen Threads arbeiten, müssen die aktuellen Statusinformationen Event gesteuert versendet werden. Dies bedeutet aber auch, dass diese Stelle auch einen direkten Überblick darüber hat, welcher Thread gerade welches DataSet versendet.

Hierzu wurde ein eigener EventService eingerichtet, welcher über den aktuellen Fortschritt von allen Threads informatiert wird. Wie in Figur 7.3 zu sehen ist, informieren die einzelnen Threads den EventService über einen wechsel des sendenden DataSets innerhalb eines Tracks.

7.2.5 Kommunikation mit der Django API über REST-Clients

Microservices zeichnen sich dadurch aus, dass sie unabhängig voneinander agieren und arbeiten können. Die Kommunikation zwischen den Services erfolgt über RESTCalls. Da in der aktuellen Architektur nur der Spring Service Daten vom Django Service nutzt, muss in Spring hierzu einen RESTClient implementieren, welcher die Anfragen an den anderen Service sendet und die Antworten verarbeitet.

Die Kommunikation zwischen den Services wird über JSON geregelt, wodurch die Kommunikation unabhängig von der Programmiersprache ist.

Anfänglich wurde hierfür die Webflux API von Spring verwendet, da die reine WebClient API nicht ausreichend war und den Implementationsaufwand unnötig erhöht hätte. Mit der Version 3.2.0 von Spring boot wurde eine neue WebClient API eingeführt, welche Funktional ähnlich der in Quarkus genutzten RestClient API⁵ ist und daher die Nutzung erleichtert (durch Reduktion des Boilerplate Codes) und das automatische Entitätsmapping übernimmt, wurde die Webflux API durch die neue WebClient API ersetzt.

Der WebClient kommt auch mit einer Failsafe API, auf die aber verzichtet wurde. Fehlercodes werden durchgereicht, aber erneutes senden von Requests wird nicht durchgeführt. Sollte die DjangoAPI nicht erreichbar sein, wird nur der Fehler an den Nutzer weitergegeben. Erneutes senden von Requests würde, sofern die Django API noch existiert, diese nur weiter belasten und der Nutzer wäre von enormen Delays betroffen, welche seine Konfiguration unvorhersehbar machen würden. Dies ist Kontraproduktiv. Unvollständige DataSets zu senden ist auch keine valide Strategie, da dies im zu testenden System zu ungewollten Verhalten führen würde.

⁵<https://quarkus.io/guides/rest-client-reactive>

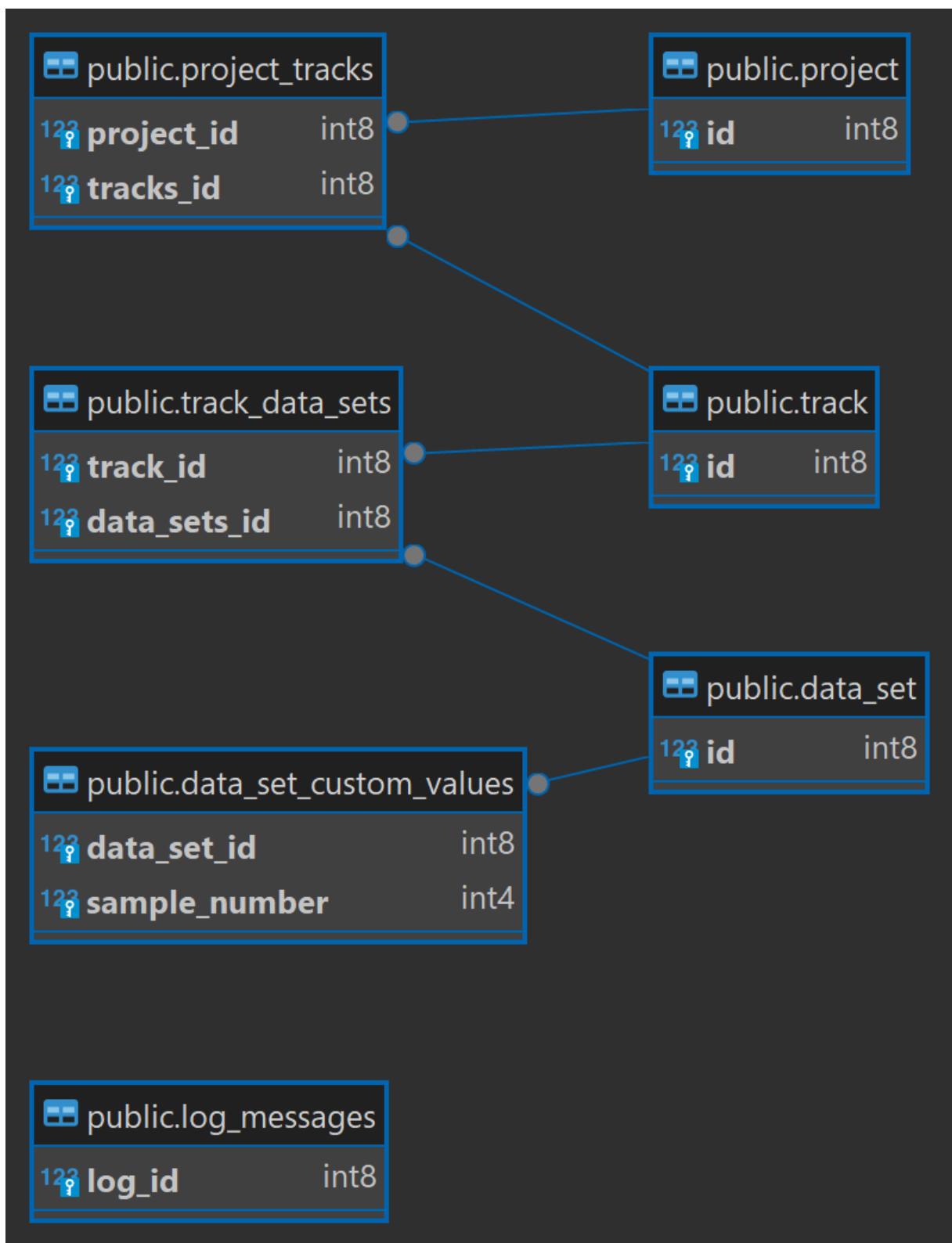


Abbildung 7.2 – Entity-Relationship-Diagramm der ursprünglichen Datenstruktur

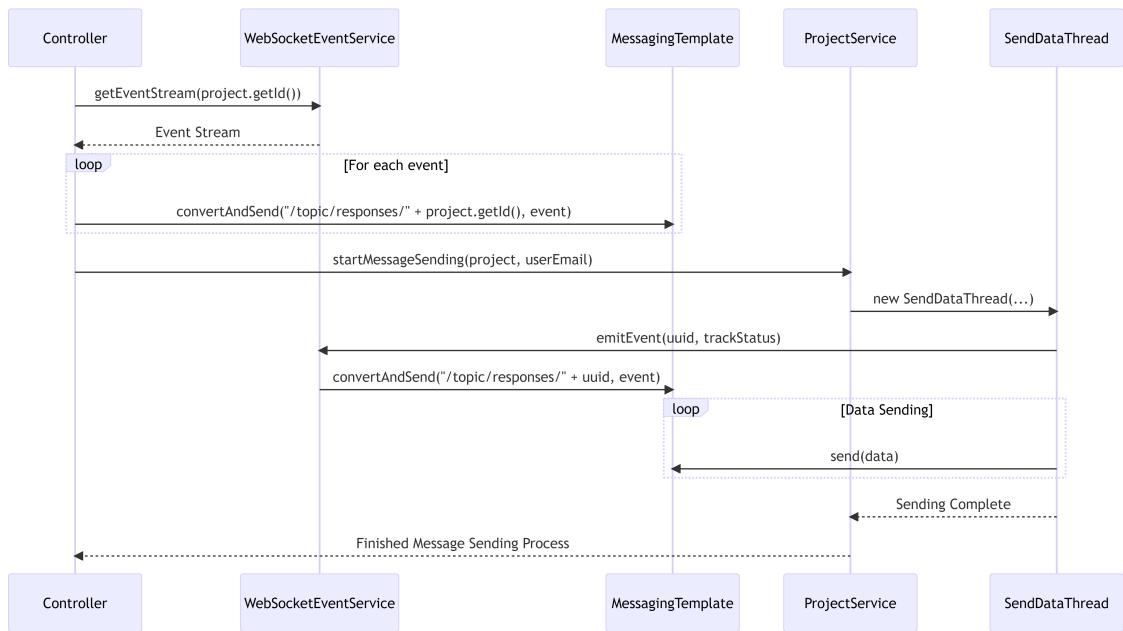


Abbildung 7.3 – vereinfachter Ablauf des Sendens von Statusinformationen mittels eines eigenen EventServices

7.3 Django

Python bietet 4 große Webframeworks an. Django, Flask, FastAPI und Pyramid. Django ist das bekannteste und am weitesten verbreitete Framework. Es ist ein Full-Stack-Framework, das viele Funktionen und Bibliotheken mitbringt, um die Entwicklung zu vereinfachen. Flask ist ein Micro-framework, das nur die grundlegenden Funktionen bietet, ansonsten eigenständig erweitert werden muss. Pyramid ist ein Framework, das sich in der Mitte dieser beiden Frameworks befindet. Es bietet mehr Funktionen als Flask, ist aber nicht so umfangreich wie Django. FastAPI hingegen ist ein neues Framework, welches auf Geschwindigkeit ausgelegt ist. Es ist noch ein recht neues Framework, welches noch in der Entwicklung steckt und daher nicht die Stabilität oder den Umfang anderer besitzt. Um eine stabile, wartbare und erweiterbare Software zu entwickeln, wurde daher Django als Framework ausgewählt.

7.3.1 Django API Architektur

Die Spring-Boot-Architektur ist von Regeln und Design Patterns geprägt, um flexibel und möglichst generisch agieren zu können. Unterstützt durch die objektorientierte Architektur von Java und Spring Injections, ist dies notwendig.

Python hingegen ist keine rein objektorientierte Sprache, sondern eine Skriptsprache, die OOP-Ansätze unterstützt, diese aber nicht erzwingt. Eine klare Trennung in Schichten ist hier nicht so einfach und würde oft zu erhöhtem Aufwand und unnötigem Boilerplate-Code führen. In Django werden Aufgaben in eigene Apps unterteilt, wobei sich die Aufgaben der Django API grob in drei Bereiche aufspalten lassen. Die Basis für beide Ansätze sind Daten, daher findet der Upload und die Vorverarbeitung von Daten in einer `sharedApp` statt. Dort sind auch die später detaillierter erklärten Imputationsalgorithmen und die Datenvorverarbeitungslogik angesiedelt.

Machine Learning und Time Series Analysis verhalten sich grundsätzlich unterschiedlich und erhalten daher jeweils ihre eigene App, was auch mit verschiedenen URL-Pfaden einhergeht und die Funktionalitäten logisch voneinander trennt.

Innerhalb der Apps wird die Architektur festgelegt. Da Interfaces und Dependency Injection nicht direkt Teil von Django sind und nur über zusätzliche Pakete und Frameworks eingebunden werden können, was zusätzliche Komplexität bedeutet, wurde hierauf verzichtet und eine dreischichtige Architektur gewählt. Diese unterteilt sich in Views, Services und Datenbankaktionen. Views sind fachlich äquivalent zu Spring-Controllern und definieren daher die REST-Endpunkte, deren Pfade zentral in einem separaten `urls.py` festgelegt werden. Die Services, hauptsächlich für CRUD-Funktionalitäten genutzt, bilden das Logik-Layer ab. Sie sind nicht für das Trainieren der ML-Modelle zuständig, da dies über Websockets gesteuert wird, welche separat definiert werden. Im TSA-Bereich kümmert sich ein Endpunkt um die Datenerstellung, da diese fast sofort zugänglich sind und kein WebSocket benötigt wird.

Um die Logik der Algorithmen von der CRUD-Funktionalität zu trennen, wurden diese in separate Ordner und Strukturen eingebunden. Die Software muss flexibel sein und eine einfache Integration neuer Algorithmen ermöglichen. Ein Builder Pattern, das sich einfach erweitern lässt und die Komplexität des Aufbaus eines komplexen Objekts reduziert, bietet sich hierfür an. Es ist durch den Einsatz von Default-Implementierungen einfach einsetzbar, ohne neu konfiguriert werden zu müssen. Dieses Konzept funktioniert jedoch nicht im Bereich der verschiedenen Algorithmen. Hier ist es notwendig, sich auf eine Basisklasse zu stützen, die gemäß dem OOP-Prinzip

bestimmte Funktionalitäten bereitstellt. Alle neuen Algorithmen müssen von dieser Klasse erben und die erforderlichen Funktionen überschreiben oder nutzen.

Im Datenbankdesign existieren die Algorithmen, und ein Strategy Pattern liefert basierend auf der gegebenen Konfiguration den passenden Algorithmus zurück.

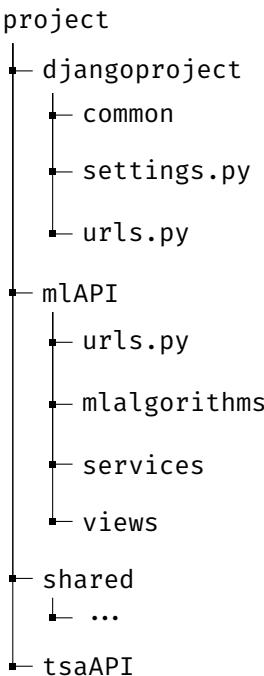


Abbildung 7.4 – Django Projekt Aufbau

7.3.2 Absicherung der Anwendung über Djangos Sicherheitskonzept

Wie bereits in Sektion 7.2.3 beschrieben, wurde für das Projekt eine JWT-Authentifizierung ausgewählt, um einerseits Nutzer zu verifizieren und andererseits die Daten an Nutzerkonten zu binden.

Django bietet hierfür eine Django-JWT-Library an, die nach einem Spring-ähnlichen Konzept funktioniert, indem eine Middleware eingesetzt wird, die die Validierung gleich zu Beginn durchführt. Hierbei traten jedoch zwei Probleme auf. Sowohl Spring als auch Django basieren ihre Authentifizierung auf einem Nutzer-Modell. Die Struktur des Django-Nutzermodells ist allerdings nicht direkt kompatibel mit dem, was Spring verwendet. Beide Modelle haben unterschiedliche Felder und sind somit nicht direkt miteinander kompatibel.

Es gibt zwei mögliche Lösungen: Einerseits könnte man auf die normalen Nutzermodelle beider Frameworks setzen. Hierbei müssten weder die Modelle angepasst werden noch die Login- bzw. Passwortvalidierung überschrieben werden. Zudem wären beide APIs stärker separiert und somit weniger voneinander abhängig. Nachteilig wäre allerdings die doppelte Nutzerverwaltung und der doppelte Anmeldeprozess. Obwohl dieser durch das Frontend abstrahiert werden könnte, bleibt die Lösung suboptimal. Eine Option wie OAuth2 wurde ebenfalls verworfen, sodass nur die Anpassung der Modelle als Lösung verbleibt.

Dabei traten einige Probleme auf. Spring und Django unterscheiden sich in der Art und Weise

der Nutzer- und Passwortvalidierung und setzen auf unterschiedliche Verschlüsselungsalgorithmen. Da die Priorität der Spring-API höher ist und die Django-API lediglich die Funktionalität der Spring-API erweitert, muss sich die Django-API nach den Vorgaben von Spring richten.

Um Nutzer in Django ähnlich wie in Spring an Methoden binden zu können, wurde ein ähnliches Modell verwendet. Python besitzt zwar kein direktes AOP-Modell, verfügt jedoch über ein ähnliches Interceptor-System.

```

1 def jwt_authenticated(view_func):
2     @wraps(view_func)
3     def _wrapped_view(request, *args, **kwargs):
4         if not request.user.is_authenticated:
5             return Response({'detail': 'Authentication credentials were not provided.'},
6                             status=status.HTTP_403_FORBIDDEN)
7         return view_func(request, request.user, *args, **kwargs)
8
9     return _wrapped_view

```

Listing 7.7 – Annotation basiertes Anbinden eines Nutzermodells an eine Methode

Wie in Listing 7.7 ersichtlich, fängt dieser Interceptor den Methodenaufruf ab und prüft, ob der Nutzer authentifiziert ist. Ist dies nicht der Fall, wird der Request abgewiesen. Andernfalls wird der Nutzer als Argument an die Methode angehängt und ist somit im Code verwendbar.

Dieses Modell ist nicht ideal, da der Nutzer bereits am Controller (oder View) angebunden ist und durch die verschiedenen Schichten weitergereicht werden muss. Daher kann er nicht einfach ersetzt oder ausgetauscht werden, ohne signifikante Teile des Codes zu verändern. Eine Alternative wäre die Nutzung eines thread-local Stores, um den Nutzer wieder in den DatenbankService zu verlagern und somit vom Business-Code zu trennen, wodurch er einfacher veränderbar wäre. Dieser Ansatz wurde zwar ausprobiert, erwies sich jedoch nicht als stabil genug und wurde letztendlich verworfen.

Im Django-Datenmodell hängen nicht alle Elemente an einem zentralen Punkt, wie es beispielsweise in der Spring-API der Fall ist. Hier ist der Nutzer nur an bestimmten Entitäten angebunden. Die grundlegenden Modelle im Bereich Machine Learning oder Time Series Analysis sind unabhängig vom Nutzer und stehen jedem zur Verfügung. Die hochgeladenen Daten und die Konfigurationen, die aus den Daten erstellt werden und später auch in Spring genutzt werden sollen, sind jedoch stark nutzerabhängig und sollten dementsprechend auch nur von ihm abrufbar sein.

7.3.3 Bidirektionale Kommunikation über Django Channels

Maschinelles Lernen ist ein sehr rechen- und zeitintensiver Prozess, der im klassischen Call-Response-Prinzip von Representational State Transfer (REST) schlecht aufgehoben ist. Nutzer reagieren, besonders im User Interface (UI)-Umfeld, sehr sensibel auf Verzögerungen und fehlendes visuelles Feedback. Es ist wichtig, dass Nutzer sehen können, dass etwas passiert, und sie sollten daher auch über den aktuellen Fortschritt informiert werden.

Die sinnvollste Lösung für dieses Problem ist die Nutzung eines Websockets, der an die Callbacks der Modelle gebunden wird und somit den Nutzer über den aktuellen Fortschritt informiert. Django unterstützt Websockets⁶ nicht direkt, weshalb hier auf das Framework Channels

⁶<https://channels.readthedocs.io/en/latest/>.

zurückgegriffen wurde. Channels integriert Websockets in Django und bietet somit die Möglichkeit, diese zu nutzen. Channels basiert auf Redis und verwendet es als Message Broker. Redis, wie bereits in 6.3 erwähnt, ist eine In-Memory-Datenbank, die aufgrund ihrer Schnelligkeit gut für die Kommunikation zwischen Threads geeignet ist.

```

1  class StatusConsumer(JsonWebsocketConsumer):
2      runners = []
3
4      def connect(self):
5          group_name = f"model{self.scope['ml_id']}solution{self.scope['sol_id']}"
6          try:
7              async_to_sync(self.channel_layer.group_add)(group_name, self.channel_name)
8              self.scope["group"] = group_name
9              self.accept()
10         except Exception as e:
11             logging.error(e)
12             raise UnknownErrorException("cannot connect", reason=str(e))
13
14     def disconnect(self, close_code):
15         async_to_sync(self.channel_layer.group_discard)
16             (self.scope["group"], self.channel_name)
17
18     def receive_json(self, message, **kwargs):
19         response = self.prepare_data(message)
20         self.send_json(response)
21
22     def chat_message(self, event):
23         self.send_json(event["message"])
24
25     def chat_disconnect(self, event):
26         self.send_json(event["message"])
27         self.close()
28         self.runners.get(event["uuid"])

```

Listing 7.8 – Websocket Konsum

Der Code in Listing 7.8 präsentiert einen JSON-Konsum, der eine WebSocket-Anfrage annimmt und eine Verbindung aufbaut. Aus der initialen Anfrage werden die trainingsrelevanten Informationen extrahiert und an die entsprechende Stelle im Prozess weitergeleitet.

Nachdem die Modelle trainiert worden sind, wird der finale Status an den Nutzer gesendet und anschließend die Verbindung getrennt.

7.3.4 Fehlerbehandlung und Verarbeitung

Das System implementiert ebenfalls ein interceptor-basiertes Error-Handling nach dem Fail-FastPrinzip. Illegale Optionen, invalide Parameter und ungültige Dateitypen führen zum Auslösen einer Exception, die von speziell dafür vorgesehenen Exception-Handlern abgefangen und verarbeitet wird. Auch hier wurde ein Format gewählt, das der Spring-Variante ähnelt, um ein kohärentes und einheitliches Fehlermeldungsprinzip zu gewährleisten.

```

1 def exception_handler(exc, context):
2     def build_error_response(exception, status_code):
3         response_data = {

```

```

4         "error": str(exception),
5         "reason": exception.reason if hasattr(exception, 'reason') else None,
6         "i18nKey": exception.i18nKey if hasattr(exception, 'i18nKey') else "UNDEFINED"
7     }
8     return JsonResponse(response_data, status=status_code)
9
10    exception_mapping = {
11        NotFoundException: status.HTTP_404_NOT_FOUND,
12        BadRequestException: status.HTTP_400_BAD_REQUEST,
13        InvalidFileTypeException: status.HTTP_400_BAD_REQUEST,
14        UnknownErrorException: status.HTTP_500_INTERNAL_SERVER_ERROR,
15        UniqueConstraintViolationException: status.HTTP_409_CONFLICT,
16        UserNotAuthenticated: status.HTTP_403_FORBIDDEN
17    }
18
19    if type(exc) in exception_mapping:
20        return build_error_response(exc, exception_mapping[type(exc)])
21
22    if isinstance(exc, UnknownErrorException):
23        logging.error(f"unknown error: {exc}")
24
25    response = drf_exception_handler(exc, context)
26    if response is not None:
27        return response
28
29    return JsonResponse({'error': 'Unexpected server error'},
30                        status=status.HTTP_500_INTERNAL_SERVER_ERROR)

```

Listing 7.9 – Exception Handler

Die in Listing 7.9 dargestellte Exception Map ermöglicht es, selbst definierte Exceptions auf einen REST Error Code zu projizieren. Dadurch ist sie flexibel in der Nutzung und leicht anpassbar.

7.3.5 Validierung der Daten und Vorverarbeitung

Es ist nicht davon auszugehen, dass alle Daten in einer homogenen Form vorliegen und normalisiert sind. Dies stellt jedoch eine grundlegende Anforderung dar, nicht nur um die Modelle effektiv konstruieren zu können, sondern auch um ...

Fehlende Werte im Datensatz Die Handhabung von Daten in Zeitreihen stellt eine besondere Herausforderung dar, insbesondere wenn nicht sichergestellt ist, dass die Daten immer im gleichen Abstand aufgezeichnet oder mit Zeitstempeln versehen wurden. Unter diesen Bedingungen wurde ein Algorithmus entwickelt, der die Struktur der Daten erkennt, sie entsprechend kategorisiert oder umwandelt und in ein uniformes Format überführt.

Angesichts der Komplexität, die mit der Unterstützung verschiedener Dateiformate einhergeht, wurden einige Annahmen getroffen:

1. Bei mehreren unbenannten Datensätzen in einer Datei wird geprüft, ob eine dieser Zeitreihen konsequent linear ansteigt. Ist dies der Fall, werden diese Werte als Zeitstempel interpretiert.
2. Ist keine konsequent steigende Zeitreihe zu finden, wird den Daten ein normaler Anstieg zugeordnet.

3. Dateiformate wie JSON oder CSV, die Header oder Dictionaries nutzen, erfordern eine spezielle Taxonomie, um die Zeitreihe im Dokument zu identifizieren, die für die Zeiterfassung gedacht ist. Wenn eine solche gefunden wird, wird sie jeder Zeitreihe hinzugefügt.

Basierend auf diesen Annahmen wurde ein Algorithmus entwickelt, abhängig vom Datenformat. Dieser durchsucht die Datei, testet auf mögliche Strukturen und bearbeitet sie anschließend, wie in Abbildung 7.5 dargestellt.

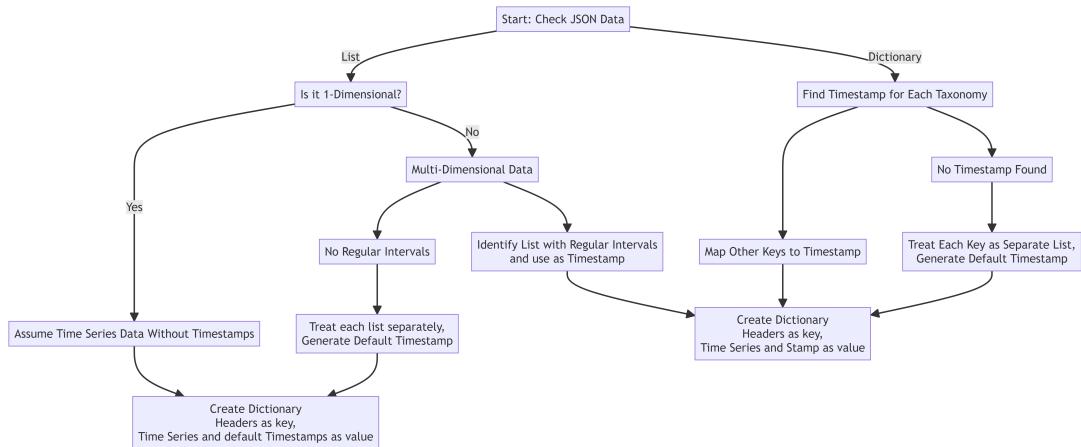


Abbildung 7.5 – Algorithmus zum zerlegen einer hochgeladenen JSON Datei

Diese Vorverarbeitung der Daten erfolgt bereits beim Hochladen und stellt sicher, dass die Daten im weiteren Verlauf immer in einem uniformen Format vorliegen. Die Überprüfung der Daten auf Lücken wird erst bei Bedarf durchgeführt. Da nur die Zeitstempel eine Rolle spielen und diese im Unix-Format vorliegen, berechnet der Algorithmus die Differenz aufeinanderfolgender Werte. Hierbei wird ein Schwellenwert definiert, um kleine Ungenauigkeiten zu ignorieren. Bei signifikanten Abweichungen von der Standarddistanz wird eine Lücke angenommen.

Der Algorithmus zum Identifizieren und Füllen von Lücken ist in Listing ?? zu finden. Er beschreibt, wie Lücken nach bestimmten Kriterien

definiert werden und zeigt auf, wie diese geschlossen werden. In diesem Schritt werden die Lücken noch nicht klassifiziert oder gefüllt, sondern lediglich mit NaN-Werten markiert. Dadurch entsteht eine Zeitreihe, in der alle Werte in regelmäßigen Abständen vorhanden sind.

```

1 def __find_differences_in_steps_and_fill_them_with_nan(train_data, timestamps_normalized):
2     processed_train_data = []
3     processed_time_stamps = []
4     for i, (train_d, timestamp_d) in enumerate(zip(train_data, timestamps_normalized)):
5         differences = np.diff(timestamp_d)
6         threshold = differences.min() + differences.min() * error_percentage
7         large_gap_indices = np.where(differences > threshold)[0]
8         start_gap = large_gap_indices
9         end_gap = large_gap_indices + 1
10        diff = np.ceil(-(differences[end_gap] - differences[start_gap]) / threshold)
11        new_timestamps = np.copy(timestamp_d)
12        new_train_d = np.copy(train_d)
13        for index, size in sorted(zip(end_gap, diff), key=lambda x: x[0], reverse=True):
14            nan_array = np.full(int(np.round(size)), np.nan)
15            new_timestamps = np.insert(new_timestamps, index, nan_array)
  
```

```

16     new_train_d = np.insert(new_train_d, index, nan_array)
17     processed_train_data.append(new_train_d)
18     processed_time_stamps.append(new_timestamps)
19     return processed_train_data, processed_time_stamps

```

Listing 7.10 – Algorithmus zum Auffüllen von Lücken

Für das Füllen von Lücken wurden unterschiedliche Ansätze abhängig von der Größe der Lücke gewählt. Kleine Lücken werden durch Imputationsalgorithmen gefüllt, indem die umgebenden Werte herangezogen werden, um die wahrscheinlichsten Ersatzwerte zu ermitteln. Es wurden verschiedene Algorithmen implementiert, zwischen denen der Nutzer per Eingabe im Frontend wählen kann. Diese Algorithmen werden als Funktion übergeben, erhalten die relevanten Daten und liefern die aufgefüllte Version zurück. Dies ermöglicht ein sehr flexibles Vorgehen und kann daher auch ohne großen Aufwand erweitert werden. Bei größeren Lücken, wo der Informationsverlust zu groß wäre, bleiben die Werte leer und erhalten den in Python üblichen NaN-Wert (Not a number). Die Handhabung dieser leeren Werte obliegt den nachfolgenden Algorithmen, die in den Sektionen 7.3.6 und 7.3.7 näher erläutert werden.

Uniformisierung der Daten Eine uniforme Version der Daten ist besonders wichtig im Bereich des maschinellen Lernens, da Trainingsdaten in unterschiedlichen Längen und Bereichen (beispielsweise von 0 bis 1) die Ergebnisse verfälschen oder das Training erschweren können. Daher ist es notwendig, die Daten zu normalisieren und auf eine einheitliche Länge zu bringen. Oft ist auch eine Reduzierung oder Komprimierung der Daten sinnvoll. Zwar gehen dabei Informationen verloren, aber da die Grundstruktur des Datensatzes erhalten bleibt und die Performance des Modells deutlich erhöht wird, sollte diese Option den Nutzern angeboten werden. Teilweise ist dies auch notwendig, da Modelle ab einer gewissen Größe nicht mehr auf jeder Hardware laufen.

Den Nutzern werden viele Möglichkeiten gegeben, selbstständig die Komprimierung vorzunehmen und eventuell lineare Trends aus den Daten zu entfernen. Weitere Optionen können problemlos hinzugefügt werden, da dafür lediglich ein JSON-Schema, die Implementierung sowie die Integration in das Strategy Pattern notwendig sind. Alle Änderungen im Vorverarbeitungsschritt werden in der Datenbank gespeichert und sind umkehrbar, sodass beim späteren Versenden der Daten diese wieder in der Form vorliegen, in der die Originaldaten hochgeladen wurden. Die verschiedenen Implementierungen werden über ein Strategy Pattern gezielt angesprochen und über JSON-Dictionaries mit ihrer spezifischen Konfiguration geladen.

7.3.6 Generalisierung und Nutzung der Machine Learning Modelle

Python ist eine der führenden Sprachen im Bereich des maschinellen Lernens und bietet daher eine umfangreiche Auswahl an Bibliotheken und Frameworks, die mit hilfreichen Komponenten und Funktionen ausgestattet sind. Große Frameworks wie TensorFlow oder PyTorch bieten zahlreiche Möglichkeiten, sind jedoch auch sehr komplex und erfordern viel Zeit, um gewünschte Modelle zu entwickeln. Aus diesem Grund wurde in diesem Projekt auf Keras gesetzt, das eine Abstraktionsebene über TensorFlow bietet und somit die Komplexität reduziert. Keras ermöglicht das Erstellen von Modellen über ein ANN, die sich durch die Konfiguration der Schichten und Neuronen anpassen lassen.

Wie im Kapitel 2 ersichtlich, sind GANs und RNNs verbreitete Ansätze, um aus einer Menge von Trainingsdaten synthetische Daten zu generieren. Da ein Ziel dieser Arbeit der Vergleich dieser

Ansätze ist, wurden verschiedene Modelle beider Varianten implementiert.

In der Praxis unterscheidet die Django-API nicht zwischen den beiden Varianten, da beide über dasselbe Interface angesprochen werden. Das zugrundeliegende Strategy Pattern liefert die passende Implementierung.

```
1 class GeneralMLModel:
2     data: List[np.ndarray] = None
3     run_information: RunInformation = None
4
5     def set_config(self, config: RunInformation):
6         """ set configuration for model """
7         pass
8
9     def set_data(self, data: List[np.ndarray]):
10        """ set data for model """
11        pass
12
13    def run(self):
14        """ start training the model """
15        pass
16
17    def predict_data_from_model(self):
18        """ predict data from model """
19        pass
20
21    def forecast(self, limit: int, starting_point: int = -1) -> np.array:
22        """ forecast data from model, does not work for every model """
23        pass
24
25    def save(self, model: Model) -> RunInformation:
26        """ write model to file and save base64 string in database """
27        pass
28
29    def load_model(self) -> Model:
30        """ write model from database to file and load it from there """
31        pass
32
33    def create_image(self, real: List[np.array], synthetic: List[np.array], title: str):
34        """ create an image and save it as base64 string """
35        pass
```

Listing 7.11 – Grundklasse zum Trainieren der Machine Learning Modelle, sie stellt Grundmethoden und geteilte Funktionalitäten bereit

Generative Modelle Zwei funktionierende GANs wurden implementiert: das normale GAN und das CGAN. Mit Wasserstein Generative Adversarial Networks (WGANs) und TGANs wurde ebenfalls experimentiert, und es existieren Testversionen dieser im Testframework (siehe Abschnitt 8.2.2). Da diese jedoch nicht die gewünschten Ergebnisse lieferten, stehen sie momentan nur in der Entwicklungsumgebung zur Verfügung und werden nicht von der API angeboten. Beide implementierten Varianten umfassen einen Generator und einen Diskriminatoren, wobei die Eingabedimensionen von der Länge der zu trainierenden Zeitreihe abhängen. Daher spielt die Normalisierung der Daten eine entscheidende Rolle, um Übereinstimmungen in den Dimensionen der

Schichten zu gewährleisten.

Rekursive Modelle RNNs analysieren Daten in ihrer zeitlichen Abfolge und eignen sich daher besonders gut für die Analyse oder Vorhersage von Zeitreihen. Die von ihnen generierten Daten sind stark abhängig von der Variabilität der Trainingsdaten. Da sie jedoch in der Lage sind, Vorhersagen über künftige Verläufe zu erstellen, bieten sie Möglichkeiten, die generative Modelle nicht bieten. Deshalb wurden sie in das System integriert. Die Auswahl der Parameter erfolgt allerdings erst in der Spring-API, da dort die entsprechenden Datensätze erstellt und die JSON-Schemata mit den dazugehörigen Regeln definiert werden.

Abhängigkeitsprobleme Ein wesentliches Problem stellte die Integration weiterer Modelle aus verschiedenen Bibliotheken dar. Viele Bibliotheken befassen sich mit Datengenerierung, sind jedoch untereinander nicht immer kompatibel, was zu Abhängigkeitskonflikten führte. Dies verhinderte die direkte Nutzung dieser Bibliotheken.

7.3.7 Generalisierung und Nutzung der Zeitreihenanalysealgorithmen

Die Zeitreihenanalyse ist der zweite Hauptansatz zur Generierung von Daten. Dabei werden die Daten in ihre Bestandteile zerlegt, um Muster zu erkennen und diese zu nutzen, um neue Daten zu generieren. Wie bereits in Kapitel 2 beschrieben, gibt es verschiedene Ansätze für die Zeitreihenanalyse, die sich in ihrer Komplexität und ihren Ergebnissen unterscheiden.

Zwei vielversprechende Ansätze in diesem Bereich sind die Singular Spectrum Analysis (SSA) und die Empirical Mode Decomposition (EMD). Diese Methoden zielen darauf ab, Muster in den Daten durch die Zerlegung in (IMFs) zu identifizieren und für die Generierung neuer Daten zu nutzen. Die Implementierung dieser Algorithmen ist zwar rechenintensiv, aber im Vergleich zu maschinellen Lernansätzen nicht so aufwendig, da sie nicht trainiert werden müssen und nur einmalig ausgeführt werden. Python bietet bereits Bibliotheken, die diese Algorithmen implementieren. Daher besteht die Herausforderung hauptsächlich darin, die Daten in das richtige Format zu bringen und dem Nutzer Zugriff auf die generierten Daten zu gewähren.

Obwohl dieser Ansatz in der Testphase nicht berücksichtigt wurde, bietet die Zerlegung der Daten in ihre Bestandteile einen weiteren interessanten Vorteil für den Nutzer. Da die Daten dem Nutzer grafisch dargestellt werden, kann er selbst Muster erkennen und diese nach Belieben verschieben. Dies ermöglicht es ihm, die Ausgangsdaten zu manipulieren und zu verändern, um so neue Daten zu generieren. Der Nutzer behält somit die Kontrolle und überlässt sie nicht nur dem Algorithmus. Weitere Details dazu werden im Kapitel 7.4.9 erläutert.

Um weitere Ansätze zu testen und anzubieten, wurden auch klassische Methoden wie die Amplitude Modulation and Noise (AMIRA) und der Cubic Spline Algorithmus implementiert. Diese Methoden interpolieren die Daten und bieten

somit eine andere Form der Datenabstraktion. Beide Methoden sind sehr schnell und liefern gute Ergebnisse. Allerdings sind sie nicht in der Lage, neue Daten zu generieren, da sie lediglich die vorhandenen Daten interpolieren und nicht nach Mustern suchen.

Die Integration dieser Algorithmen in die API erfolgt ähnlich wie im Kapitel 7.3.6 und sie sind ebenfalls über ein Strategiemuster auswählbar.

7.3.8 Das Datenbank Modell

Aus den in den vorherigen Sektionen beschriebenen Anforderungen an die Datenstruktur wurde ein Datenbankmodell entwickelt, das die Anforderungen erfüllt und gleichzeitig flexibel genug ist, um die Erstellung komplexer Simulationsumgebungen zu ermöglichen. Dieses ist in Abbildung 7.6 dargestellt und zeigt den aktuellen Stand der finalen Software.

Wie in dem Bild zu sehen ist, dient der Nutzer als zentraler Punkt und besitzt viele verschiedene Entitäten, die mit ihm verknüpft sind. Wichtig sind hiervor allem die TrainingData, welche die hochgeladenen Daten beinhalten, die Projects, welche seine Projekte aufspannen, und die jeweiligen Konfigurationen für ML und Time Series Analysis (TSA) Modelle. Die ausenstehenden Entitäten der Schemata sind für die flexible Zusammensetzung der jeweiligen JSON-Schemata notwendig.

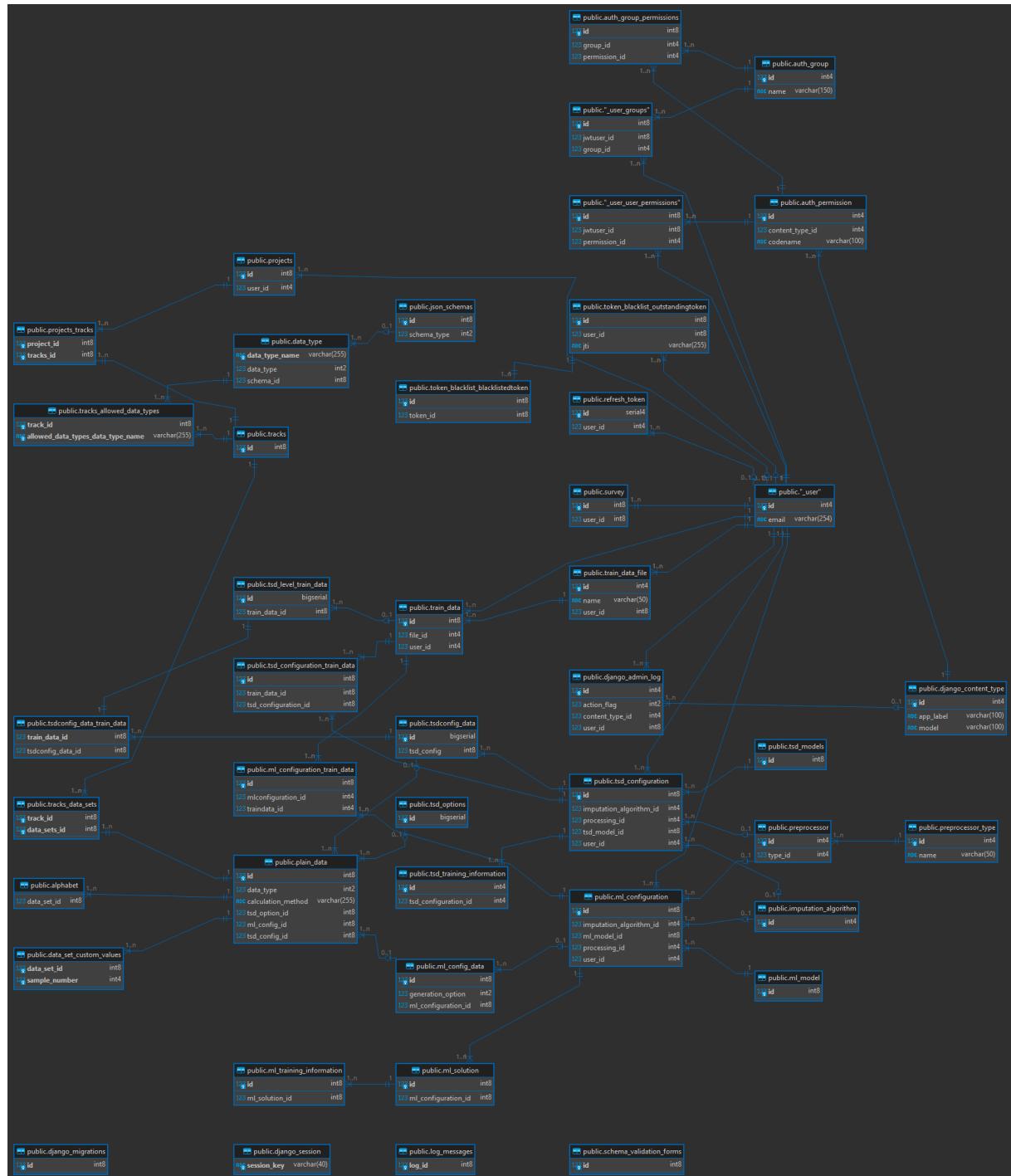


Abbildung 7.6 – Entity-Relationship-Diagramm der aktuellen Datenstruktur

7.4 React Implementation

7.4.1 Grundlagen der UI

Ein zentrales Konzept von React ist die Verwendung von Komponenten. Diese Komponenten sind wiederverwendbare UI-Elemente, die jeweils über eigenes Styling und Verhalten verfügen. Der wesentliche Vorteil dieses Ansatzes ist die Förderung von Wiederverwendbarkeit und Konsistenz innerhalb der Applikation. Durch die Wiederverwendung von Komponenten wird eine einheitliche Benutzeroberfläche gewährleistet. UI-Elemente verhalten sich in der gesamten Anwendung konsistent. Ein kohärentes Design verbessert die Benutzererfahrung und erfüllt damit zwangsläufig einen der in Sektion 5.2.3 vorgestellten Punkte der ISO 9241-110. Zudem fördert dieser Ansatz die Entwicklungseffizienz und hält die Codebasis überschaubar.

7.4.2 Grundideen des UI-Designs

Im Gegensatz zur ursprünglichen Version, die in mehrere Aufgabenbereiche unterteilt war, empfiehlt sich nun die Integration einer zentralen Navigationsleiste für eine bessere Übersichtlichkeit. Diese ermöglicht es den Nutzern, problemlos zwischen verschiedenen Bereichen und Menüs zu wechseln, was die Anzahl der Klicks zwischen den Interaktionen reduziert. Dies minimiert nicht nur den Aufwand für den Nutzer, sondern verbessert auch die allgemeine Benutzererfahrung. Die im Kapitel 10 vorgestellte Studie dient zur Überprüfung der hier vorgestellten Konzepte.

Nach dem Login landet der Nutzer auf der Hauptseite, die eine Übersicht über alle verfügbaren Möglichkeiten bietet. Dies umfasst eine kurze Beschreibung des Ablaufs sowie Erklärungen zur Funktionsweise der gewählten Technologien. In der Navigationsleiste kann der Nutzer farblich hervorgehoben sehen, wo er sich befindet, und kann zu anderen Bereichen wie der Konfiguration der Modelle, der Verwaltung der Daten oder dem Erstellen von Projects navigieren. Die Elemente Studie und Dokumentation sind nur temporär und sollten in der Hauptanwendung entfernt werden. Sie dienen lediglich der Durchführung und Dokumentation der Studie.

7.4.3 Kommunikation mit den APIs

Um die Kommunikation und Fehlerverarbeitung zwischen den Services zu vereinheitlichen, wurde die Axios-Bibliothek integriert. Axios implementiert die Promise-API und ermöglicht somit asynchrones Arbeiten. Es bietet eine automatische Serialisierung von JSON zu TypeScript-Typen und Interfaces, was ein direktes, typenbasiertes Arbeiten mit der React-Umgebung erlaubt. Da React zudem Multi-Browser-Unterstützung bietet, kann durch die Auslagerung dieser Problematik auf eine externe Bibliothek die Webanwendung mit allen gängigen Browsern kompatibel gemacht werden.

Zusätzlich erlaubt Axios die Implementierung von Interceptors. Diese überprüfen eingehende und ausgehende Requests auf ihren Status und ermöglichen eine direkte Verarbeitung. Das Konzept dahinter ist in Figur 7.8 beschrieben. Fehlercodes wie '403 Not Allowed', die beispielsweise auftreten, wenn der JWT-Token abgelaufen ist, können durch einen separaten Refresh-Token erneuert werden, ohne dass eine erneute Anmeldung vom Nutzer erforderlich ist. Andere Fehlercodes kommen mit einem I18N-Fehlercodefeld aus dem Backend, um diese übersetzen zu können und so dem Nutzer genau anzuzeigen, was und warum etwas schiefgelaufen ist.

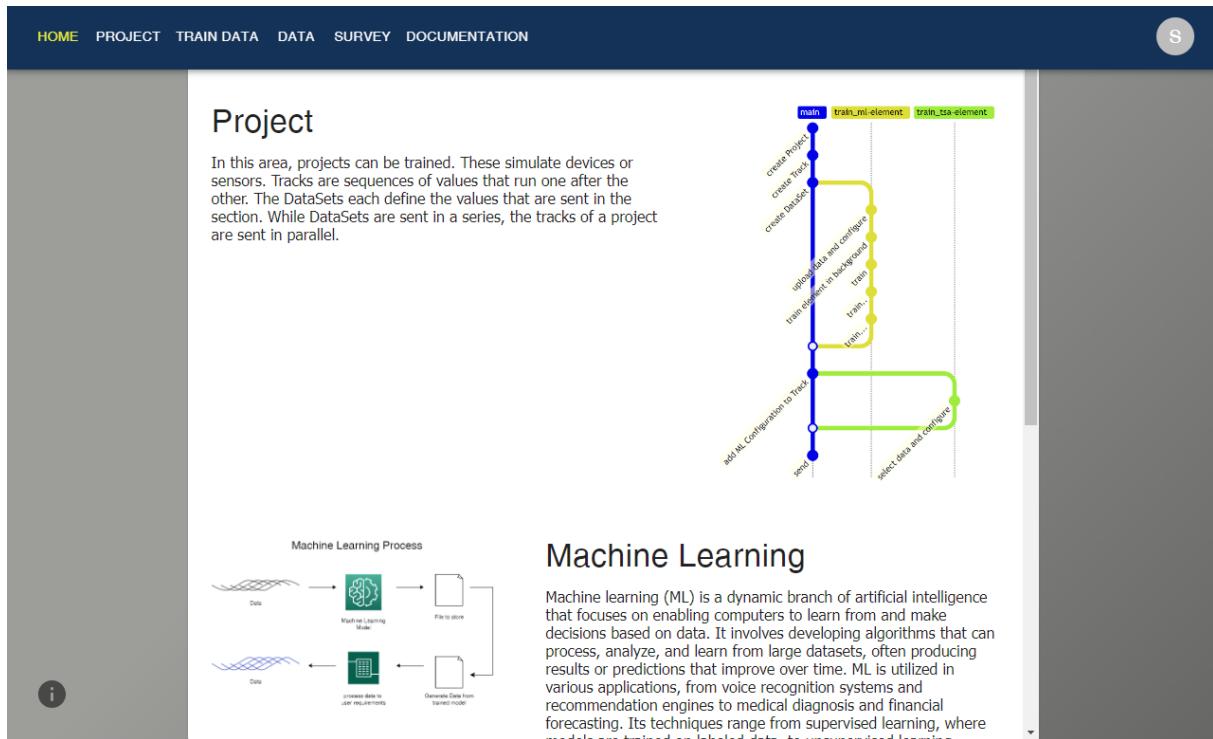


Abbildung 7.7 – vereinfachter Ablauf des Sendens von Statusinformationen mittels eines eigenen EventServices

7.4.4 Eingabenvierlidierung über JSON Schema

Nutzern die Eingabe möglichst einfach und verständlich zu gestalten und dabei auch Fehler zu vermeiden ist eine der wichtigsten Aufgaben eines angenehmen User Interfaces und trägt ungemein zur allgemeinen User Experience bei. Um dies dynamisch und flexibel zu erreichen, wurde eine Validierung der Eingaben von den ursprünglichen JsonForms durch rsjf⁷ ersetzt. Diese überprüft die Eingaben auf ihre Korrektheit und gibt dem Nutzer ein visuelles Feedback, sollte etwas nicht stimmen. Im Gegensatz zur alten Variante aber erlaubt rsjf auch die Validierung von komplexeren Schemata und kann diese sogar in Kombination mit einer UI Definition darstellen. Dies erlaubt es Validierung der Eingaben und Erstellung der UI zu verbinden und somit eine einheitliche und konsistente Darstellung zu gewährleisten.

Um die Felderbeschreibung und Fehlercodes zu übersetzen wurden Funktionen überschrieben welche das gesendete JsonSchema überschreiben und Fehlercodes sowie die dazugehörigen Argumente übersetzen. Dies erlaubt es dem Nutzer die Anwendung komplett in seiner eigenen Sprache zu nutzen und somit die User Experience zu verbessern. Einen Nachteil hat dieses Vorgehen aber. Die Übersetzung der Beschreibungen setzt voraus, dass die Beschreibungen aus I18n Schlüsseln bestehen. Fremde, oder neue User Interfaces, können die vom Backend bereitgestellten Schemata nur vernünftig darstellen, wenn diese die gleichen Schlüssel sowie Übersetzungsdateien verwenden. Ohne eine Übersetzung ist nur der Schlüssel sichtbar, was für den Nutzer wenig hilfreich ist.

⁷react-jsonschema-form ist ein react component, der ähnlich wie jsonforms, aus JSON Schemas React Components zusammenbaut

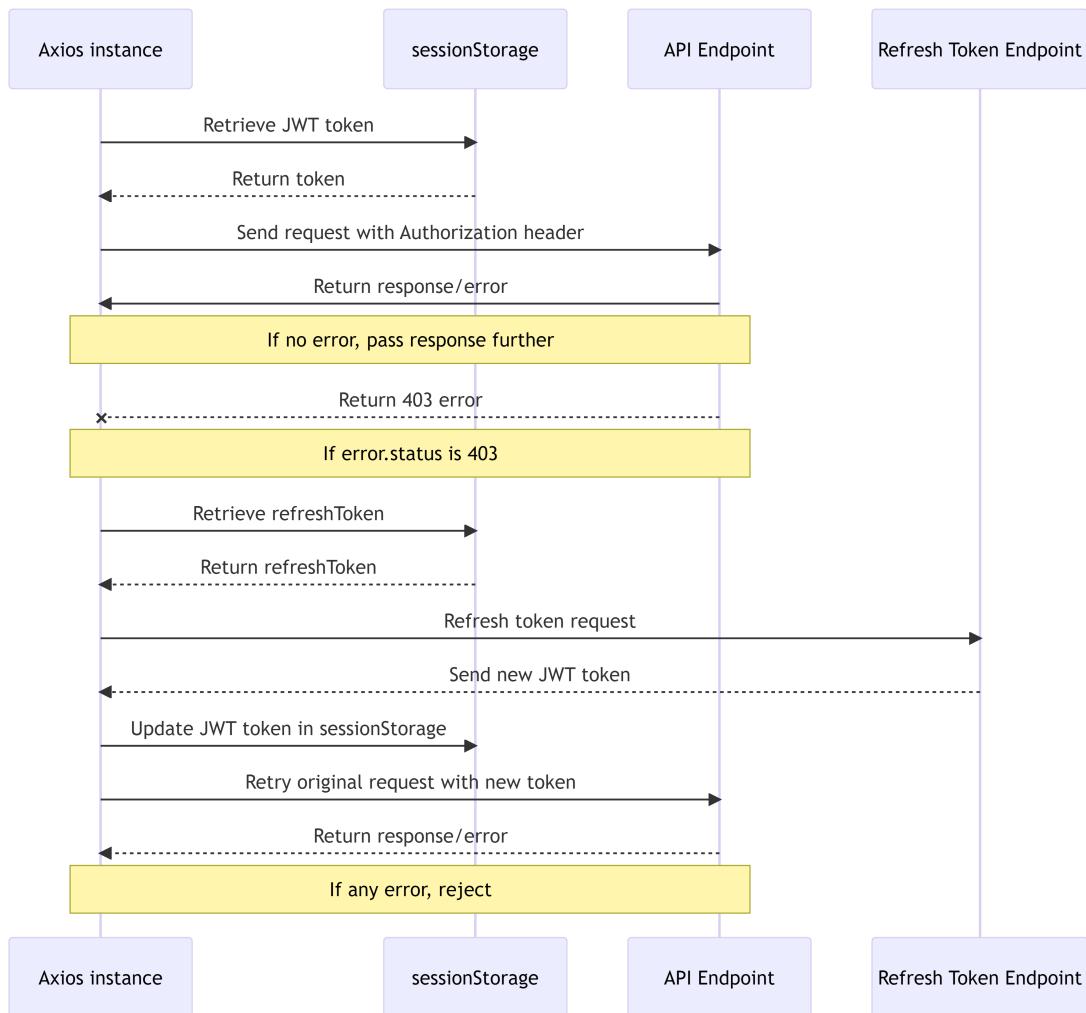


Abbildung 7.8 – vereinfachter Ablauf des Sendens von Statusinformationen mittels eines eigenen EventServices

Ein weiterer großer Vorteil von rsjf ist die Möglichkeit, die Schemata dynamisch zu laden. Anstatt wie bereits in Abschnitt 7.2.1 angedeutet die Schema fest im Code zu integrieren, was bedeutet hätte diese einzeln und für jeden Datentyp neu erstellen zu müssen, können diese durch das laden der Schema aus der API, unabhängig vom Frontend erstellt werden und das Frontend muss lediglich eine Komponente zum laden, anzeigen und verarbeiten der Schemata bereitstellen. Dies erleichterte besonders die Implementierung der primitiven "Datentypen", da diese alle in der gleichen Komponente verarbeitet werden können. Komplexere Schemata wie die der Machine Learning- und Zeit Serien Analyse Datentypen müssen aber weiterhin einzeln behandelt werden, da diese mehr Informationen benötigen als die primitiven "Datentypen". Beide Datentypen verfügen über Auswahlmöglichkeiten zum genutzten Modell und dessen Konfiguration, die der Nutzer treffen muss. Diese werden in der UI als Dropdown Menüs dargestellt und in mehrere Schritte unterteilt, um die Mentale Belastung für den Nutzer möglichst gering zu halten.

7.4.5 Bereitstellung von generischen Komponenten für Nutzerfeedback

Feedback sollte konsistent sein. Egal bei welcher Aktion muss im Fehlerfall der Nutzer darüber benachrichtigt werden. MUI liefert hierfür bereits eine vorgefertigte Komponente und kann 4 verschiedene Feedbacks dem Nutzer geben. Durch farbliche Unterscheidungen stellt es Informationen, Erfolgreiche Aktionen, Warnungen und Fehler dar. Dies natürlich in Kombination mit einer dazu passenden Erklärung. Um diese Komponenten innerhalb der Gesammten App einzubauen wurde eine Wrapper Komponente geschrieben. Es erzwingt ein einheitliches Fehlerkonzept und vereinfacht die Nutzung. Da es sich um eine Austauschbare Komponente handelt, lässt sich diese später natürlich austauschen, erweitern und neuen Anforderungen anpassen. Über die Zugabe von optionalen Übersetzungskeys ist die Fehleranzeige multilingual. Es erlaubt damit eine einfache, dem Nutzer aus anderen Anwendungen bereits bekannte Lösung der in 5.2.3 definierten Anforderung an eine Bereitstellung informativen Feedbacks.

7.4.6 Bereitstellung von Mehrsprachigkeit durch i18n

Fachsprache stellt, besonders für Nicht-Experten, eine grundlegende Herausforderung dar. Unabhängig von den Sprachkenntnissen einer Person, ist gerade die fachspezifische Terminologie oft schwierig zu verstehen. Nehmen wir beispielsweise Dieter Maibach, der seit vielen Jahren in einem deutschen Unternehmen arbeitet. Obwohl er Englisch sprechen kann, ist er nicht mit der englischen Fachsprache vertraut. Eine Software nur in einer Sprache anzubieten, die er nicht vollständig versteht, würde ihm erhebliche Schwierigkeiten bereiten. Eine Übersetzung der Anwendung wäre daher vorteilhaft.

Es gibt bereits zahlreiche Lösungen für dieses Problem. Im Bereich von React ist die i18next-Bibliothek weit verbreitet. Sie ermöglicht es, Anwendungen mithilfe von JSON-Dateien, in denen die Übersetzungen gespeichert sind, mehrsprachig zu gestalten. Der Nutzer hat dann zwei Optionen: Entweder wählt er die Sprache selbst aus, oder ein Detektor übernimmt die Sprach-einstellung des Browsers. Eine I18N-Integration führt somit zu einer deutlich verbesserten Benutzererfahrung, da die Anwendung in der bevorzugten Sprache des Nutzers verwendet werden kann.

Es ist daher sinnvoll, dem Nutzer die Möglichkeit zu bieten, die Anwendung in seiner eigenen Sprache zu nutzen. Durch eine I18n-Integration kann die Anwendung ohne erheblichen Aufwand in verschiedene Sprachen übersetzt und somit die Benutzerfreundlichkeit erheblich gesteigert werden. Die Übersetzung der Texte erfolgt direkt im Frontend. Nutzer können ihre bevorzugte Sprache angeben; falls sie dies nicht tun, wird versucht, die Standardsprache des verwendeten Webbrowsers zu verwenden. Die Übersetzungen werden in JSON-Dateien innerhalb des Projekts gespeichert. Diese können dann über eine Komponente in die Anwendung geladen werden.

Etwas komplizierter gestaltete sich die Übersetzung der rsjf-Komponente. Da diese die Beschreibungen der Felder aus dem Schema lädt, müssen diese ebenfalls übersetzt werden. Ein speziell hierfür entwickelter Konverter liest das Schema aus. Anstatt mit normalen Beschreibungen zu arbeiten, mussten die Feldbeschreibungen als i18n-Schlüssel gespeichert werden. Dies gilt auch für die Fehlerbehandlung und -anzeige, die ebenfalls über ihre spezifischen Fehlercodes übersetzt werden müssen.

The screenshot shows a form with two fields: 'name *' and 'sleepTime *'. Below each field is a descriptive text and a red error message indicating it's required. At the bottom are 'ABBRECHEN' and 'SENDEN' buttons.

name *
Der Name des Datensatzes.
Das Feld 'name' ist erforderlich.

sleepTime *
Die Gesamtanzahl der Sekunden, die der Sender ruht.
Das Feld 'sleepTime' ist erforderlich.

ABBRECHEN SENDEN

Abbildung 7.9 – Analyse des CGAN tsgm Modells

The screenshot shows a form with two fields: 'name *' and 'sleepTime *'. Below each field is a descriptive text and a red error message indicating it's required. At the bottom are 'СКАСУВАТИ' and 'НАДІСЛАТИ' buttons.

name *
Назва набору даних.
Поле 'name' є обов'язковим.

sleepTime *
загальна кількість секунд відпочинку відправника.
Поле 'sleepTime' є обов'язковим.

СКАСУВАТИ НАДІСЛАТИ

Abbildung 7.10 – Analyse des CGAN Keras Modells

7.4.7 Websockets und Ihre Auswirkungen für den Nutzer

Da, wie bereits in Spring erwähnt, es viele lang laufende Aufgaben gibt und diese für den Nutzer wenig Transparent sind, wurden in beiden APIs bereits WebSocket-Endpunkte eingefügt um dem Nutzer dieses fehlende Feedback zu geben. Diese jedoch zu implementieren stellte sich als deutlich komplexer heraus als anfangs angenommen. Security beider APIs und teilweise unterschiedlicher Protokolle, Zentralisierung und Verwaltung (Erstellen, auf Nachricht warten und Beenden der Verbindungen) der Websockets, das Serialisieren der Nachrichten und die multivariate Darstellung spielten hier alle mit rein.

Die Webapp muss mehrere Websockets gleichzeitig halten und verwalten können. Nutzer können, sofern die Ressourcen des Servers dies natürlich zulassen, gleichzeitig mehrere Modelle trainieren. Um es dem Nutzer hier so einfach wie möglich zu machen, hängt am Modell ein Fortschritt Element, welches numerisch und visuell den gesamten Fortschritt zeigt. Man kann aber nicht erwarten, dass der Nutzer auf der Seite des Modelles wartet, ob dieses bereits fertig trainiert ist oder noch läuft (siehe Abbildung 7.11). Hier kommt die zentrale Steuerung der Websockets ins Spiel. Über eine Drawer Komponente kann ein Live-Fortschritt des Modells überwacht werden. Dieses ist von jedem Punkt innerhalb der Anwendung erreichbar. Nutzer können somit, ohne in ihrem Arbeitsfluss unterbrochen zu werden schauen, ob sie bereits neue Modelle starten können oder das trainierte Modell in ihre Tracks integriren können. Ebendso gilt dies auch für die Projects welche gerade senden. Sollten mehrere verschiedene Projects gleichzeitig senden und diese darüber hinaus auch mit zeitlich unterschiedlich langen Tracks, so kann der Nutzer dies nur schwer nachvollziehen, ohne eine Kafka Consumer implementation nebenbei laufen zu haben. Die gleiche Drawer Komponente sorgt auch hier für Klarheit, in dem sie den Fortschritt prozentual (auf die Anzahl der Tracks gerechnet) in einem Balkendiagramm anzeigt.

7.4.8 Konfiguration der Komponenten im Bereich maschinelles Lernen

Der Bereich des Machine Learnings in der Anwendung ist bewusst einfach gehalten, um eine intuitive Bedienung zu gewährleisten und Nutzer nicht mit, im Überblick unwichtigen Daten zu überlasten. Auch spiegelt dies das aktuelle API-Design wieder, welches die Konfigurationen der Modelle über einen separaten Endpunkt verwaltet. Alle möglichen Konfigurationen werden beim Erstellen oder Aktualisieren des Modells vorgenommen. Diese Entscheidung basiert auf einem wichtigen Grund: Wie bereits in Abschnitt 7.3.6 erörtert, stellt das Training von Modellen ein komplexes

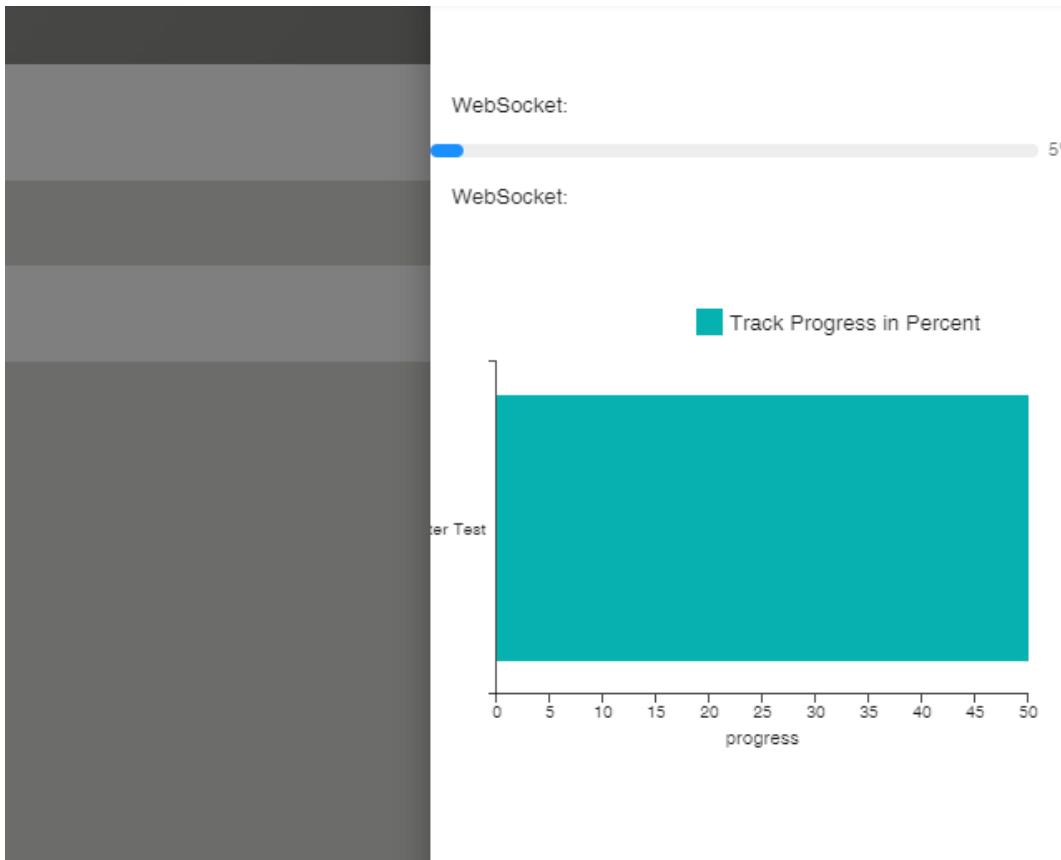


Abbildung 7.11 – Websocket Drawer für das Trainieren von Modellen und als Fortschrittsanzeige für sendende Projects

Unterfangen dar. Eine Vielzahl an Nutzereingaben kann insbesondere für fachfremde Benutzer mental anstrengend sein. Um den Prozess zu vereinfachen, wurde der Dialog in verschiedene, abgeschlossene Teile gegliedert. Dies reduziert die kognitive Belastung für den Benutzer. Im ersten Schritt gibt der Nutzer lediglich grundlegende Daten wie Namen und Beschreibung ein. Anschließend werden im nächsten Schritt die Trainingsdaten ausgewählt. Hierfür hat der Nutzer mehrere Möglichkeiten. Wie in vielen Anwendungen bereits bekannt, gibt es eine Fläche, in die Daten über das Drag'n'Drop Prinzip abgelegt werden, alternativ wird ein Dialog zur manuellen Suchen geöffnet. Dann besteht noch die Möglichkeit zwischen Dateien und Elementen innerhalb der Datein zu wählen. Dieses Konzept wurde gewählt, da gerade größere CSV Dateien eine Vielzahl an Spalten enthalten können und es für den Nutzer einfacher ist, die Daten direkt in der Anwendung zu filtern, als dies in einer externen Anwendung zu tun. Der letzte Schritt beinhaltet die spezifischen Konfigurationen für das Training des Modells. Durch ein strukturiertes und dynamisch angepasstes Eingabefeld kann der Nutzer die verschiedenen Optionen prüfen und, sofern die Daten bereits hochgeladen sind, sich einen Überblick über die Auswirkungen seiner Auswahl verschaffen. Der dargestellte Graph vergleicht eine originale Datei mit ihrer nach der Konfiguration bearbeiteten Version (Siehe Abbildung 7.13) und, sofern die Zeitreihe nicht vollständig ist, zeigt auch den Unterschied der verwendeten Imputationsalgorithmen. Da dieser Prozess rechenintensiv ist, wird er nicht für alle Zeitreihen angeboten und ist nur im Dialog zu finden.

In der Hauptansicht hat der Nutzer Zugriff auf mehrere wichtige Komponenten. Er kann sei-

ne Beschreibungen einsehen, was insbesondere bei einer Vielzahl an Modellen sehr hilfreich ist. Zudem bietet die Anwendung eine Übersicht, in welchen Projekten das Modell bereits eingesetzt wird, und ermöglicht einen direkten Zugriff auf diese. Weiterhin kann der Nutzer sein Modell starten und dabei einige Optionen festlegen, die den Trainingsverlauf wesentlich beeinflussen. Hat der Nutzer ein Modell bereits trainiert, erhält er eine Übersicht mit allen relevanten Trainingsinformationen, einschließlich Datum, Laufzeit und einem Beispielbild (Siehe Abbildung 7.15). Dieses zeigt im direkten Vergleich die Ausgabe des Modells im Vergleich zu einem Element aus dem Trainingsdatensatz. Sollte das Ergebnis nicht den Erwartungen entsprechen, erkennt der Nutzer dies sofort und hat die Möglichkeit, das Modell mit neuen Parametern erneut zu trainieren.

Zugriff auf interne Parameter der Modelle ist nicht möglich. Dies ist eine bewusste Entscheidung, um die Komplexität der Anwendung zu reduzieren und die Benutzerfreundlichkeit zu erhöhen. Es könnte durch die API angeboten werden, bedarf aber eines enormen Validierungsaufwandes, da die Parameter je nach Modell stark variieren, freie Kombinationen meist nicht funktionieren und eine vernünftige API Dokumentation hierfür enorm aufwändig wäre.

7.4.9 Konfiguration der Komponenten im Bereich der Zeitreihenanalyse

Der Bereich der Zeitreihenanalyse ist ähnlich konzipiert wie der des Machine Learnings. Auch hier wurde Wert darauf gelegt, dieselben Komponenten zu verwenden, um den Nutzern eine konsistente und einheitliche Erfahrung zu bieten. Der Dialog zur Erstellung und Aktualisierung einer Konfiguration für ein Modell ist ebenfalls in mehrere Schritte unterteilt. Zunächst werden spezifische Daten für die Konfiguration eingegeben. Im zweiten Schritt erfolgt die Auswahl oder das Hochladen der Trainingsdaten. Im letzten Schritt kann der Nutzer die Verarbeitung der Daten anpassen, genau wie es auch in den Abbildungen 7.12 und 7.13 zu sehen ist.

Ein wesentlicher Unterschied zum maschinellen Lernen besteht darin, dass die Nutzer die Modelle hier nicht trainieren müssen. Die Operationen werden direkt auf die Daten angewendet, und das Ergebnis wird dem Nutzer in verschiedenen Grafiken dargestellt. Bei Methoden wie der Singulärwertzerlegung (SSA) und der empirischen Moduszerlegung (EMD), bei denen das Signal zuerst in mehrere intrinsische Modenfunktionen (IMFs) zerlegt wird, erhält der Nutzer eine Übersicht über die einzelnen IMFs. Zusätzlich gibt es die Möglichkeit, diese zu verschieben, um das Ergebnis zu beeinflussen (siehe Abbildung 7.16). Dies kann nützlich sein, um beispielsweise Phasen zu verschieben, obwohl es kein Bestandteil der späteren Analyse ist.

Methoden wie AMIRA, die grundlegend anders funktionieren, präsentieren lediglich einen Graphen, der das Ergebnis in Arbeitsform zeigt. Dies dient dazu, die Menge der Daten, die regelmäßig zwischen der API ausgetauscht werden müssen, zu reduzieren.

Änderungen an den Modellen können nur über den Aktualisierungsdialog vorgenommen werden, indem die Konfiguration angepasst wird. Dies ist eine bewusste Entscheidung, um die Komplexität der Anwendung zu reduzieren, da die ganzen Informationen nicht überwältigend wirken. Auch räumt es die Ansicht auf und konzentriert den Fokus auf die einzelnen Zeitreihen. Informationen wie Laufzeit und Datum des letzten Trainings, wie es beispielsweise bei den ML Modellen der Fall ist, ist hier nicht relevant, da die Daten in fast Echtzeit bereitgestellt werden können.

7.4.10 Verwalten der hochgeladenen Daten

Zugriff und Verwaltung der Trainingsdaten ist, wie bereits im Kapitel 5.2.1 beschrieben, ein wichtiger Bestandteil der Anwendung. Da in den meisten Fällen Daten innerhalb der Dialoge beim Er-

stellen neuer Machine Learning und Time Series Analysis Konfigurationen hochgeladen werden, ist es wichtig dem Nutzer eine Möglichkeit zu bieten diese Daten auch zu verwalten. Gerade bei komplexeren Dateien, wie sie CSV Dateien mit vielen Spalten produzieren, ist ein visuelles Feedback durchaus praktisch. Sollten Daten, oder Graphen, nicht in das gewünschte Konzept passen, kann der Nutzer diese auch direkt aus der Datei entfernen. Dies ist aber nur möglich, sofern die spezifische Datei nicht an eine Konfiguration gebunden ist. Direkt werden dem Nutzer nur seine Dateien angezeigt (Siehe Abbildung 7.17). Die einzelnen Graphen werden erst geladen, wenn die Datei ausgeklappt wird. Somit wird die Auslastung etwas reduziert und die Anwendung läuft flüssiger.

Add a new solution

Select Parameter ^

Data Processing Configuration

Schema for specifying data processing configuration with customizable options

min_length *

minimal length of the dataset, indicates the reduction of each element, smaller values mean faster training time

Processor Type *

LinearTrendRemove SimpleReduction

select your processing option, could improve training result and time

Imputation_Algorithm *

LOCF MEAN MEDIAN NOCF
 SPLINE INTERPOLATION

select an algorithm to fill in your gaps

configuration option *

default option customize

Choose to use the default configuration or customize the settings for Linear Trend Removal

SimpleReductionOption

max_red *

max reduction factor

Preview ▾

Information — Data — Preview

CANCEL BACK SEND

Abbildung 7.12 – Eingabefeld für die Datenvorverarbeitung sowohl für die Zeit Serien Analyse als auch für das maschinelle Lernen

Add a new solution

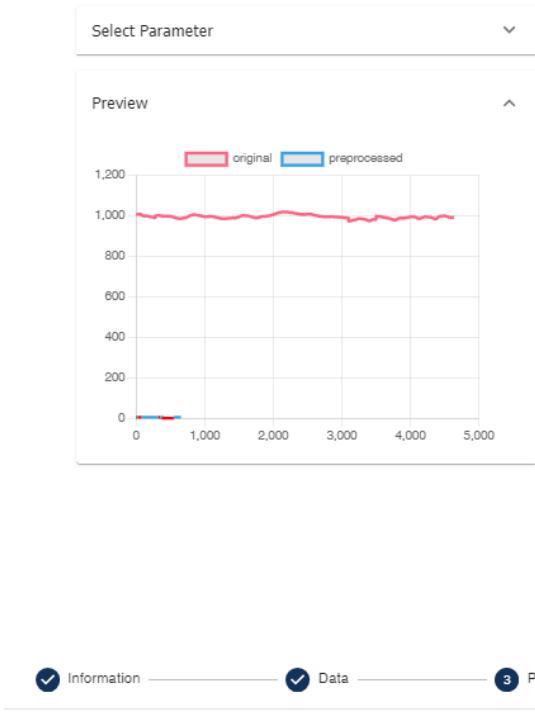


Abbildung 7.13 – Eingeabefeld für die Datenvorverarbeitung sowohl für die Zeit Serien Analyse als auch für das maschinelle Lernen, hier können grafisch die Unterschhiede zwischen den Originaldaten und den verarbeiteten Daten interaktive angesehen werden.

Add a new solution

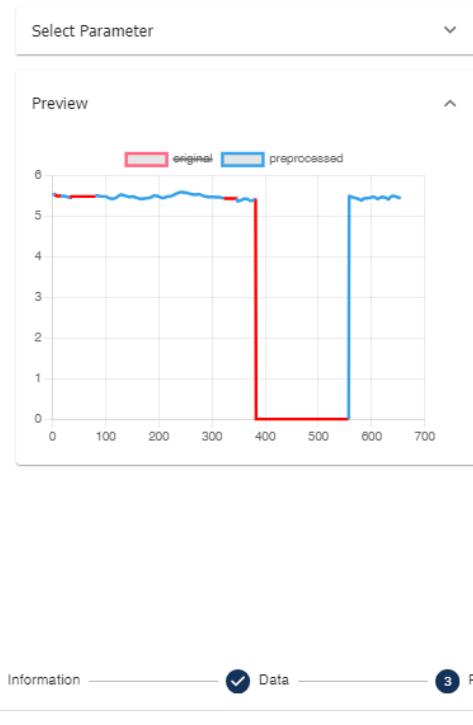


Abbildung 7.14 – In diesem Beispiel werden die originalen Daten ausgeblendet. Die roten Bereiche sind fehlende Werte, welche durch den Imputationsalgorithmen ersetzt wurden. Die große Lücke in der Mitte ist zu groß und wird nicht ersetzt, sondern durch ein ML Modell gefüllt.

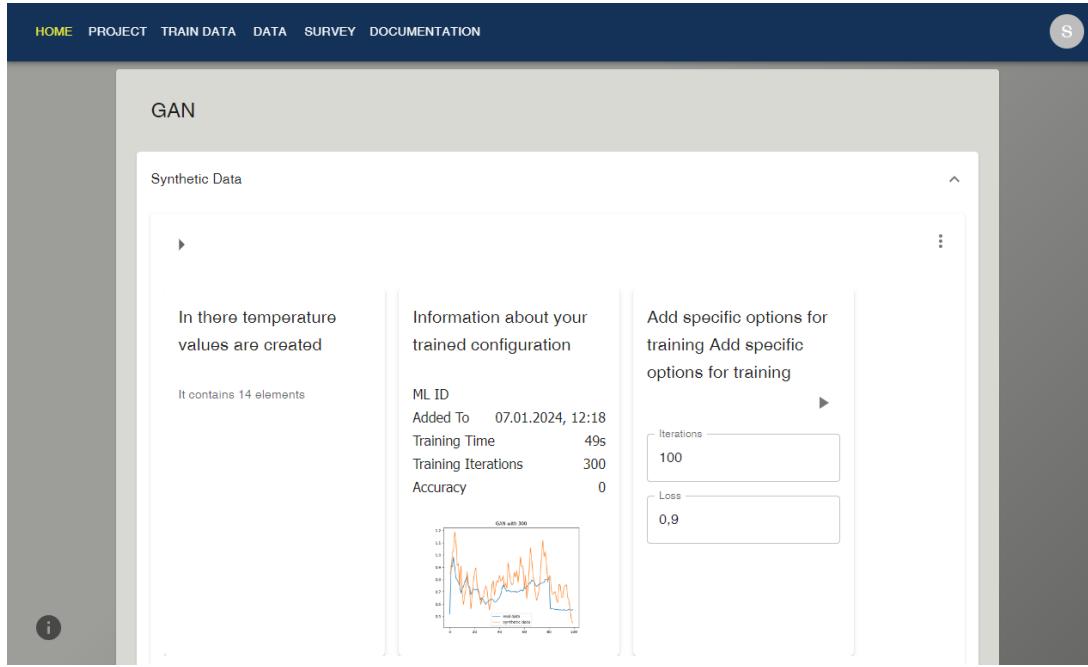


Abbildung 7.15 – Ansicht eines trainierten Modells mit den wichtigsten Informationen und einem Beispielbild



Abbildung 7.16 – Ansicht einer Zeitreihenzerlegung über den EMD Algorithmus

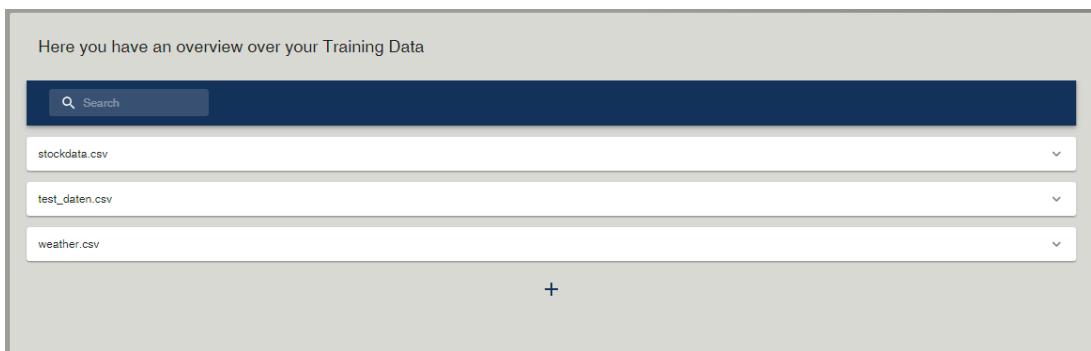


Abbildung 7.17 – Übersicht über die hochgeladenen Dateien und deren Inhalte

8 Aufbau der Testumgebung

8.1 Metriken und Methoden der Testumgebung

In den Abschnitten 7.3.6 und 7.3.7 wurde bereits auf die grundlegende Implementierung der Modelle oder Algorithmen beider Varianten in Grundzügen eingegangen. In diesem Abschnitt wird nun die Implementierung des Testframeworks beschrieben, das die beiden Varianten vergleicht.

Dies ist nicht trivial, da die Kriterien für synthetische Daten im Vergleich zu ihrem realen Pendant auf Ähnlichkeit, jedoch bewusst nicht auf Identität beruhen. Eine einfache euklidische Distanzberechnung ist daher nicht sinnvoll. Stattdessen wird auf verschiedene Methoden zurückgegriffen, die im Folgenden beschrieben werden.

8.1.1 Nicht-Statistische Maße

Wasserstein-Distanz Die Wasserstein-Distanz, auch als Earth Mover's Distance[Rah23] bekannt, ist hilfreich, um Unterschiede zwischen zwei Wahrscheinlichkeitsverteilungen zu quantifizieren. Im Gegensatz zur euklidischen Distanz, bei der die direkten Differenzen zwischen einzelnen Datenpunkten berechnet werden, berücksichtigt die Wasserstein-Distanz die gesamte Struktur der Verteilungen. Somit ist sie als Metrik in Szenarien, in denen nicht nur identische, sondern ähnliche Daten verglichen werden, durchaus aussagekräftig.

$$[Wasserstein - Distanz] W_p(\mu, \nu) = \left(\inf_{\gamma \in \Gamma(\mu, \nu)} \int_{M \times M} d(x, y)^p d\gamma(x, y) \right)^{\frac{1}{p}} \quad (8.1)$$

Für die in Gleichung 8.1 verwendeten Variablen gilt:

$W_p(\mu, \nu)$ ist die Distanz zwischen den beiden Verteilungen μ und ν , $\Gamma(\mu, \nu)$ die Menge aller Kopplungen zwischen μ und ν und $d(x, y)$ ist die euklidische Distanz zwischen x und y .

Speziell in der Zeitreihenanalyse ist dies besonders relevant, da synthetische Zeitreihen im Vergleich zum Original Verschiebungen aufweisen können. Die Wasserstein-Distanz kann solche Verschiebungen handhaben, da sie die gesamte Verteilung der Datenpunkte berücksichtigt, anstatt eines Punkt-zu-Punkt-Vergleichs wie bei der euklidischen Distanz.

Sofern die Trainingsdaten jedoch eine zu heterogene Verteilung aufweisen, ist die Wasserstein-Distanz wenig geeignet, da ihre Aussagekraft zu unterschiedlich sein kann. Dies ist bei variablen Daten grundsätzlich der Fall, da die Daten nicht nur unterschiedliche Werte, sondern auch unterschiedliche Verteilungen aufweisen können. Als Metrik für vorher bekannte Daten ist die Wasserstein-Distanz jedoch durchaus geeignet.

Autokorrelation Eine weitere Methode ist die Analyse der Autokorrelation[Hem16] innerhalb von Zeitreihen. Die Autokorrelationsfunktion (ACF) misst, wie stark eine Zeitreihe mit verzögerten Versionen von sich selbst korreliert ist. Dies ist besonders nützlich, um die interne Struktur

von Zeitreihendaten zu verstehen, insbesondere im Hinblick auf ihre Periodizität und saisonale Muster.

$$ACF(k) = \frac{\sum_{t=1}^{n-k} (X_t - \bar{X})(X_{t+k} - \bar{X})}{\sum_{t=1}^n (X_t - \bar{X})^2} \quad (8.2)$$

In Gleichung 8.2 repräsentiert $ACF(k)$ die Autokorrelation zum Lag k , X_t ist der Wert der Zeitreihe zum Zeitpunkt t , und \bar{X} ist der Mittelwert der Zeitreihe. Diese Funktion hilft dabei, Periodizitäten und wiederkehrende Muster in den Daten zu identifizieren, die für die Prognose und Analyse zukünftiger Datenpunkte nützlich sein können.

Fourier-Transformation Ein weiteres wichtiges Maß ist die Spektralanalyse, die Frequenzkomponenten einer Zeitreihe identifiziert. Hierfür kann der Fast Fourier Transformation (FFT) Algorithmus genutzt werden, um Zeitreihen von einer Zeit- in eine Frequenzdomäne zu überführen, um so dominierender Frequenzen und periodischer Signale in den Daten zu identifizieren [Sze11].

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-2\pi i \omega t} dt \quad (8.3)$$

In Gleichung 8.3 ist $F(\omega)$ die Fourier-Transformierte der Zeitreihe $f(t)$, wobei ω die Frequenz und t die Zeit darstellt. Diese Analyse ist besonders wertvoll, um zyklische Verhaltensweisen in Zeitreihendaten zu erkennen, die durch einfache Abstandsmetriken nicht erfasst werden können.

8.1.2 Statistische Maße

Zeitreihen, wie alle Datensätze, folgen spezifischen Verteilungen. Wenn man annimmt, dass die originalen Daten einer bestimmten Verteilung folgen, sollten die synthetischen Daten diese Verteilung widerspiegeln. Daher kann der Einsatz statistischer Methoden zur Bewertung der Ähnlichkeit zwischen synthetischen und realen Daten als aussagekräftig angenommen werden. Dies gilt insbesondere, wenn man davon ausgeht, dass die synthetischen Daten zwar nicht identisch, aber statistisch ähnlich zu den realen Daten sein sollen.

Kullback-Leibler Divergenz Ein wichtiges statistisches Maß in diesem Kontext ist die Kullback-Leibler Divergenz (KLD) (siehe [Kur17]). Die KLD quantifiziert den Informationsverlust, der auftritt, wenn eine Verteilung genutzt wird, um eine andere Verteilung anzunähern. Ein hoher Wert der KLD signalisiert erhebliche Unterschiede zwischen den Verteilungen, was darauf hinweist, dass die synthetischen Daten die realen nicht adäquat abbilden und wichtige Informationen verloren gehen. Im Gegensatz dazu indiziert ein niedriger KLD-Wert eine starke Ähnlichkeit der Verteilungen, was nahelegt, dass die synthetischen Daten eine präzise Repräsentation der realen Daten sind und der Informationsverlust minimiert ist. Mathematisch ist es definiert als:

$$D_{KL}(P||Q) = \sum_i P(i) \log \left(\frac{P(i)}{Q(i)} \right) \quad (8.4)$$

Wobei $D_{KL}(P||Q)$ die Kullback-Leibler Divergenz zwischen den Verteilungen P und Q ist. $P(i)$ und $Q(i)$ sind die Wahrscheinlichkeiten für das Ereignis i in den Verteilungen P und Q.

Maximum Mittelwert Diskrepanz Maximum Mean Discrepancy (MMD) ist eine nicht-parametrische Methode, die Unterschiede zwischen zwei Verteilungen auf Basis von Stichproben aus jenen Verteilungen evaluiert. Definiert ist sie über:

$$\text{MMD}^2(P, Q) = \mathbb{E}_{x, x' \sim P}[k(x, x')] + \mathbb{E}_{y, y' \sim Q}[k(y, y')] - 2\mathbb{E}_{x \sim P, y \sim Q}[k(x, y)] \quad (8.5)$$

Wobei, wie in Formel 8.4 bereits definiert, P und Q die beiden Verteilungen sind, x, x' und y, y' sind die jeweiligen Stichproben und \mathbb{E} der Erwartungswert.

Sie verwendet eine Kernel-Funktion k , um die Distanz zwischen den Mittelwerten beider Verteilungen im Feature-Raum zu messen. Da sie keinerlei Annahmen über die Form der Verteilungen macht, ist MMD ein flexibles Werkzeug zur Analyse, insbesondere wenn die Form der Verteilungen unbekannt oder komplex ist, wovon bei variablen Datensätzen grundsätzlich auszugehen ist. Es eignet sich daher sehr gut, um festzustellen, ob zwei Stichproben aus der gleichen Verteilung stammen, was in vielen Anwendungen der Zeitreihenanalyse von Bedeutung ist.

Mann-Whitney-U-Test Der Mann-Whitney-U-Test, auch bekannt als Wilcoxon-Rangsummentest, ist ein nicht-parametrischer Test, der verwendet wird, um zu beurteilen, ob zwei unabhängige Stichproben aus identischen Populationen stammen. Der Test ist besonders geeignet für Daten mit unbekannter oder nicht-normaler Verteilung. Er vergleicht die Rangwerte der Daten in den beiden Gruppen.

Sei U die Teststatistik, die wie folgt berechnet wird:

$$U = R_1 - \frac{n_1(n_1 + 1)}{2} \quad (8.6)$$

wobei R_1 die Summe der Ränge in der ersten Gruppe, n_1 die Anzahl der Beobachtungen in der ersten Gruppe ist. Der Wert von U wird dann verwendet, um die Signifikanz des Unterschieds zwischen den beiden Gruppen zu beurteilen.

T-Test Der T-Test ist ein statistischer Hypothesentest, der verwendet wird, um zu prüfen, ob sich die Mittelwerte zweier Gruppen signifikant unterscheiden. Er setzt voraus, dass die Daten normalverteilt sind. Der unabhängige T-Test für zwei Gruppen ist definiert als:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \quad (8.7)$$

wobei \bar{X}_1 und \bar{X}_2 die Mittelwerte der beiden Gruppen, s_1^2 und s_2^2 die Varianzen der Gruppen und n_1 und n_2 die Anzahl der Beobachtungen in den Gruppen sind. Der berechnete t-Wert wird dann mit einem kritischen Wert aus der T-Verteilung verglichen, um die Hypothese zu testen.

8.1.3 Evaluation durch maschinelles Lernen

Machine Learning-Algorithmen sind besonders effizient darin, Muster in Daten zu erkennen. Insbesondere die Diskriminatoren in generativen Modellen sind darauf ausgelegt zu beurteilen, ob die vom Generator erzeugten Daten realistisch sind oder nicht. Daher bietet sich die Genauigkeit des Diskriminators als Maß für die Ähnlichkeit zwischen synthetischen und realen Daten an. Wenn

der Diskriminator die synthetischen Daten als real einstuft, deutet dies auf eine hohe Ähnlichkeit oder grundsätzlich gleiche Muster mit den tatsächlichen Daten hin.

Eine in der Wissenschaft verbreitete und oft genutzte Methode baut auf dem Konzept auf und erweitert dieses um dieses Konzept im Kontext der Privatsphäre und ethischer Aspekten synthetischer Daten, insbesondere personenbezogener Daten, anzuwenden. Ein Ansatz zur Bewertung des Datenschutzes ist die Überprüfung der Outputs eines trainierten Modells auf Ähnlichkeiten, die Rückschlüsse auf die ursprünglich verwendeten Trainingsdaten zulassen könnten. In dem Paper 'User-Level Membership Inference Attack against Metric Embedding Learning' von Guoyao Li, Shahbaz Rezaei und Xin Liu [LRL22] wird eine solche Methodik vorgestellt. Wenn ein Angreifermodell auf der Basis von synthetischen Daten trainiert wird und eine hohe Wahrscheinlichkeit besteht, dass dieses Modell die realen Trainingsdaten identifiziert, ist die Privatsphäre der Daten möglicherweise gefährdet. Dieses Konzept wurde beispielsweise von Nikitin et al. in [NIK23] untersucht.

Ein weiterer wichtiger Aspekt ist die Erkennung von Ausreißern in den Daten. Wenn die realen Daten signifikante Anteile von Ausreißern aufweisen, sollten diese auch in den synthetischen Daten vorhanden sein. Zur Bewertung dieser Eigenschaft kann ein Random-Forest-Algorithmus eingesetzt werden, der auf den realen Daten trainiert wird und sowohl auf die realen als auch auf die synthetischen Daten angewendet wird. Eine ähnliche Verteilung von Ausreißern in beiden Datensätzen kann darauf hindeuten, dass die synthetischen Daten die extrema der realen Daten gut nachbilden.

8.2 Bewertung der Methoden

All diese Methoden wurden in einem Testframework implementiert. Dieses Framework besitzt die reduzierten und leicht modifizierten Algorithmen, die auch in der API verwendet werden. Es trainiert die Modelle sowie speichert deren Laufzeit und die von ihm generierten synthetischen Daten.

```
1 def run_test_for_synthetic_data(data, syn_date_file_name, filename="statistic"):
2     syn_data = load_processed_data(syn_date_file_name)
3     result = {}
4     for i in syn_data.keys():
5         result[i] = {}
6         if len(syn_data[i]) == 99:
7             x, y = build_summed_version(data), np.array(syn_data[i][0])
8         else:
9             x, y = build_summed_version(data), np.array(syn_data[i][0])
10        result[i]["statistic"] = run_static_tests(x, y)
11        result[i]["outlier"] = outlier_detection(data, y)
12        result[i]["attacker"] = run_syn_analyzer_(data, syn_data[i])
13        result[i]["kl"] = run_kullback_leibler_divergence(x, y)
14    write_to_file(f"data/{filename}.json", result)
```

Listing 8.1 – Methode um Statistiken aus den generierten Daten zu erstellen

Abgespeichert werden die Daten in einer JSON Datei, welche dann von den verschiedenen Methoden gelesen werden kann. Eine Methode nimmt dann die gesammelten Daten und wertet diese in verschiedenen Schritten aus und erstellt die notwendigen Statistiken (Siehe Listing 8.1). Da sie

am Ende des Prozesses wieder gespeichert werden, können sie zu einem späteren Zeitpunkt analysiert und geplottet werden.

Im letzten Schritt werden die Daten dann an einen Clustering Algorithmus übergeben, welcher die Daten in verschiedene Gruppen unterteilt. Über Pyplot¹ können die Daten visualisiert werden und sind somit leichter zu bewerten. Für die Clustering Methoden wurde Plotly² verwendet, da diese eine interaktive Visualisierung ermöglichen und die generierten HTML Seiten problemlos in die Hugo Dokumentation eingebunden werden können.

Das Testframework ist so konzipiert, dass, abgesehen der Visualisierung, alle Schritte automatisch über die main Methode ausgeführt werden können und lediglich der Pfad zur Datei mit den Trainingsdaten ausgetauscht werden muss. Auch folgen die Modelle dem gleichen Vererbungsdesign, sodass sie alle über die gleiche Schnittstelle angesprochen werden können.

```

1 modules = [
2     (RNN, "RNN"),
3     ...
4 ]
5
6 tsa_modules = [
7     (SingularSpectrumAnalysis, "SSA"),
8     ...
9 ]
10
11 def run_training_for_all_modules(data:[], name:str, use_ml=True):
12     data = np.array(data)
13     iterations = [30, 50, 70, 100, 140, 210, 300, 500, 700, 1400, 2100]
14     name = f"{name}_{'ml' if use_ml else 'tsa'}"
15     filename = f"data/{name}.json"
16     all_data = load_data(filename)
17
18     if use_ml:
19         for module, class_name in modules:
20             for iteration in iterations:
21                 run_info = RunInformation(iterations=iteration, input_length=100)
22                 value = run_class(module, class_name, data, run_info, {})
23                 all_data[f"{class_name}_{iteration}"] = value
24                 write_to_file(filename, all_data)
25     else:
26         for module, class_name in tsa_modules:
27             run_info = RunInformation(iterations=0, input_length=100)
28             value = run_class(module, class_name, data, run_info, {})
29             all_data[f"{class_name}"] = value
30             write_to_file(filename, all_data)

```

Listing 8.2 – Training von Modellen im Testframework

Ziel des Testframeworks ist es aber nicht nur die synthetischen Daten der Algorithmen und Modelle mit den originalen Daten zu vergleichen, sonder auch zu schauen inwieweit die Anzahl der Iterationen, gerade bei den generativen Modellen, die Ergebnisse beeinflusst und dies in Relation zum erhöhten Rechenaufwand zu setzen.

¹<https://matplotlib.org/stable/tutorials/pyplot.html>

²<https://plotly.com/>

8.2.1 Clustering

Clustering ist ein Verfahren im Bereich des überwachten Lernens mit dem Ziel Daten in Gruppen zu unterteilen. Hierbei beinhalten diese Gruppen ähnliche Daten, während die Daten zwischen den Gruppen möglichst unterschiedlich sind.

Für die Analyse von synthetischen Zeitreihen ist dies interessant, da eine Liste an Eigenschaften der synthetischen und realen Daten erstellt werden kann und diese dann in Gruppen unterteilt werden können. Sind die Gruppen sehr weit auseinander, dann sind die synthetischen Daten nicht ähnlich zu den realen Daten. Sind sie sehr nah, dann sind die synthetischen Daten wiederum sehr ähnlich zu den realen Daten. Ideal wäre also eine Gruppe, welche einen geringen Abstand zum Cluster der originalen Daten besitzt.

Um einen Clustering-Algorithmus zu verwenden, müssen zunächst die Eigenschaften der Daten extrahiert werden. Hierzu wurden verschiedene Methoden verwendet:

Clustering ist ein Verfahren im Bereich des unüberwachten Lernens, das darauf abzielt, Daten in Gruppen zu unterteilen. Diese Gruppen bestehen aus ähnlichen Daten, während die Daten zwischen den Gruppen möglichst unterschiedlich sind.

Die Anwendung von Clustering-Verfahren auf die Analyse von synthetischen Zeitreihen ist somit besonders interessant um zu sehen wie die jeweiligen Modelle ihre Daten generieren. Hierfür wird eine Liste von Eigenschaften sowohl der synthetischen als auch der realen Daten erstellt. Diese Eigenschaften werden anschließend genutzt, um die Daten in Gruppen zu unterteilen. Wenn die Gruppen weit auseinanderliegen, deutet dies darauf hin, dass die synthetischen Daten nicht den realen Daten ähneln. Sind die Gruppen hingegen nahe beieinander, impliziert dies eine hohe Ähnlichkeit zwischen den synthetischen und den realen Daten. Das ideale Ergebnis wäre eine Gruppe synthetischer Daten, die einen geringen Abstand zum Cluster der originalen Daten aufweist.

Bevor ein Clustering-Algorithmus angewendet werden kann, ist es jedoch notwendig, die Eigenschaften der Daten zu extrahieren. Für diesen Zweck wurden verschiedene Methoden eingesetzt:

1. Statistische Eigenschaften:

- **Mittelwert und Standardabweichung:** Diese Metriken geben einen Überblick über die zentrale Tendenz und die Streuung der Zeitreihe.
- **Maximum und Minimum Werte:** Diese Informationen geben einen Überblick in welchem Wertebereich sich die Zeitreihe befindet.
- **Skewness und Kurtosis:** Skewness ist eine Metrik, die die Asymmetrie der Verteilung der Zeitreihe misst. Kurtosis misst Ausreißer der Verteilung der Zeitreihe. Ein hoher Kurtosis-Wert bedeutet viele Ausreißer, während ein niedriger Wert bedeutet, dass die Verteilung weniger Ausreißer aufweist.

2. Wavelet Transform Eigenschaften:

- Daubechies Wavelet ist eine Methode, die eine Zeitreihe in verschiedene Frequenzkomponenten zerlegt. Diese Frequenzkomponenten können dann als Eigenschaften verwendet werden [Sze11].

3. Time Series Decomposition (Saisonale Aufspaltung):

- Aufspalten einer Zeitreihe in Trend, Saison und Residuen. Dieses sind die klassischen Komponenten wie sie in Sektion 3.1 beschrieben wurden.

4. Spectral Analysis (Welch Methode):

- Die Welch Methode ist ein Algorithmus der Spektralanalyse und wird zur Berechnung der Leistungsspektraldichte von Zeitreihen eingesetzt. Diese Leistungsspektraldichte gibt an wie eine Reihe über mehrere Frequenzen verteilt ist.

Hinzu kamen noch die Werte, welche in Sektionen 8.1.1, 8.1.2 und 8.1.3 gesammelt wurden.

Diese gesammelten Eigenschaften können nun mit einem Clustering-Algorithmus, wie beispielsweise dem k-Means-Algorithmus, verarbeitet werden. Der k-Means-Algorithmus ist besonders beliebt, da er effizient ist und sich einfach implementieren lässt. Er teilt eine Liste von Datenpunkten in k Gruppen auf. Allerdings liefert k-Means oft eine Vielzahl von Parametern zurück, die schwer zu interpretieren sein können. Aus diesem Grund ist es sinnvoll, Techniken zur Reduktion der Dimensionalität der Daten einzusetzen, um die Clustering-Ergebnisse vernünftig interpretieren zu können.

Ein verbreiteter Ansatz hierfür ist die Principal Component Analysis (PCA), die sich besonders effektiv bei der Extraktion und Identifikation wichtiger Merkmale zeigt. Die PCA kann, wie in Agostas Arbeit [Ago20] hervorgehoben, periodische Komponenten isolieren und Rauschen reduzieren. Dieser Prozess der Identifizierung und Isolierung wesentlicher Komponenten in den Daten ist entscheidend für die Dimensionsreduktion und ermöglicht eine tiefere Analyse aufbauend auf dem Clustering-Ergebnis.

Principal Component Analysis (PCA) Die Fähigkeit der PCA, die Dimensionalität eines Datensatzes zu reduzieren und gleichzeitig dessen Varianz zu erhalten, macht sie zu einem wertvollen Werkzeug in der Analyse von Zeitreihendaten nach der Merkmalsextraktion. Sie identifiziert und isoliert effektiv die Hauptkomponenten der Variation innerhalb der extrahierten Merkmale [Ago20].

t-Distributed Stochastic Neighbor Embedding (t-SNE) t-SNE eignet sich hervorragend zur Visualisierung und Analyse der nichtlinearen Beziehungen und Cluster in hochdimensionalen Daten, insbesondere nach der Extraktion wichtiger Merkmale. Seine Anwendung in komplexen Zeitreihendatensätzen ermöglicht eine nuanciertere Erforschung der Datenstruktur [WC19].

Uniform Manifold Approximation and Projection (UMAP) Die Flexibilität von UMAP im Umgang mit linearen und nichtlinearen Daten macht es für die Zeitreihenanalyse geeignet, in der die extrahierten Merkmale von einfachen bis zu komplexen Mustern reichen können. Seine Fähigkeit, lokale und globale Strukturen zu erhalten, ist besonders vorteilhaft in Datensätzen mit variierenden Zeitskalen [PBC21].

Isometric Mapping (Isomap) In Zeitreihendaten, die auf einem nichtlinearen Manifold liegen, bietet Isomaps Fokus auf die Erhaltung geodätscher Distanzen eine einzigartige Perspektive. Nach der Merkmalsextraktion kann Isomap Einblicke in die intrinsische geometrische Struktur der Daten geben, was es ideal für komplexe Datensätze macht [DDM15].

8.2.2 Getestete Modelle

Viele der Grundlegenden Modelle unterscheiden sich nur durch leicht abgewandelte Architekturen oder Loss-Funktionen. Dann gibt es noch Bibliotheken, welche komplexere Modell-Architekturen bereitstellen, wie beispielsweise die tsgm Bibliothek [NIK23]. Somit wurde für die Tests eine Vielzahl an Modellen verwendet, welche in den folgenden Abschnitten beschrieben werden.

Generative Modelle

Native Modelle Nativ bezieht sich in diesem Kontext auf Modelle, die direkt auf der Keras Bibliothek aufgebaut sind und keine weiteren Abhängigkeiten haben. Sie nutzen die Standard-Implementierung von Keras und Tensorflow, ihre Parameter sind definiert und stehen in Abhängigkeit zur Größe der Daten. Sie besitzen, im Vergleich zu anderen Modellen, eine geringe Komplexität mit weniger Parametern und Schichten und sind daher schneller zu trainieren. Gerade die Schnelligkeit der Modelle macht diese hoch interessant, da sie mit wenig Rechenaufwand synthetische Daten in kurzer Zeit generieren können und damit ideal für den Einsatz in der API sind.

Modelle dieser Art sind:

1. GAN
2. CGAN
3. WGAN (Wasserstein GAN) nur leider erzielt es extrem schlechte Ergebnisse
4. TGAN (Time Series GAN), auch dieses erzielte nur schlechte Ergebnisse und wurde daher nicht weiter verfolgt

tsgm Modelle Die tsgm Bibliothek [NIK23] baut auf Keras auf und bietet einige Modelle an, welche speziell für die Generierung synthetischer Zeitreihen entwickelt wurden:

1. TGAN
2. TCGAN
3. CGAN
4. RCGAN

Leider konnten von diesen aber nur die ersten drei genutzt werden, da RCGAN Abhängigkeiten zur Tensorflow-Privacy besitzt und diese nicht mit der genutzten Python Version (3.11) kompatibel war.

Rekursive Modelle

Um die Rekursiven Modelle zu testen wurde wieder auf die Keras Bibliothek zurückgegriffen und mit Ihrer Hilfe die folgenden Modelle implementiert:

1. RNN
2. LSTM
3. CNN-LSTM
4. CONV-LSTM

8.2.3 Zeitreihenanalyse Algorithmen und Modelle

Für die Zeitreihenanalyse wurde, wie bereits im Detail in Sektionen und 3.1.2 erklärt, die EMD und SSA Algorithmen verwendet. Weitere gängige Methoden sind **AMIRAI** (**AMIRAI!**) und der Cubic Spline Algorihtmus, deren Bibliotheken aber lediglich in die grundlegende Code Struktur eingebunden werden mussten. Somit wurden folgende Algorithmen verwendet:

1. EMD
2. SSA
3. Amira
4. Cubic Spline

8.3 Deployment

Wie aus den vorhergenden Kapiteln hervorgeht, ist die Anwendung recht groß und besteht aus vielen Komponenten. Dies macht leider das Deployment der Gesamtanwendung kompliziert. Daher wurde der Deployment Prozess in mehrere Schritte aufgeteilt, welche in diesem Kapitel beschrieben werden.

8.3.1 Deployment der Anwendung

Die Anwendung bzw. Anwendungen, aus welchen das Projekt zusammengebaut wurde, sind alle um das Docker Ökosystem aufgebaut (siehe 6.3). Einzelne Komponenten sind mit Docker Files ausgestattet um die entsprechenden Images zu bauen. Zum starten aller Container wurden zwei unterschiedliche Docker Compose Dateien erstellt, eines für die Entwicklung und eines für die Produktion. Das zweite wurde mit den auch in 6 vorgestellten Überwachungstools ausgestattet.

Es wurde auch ein Template für die Docker Swarm Orchestrierung erstellt, welches das ganze Projekt in einer stabilen Umgebung laufen lässt und dem Betreiber die Möglichkeit gibt, die Anwendung auf mehreren Servern zu verteilen. Hier müssten lediglich die gewünschten Bedienung noch definiert werden.

8.3.2 Automatisierung der Deployment Pipeline

Da der ganze Prozess des manuellen Deployment sehr aufwendig ist, wurde eine automatisierte Deployment Pipeline erstellt. Diese läuft über die GitLab CI/CD Funktionen und wird bei jedem Push auf den Master Branch ausgeführt. Hierfür muss lediglich ein passender Runner auf dem Server installiert sein, welcher die Jobs ausführt. Die entsprechenden Variable müssen in der GitLab CI/CD Konfiguration hinterlegt werden und schon wird der Prozess bei jedem Push ausgeführt.

Der Prozess, abgebildet in Abbildung 8.1, besteht aus folgenden Schritten:

1. Baue die Django, Spring, Hugo und React Images
2. Lade die Images auf den Server
3. Ersetzte die Container Image Tags durch die neuen Tags der Images, hierfür wird das Template genutzt
4. Deploye den Stack auf dem Server, Trafik wird als Manager gesetzt

Der Prozess ist in der Datei `.gitlab-ci.yml` hinterlegt und kann dort angepasst werden. Wie in Abbildung 8.3 zu sehen ist, werden die einzelnen Schritte als Jobs definiert und können auch einzeln ausgeführt werden. Aktuell ist der Deployment Prozess nur für den Master Branch konfiguriert. Hier kann die eigene Vorgehensweise später gewählt werden. Da aber im aktuellen Prozess nur vom Master abgezweigt wird und dieser immer im Produktivmodus zu laufen hat, gab es keine Notwendigkeit auf eine Git Flow basierte Branching Strategie zu wechseln. Dies wäre aber auch möglich, da die GitLab CI/CD Konfiguration auch auf Branches angewendet werden kann.

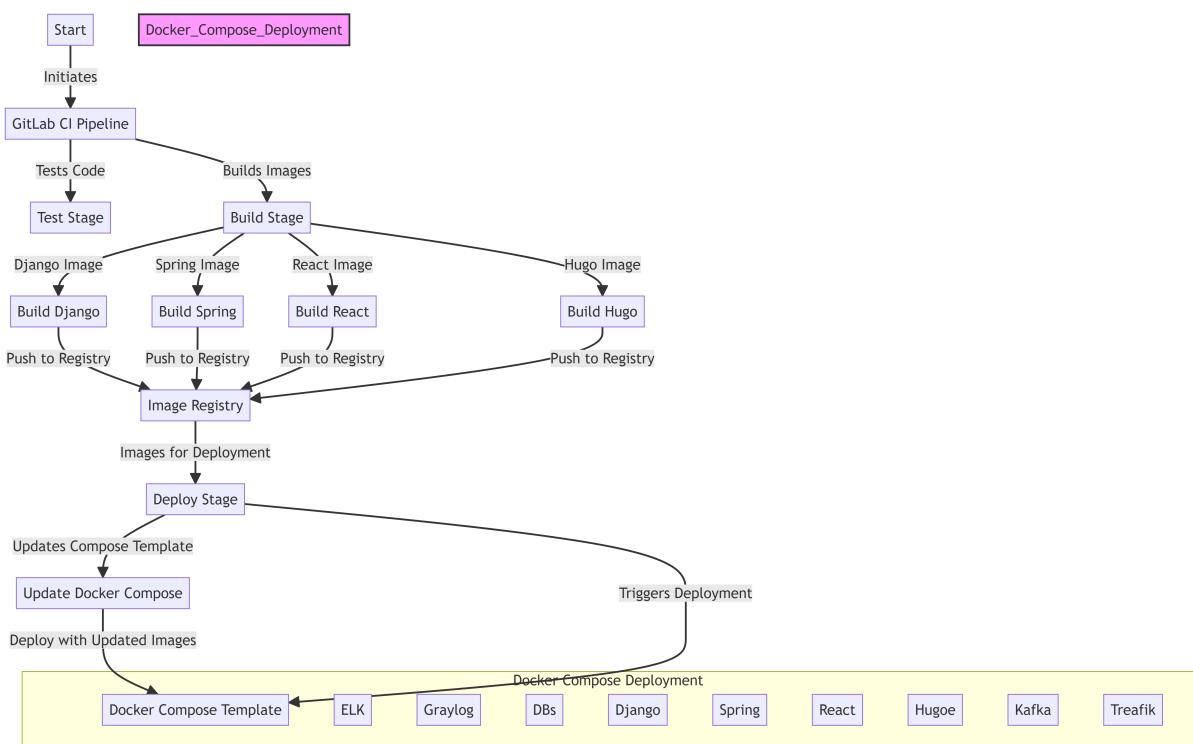


Abbildung 8.1 – Visuelle Darstellung des Deployment Prozesses in GitLab CI/CD und lokaler Umgebung

8 Aufbau der Testumgebung

```

version: '3.8'

services:
  prometheus:
    build:
      context: prometheus
      container_name: sus-prometheus
    ...
  elasticsearch:
    image: docker.io/elastic/elasticsearch:7.16.3
    ...
  mongo:
    image: mongo:6.0.8
    ...
  graylog:
    image: graylog/graylog:5.1.3
    ...
  traefik:
    image: traefik:v2.0.0-beta5
    container_name: traefik-proxy
    ...
  db:
    image: postgres:14
    container_name: sus-db-dev
    ...
  django:
    build:
      context: ../../data-training-app/djangoProject
      container_name: django-dev
    ...
  springapp:
    build:
      context: ../../spring-backend
      container_name: spring-backend-dev
    ...
  hugo-site:
    build:
      context: ../../documentation
      container_name: hugo-site-dev
    ...
  frontend:
    build:
      context: ../../frontend
      container_name: react-dev
    ...
  redis:
    image: "redis:alpine"
    container_name: redis-dev
    ...
  zookeeper:
    image: wurstmeister/zookeeper
    container_name: sus-zookeeper
    ports:
      - "2181:2181"
    restart: unless-stopped
    networks:
      sus-network:
        aliases:
          - zooAlias
  kafka:
    image: wurstmeister/kafka
    container_name: sus-kafka
    ...
  networks:
    sus-network:
      name: sus-network
      driver: bridge
      external: true

```

Abbildung 8.2 – Docker Compose File für den Produktionsprozess

```

image: ubuntu
services: [ ]
stages:
  - test
  - build
  - deploy
variables:
  DOCKER_HOST: "unix:///var/run/docker.sock"
  DOCKER_CERTDIR: ~
  DOCKER_TLS_CERTFILE: ./certs/tls/cert.pem
  STACK_NAME: SUS_APP
  IMAGE_TAG: $CI_COMMIT_SHA
  DOCKER_REGISTRY: $CI_REGISTRY_IMAGE
  DOCKER_LOGIN_URL: $CI_LOGIN_URL
  SSH_PRIVATE_KEY: $SSH_PRIVATE_KEY
build-django:
  stage: build
  image: python:3.11
  before_script:
    - cd data-training-app/djangoProject
    - apt-get update -qq && apt-get install -y curl gnupg
    - curl -fsSL https://get.docker.com -o get-docker.sh
    - sh get-docker.sh
    - echo
    - docker login $DOCKER_LOGIN_URL -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
  script:
    - docker build -t $DOCKER_REGISTRY/sus_apl_django:$IMAGE_TAG .
    - docker push $DOCKER_REGISTRY/sus_apl_django:$IMAGE_TAG
build-spring:
  stage: build
  image: maven:3.9-eclipse-temurin-17-focal
  cache:
    key: $CI_COMMIT_REF_SLUG
  before_script:
    - cd ..../spring-backend
    - apt-get update -qq & apt-get install -y curl gnupg2
    - curl -fsSL https://get.docker.com -o get-docker.sh
    - sh get-docker.sh
    - docker login $DOCKER_LOGIN_URL -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
  script:
    - docker build -t $DOCKER_REGISTRY/sus_apl_spring:$IMAGE_TAG .
    - docker push $DOCKER_REGISTRY/sus_apl_spring:$IMAGE_TAG
build-react:
  stage: build
  image: node:16
  before_script:
    - cd frontend
    - apt-get update -qq & apt-get install -y curl gnupg2
    - curl -fsSL https://get.docker.com -o get-docker.sh
    - sh get-docker.sh
    - docker login $DOCKER_LOGIN_URL -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
  script:
    - docker build -t $DOCKER_REGISTRY/sus_frontend:$IMAGE_TAG .
    - docker push $DOCKER_REGISTRY/sus_frontend:$IMAGE_TAG
build-hugo:
  stage: build
  image: ubuntu:latest
  before_script:
    - cd documentation
    - apt-get update -qq & apt-get install -y curl gnupg2
    - curl -fsSL https://get.docker.com -o get-docker.sh
    - sh get-docker.sh
    - docker login $DOCKER_LOGIN_URL -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
  script:
    - docker build -t $DOCKER_REGISTRY/sus_hugo:$IMAGE_TAG .
    - docker push $DOCKER_REGISTRY/sus_hugo:$IMAGE_TAG
deployment:
  stage: deploy
  image: ubuntu:latest
  before_script:
    - cd deployment
    - apt-get update -qq & apt-get install -y curl gnupg2
    - curl -fsSL https://get.docker.com -o get-docker.sh
    - sh get-docker.sh
    - chmod 644 traefik.yml
  script:
    - docker login $DOCKER_LOGIN_URL -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
    - sed -e "$!Django$DOCKER_REGISTRY/sus_apl_django:$IMAGE_TAG" \
      -e "$!Spring$DOCKER_REGISTRY/sus_apl_spring:$IMAGE_TAG" \
      -e "$!React$DOCKER_REGISTRY/sus_frontend:$IMAGE_TAG" \
      -e "$!Hugo$DOCKER_REGISTRY/sus_hugo:$IMAGE_TAG" \
      docker-compose.yaml.template > docker-compose.yml
    -
if [ $(docker network ls | grep -e sus-network) ]; then docker network create --driver=overlay --attachable sus-network; fi
only:
  - master

```

Abbildung 8.3 – Gitlab CI/CD Pipeline

9 Ereignisse und Auswertung der Modelle und Algorithmen

9.1 Trainingsdaten

Die Bewertung der einzelnen Metriken ist nicht trivial, da die synthetischen Daten nicht nur ähnlich, sondern nicht identisch zu den realen Daten sein sollen. Während bei den meisten Metriken hauptsächlich auf die Übereinstimmung der Daten geschaut wird, oder wie stark deren Verteilungen übereinstimmen, so ist dieser Wert wenig aussagekräftig. Abweichungen, Werte die nicht in die normale Verteilung passen und andere Abweichungen sind gewollt und dürfen nicht als schlecht bewertet werden. Visuell lässt sich leicht ein Unterschied zwischen den Daten erkennen, auch kann eingeschätzt werden in wie weit die Daten dem gleichen Muster folgen (siehe Abbildung 9.1), nur ist dies kein klassifizierbares Maß.

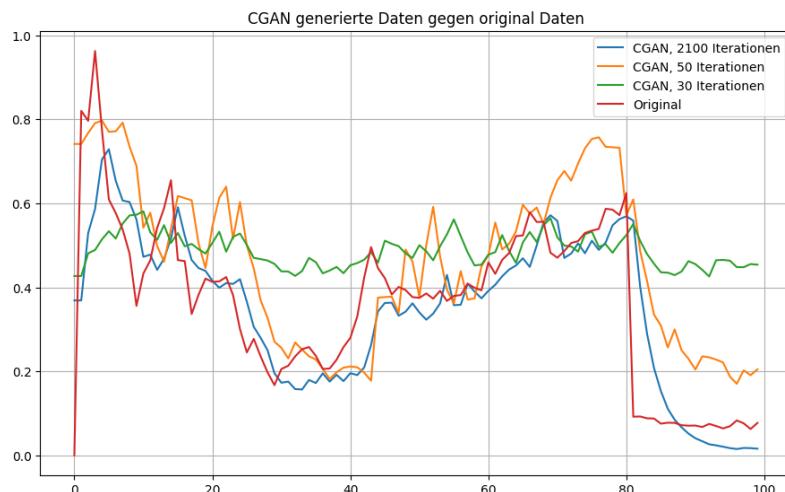


Abbildung 9.1 – Vergleich von realen und synthetischen Daten

Um die Modelle und Algorithmen später vernünftig zu testen und vergleichen zu können, wurden sie mit 5 verschiedenen Trainingsdaten trainiert. Diese besitzen jeweils unterschiedliche Komplexität, welche hier durch ihre Autokorrelationswerte angeben und in Tabelle 9.1 zu finden ist, und sollen so die Stärken und Schwächen der einzelnen Algorithmen aufzeigen. Die wohl am einfachsten zu erlernenden Daten sind Sinus Kurven. Diese sind leicht zu erlernen und können mit wenigen Parametern beschrieben werden. In Abbildung 9.2 sind die Daten zu sehen.

Die genutzten Corona Daten zeigen den Anstieg zum Beginn der Pandemie, und sind daher wenig komplex. Diese stammen von [Geh20] und sind in Abbildung ?? zu sehen.

Tabelle 9.1 – Autokorrelation der Datensätze

Dataset Name	Auto-correlation Coefficient
weather_normalized	0.7531247393640502
stockdata_normalized	0.8705377400789976
electricity_normalized	0.9213733429685765
corona_normalized	0.9705517108865852
sinus	0.992051245850721

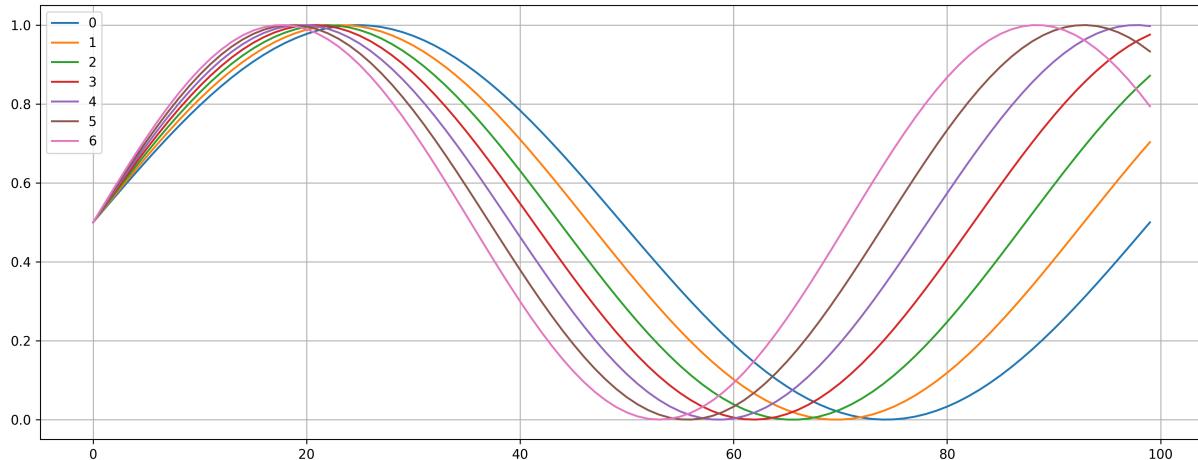


Abbildung 9.2 – Sinus Kurven als Testdaten

Etwas komplexer sind die Daten über den Stromverbrauch von unterschiedlichen Verbrauchern über mehrere Tage hinweg, der genutzte Datensatz misst die Werte für verschiedenen Haushalte über etwas mehr als 30 Tage im stündlichen Takt. Sie stammen von [giv20] und hatten leider wenige Information über die Art der Daten. Hier treten periodische Muster auf, welche aber nicht so einfach zu erlernen sind wie die Sinus Kurven und damit eine größere Herausforderung darstellen. Auch sind hier Außreißer und andere Abweichungen von der Norm zu erwarten. Die Daten sind in Abbildung 9.4 zu sehen.

Sehr interessant sind auch Aktiendaten. Diese sind sehr komplex, nicht zwangsläufig periodisch und besitzen viele Abweichungen von der Norm. Sie sind deutlich komplexer zu erlernen, da sie keinen direktem Muster folgen. Die Daten sind in Abbildung 9.5 zu sehen und wurden über Yahoo Finanzen¹ erworben. Da es sich um die selber Aktie und verschieden Werte handelt, wie beispielsweise deren höchsten, niedrigersten oder durchschnittlichen Preis, sind die Daten sehr ähnlich, aber nicht identisch.

Interessanterweise sind die Wetterdaten über einen gewissen Zeitraum am komplexesten. Diese sind Teilweise periodisch, aber auch nicht immer. Dazu kommt der Umstand, dass nicht nur Temperaturen, sondern auch Luftdruck und andere Werte mit aufgenommen wurden. Ein Überblick ist in Abbildung 9.6 zu sehen. Sie stammen auch von [giv20] und es gibt daher auch keine weiteren Informationen über die Daten.

¹<https://de.finance.yahoo.com/>

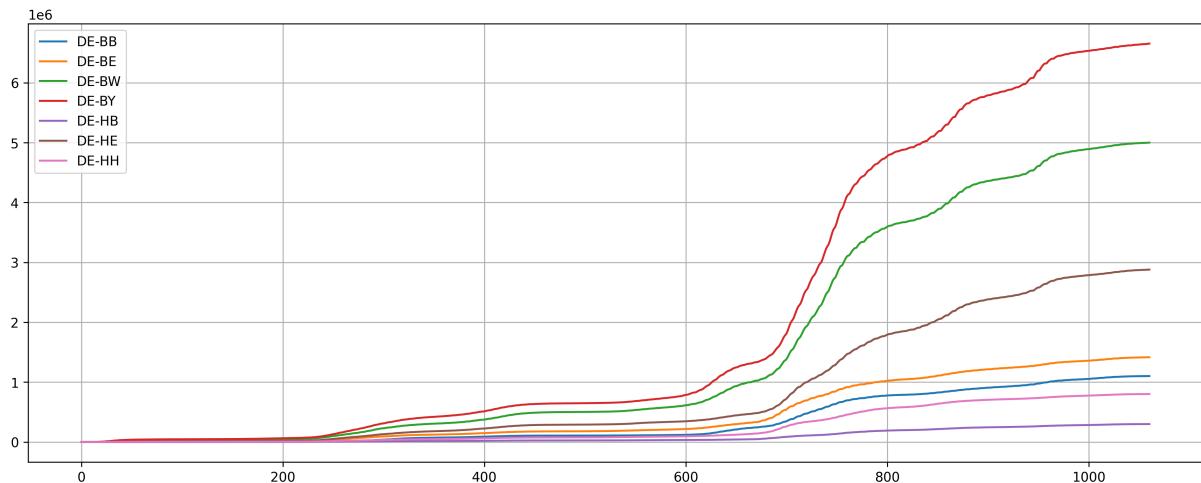


Abbildung 9.3 – Infektionszahlen einiger Bundesländer zu Beginn der Pandemie als Testdaten

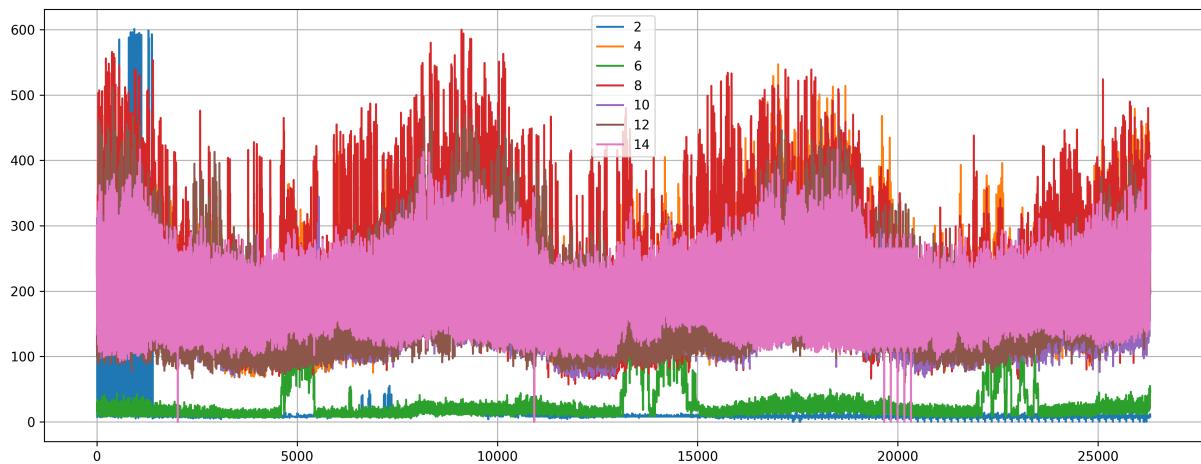


Abbildung 9.4 – Elektrizitätsverbrauch als Testdaten

9.2 Vergleich der Algorithmen und Modelle

9.2.1 Performance Metriken der ML Modelle

Um die Vergleichbarkeit der einzelnen Metriken zu gewährleisten, wurden die realen Daten alle erst einmal uniformiert. Dies bedeutet, dass die Daten auf einen Wertebereich von 0 bis 1 skaliert wurden und auf die gleiche Anzahl an Elementen je Zeitreihe gebracht wurden. Somit lässt sich der Vorverarbeitungsschritt entfernen und die Modelle können direkt auf den Daten trainieren.

Grundsätzlich sind die Trainingszeiten der generativen Modelle stark unterschiedlich. Modelle aus der tsgm Bibliothek [NIK23] sind deutlich deutlich langsamer als die nativen keras Modelle, welche selber implementiert wurden. Da aber die tsgm Modelle bei weitem komplexer sind und bereits nutzbare Ergebnisse mit weniger Iterationen liefern, wurde hier die Zahl der Iterationen reduziert, um die Laufzeit zu verkürzen. Der Test wurde nur mit wenigen Iterationen durchgeführt, da hier am entscheidendsten die Menge an Daten zum Training relevant war. Um die Einflüsse anderer Programme auf dem System zu eliminieren oder wenigstens zu minimieren

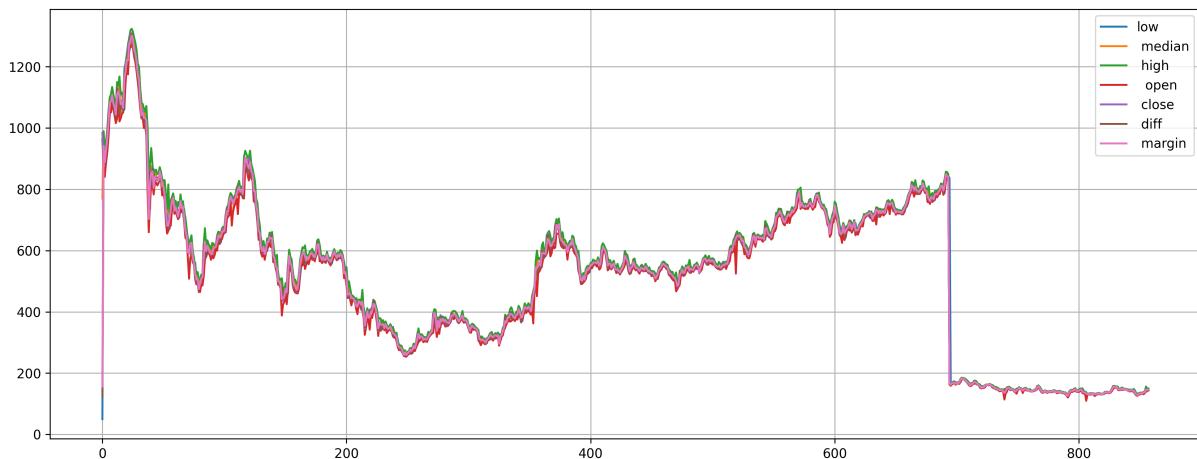


Abbildung 9.5 – Aktiendaten als Testdaten

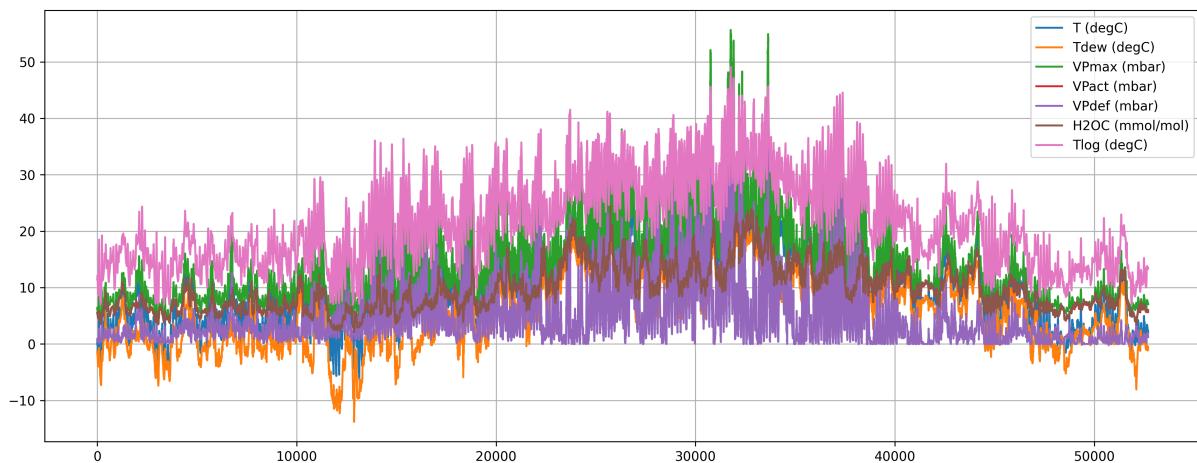


Abbildung 9.6 – Wetterdaten als Testdaten

wurde der Test in einen eigens hierfür erstellten Docker Containern durchgeführt. Dieser Container wurde mit 2 Central Processing Unit (CPU) Kernen und 8 Gigabyte (GB) Random Access Memory (RAM) ausgestattet.

Da sowohl die tsgm Bibliothek als auch selber eine Implementierung der CGAN Architektur erstellt wurde, bat es sich an diese als Vergleich zu nutzen. Wie in Abbildung 9.7 zu sehen ist, wird der Docker Container über ein Powershell Script gebaut, startet das Python Script, speichert alle notwendigen Metriken und startet den Container neu. Dies sorgt dafür, dass alle gesammelten Ressourcen wieder freigegeben werden und somit ein klares Bild über die realen Anforderungen des Modells entstehen. Damit fremde Einflüssen auf die Messungen minimiert werden, wurde der Test 6 mal wiederholt und der Durchschnitt gebildet. Die Ergebnisse sind in Abbildung 9.9 zu sehen. Es ist klar, dass die tsgm Implementierung deutlich länger braucht als die des nativen keras Modells. Dies lässt sich auf mehrere entscheidende Faktoren zurückzuführen. Die tsgm Bibliothek ist noch sehr neu und, zur Zeit der Implementierung, erst in Version 0.0.4. Auch sind die Architektur der Modelle komplexer, da sie mit deutlich mehr Parametern in mehreren Schichten trainieren.

Dennoch ist der nur sehr geringe Anstieg der Laufzeiten des Keras Modells sehr beeindruckend.

```
$runs = "50", "100", "150", "200", "250", "300", "350"
$algos = "CGAN-keras", "CGAN-tsgm"
$repeatCount = 10

docker build -t test_runs:1.0 .
docker run -m 8g --cpus 2 -d --name test_statistics test_runs:1.0

for ($i=1; $i -le $repeatCount; $i++) {
    Write-Host "Repeat count: $i"
    foreach ($algo in $algos) {
        {
            foreach ($run in $runs)
            {
                Write-Host "Running Docker command with run = $run and algo = $algo"
                docker exec -it test_statistics python main.py $run $algo
                $fileName = "<Path>\stats_${algo}_${run}.txt"
                Start-Sleep -s 10
                docker stats --no-stream --format
                    "{`container': `{{.Name}}`,`memory': `{{.MemUsage}}`,`cpu': `{{.CPUPerc}}`}" >> $fileName
                docker restart test_statistics
            }
        }
    }
}
```

```
FROM python:3.11-slim-buster
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1
WORKDIR /app
COPY requirements.txt /app/
RUN pip install --upgrade pip \
    && pip install -r requirements.txt
COPY . /app/
CMD ["python", "main.py"]
```

Abbildung 9.8 – Dockerfile zum Erstellen des Docker Containers

Abbildung 9.7 – PowerShell Script zum Testen der Modelle in einem Docker Container

Keras hat einen minimalen Anstieg, während der Anstieg der tsgm Modelle linear zur Anzahl der Iterationen ist. Da der RAM Verbrauch der tsgm Modelle sehr hoch war, wurde dieser auch gemessen. Hier ist auch wieder der Docker Container hilfreich, da die Isolation der Container dafür sorgt, dass der Verbrauch nicht durch andere Programme beeinflusst wird. Wie in Abbildung 9.10 zu sehen ist, ist selbst bei kleinen Modellen der RAM Verbrauch der tsgm Modelle deutlich höher als der der nativen Modelle. Daher ist besonderst für diese eine vernünftige Vorverarbeitung wichtig, da sonst weder die Laufzeit noch die Hardware Anforderungen von normalen Computern akzeptabel bzw ausreichen sind.

In Abbildung 9.11 ist noch einmal der Verlauf aller getesteten Modelle zu sehen. Hier ist deutlich zu erkennen, dass die meisten tsgm Modelle deutlich mehr Zeit benötigen als die nativen Modelle.

9.2.2 Performance Metriken der TSA Algorithmen

Die Trainingszeiten der TSA Algorithmen sind deutlich geringer als die der ML Modelle, wie in Tabelle 9.2 zu sehen ist. Der Cubic Spline Algorithmus ist hierbei der schnellste, da er nur eine Interpolation durchführt und keine weiteren Berechnungen benötigt. Schaut man auf die anderen Modelle, so fällt Amira auf, dieser ist weitaus langsamer als die anderen Algorithmen. Dies liegt daran, dass Amira eine sehr komplexe Architektur besitzt und daher deutlich mehr Rechenleistung benötigt.

9.2.3 Auslastung der Hardware in Produktion

Wie bereits vorher einmal angemerkt, sind die ML Modelle deutlich fordernder als es die TSA Algorithmen sind.

Um dies aber einmal in Produktion sehen zu können, wurde für die Django API ein separater Prometheus Importer für die CPU Auslastung geschrieben. Dieser sammelt in 2 Sekunden Inter-

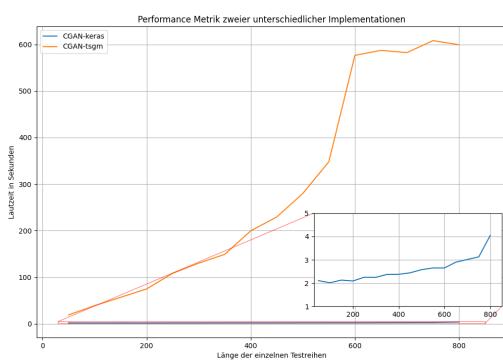


Abbildung 9.9 – Laufzeit des nativen CGAN Modells im Vergleich zur tsgm Implementierung in Abhängigkeit der Länge der Zeitreihen

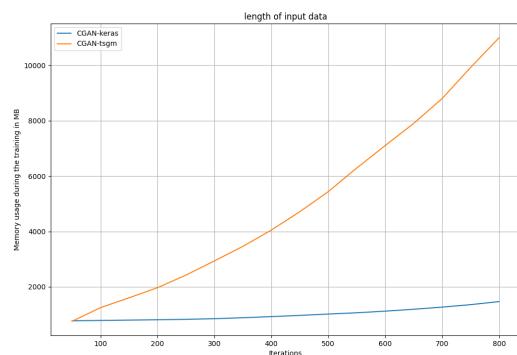


Abbildung 9.10 – RAM Auslastung des nativen CGAN Modells im Vergleich zur tsgm Implementierung in Abhängigkeit der Länge der Zeitreihen

Tabelle 9.2 – Trainingszeiten der TSA Algorithmen

Algorithm	Iteration	Trainingszeit in Sekunden
Amira	/	0.452361
Cubic	/	0.004374
EMD	0	0.038583
EMD	1	0.038053
EMD	2	0.040761
EMD	3	0.039456
SSA	0	0.013947
SSA	1	0.014150
SSA	2	0.012575
SSA	3	0.012223

vallen die Auslastung der CPU und wird über das Elasticsearch Logstash Kibana (ELK) Stack visualisiert. Leider konnte dies aber nicht auf gleiche Art und Weise mit der RAM-Auslastung getestet werden, da Python's Speichermanagement und seine Garbage Collection nicht vorhersehbar sind. Trotz separater Reinigung zeigte sich kein wirklicher Anstieg der RAM-Auslastung oder stieg nach Abschluss des Trainingsprozesses sogar noch.

In Abbildung 9.12

9.2.4 Ähnlichkeit der Zeitreihen der ML Modelle

Um die Ähnlichkeit der Zeitreihen zu bewerten wurden die in Sektion 8.1.1 vorgestellten Metriken genutzt. Diese Metriken zeigen direkt, dass Beispielsweise die Ähnlichkeit der ML Modelle basierend auf Ihrer Architektur sehr unterschiedlich ist. Rekursive Modelle wie beispielsweise das LSTM Modell, zeigen eine sehr viel schnellere Annäherung zu den realen Daten als die generativen Modelle. Da diese aber versuchen eine Funktion zu erlernen, welche die Daten generiert

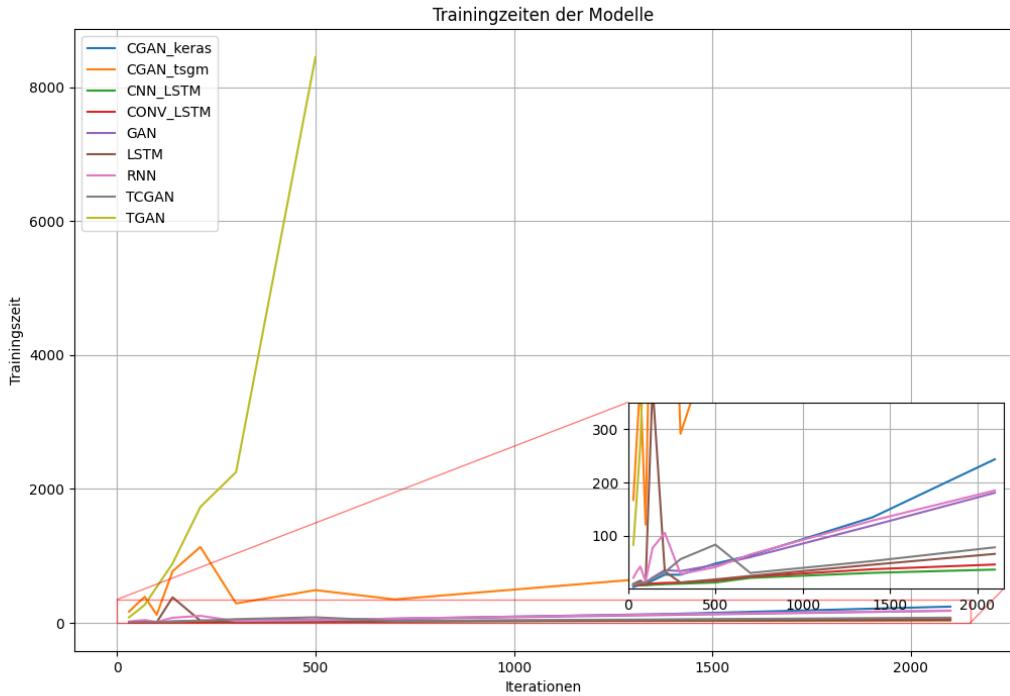


Abbildung 9.11 – Laufzeit der einzelnen Modelle im Vergleich zur Anzahl der Iterationen

anstatt Muster aus einem Fenster an vorherigen Werten zu lernen und daraus den nächsten Wert zu berechnen, ist dies wenig verwunderlich. Interessant ist aber, dass über einen Anstieg der Iterationen die Genauigkeit der generativen Modelle zwar tendenziell besser wird, aber von Außereinflüssen unterbrochen wird. Um dennoch eine vernünftige Vergleichbarkeit zu erziehen, wurden alle Algorithmen mit der gleichen Anzahl an Iterationen trainiert. Sie wurden in Intervallen von 10, 30, 50, 70, 100, 140, 210, 300, 500 getestet. Die nativen Keras Modelle wurden noch in Intervallen von 700, 1400 und 2100 Iterationen getestet. Für die tsgm Modelle war dies nur Teilweise noch möglich, da sich die Laufzeit im Extremfall des TGANs schon auf über 1h16min für 500 Iterationen belief. Da hier auch die Trainingszeit je Iteration Anstieg hätte dies, gerade über mehrere Trainingsdatensätze, zu lange gedauert und wäre daher wenig sinnvoll.

Interessante Punkte sind beispielsweise die Laufzeit der Modelle, die Anzahl an Iterationen im Verhältnis zur Wasserstein Distanz und die Anzahl an Iterationen im Verhältnis zur Vektor Distanz. Andere interessante Punkte sind beispielsweise das Privatsphäre Maß auch in ein Verhältnis zur Distanz und zur Laufzeit zu setzen. Gerade der letzte Punkt sollte interessante Ergebnisse liefern, da die generativen Modelle eine sehr hohe Laufzeit haben und daher auch ein hohes Maß an Privatsphäre bieten sollten, aber die Distanz zum original nicht zu gering sein soll, da es sonst diese nicht gut genug nachbildet.

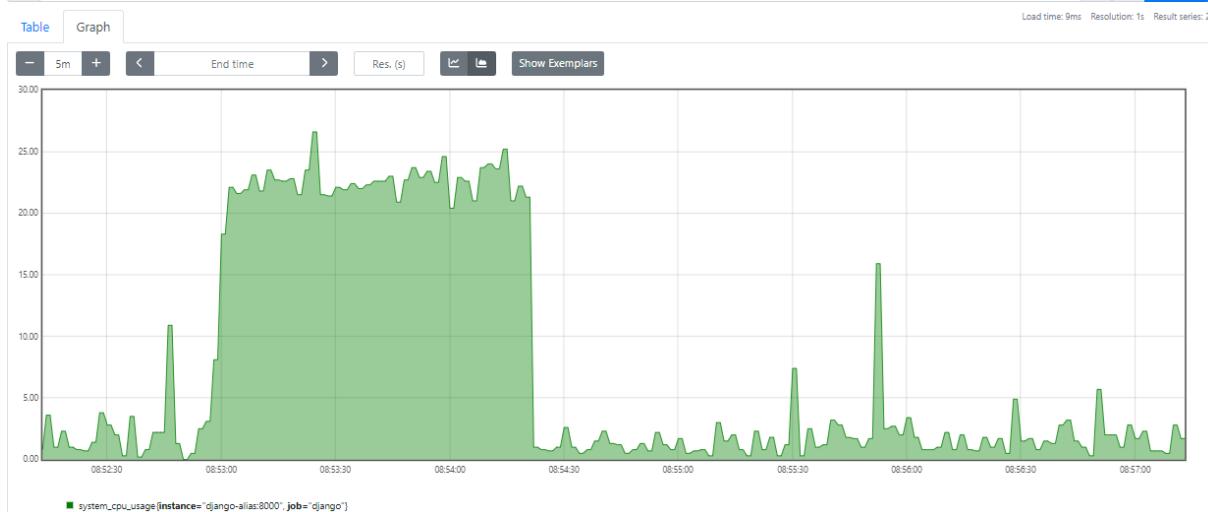


Abbildung 9.12 – Der erste große Anstieg zeigt die Auslastung während des Trainings eines GAN Modells (Zwischen 08:53 und 08:54), der kurze Anstieg während 08:56 zeigt die CPU Auslastung während der SSA-Algorithmus aufgerufen wurde

Analyse der rekursiven Modelle

RNN Betrachtet man die Grafik des RNN, zeigt sich eine erhebliche Fluktuation, aber im Vergleich zu anderen Modellen weist es einen sehr stabilen Wert auf. Die Wasserstein-Distanz divergiert bei 500 Iterationen, erreicht aber hier nur einen Wert von 0,124 – eine sehr gute Annäherung. Ähnlich verhält es sich mit der Vektordistanz, die bei nur 0,02 liegt. Beide Werte, auch die sehr niedrige Standardabweichung, deuten auf eine hohe Übereinstimmung mit den Originaldaten hin, und das fast unmittelbar. Eine recht niedrige Membership Inference Attack (MIA)-Metrik zeigt ebenfalls, dass das Modell die Privatsphäre nicht ausreichend schützt und Rückschlüsse auf Originalmuster zulässt.

LSTM Die Daten dieses Modells verhalten sich interessant. Die Wasserstein-Distanz steigt kontinuierlich an, zeigt also eine Entfernung von den Originaldaten, allerdings in einem, verglichen mit anderen Modellen, langsamen und geringen Rahmen. Die Standardabweichung steigt ähnlich zur Wasserstein-Distanz; gleiches gilt für die MIA-Metrik, die leider dennoch sehr gering ist. Die Vektordistanz liegt fast bei 0 und zeigt einen Ausreißer, der sich schnell wieder einpendelt. Aus den Daten des Mann-Whitney-U-Tests lässt sich ein Trainingseffekt erkennen, aber die P-Werte fallen nie unter die 0,05-Marke.

Convolutional LSTM Die Wasserstein-Distanz dieses Modells startet sehr hoch, fällt schnell ab und fluktuiert, bevor sie sich bei einem Wert von 0,1 einpendelt. Das Modell kann also sehr schnell die Verteilung der Originaldaten erfassen. Ähnlich verhält es sich mit der Vektordistanz, die ebenfalls hoch beginnt und sich etwas langsamer einpendelt als die Wasserstein-Distanz. Mit einem Wert von ungefähr 0,45 ist sie aber dennoch akzeptabel. Die MIA-Metrik und die Standardabweichungen folgen einem ähnlichen Verlauf: Sie starten hoch, fallen, fluktuiieren und pendeln sich ein. Am Ende erreicht das Modell nur einen Wert von 0,65 in der MIA, was recht gut ist. Die Ergebnisse des Mann-Whitney-U-Tests zeigen niedrige P-Werte, die aber nie den Schwellwert von

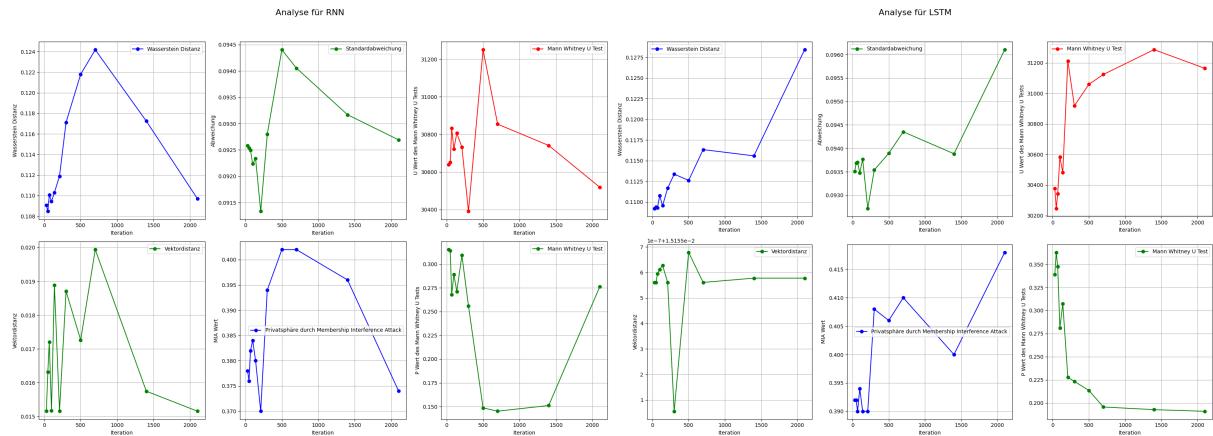


Abbildung 9.13 – Analyse des RNN Modells

Abbildung 9.14 – Analyse des LSTM Modells

0,05 erreichen. Das Modell erreicht somit nie eine exakte Annäherung an die Originaldaten, was durchaus erwünscht sein kann. Der starke Anstieg um Iteration 1400 sollte durch weitere Tests untersucht werden.

CNN LSTM Die Wasserstein-Distanz zeigt bei ungefähr 700 Iterationen eine signifikante Spitze. Da der Wert über mehrere Datensätze hinweg berechnet wurde und die Extrema nicht stärker als bei anderen Iterationen abweichen, scheint das Modell in diesem Bereich zu divergieren. Die Standardabweichung zeigt keine Auffälligkeiten. Die Vektordistanz verhält sich ähnlich wie die Wasserstein-Distanz, nur mit einem niedrigeren Wert, der gegen Ende der Iterationen wieder ansteigt. Interessant ist der Abfall der MIA-Metrik, als die Wasserstein- und Vektordistanz ansteigen, was etwas kontraintuitiv erscheint. Die Werte des Mann-Whitney-U-Tests zeigen ähnliche Muster und normalisieren sich nach der initialen Trainingsphase.

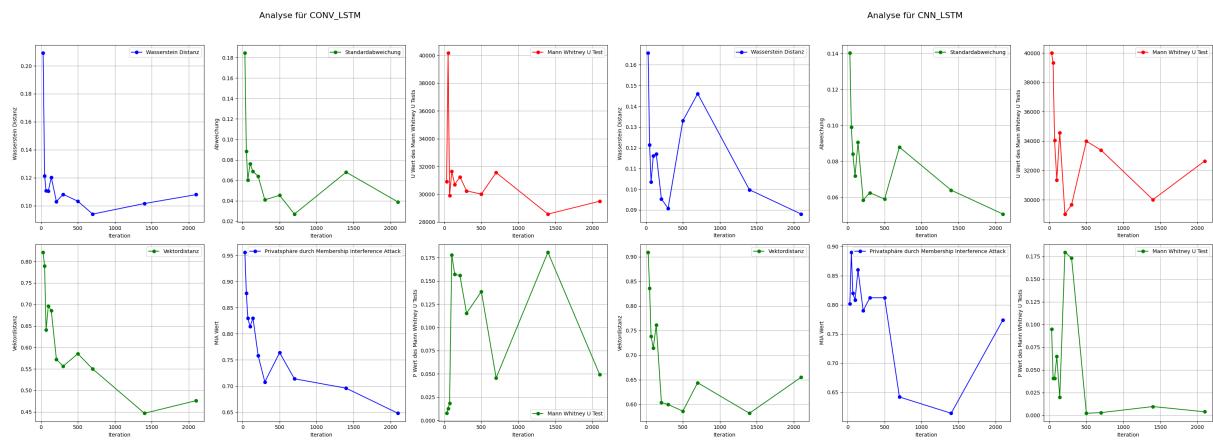


Abbildung 9.15 – Analyse des Convolutional LSTM Modells

Abbildung 9.16 – Analyse des CNN LSMT Modells

Analyse der generativen Modelle

GAN Das GAN zeigte eine signifikante anfängliche Verbesserung und verringerte schnell die Diskrepanz in der Verteilung zwischen generierten und realen Daten, wie die Wasserstein- und Vektordistanz-Metriken anzeigen. Die Standardabweichung des Modells nimmt ab, was auf eine Verbesserung der Konsistenz der Datengenerierung hindeutet.

Schwankungen in der Mann-Whitney-U-Test-Statistik und ihren P-Werten weisen auf Momente der Divergenz zwischen den synthetischen und realen Datenverteilungen hin, was auf eine potenzielle Überanpassung oder Instabilität des Modells im Laufe des Trainings schließen lässt. Die MIA-Werte deuten auf eine erhöhte Anfälligkeit für Inferenzangriffe hin, sind aber verglichen mit anderen Algorithmen recht gut.

Die anfänglichen und schnellen Erfolge des Modells zeigen einen schnellen Lernprozess, der ab ungefähr 1000 Iterationen stagniert.

TGAN Das TGAN zeigte gute Fähigkeiten bei der Annäherung an reale Datenverteilungen, wie die Trends der Wasserstein- und Vektordistanz zeigen. Bereits die ersten Iterationen des Modells zeigen Fortschritte bei der Generierung von synthetischen Daten. Gleichzeitig zeigte das TGAN Schwankungen in den Datenschutzmetriken, was eine Konsequenz der schnellen Verbesserung der Wasserstein- und Vektordistanzmetriken sein kann.

Die beobachtete Standardabweichung und die Mann-Whitney-U-Test-Statistiken wiesen zudem auf Perioden signifikanter Verbesserung und Stabilität in der Leistung des TGAN hin. Diese Metriken wiesen jedoch auch auf Fälle hin, in denen die Ergebnisse des Modells von den gewünschten Ergebnissen abwichen, was auf potenzielle Bereiche zur Verfeinerung hindeutet.

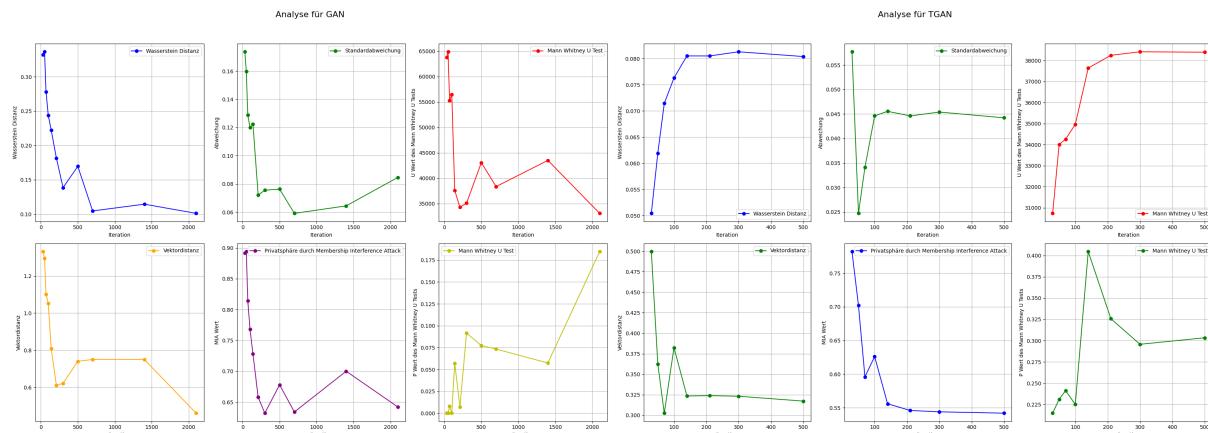


Abbildung 9.17 – Analyse des GAN Modells

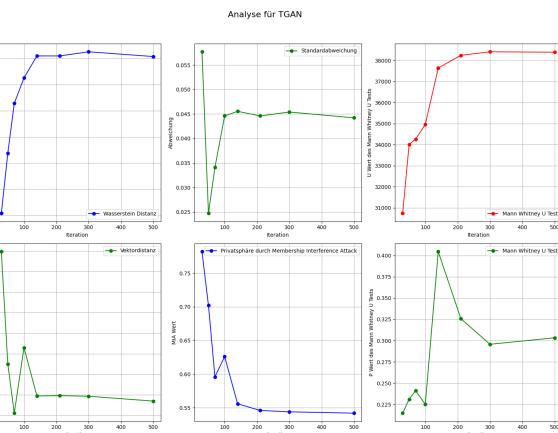


Abbildung 9.18 – Analyse des TGAN Modells

TCGAN Während des gesamten Trainingsprozesses zeigte das TCGAN eine gute Verbesserung mit einem abfallenden Lerneffekt, wie die Werte der Standardabweichung, Vektordistanz und Wasserstein-Distanz anzeigen.

Bezüglich der Privatsphäre zeigte das TCGAN eine schlechte Leistung, wie die MIA-Metrik zeigt.

Darüber hinaus wiesen die P-Werte des Mann-Whitney-U-Tests auf Momente hin, in denen sich die synthetischen Daten signifikant von den Originaldaten unterschieden, was Fragen hin-

sichtlich der Fähigkeit des Modells aufwirft, Daten mit konsistenten statistischen Eigenschaften über Iterationen hinweg zu generieren.

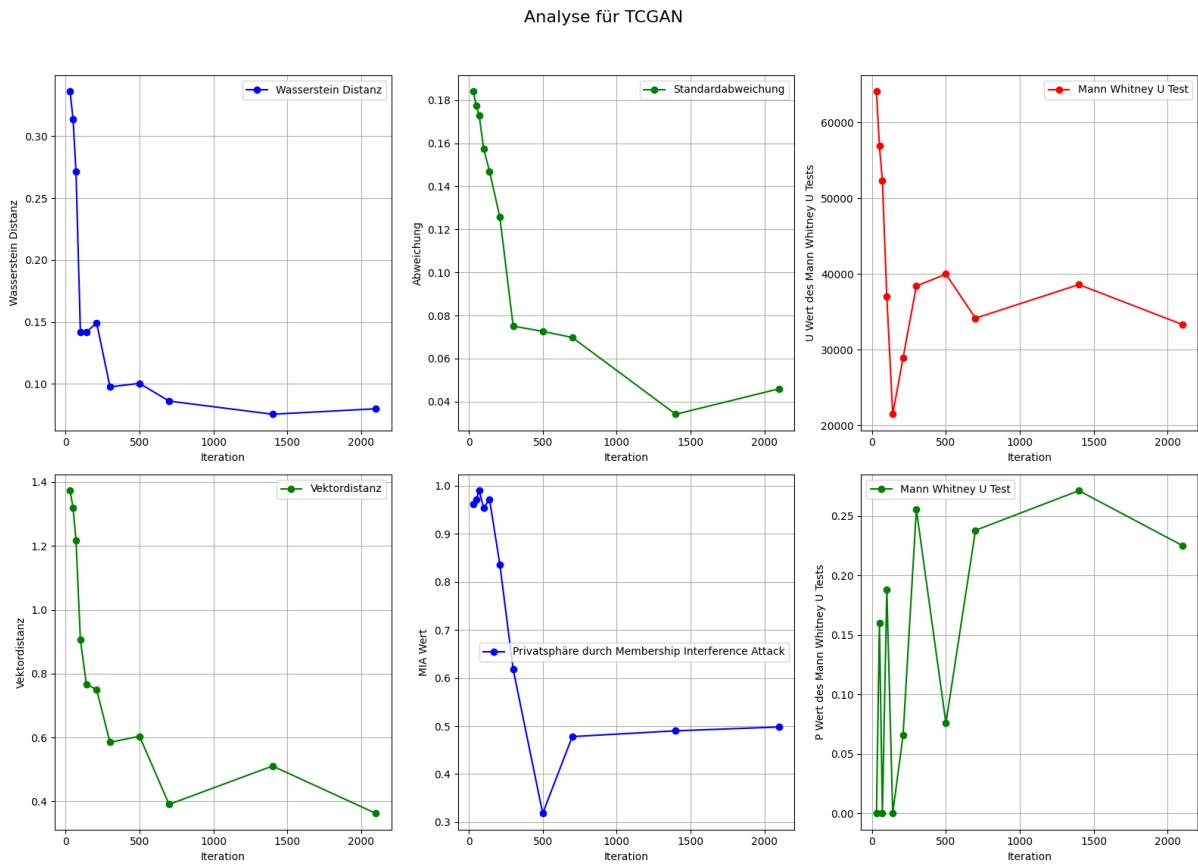


Abbildung 9.19 – Analyse des TCGAN Modells

CGAN-keras Die CGAN-Keras zeigten einen sehr guten Start mit einer steilen Verbesserung des Wasserstein-Abstands und einem erstaunlich niedrigen Ausgangswert. Die Vektordistanz zeigt einen ähnlichen Verlauf, nähert sich extrem schnell den Daten an und entfernt sich dann wieder. Hier ist eine hohe Fluktuation zu beobachten. Trotzdem sind die Standardabweichungen sehr gering und die Privatsphäre-Metriken zeigen einen durchaus guten Wert mit einer steigenden Tendenz. Die Analyse der Mann-Whitney-U-Test-Statistiken und der P-Werte ergab jedoch Zeiträume, in denen die statistischen Eigenschaften der synthetischen Daten signifikant von denen der realen Daten abwichen, während zu manchen Zeiten der Schwellwert von 0,05 unterschritten wurde.

CGAN-tsgm Die CGAN-Variante der tsgm-Bibliothek zeigt ähnlich gute und schnelle Annäherungen und Trainingsfortschritte wie die keras-Variante. Nach einer rapiden Lernphase pendeln sich die Werte ein und zeigen eine gute Annäherung an die Originaldaten, wie sich anhand der Wasserstein- und Vektordistanz sowie der Standardabweichung ablesen lässt. Leider zieht das

gute Erlernen auch Konsequenzen für die Privatsphäre nach sich, da die MIA-Metrik einen sehr niedrigen Wert zeigt.

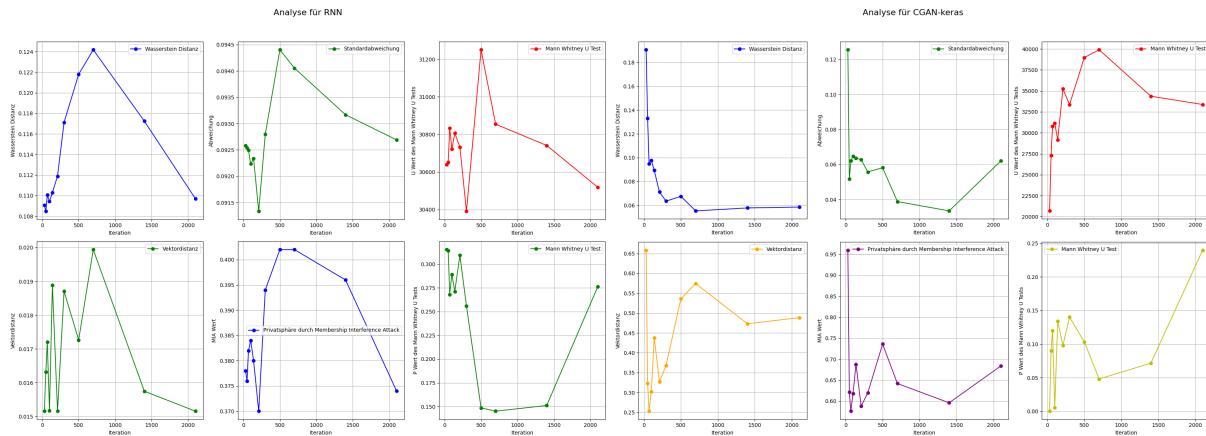


Abbildung 9.20 – Analyse des CGAN tsgm Modells

Abbildung 9.21 – Analyse des CGAN Keras Modells

Da in den vorherigen Abschnitten primär die reinen Verteilungen über verschiedene Datensätze gezeigt wurden, ist es auch interessant zu sehen, wie sich die einzelnen Modelle auf die einzelne Datensätze auswirken. Da die Modelle (mit ausnahme der tsgm Modelle) in der Lage sind spezifisch für die einzelnen Datensätze passende synthetische Varianten zu generieren, lässt sich die jeweilige Iteration eine Histogramms der synthetischen Daten erstellen, in dem sowohl die Verteilung als auch die direkten Werte verglichen werden.

Wie Anhand der in Abbildung 9.22 und 9.23 gezeigten Histogramme zu sehen ist, sind die synthetischen Daten sehr gut an die Originaldaten angepasst. Es gibt, auch nach etwa 2100 Iterationen noch Unterschiede im CGAN-keras Modell zu sehen, diese sind aber nur minimal, da es aber auch das Ziel war die synthetischen Daten dem original nur anzunähern, scheint dieses Ergebnis doch durchaus gut.

9.2.5 Ähnlichkeit der Zeitreihen der TSA Modelle

Um einen Vergleich von den TSA Algorithmen zu den ML Modellen zu ermöglichen, wurden gerade beim SSA und EMD Algorithmus ein paar Veränderungen vorgenommen. Da diese ja mit IMFs arbeiten, kann die Genauigkeit der Rückgabe über die verschiedenen IMFs variieren. Daher wurden die Tests mit steigender Anzahl an IMFs durchgeführt. Bei Modellen wie AMIRA oder Cubic-Spline war dies nicht ganz möglich, weshalb hier die Metriken nicht variieren und sich nur auf das Gesamtergebnis beziehen.

Analyse von SSA und EMD

SSA Der SSA Algorithmus zeigt, wie zu erwarten ist einen enormen Abfall in der Wasserstein Distanz sowie der Vektor Distanz und der Standardabweichung. Dies ist soweit auch verständlich, da hier mehr Informationen aus dem originalen Signal übernommen werden. Aber der Anstieg am Ende, in der alle IMFs genutzt werden ist interessant. Da die IMFs aber mit steigender Position die entsprechend Elemente der Reihe aufnehmen, könnte dies im Vergleich zur gesammten

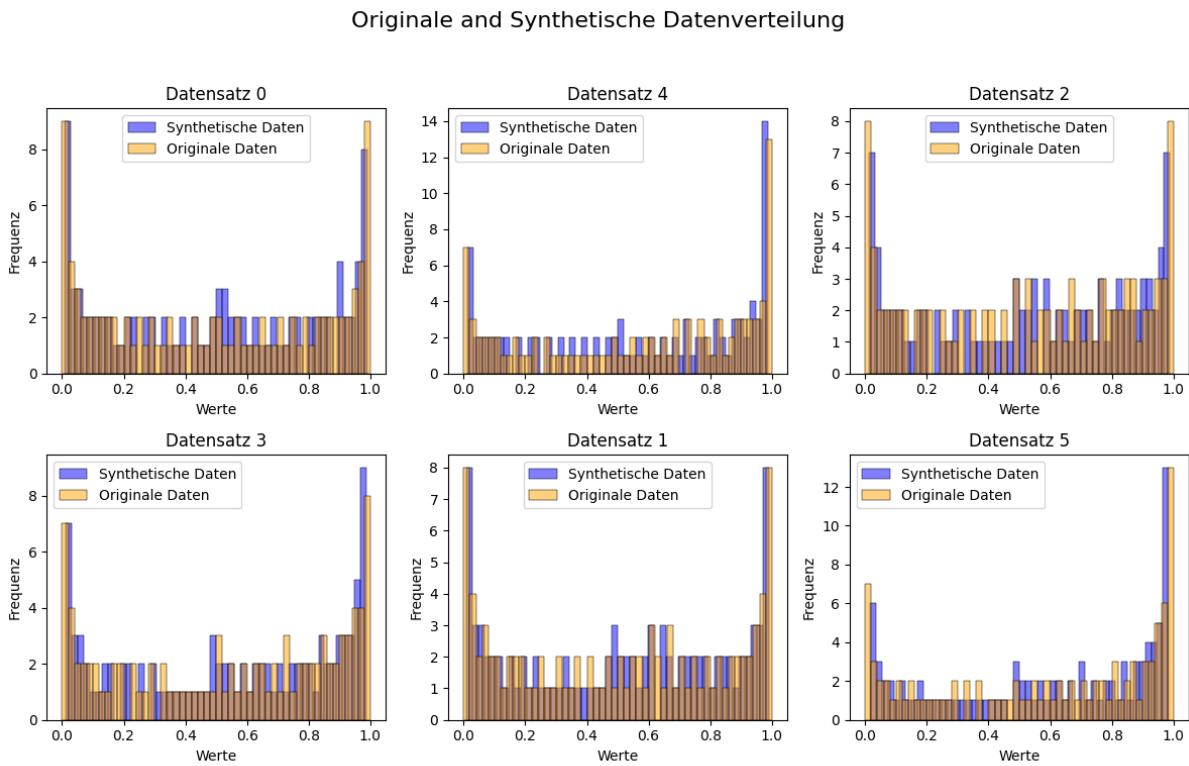


Abbildung 9.22 – Verteilungen des jeweiligen einzelnen Trainingsdaten zu ihren synthetischen Varianten

analysierten Datenmenge eine Entfernung von der originalen Verteilung mitsich ziehen. Die Vektordistanz ist davon ja nicht betroffen. Spannend ist die MIA Metrik, da diese einen erstaunlichen hohen Wert zeigt, und dies obwohl die Vektordistanz sehr niedrig ist, also eine gute Angleichung andeutet. Schaut man auf die Ergebnisse des Mann Whitney U Tests, so steigen diese an, anstatt abzufallen. Während dies eigentlich eine Entfernung von originalen Daten darstellen sollte, zeigen die anderen Metriken genau das Gegenteil. Dies müsste genauer untersucht werden.

EMD Der EMD Algorithmus zeigt ein sehr ähnliches Bild zu SSA Algorithmus und zeigt den Abfall in Standardabweichung, Wasserstein- und Vektor Distanz mit einem Anstieg, sofern alle IMFs einbezogen werden. Ein gleiches Verhalten zeigt auch die MIA Metrik, welche einen sehr hohen Wert zeigt, obwohl die anderen Metriken eine gute Angleichung andeuten. Hier steigt aber zwischenzeitlich der P-Wert des Mann Whitney U Tests enorm an, fällt aber wieder auf einen normalen Wert und stabilisiert sich hier. Auch dies ist interessant und sollte genauer untersucht werden.

Analyse von AMIRA und Cubic Spline

Da beide Modelle nicht auf ähnliche Weise flexibel sind wie die vorherigen Modelle und Algorithmen ist es hier nicht möglich eine Unterscheidung zwischen Iterationen oder IMFs zu treffen. Daher lies sich nur ein Wert jeweil berechnen. Beide Varianten besitzen einen sehr niedrigen Wert in der Wasserstein- und Vektor Distanz sowie in der Standardabweichung, relativ hohe Werte in

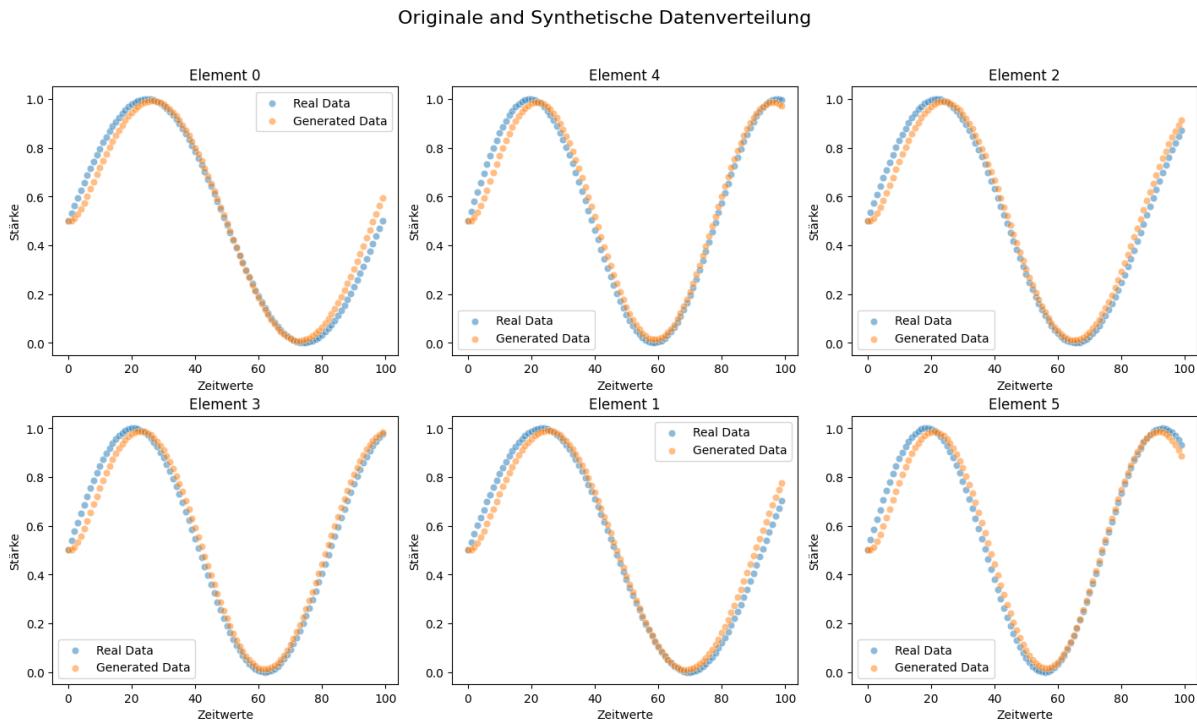


Abbildung 9.23 – Vergleich der einzelnen Trainingdaten zu ihren synthetischen Varianten

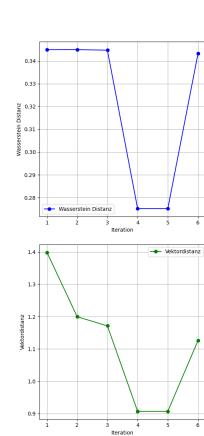
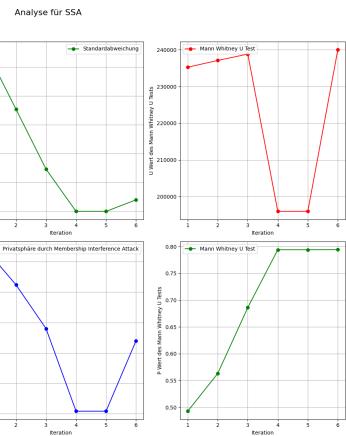
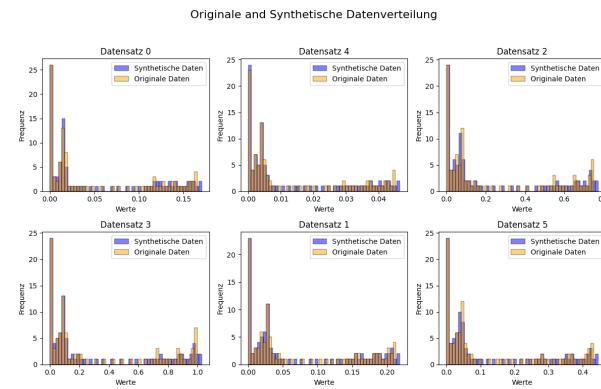
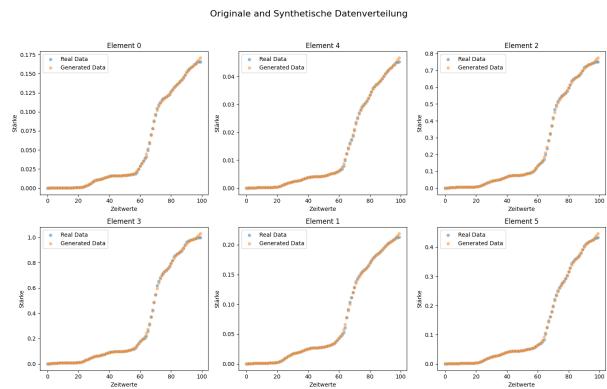
der MIA Metrik und einen recht hohen p Wert im Mann Whitney U Test.

9.3 Vorhersagefähigkeiten der rekusiven Modelle

Rekursive Modelle haben einen weiteren Vorteil gegenüber den generativen Modellen. Sie sind in der Lage auch einen nächsten Wert vorherzusagen, was bei den generativen Modellen nicht möglich ist. Hiermit bietet sich eine weitere Möglichkeit, neue Daten zu generieren. Forecasts sind nicht so genau und variieren stark, abhängig von der Art der Daten. Sind die Muster Ihnen bekannt, können sie recht gute vorhersagen treffen, bei komplexen Daten wie beispielsweise den Corona-Fallzahlen (Siehe Abbildung 9.3) ist dies aber nicht möglich, da sich hier noch kein Muster herausgebildet hat.

9.4 Zusammenfassung

Die gesammelten Daten zeigen ein gemischtes Bild. Wird grundsätzlich ein Vergleich zwischen ML und TSA angestrebt, so gewinnt immer der TSA Algorithmus, sollte es ein Wunsch nach Performance und Ressourcenverbrauch geben. Hier können ML Algorithmen einfach nicht mithalten. Schaut man auf die MIA Werte, so ist auch hier TSA ein wenig dem ML Modellen vorraus. Aber dies nur sofern viele Informationen verloren gehen. GANs und CGANs sind sehr gut im erzeugen synthetischer Daten, neigen aber zu Überanpassungen, welches sie unvorhersehbar macht. Hier lässt aber eine direkte Korrelation zwischen der Komplexität der Daten und den entstehenden Resul-


Abbildung 9.24 – Analyse des EMD Algorihtmus

Abbildung 9.25 – Analyse des SSA Algorihtmus

Abbildung 9.26 – Verteilung der originalen Daten mit den synthetischen Daten des SSA Algorihtmus

Abbildung 9.27 – Übereinstimmung des SSA Algorihtmus mit den originalen Daten

taten herausfinden, wie in Tabelle 9.1 zu sehen ist. Die Sinus Daten trainieren deutlich schneller als die Wetterdaten und erreichen schneller bessere Werte, wie auch an den extern beigefügten Tabellen zu erkennen ist. Aber um eine definitive Aussage zu treffen, müsste man mehrere Datensätze ähnlicher Autokorrelationswerte testen. Dies macht die gezielte Nutzung dieser Modelle nicht einfach. Um hier ein gewünschtes Ergebnis zu bekommen, muss im Notfall experimentiert werden.

Dennoch zeigt das CGAN eine durchschnittlich deutlich bessere Performance als das normale GAN Modell. Dies lässt sich auch an den Clustering Ergebnissen sehen. Diese sind separate im Anhang als 3D html Plots hinterlegt. Vergleicht man die generativen Modelle mit den rekursiven Modellen, so muss eindeutig gesagt werden, dass diese eine deutlich besser Annäherung an die originalen Daten zeigen. Sie sind dabei auch deutlich schneller, benötigen also weniger Ressourcen und sind einfacher zu trainieren. Leider kommt durch die Annäherung auch eine geringe Privatsphäre, da die MIA Metrik einen sehr niedrigen Wert zeigt. Dann kommt dazu die Uneindeutigkeit der Ergebnisse. Steigt die Komplexität, so steigt die Trainingszeit der Daten. Zu langes training kann aber zu Overfitting führen, eine zu kurze Trainingszeit zu einer schlechten Annäherung.

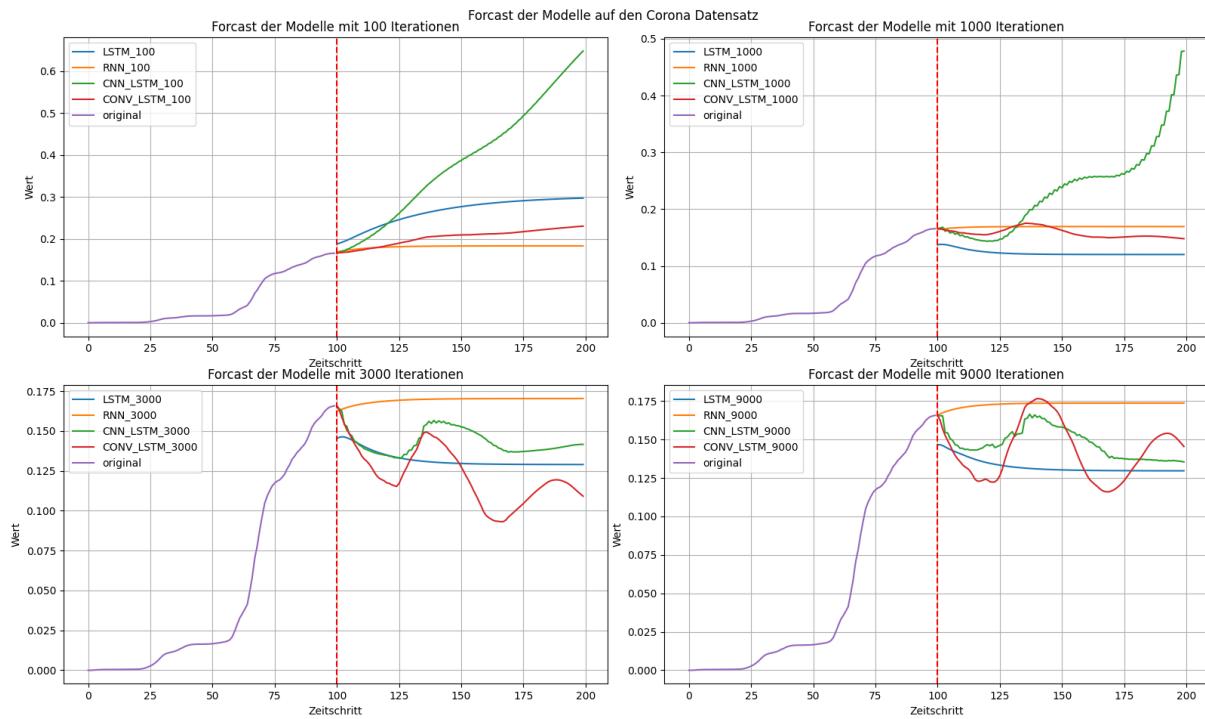


Abbildung 9.28 – Forcast der jeweiligen rekursiven Modelle

Dies gibt dem Nutzer viel Kontrolle, aber dies nur durch ausgedehntes Testing oder einen guten loss Wert. Hier wäre Erfahrung mit den Modellen aber wichtig.

Vergleicht man diese Werte mit den TSA Algorithmen, so wirken diese primär erstmal deutlich attraktiver. Sie sind schneller, brauchen weniger Ressourcen und sind auch recht flexibel. Selbst komplexe Zeitreihen können schnell approximiert werden. Probleme wie Overfitting gibt es nicht. Underfitting lässt sich, jedenfalls beim Vorhandensein von genügend IMFs, auch vermeiden, sogar steuern. In Kombination mit dem Möglichkeiten des Frontends, welches, wie in Abbildung 7.16 aus Sektion 7.4.9 hier direkt Veränderungen an der Konfiguration zulässt, ist dies ein sehr mächtiges Werkzeug, welches deutlich mehr Kontrolle an den Nutzer abgibt und ihn deutlich schneller zum Ziel führt. Auch sind Modelle wie Amira, Cubic Spline oder das nicht implementierte Prophet in der Lage größere Datenmengen in kürzester Zeit zu verarbeiten.

Hier ist also eine Abwegung notwendig. Braucht man schnell die Daten, so sollte man auf jeden Fall zu Algorithmen aus der Zeitreihenanalyse greifen. Hier erhält man direkt ein Ergebnis und kann dieses bei Bedarf anpassen. Maschinelles Lernen wird bei Datenmengen interessant, welche groß und komplex sind, und bei denen meist einen ungefähren Verlauf benötigt. Für komplexer Systeme, welche beispielsweise mit Aufreibern klar kommen müssen, sind generative Modelle interessanter als TSA Algorithmen.

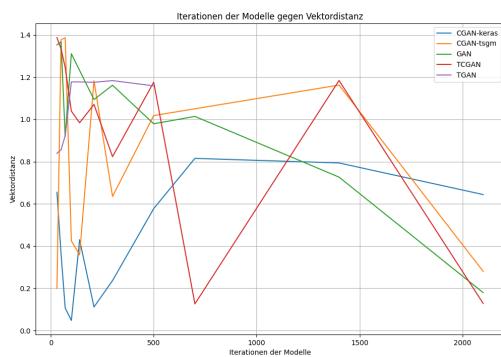


Abbildung 9.29 – Vektordistanz zwischen den Lösungen und dem synthetischen Daten, Trainingsdaten: corona.csv

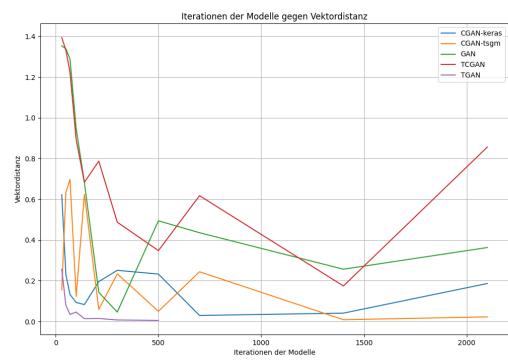


Abbildung 9.30 – Vektordistanz zwischen den Lösungen und dem synthetischen Daten, Trainingsdaten: sinus.csv

10 Nutzerstudie

10.1 Inhalte der Studie

Um eine Aussage über die Nutzbarkeit der Anwendung zu treffen, wurden zwei Verschiedene Studien durchgeführt. Die erste Studie nutzt eine Thinking Aloud Technik, bei der die Probanden unter einer vorgegebenen Aufgabenstellung die Anwendung verwenden und dabei ihre Gedanken laut äußern. Es wird dabei nicht nur die Aufgabe bewertet, sondern auch die wie die Nutzer Anwendung selbst wahrnehmen, welche Probleme sie haben und wie sie diese lösen und ob sie die Anwendung intuitiv bedienen können. Dadurch kann im Nachhinein eine Aussage getroffen werden inwieweit die erstellten Konzepte vom Nutzer angenommen werden und ob sie so intuitiv bedienbar sind, wie angenommen. Gerade im Bezug auf Shneidermans Golden Rules [Shn+16] lässt sich dies bewerten.

Die zweite Studie ist etwas größer angelegt und soll die Anwendung in einem größeren Rahmen bewerten. Hierzu wurden die Nutzer in zwei Gruppen aufgeteilt und bekommen jeweils eine leicht abgewandelte Aufgabe gestellt. Diese können sie alleine lösen um anschließend ihre Erfahrungen und Meinungen in einem Fragebogen zu äußern. Der Fragebogen ist in drei Teile aufgeteilt. Im ersten Abschnitt werden die Nutzer gebeten, die Anwendung zu bewerten und ihre Erfahrungen zu äußern. Dies wird mit Hilfe des System Usability Scale (SUS) Fragebogens [Bro95] durchgeführt. Im zweiten Teil wird die mentale Leistung der Nutzer gemessen und dafür der Nasa Task Load Index (NASA-TLX) [HS88] verwendet. Um die in den ersten beiden Teilen in einen Kontext zu setzen, werden im dritten Teil noch einige Fragen zur Person und dem Umgang mit der Anwendung gestellt.

Der NASA-TLX Fragebogen existiert seit 1988, ist anerkannt und weit verbreitet. Ziel des Fragebogens ist es zu messen, wie mental Fordernd die Anwendung ist. Mithilfe von 6 Fragen wird die mentale Belastung in den 6 folgenden Kategorien bewertet:

Geistige Anforderung Wie geistig anspruchsvoll war die Aufgabe?

Körperliche Anforderung Wie körperlich anspruchsvoll war die Aufgabe?

Zeitliche Anforderung Wie gehetzt oder eilig war das Tempo der Aufgabe?

Leistung Wie erfolgreich waren Sie bei der Erledigung dessen, was von Ihnen verlangt wurde?

Anstrengung Wie hart mussten Sie arbeiten, um Ihre Leistung zu erreichen?

Frustration Wie unsicher, entmutigt, gereizt, gestresst und genervt waren Sie?

Die Bewertung erfolgt auf einer Skala von 1 bis 10, wobei 0 für sehr geringe Anforderung steht und 10 für sehr hohe Anforderung. Aufsummiert ergibt der finale Wert die mentale Anforderung. Ist der Wert hoch, dann muss der Nutzer sich stark auf die Anwendung konzentrieren. Bei einem geringen Wert hat der Nutzer wenig Probleme mit der Anwendung klar zu kommen.

Der System Usability Scale Fragebogen ist ebenfalls sehr weit verbreitet und wird seit 1986 verwendet. Er besteht aus 10 Fragen (Siehe 10.1), die in einer 5-Punkte Likert-Skala bewertet werden. Eine 5 steht dabei für „Stimme voll zu“ und eine 1 für „Stimme überhaupt nicht zu“. Die Fragen sind im Wechsel positiv und negativ formuliert, um eine Verzerrung der Ergebnisse zu verhindern. Dies macht die Auswertung ein wenig komplizierter, da die negativ formulierten Fragen umgerechnet werden müssen. Nach einer Normalisierung werden die negativen Werte umgerechnet und danach aufsummiert und mit 2.5 multipliziert.

Die Fragen sind:

- Ich denke, ich würde die Website/die App regelmäßig nutzen.
- Die Website/die App scheint mir unnötig kompliziert zu sein.
- Ich finde, die Website/die App ist einfach zu bedienen.
- Ich denke, ich würde technische Unterstützung benötigen, um die Website/die App nutzen zu können.
- Ich denke, dass die verschiedenen Funktionen der Website/des Apps gut integriert sind.
- Die Website/die App scheint mir zu inkonsistent.
- Ich glaube, dass die meisten Menschen schnell lernen können, wie man die Website/die App benutzt.
- Die Website/die App scheint sehr umständlich in der Nutzung zu sein.
- Ich fühle mich sehr sicher bei der Nutzung der Website/des Apps
- Ich musste viel lernen, um mich mit der Website/dem App zurechtzufinden.

10.2 Studienablauf

Die für dieses Projekt gewählten Personas sind divers. Sie haben einen gewissen technischen Hintergrund, sind aber keine Experten. Dies erlaubt daher auch eine breite Auswahl an Teilnehmern in die Studie einzubeziehen. Für die Studie wurden 11 Probanden ausgewählt. Die Auswahl erfolgte über persönliche Kontakte, weitere Nutzer haben die Software ausgetestet, aber nicht an der Studie teilgenommen.

Schaut man sich die Hintergründe der Probanden an, so fällt auf, dass diese sehr unterschiedlich sind. Die Probanden sind zwischen 24 und 40 Jahre alt, sind in unterschiedlichen, auch nicht technischen Berufen tätig und zeigen eine ziemlich unterschiedliche technische Grundfähigkeit auf. Es gibt bereits Teilnehmer mit einem geringen Vorwissen im Bereich der Simulationssoftware.

Da es, wie in der im Anhang beigefügten Aufgabenstellung zwei unterschiedliche Aufgaben gibt, wurden die Probanden in zwei Gruppen aufgeteilt, welche aus jeweils 5 Teilnehmern bestanden. Teilnehmer Nr. 11 hatte beide Aufgaben durchgeführt. Mit außnahme einiger weniger Probanden, welche zusätzlich noch eine kurzen Thinking Aloud Technik unterzogen wurden, durften die meisten Probanen sich selber ohne weitere Unterstützung an der Aufgabe versuchen. Da die Webanwendung auf einem Server läuft, mussten die Probanden nicht extra eine Anwendung installieren,

sondern konnten diese direkt über den Browser aufrufen. Aufgabenstellung und Testdaten wurden den Probanden in einer Datei zur Verfügung gestellt.

Teilnehmer der Thinking Aloud Studie bekamen die gleiche Aufgabe gestellt, diese sollten sie aber unter Überwachung durchführen und wurden gebeten, ihre Gedanken während der Aufgabenlösung laut auszusprechen. Diese, sowie ihre Interaktionen, wurden alle protokolliert. Wichtig ist, dass die Probanden dabei nicht beeinflusst wurden und somit die Aufgabe frei und selbstständig lösen mussten. Die Protokolle erlauben einen Einblick in die Gedanken der Nutzer und erlauben Rückschlüsse darüber, inwieweit der Nutzer die Prozesse und Möglichkeiten der Anwendung realisiert und versteht.

10.3 Auswertung der Studie

10.3.1 Ergebnisse der Thinking Aloud Studie

Die Studie wurde mit fünf Probanden durchgeführt, die aufgefordert wurden, die Aufgaben zu lösen und dabei ihre Gedanken laut auszusprechen, wie bereits in Sektion 10.1 beschrieben.

Zu Beginn erhielten die Teilnehmer etwas Zeit, um sich mit der Software vertraut zu machen, da sie zuvor noch keine Erfahrungen mit dieser hatten. Dieser Schritt war entscheidend, um ihnen ein grundlegendes Verständnis für die Funktionsweise der Software zu vermitteln.

Während der Aufgabenbearbeitung hatten die Teilnehmer im Allgemeinen wenig Schwierigkeiten. Die meisten verstanden die Aufgabenstellung schnell und konnten durch das Durchsuchen der Menüs die benötigten Funktionen finden. Oft wichen sie anfangs von der eigentlichen Aufgabe ab und erforschten eigenständig die Möglichkeiten der Software. Positiv hervorgehoben wurden dabei die integrierten Tooltips.

Bei der Definition der Generierung synthetischer Daten kamen die meisten Teilnehmer gut mit dem Dialog zurecht. Der dritte Schritt, die Definition der Datenvorverarbeitung, stellte sich jedoch als verwirrend heraus. Die Teilnehmer waren sich über die Bedeutung der einzelnen Optionen und deren Auswirkungen auf die Daten unsicher. Hinzu kam, dass das verwendete Schema zwar eine Beschreibung für alle Optionen bot, die eingesetzte rsjf-Bibliothek jedoch keine Erklärungen für die Auswahl der Imputationsalgorithmen bei den Radio Buttons zur Verfügung stellte. Drei der Nutzer entdeckten zudem nicht die Möglichkeit, sich einen Graphen der Daten anzeigen zu lassen, obwohl dieser direkt unter der Konfiguration zu finden war. Der Teilnehmer, der den Graphen entdeckte, war sich jedoch nicht bewusst, dass man einzelne Graphen dynamisch ausblenden kann, und konnte daher nur wenig Nutzen aus der Visualisierung ziehen.

Nach dem Absenden der Konfiguration hatten die Teilnehmer wenig Schwierigkeiten, entweder testweise die Graphen zu verschieben und zu speichern oder das Training der Modelle zu starten. Besonders positiv wurde die prozentuale Statusanzeige während des Trainings der Modelle aufgenommen. Bei der Erstellung der Projekte hatte ein Teilnehmer bereits an der Nutzerstudie der ursprünglichen Anwendung teilgenommen und verfügte daher über einen guten Überblick über die notwendigen Schritte. Die anderen Teilnehmer hatten jedoch Schwierigkeiten, die Unterscheidung zwischen Projekten, Tracks und Datasets zu verstehen. Da diese Struktur jedoch zweckmäßig ist und einen bestimmten Nutzen verfolgt, könnten zukünftige Einarbeitungen diese Probleme beheben.

Auffällig war, dass viele Teilnehmer den Informationsbutton nicht näher betrachteten. Dieser bietet eine Visualisierung der Websockets, also der trainierenden Modelle und der gesendeten

Projekte. Die Personen, die ihn benutzten, waren verwirrt, als sie keine Aktivität sahen, weil gerade nichts trainiert wurde. Hier ist also weitere Arbeit nötig, um die Nutzer besser über die Bedeutung und Funktion dieser Anzeige zu informieren.

10.3.2 Auswertung der Fragebögen

Die Fragebögen wurden von allen Teilnehmern ausgefüllt, einschließlich jener, die an der Thinking Aloud Technik teilnahmen. Die einzelnen Ergebnisse sind in der Tabelle A.1 im Anhang dargestellt. Generell lässt sich sagen, dass die Nutzer die Anwendung als gut bedienbar empfanden. Die durchschnittlichen Ergebnisse auf der SUS-Skala lagen bei 76.32, was darauf hindeutet, dass die Anwendung als benutzerfreundlich wahrgenommen wurde. Mental fühlten sich die Nutzer nicht stark gefordert, was sich in einem durchschnittlichen TLX-Wert von 36.5 widerspiegelt, ein Indikator dafür, dass die Aufgaben ohne große Anstrengungen gelöst werden konnten.

Besonders interessant ist der Vergleich der Ergebnisse der Fragebögen im Zusammenhang mit Alter und technischer Begabung der Teilnehmer (siehe Abbildung 10.1). Hierbei zeigte sich, dass Teilnehmer mit geringerer technischer Begabung die Anwendung wenigerfordernd fanden, während jene mit durchschnittlicher Begabung dies als anspruchsvoller empfanden. Teilnehmer mit hoher technischer Begabung fanden die Aufgaben wiederum leichter. Dies mag zunächst kontraintuitiv erscheinen, aber unter Berücksichtigung der Beobachtungen aus der Thinking Aloud Studie ergibt sich ein schlüssiges Bild: Nutzer mit geringer technischer Begabung gingen erstaunlich schnell durch die Anwendung und klickten die meisten Funktionen direkt an, ohne sich zunächst auf die Aufgabe zu konzentrieren. Bei Nutzern mit durchschnittlichem technischen Verständnis war dies nicht der Fall; sie verhielten sich deutlich zögerlicher.

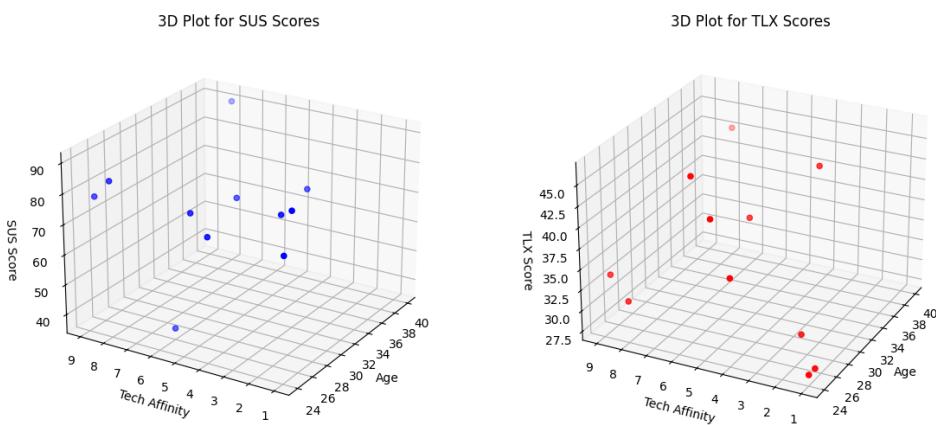


Abbildung 10.1 – Grafische Darstellung der Ergebnisse des NASA-TLX und SUS-Fragebogens in Abhängigkeit des Alters und der technischen Begabung

Eine Korrelation des Alters mit dem Testergebnis war nicht erkennbar. Die Altersverteilung war jedoch zu klein, um aussagekräftige Schlussfolgerungen zu ziehen, insbesondere im Hinblick auf die technische Begabung. Während sich die technische Begabung bei den Teilnehmern in den mittleren Zwanzigern noch recht gut verteilte, war sie bei den älteren Teilnehmern deutlich höher.

Wie in Abbildung 10.3 dargestellt, war insbesondere bei Frage 1, 'Ich denke, ich würde die Webseite/die App regelmäßig nutzen.', eine hohe Zustimmung zu verzeichnen. Die Nutzer wurden gebeten, sich in die Situation zu versetzen, solch eine Software zu benötigen. Obwohl die meisten Teilnehmer in einer solchen Situation nicht sind, war es notwendig um über den Fragebogen eine valide Aussage zu erhalten.

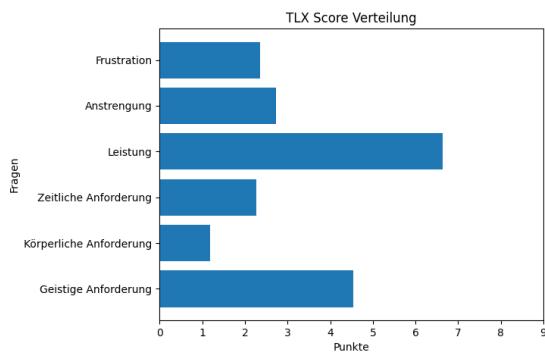


Abbildung 10.2 – Auswertung des NASA-TLX Fragebogens

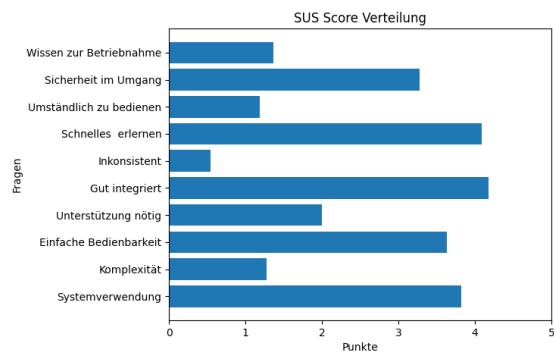


Abbildung 10.3 – Auswertung des SUS Fragebogens

11 Ausblick

11.1 Einführung neuer Modelle

In dieser Arbeit lag der Fokus auf rekursiven und generativen Modellen. Variational Autoencoder, die aktuell große Beliebtheit genießen, stellen eine vielversprechende Erweiterung dar. Diese Autoencoder sind besonders effektiv in der Generierung synthetischer Daten, da sie fähig sind, aus erlernten Datenmustern realistische und vielfältige Beispiele zu erzeugen. Für Variational Autoencoder wurde in der Testumgebung bereits ein Grundgerüst entwickelt, jedoch konnte aufgrund zeitlicher Beschränkungen keine umfassende Bearbeitung zur Erzielung aussagekräftiger Ergebnisse vorgenommen werden. Eine Weiterentwicklung in diesem Bereich könnte daher sehr fruchtbar sein.

11.2 Erweiterungsmöglichkeiten innerhalb des ML Bereiches

In der Auswertung wurde viel über die tsgm Modelle geschrieben. Sie produzierten gute Ergebnisse, nur kamen sie mit Hardwareanforderungen und einer nicht direkt lösbar Speicherinflizienz. Dies macht sie ungeeignet für den Einsatz in der geschriebenen Django API. Wenn sich dieser Umstand ändert, wäre es möglich die tsgm Modelle in die API zu integrieren. Hierzu müssten lediglich die Datenbankeinträge dafür erstellt werden. Im Testprojekt sind diese bereits an das in der API genutzte Konzept angepasst worden. Da die Modelle auch zeitlich nicht mit den doch relativ kurzen Trainingszeiten der nativen Modelle mithalten können, aber bereits mit weniger Iterationen nutzbare Ergebnisse liefern, wäre hierfür auch noch ein Informationskonzept zu erstellen. Der letzte Punkt betrifft wie Erstellung von Projects. Die tsgm Modelle können zwar ebenso aus der Datenbank geladen werden, bemüthen aber einen anderen Ladeprozess. Dieser ist weitestgehend integriert, aber kann zeitaufwendig sein. Da lange Prozesse nicht in das REST Prinzip passen, bräuchte es hier eine Websocketschnittstelle, die den Ladeprozess überwacht und die fertig generierten Daten an die Spring API sendet. Da diese aber über das Frontend angesprochen wird, muss dieses auch die Websockets integrieren und ist daher auch zu ändern, wodurch eine grundsätzlich kleine Änderung der Funktionalitäten eine große Änderung der Gesamtarchitektur nach sich zieht.

Darüber hinaus kann auch mit freieren Parametern experimentiert werden. So kann die Anzahl der Neuronen in den Layern verändert werden, oder die Anzahl der Layer selbst. Dies bedeutet aber einen enormen Validierungsaufwand, da nicht alle Kombinationen sinnvoll sind oder gar funktionieren. Eine weitere Möglichkeit ist die stärkere Integration von Loss-Funktionen. Diese wird aktuell nur sporadisch genutzt.

Aufgrund von erhobenen Statistiken könnten auch Vorschläge erstellt werden. Über ein dynamisches Auswahlverfahren könnten dem Nutzer passende Modelle mit passenden Konfigurationen vorgeschlagen werden, oder das dieser sich um die genauen Parameter zu kümmern hat. Hierfür kann ein Language Level Model (LLM) genutzt werden, welches in einem Dialog mit dem

Nutzer die Parameter erfragt und direkt mit der API kommuniziert. Dies würde die Nutzerfreundlichkeit enorm erhöhen. Die Grundlagen hierfür sind bereits teilweise gelegt, da es eine umfangreiche OpenAPI Definition gibt, mti welcher LLMs umgehen können.

11.3 Erweiterungsmöglichkeiten außerhalb des TSA Bereiches

Gerade die rekursiven Modelle sind in der Lage eine Forcast zu erstellen. Eine durchaus interessante und praktische Funktion. Da dies grundsätzlich auch durch IMFs möglich ist, wäre eine mögliche Erweiterung hier auch interessant. Während die Grundlagen hierfür in der Testumgebung bereits geschaffen wurden, waren die Ergebnisse noch nicht zufriedenstellend und wurden daher nicht in die Auswertung mit übernommen. Dennoch bietet sich hier ein Potenzial, welches es zu nutzen gilt. Ansätze wie Fouriertransformationen könnten hierbei helfen. Modelle wie Prophet könnten weitere interessante Ansätze liefern, welche es zu untersuchen gilt.

11.4 Erweiterungsmöglichkeiten innerhalb des Frontends

Obwohl die Studie eine insgesamt gute User Experience des Frontends bestätigt, gibt es Bereiche, die verbessert, vereinfacht oder fehlertoleranter gestaltet werden könnten. Der umfangreiche Einsatz von Json Schema bietet zwar flexible Erweiterungsmöglichkeiten für die Funktionalitäten, zeigt jedoch im Bereich der Benutzerunterstützung und Hilfestellungen Grenzen auf.

11.5 Testabdeckung

Eine Erhöhung der Testabdeckung wäre ein wichtiger Schritt zur weiteren Verbesserung der Anwendung. Dazu ist die Integration von Testcontainern notwendig, sowie die Einbindung einer gemeinsamen Datenbankinstanz in die Testumgebung, um effektive Unit Tests zu ermöglichen.

Für Component- und Systemtests, die auch das Frontend einbeziehen, wären Tools wie Pact und Playwright geeignet. Pact ermöglicht das Testen der Kommunikation zwischen zwei Systemen, indem ein simulierter Pactserver die Anfragen des zu testenden Systems empfängt und entsprechende Antworten liefert. Dieser Prozess kann automatisiert werden. Playwright ist für das Testen des Frontends vorgesehen, wobei ein Browser gestartet wird, der bestimmte Aktionen ausführt und die Ergebnisse überprüft. Diese Tests können direkt in Java integriert und automatisiert ausgeführt werden.

Eine unzureichende Testabdeckung oder das Fehlschlagen von Tests könnte dazu führen, dass die Pipeline automatisch abgebrochen wird, um die Qualität der Anwendung zu gewährleisten.

12 Fazit

12.1 Ergebnisse der Nutzererfahrung

Die Nutzererfahrung war der zweite wichtige Punkt innerhalb der Arbeit. Nutzer müssen ihre Aufgaben möglichst schnell und einfach erledigen können. Aus der Studie ließ sich ableiten, dass die Nutzer die Aufgaben recht problemlos erledigen konnten, sobald sie verstanden hatten, was zu tun ist. An diesen Punkt zu gelangen, war jedoch nicht immer einfach. Dies ist zwar nicht verwunderlich, da die Nutzer die Aufgaben zum ersten Mal sahen, aber dennoch ein wichtiger Punkt.

Die Ergebnisse der Evaluation zeigen, dass die Zeitreihenzerlegung in den meisten Fällen bessere Ergebnisse liefert. Dies ist vor allem darauf zurückzuführen, dass die Zeitreihenzerlegung die Datenpunkte der Originaldaten nicht verändert, sondern nur deren Anzahl reduziert. Die generativen Modelle hingegen erzeugen neue Datenpunkte, die nicht immer den Originaldaten entsprechen. Dies führt zu einer höheren Abweichung der generierten Daten von den Originaldaten.

12.2 Auswertung der umgesetzten Anforderungen

Zu Beginn der Arbeit wurde eine Liste an Anforderungen (siehe 8) aufgestellt, welche im Verlauf der Arbeit erfüllt werden sollten. Diese Anforderungen werden nun einzeln ausgewertet.

Vielseitige Datenintegration und Unterstützung mehrerer Datenformate Die Software unterstützt Datenformate wie CSV, JSON und NumPy-Arrays, welche einen großen Teil der gängigen Datenformate für Zeitreihen abdecken sollten. Die Daten können über die Weboberfläche oder über die REST API importiert werden, beispielsweise über Tools wie curl oder Postman¹.

Effiziente Datenübertragung, -validierung und Systemressourcennutzung Die Datenübertragung, insbesondere im Frontend, ist zweiteilig aufgebaut. Grundsätzlich können die Daten in einem separaten Bereich hochgeladen und durch die Django-API überprüft werden. Die zweite Möglichkeit, Daten hochzuladen, besteht bei der Erstellung von Konfigurationen. Um die Datenübertragung hier möglichst gering zu halten, werden die Daten erst hochgeladen, wenn die Konfiguration vollständig ist. Der Preview ist hiervon ausgenommen. Um die Anzahl an Datenbankoperationen möglichst gering zu halten und damit die Ressourcenauslastung niedrig, werden hier Daten nur gecacht und nichtpersistiert, sofern keine bereits persistenten Daten vorhanden sind. Das Konzept der Datenvorverarbeitung wurde in Sektion 7.3.5 genauer erläutert.

Benutzerorientierte Datenauswahl und -vorverarbeitung sowie Verwaltung: Um die Datenauswahl für Nutzer flexibel zu gestalten, wurde ein zweischneidiges Konzept implementiert. Da

¹Postman ist eine API-Plattform, <https://www.postman.com/>

gerade bei Zeitreihen die Datenmenge schnell sehr groß werden kann, werden nur die einzelnen Zeitreihen an die Konfiguration angehängt, um ungewollte Daten in einer Konfiguration zu vermeiden. Um dennoch die Datenauswahl effektiv zu gestalten, sind alle Zeitreihen an ein Dokument gebunden, welches referenziert werden kann und alle Zeitreihen enthält. Dies kann auch in der passenden Übersicht den gewünschten Anforderungen angepasst werden.

Die Vorverarbeitungsschritte werden dem Nutzer über anpassbare und erweiterbare JsonSchemas zur Verfügung gestellt. Diese sind in der Django-API einfach erweiterbar. Es existieren Imputations- sowie Transformationsalgorithmen, welche die Daten vorverarbeiten können. Der Nutzer hat hier freie Auswahlmöglichkeiten und kann die Transformationsalgorithmen seinen speziellen Anforderungen anpassen.

Visuelles Feedback, Datenexploration und transparente Ergebnisdarstellung Für visuelle Rückmeldungen wurde viel mit Graphen und Diagrammen innerhalb der Weboberfläche gearbeitet. Diese sind teilweise interaktiv und erlauben somit einen tiefen Einblick in die Unterschiede, die beispielsweise durch die Datenverarbeitung entstehen. Für die Fehlerbehandlung wurde eine eigene Komponente dem Frontend hinzugefügt. Die Fehlermeldungen aus den APIs sind standardisiert und auch mit i18n-Keys versehen, wodurch die spezifischen Fehler direkt in der Nutzersprache angezeigt werden können. Durch die Implementierung von Websockets ist eine direkte Status-/Fortschrittsanzeige möglich, welche den Nutzer über den aktuellen Stand der Verarbeitung informiert.

Sicherheit, Privatsphäre und sicherer Umgang mit Benutzerdaten Die Sicherheit der Anwendung ist mehrschichtig aufgebaut. Ein zentraler Punkt sind die APIs, die sowohl durch JWT-Authentifizierung als auch durch Cross-Site Request Forgery (CSRF)- und Cross-Origin Resource Sharing (CORS)-Richtlinien geschützt sind. Invalide Tokens haben nur Zugriff auf wenige öffentliche Endpunkte. Die Weboberfläche ist durch Routenschutz abgesichert und lässt nicht eingeloggte Nutzer lediglich auf die Login-Seite. Aktuell ist die Anwendung nur über HTTP erreichbar, jedoch kann Traefik problemlos konfiguriert und mit entsprechenden Zertifikaten ausgestattet werden.

Die spezifischen Daten der Nutzer werden in der Datenbank gespeichert, allerdings wurde auf eine zusätzliche Verschlüsselung der einzelnen Datenbankspalten verzichtet. Der Zugriff auf die entsprechenden Daten ist über die API nur für berechtigte Nutzer möglich, während die Datenbank selbst durch Docker Compose abgesichert ist und keinen Port nach außen freigibt, also nur innerhalb des Docker-Netzwerks erreichbar ist. Dies sollte ein ausreichendes Sicherheitsniveau für die Nutzerdaten bieten.

Trainieren der Modelle und Bereitstellung einer Vielzahl von Algorithmen Im Frontend werden aktuell vier verschiedene ML-Algorithmen zur Datengenerierung angeboten, von denen zwei rekursiv und die anderen beiden generativ sind. Für die Tests wurden fünf weitere Modelle geprüft: zwei rekursive und drei generative. Die rekursiven Modelle könnten direkt in die Anwendung integriert werden, während die generativen Modelle aufgrund unzureichender Performance bezüglich der genutzten Hardware nicht Bestandteil der Anwendung sind.

Im Bereich der TSA werden EMD, SSA, Amira und Cubic Spline angeboten. Der Algorithmus Prophet, der in der Testumgebung vorhanden ist, kann jedoch keine synthetischen Daten erzeugen und wurde daher nicht in die Anwendung integriert.

Benutzerschnittstelle, Benutzererfahrung und intuitive Konfiguration der Algorithmen Die Auswertung der SUS- und TLX-Studie (siehe 12.1) zeigt positive Ergebnisse. Der SUS-Score von 76.32 wird als *gut* eingestuft und der TLX-Score von 36.5 deutet auf eine *geringe mentale Belastung* hin, was insgesamt eine gute Benutzererfahrung widerspiegelt. Die intuitive Konfiguration der Algorithmen wurde durch den Einsatz von JsonSchema erreicht. Allerdings könnte, laut der Thinking Aloud Studie, noch Raum für Verbesserungen bestehen.

Dokumentation, Fehlerbehandlung, Überwachung und Deployment In der Implementierungsphase wurde die Dokumentation nicht vollständig abgeschlossen. Für die umfassende Dokumentation, die verschiedene Bereiche umspannt, wurde sich für eine Hugo-Dokumentation entschieden. Hugo, ein Static Site Generator, wie in Paragraph 6.3 erwähnt, ermöglicht es, komplexe Abhängigkeiten mit Mermaid² zu visualisieren. Die Dokumentation liegt in Markdown-Dateien vor und wird bei jedem Deployment aktualisiert und integriert. Die OpenAPI Spezifikation wurde für die Dokumentation der API-Schnittstellen genutzt, zugänglich im lokalen Deployment. Die Swagger.yaml-Dateien werden im externen Anhang bereitgestellt, ebenso wie eine Postman-Konfiguration.

Der Fehlerbehandlungsprozess könnte noch verbessert werden, da nicht alle Fehlercodes übersetzt sind und einige technische Fehlerbeschreibungen für Nutzer unklar sein könnten. Insbesondere die WebSocket-Komponenten benötigen eine bessere Fehlerbehandlung.

Das Projekt wird durch Prometheus und Graylog überwacht. Graylog muss initial konfiguriert werden und kommuniziert über das UDP-Protokoll. Sobald eingerichtet, werden Logs angezeigt und können gefiltert werden. Prometheus ist bereits konfiguriert und zeigt die relevanten Metriken an. Die Konfigurationen werden im externen Anhang bereitgestellt und sind Teil des Deployment-Projekts.

Das automatische Deployment erfolgt über einen GitLab Runner und eine entsprechende GitLab-CI-Konfiguration, wobei Docker Images erstellt und auf einem lokalen Server bereitgestellt werden.

12.3 Ergebnisse der Evaluation

Aufbauend auf den Ergebnissen aus Kapitel 9 lässt sich ein Fazit ziehen, das die Ergebnisse zusammenfasst und einen Ausblick auf mögliche weitere Forschungen gibt. Die Ergebnisse der Evaluation können in verschiedene Punkte gegliedert werden.

Effizienz Effizienz ist grundsätzlich ein wichtiges Thema. Vergleicht man die beiden Ansätze, so ist die Zeitreihenzerlegung deutlich effizienter als maschinelles Lernen. Dies war jedoch von Anfang an klar. Innerhalb der ML-Ansätze gibt es aber auch deutliche Unterschiede. Die nativen Keras-Modelle sind bei weitem effizienter als die aus externen Bibliotheken, nutzen jedoch auch deutlich weniger Parameter. Wie jedoch in Sektion 9.2.4 dargestellt wurde, bedeutet dies keinesfalls schlechtere Ergebnisse. Benötigt man jedoch Daten, die sehr nah an den Originaldaten liegen, so eignen sich rekursive Modelle besonders gut, da sie sehr schnell sind und bereits mit wenigen Iterationen eine gute Nachbildung liefern können.

²Mermaid ist eine Diagrammsprache, <https://mermaid-js.github.io/mermaid/>

Genauigkeit Die Genauigkeit der generierten Daten ist ein weiterer wichtiger Punkt. Ziel war es, synthetische Daten zu schaffen, die das Original abbilden, um einen Mehrwert zu bieten. Ausreißer an den falschen Stellen oder grundsätzlich andere Verläufe können beispielsweise zu falschen Schlussfolgerungen führen. Betrachtet man daher die Ergebnisse der beiden Methoden, so liefert die Zeitreihenzerlegung in den meisten Fällen eine deutlich bessere Annäherung. Generative Modelle fügen der neuen Zeitreihe Variabilität hinzu, die stark schwankt und von der Anzahl an Trainingsiterationen und der Korrelation der Originaldaten abhängt. Rekursive Modelle hingegen liefern eine schnelle Abbildung der Daten, tendieren nicht zum Overfitting und sind in der Lage, auch Prognosen zu treffen.

Limitationen Das Projekt war recht umfangreich. Es wurden insgesamt 4 TSA-Algorithmen und 9 ML-Modelle getestet, doch bieten sich hier weitere Möglichkeiten. Auch könnte ein gezieltes Fine-Tuning der Parameter die Ergebnisse in manchen Bereichen deutlich verbessern. Sollten die Tests mit viel mehr Daten durchgeführt werden, könnte eventuell auch eine Korrelation zwischen der Komplexität oder Autokorrelation der Daten und ihrer Effizienz und Genauigkeit festgestellt werden. In den Tests wurde zwar eine festgestellt, diese kann aber aufgrund der geringen Datensetze nicht als allgemeingültig angesehen werden.

12.3.1 Schlusswort

Abschließend lässt sich sagen, dass sowohl ML- als auch TSA-Ansätze ihre Vor- und Nachteile haben. Generative Modelle eignen sich am besten für ähnliche Daten, während rekursive Modelle und TSA-Algorithmen sich den Originaldaten schnell annähern, aber schneller und ressourceneffizienter sind. Diese Nachteile können durch sorgfältige Auswahl des Algorithmus und geeignete Vorverarbeitungsschritte ausgeglichen werden.

A Appendix

Tabelle A.1 – Teilnehmerbewertungen: Der SUS-Score sowie der TLX-Score reichen bis 100. Ein hoher SUS-Score und ein niedriger TLX-Score sind erstrebenswert. Die technische Erfahrung wurde auf einer Skala von 0 (kein technisches Verständnis) bis 9 (sehr gutes technisches Verständnis) bewertet.

Alter	technische Begabung	SUS	TLX	Vorwissen	Beruf	Aufgabe
25	5	79.16	42.59	8	Stylist	1
25	1	87.50	27.77	keine	/	2
31	5	75.00	38.88	keine	Elektroniker	1
25	9	77.08	33.33	keine	E-Learning Producer	1
24	8	85.41	31.48	ja	Studium	2
34	3	77.08	44.44	/	/	2
24	1	75.00	27.77	None	Wirtschaftsstudent	2
24	4	75.00	37.03	wenig	Sachbearbeiter	1
40	8	89.58	42.59	/	/	/
27	2	81.25	29.62	none	Historian	1
26	6	37.50	46.29	Simulink	Student	2

A.1 Weitere Tabellen

Alle weiteren Daten, Tabellen, Grafiken und HTML Plots sind im externen Anhang dieser Arbeit beigefügt. Diese sind zu groß oder unmöglich in dieser Arbeit darzustellen.

Demonstration von Time Series Analytics Software auf einer Fachmesse

Hintergrund

Sie sind verantwortlich für die Demonstration der neuen Time Series Analytics Software Ihres Unternehmens auf einer Fachmesse. Die Software verwendet Kafka-Konsumenten, um in Echtzeit Daten von Sensoren und anderen Quellen zu streamen und wertvolle Einblicke in die Betriebsabläufe zu liefern. Allerdings erlaubt Ihre aktuelle Einrichtung nicht, komplexe Muster und Verhaltensweisen zu demonstrieren. Deshalb haben Sie im Vorfeld Daten vorbereitet.

Ihr Ziel ist es, potenziellen Kunden die Software erfolgreich zu demonstrieren und sie davon zu überzeugen, das Produkt Ihres Unternehmens zu kaufen.

Aber wie werden Sie das erreichen?

Sie haben die SUS-Software gefunden, eine kleine Web-App, die es Ihnen ermöglicht, eigene Sensor-Simulationsprojekte zu erstellen und sogar synthetische Daten nach Ihren Wünschen zu generieren.

Aufgabe 1:

Hier anmelden

Das Einzige, was Sie tun müssen, ist:

1. Ihre Daten abrufen (sie befinden sich im Ordner).
2. Ein Konto einrichten (verwenden Sie beliebige Werte, aber die E-Mail sollte den üblichen Konventionen folgen).
3. Sie haben sich entschieden, für Ihre Aufgabe maschinelles Lernen zu verwenden, da Sie nicht an Prognosen interessiert sind, wählen Sie ein generatives gegnerisches Netzwerk (GAN).
4. Erstellen Sie eine neue Trainingskonfiguration, geben Sie ihr einen vernünftigen Namen und eine Beschreibung und fügen Sie Ihre eigenen Konfigurationen hinzu.
5. Beginnen Sie mit dem Training des Modells. Während es trainiert, schauen Sie sich noch einmal in der Software um und sehen Sie, ob Sie etwas Interessantes finden.
6. Nach Abschluss des Trainings erstellen Sie Ihr eigenes Projekt mit einem Track.
 1. Beginnen Sie mit einem kleinen Schlafzyklus (10s).
 2. Öffnen Sie Ihr maschinelles Lernmodell.
 3. Erstellen Sie einen weiteren Schlafzyklus (10s).
 4. Erstellen Sie einen linearen Anstieg.
7. Beginnen Sie mit dem Senden Ihrer Daten.

8. Am Ende klicken Sie auf den Umfrage-Button und füllen Sie ihn aus, bitte fügen Sie die Aufgabenummer am Ende hinzu.

Danke!

Aufgabe 2:

Hier anmelden

Das Einzige, was Sie tun müssen, ist:

1. Ihre Daten abrufen (sie befinden sich im Ordner).
2. Ein Konto einrichten (verwenden Sie beliebige Werte, aber die E-Mail sollte den üblichen Konventionen folgen).
3. Sie haben sich entschieden, für Ihre Aufgabe Zeitreihenanalyse zu verwenden, wählen Sie den SSA-Algorithmus.
4. Erstellen Sie eine neue Trainingskonfiguration, geben Sie ihr einen vernünftigen Namen und eine Beschreibung und fügen Sie Ihre eigenen Konfigurationen hinzu.
5. Erstellen Sie Ihr eigenes Projekt mit einer Spur.
6. Beginnen Sie mit einem kleinen Schlafzyklus (10s).
 1. Öffnen Sie Ihr Zeitreihenanalysemodell.
 2. Erstellen Sie einen weiteren Schlafzyklus (10s).
 3. Erstellen Sie einen linearen Anstieg.
 4. Beginnen Sie mit dem Senden Ihrer Daten.
7. Am Ende klicken Sie auf den Umfrage-Button und füllen Sie ihn aus,

bitte fügen Sie die Aufgabenummer am Ende hinzu.

Danke!

List of Abbreviations

TSA	Time Series Analysis	58
PCA	Principal Component Analysis	6
ML	Machine Learning	V
GAN	Generative Adversarial Network	12
LSTM	Long Short-Term Memory	12
UCD	User Centered Design	
RNN	Recurrent Neural Network	12
ANN	Artificial Neural Network	12
IMF	Intrinsic Mode Function	5
HSA	Hilbert Spectral Analyse	6
EMD	Empirical Mode Decomposition	5
HHT	Hilbert-Huang Transform	6
SSA	Singular Specturm Analysis	6
SVD	Singular Value Decomposition	11
API	Application Programmable Interface	2
MMD	Maximum Mean Discrepancy	75
MUI	Material-UI	
SPA	Single-Page-Applikation	19
KLD	Kullback-Leibler Divergenz	74
t-SNE	t-Distributed Stochastic Neighbor Embedding	79
Isomap	Isometric Mapping	79
UMAP	Uniform Manifold Approximation and Projection	79
GAN	Generative Adversarial Network	12
WGAN	Wasserstein Generative Adversarial Network	56
TGAN	Time Series Generative Adversarial Network	13
CGAN	Conditional Generative Adversarial Network	13
RAM	Random Access Memory	88
CPU	Central Processing Unit	88
GB	Gigabyte	88
CICD	Continuous Integration Continuous Delivery	33

ACF	Autokorrelationsfunktion	73
FFT	Fast Fourier Transformation	74
MIA	Membership Inference Attack	92
LLM	Language Level Model	109
REST	Representational State Transfer	51
CSRF	Cross-Site Request Forgery	112
CORS	Cross-Origin Resource Sharing	112
UI	User Interface	51
ELK	Elasticsearch Logstash Kibana	90

Abbildungsverzeichnis

3.1	Grundidee einer LSTM-Architektur	13
5.1	Hauptansicht der originalen Anwendung, hier sind alle Projekte aufgelistet	19
5.2	DataSet Ansicht der originalen Anwendung	20
5.3	DataSet Ansicht der originalen Anwendung	20
6.1	Aktuelle Infrastruktur, sie verbindet das Frontend mit beiden APIs, welche Zugriff auf die PostgreSQL Datenbank haben. Überwacht werden sie von Prometheus und Graylog. Ein Kafka Producer wird genutzt um die generierten Daten als Streams an entsprechende Consumer zu senden. Redis dient als Cache für die Websockets des Django-Channel Frameworks. Hugo wird genutzt um die Dokumentation zu generieren und über Nginx verfügbar zu machen. Dies alles wird in Docker Containern ausgeführt.	34
7.1	Alte Datenstruktur (links) und neue (rechts)	37
7.2	Entity-Relationship-Diagramm der ursprünglichen Datenstruktur	47
7.3	vereinfachter Ablauf des Sendens von Statusinformationen mittels eines eigenen EventServices	48
7.4	Django Projekt Aufbau	50
7.5	Algorithmus zum zerlegen einer hochgeladenen JSON Datei	54
7.6	Entity-Relationship-Diagramm der aktuellen Datenstruktur	59
7.7	vereinfachter Ablauf des Sendens von Statusinformationen mittels eines eigenen EventServices	61
7.8	vereinfachter Ablauf des Sendens von Statusinformationen mittels eines eigenen EventServices	62
7.9	Analyse des CGAN tsgm Modells	64
7.10	Analyse des CGAN Keras Modells	64
7.11	WebSocket Drawer für das Trainieren von Modellen und als Fortschrittsanzeige für sendende Projects	65
7.12	Eingabefeld für die Datenvorverarbeitung sowohl für die Zeit Serien Analyse als auch für das maschinelle Lernen	68
7.13	Eingabefeld für die Datenvorverarbeitung sowohl für die Zeit Serien Analyse als auch für das maschinelle Lernen, hier können grafisch die Unterschiede zwischen den Originaldaten und den verarbeiteten Daten interaktiv angesehen werden.	69
7.14	In diesem Beispiel werden die originalen Daten ausgeblendet. Die roten Bereiche sind fehlende Werte, welche durch den Imputationsalgorithmus ersetzt wurden. Die große Lücke in der Mitte ist zu groß und wird nicht ersetzt, sondern durch ein ML Modell gefüllt.	69

7.15 Ansicht eines trainierten Modells mit den wichtigsten Informationen und einem Beispielbild	70
7.16 Ansicht einer Zeitreihenzerlegung über den EMD Algorithmus	70
7.17 Übersicht über die hochgeladenen Dateien und deren Inhalte	71
8.1 Visuelle Darstellung des Deployment Prozesses in GitLab CI/CD und lokaler Umgebung	83
8.2 Docker Compose File für den Produktionsprozess	84
8.3 Gitlab CI/CD Pipeline	84
9.1 Vergleich von realen und synthetischen Daten	85
9.2 Sinus Kurven als Testdaten	86
9.3 Infektionszahlen einiger Bundesländer zu Beginn der Pandemie als Testdaten	87
9.4 Elektrizitätsverbrauch als Testdaten	87
9.5 Aktiendaten als Testdaten	88
9.6 Wetterdaten als Testdaten	88
9.7 PowerShell Script zum Testen der Modelle in einem Docker Container	89
9.8 Dockerfile zum Erstellen des Docker Containers	89
9.9 Laufzeit des nativen CGAN Modells im Vergleich zur tsgm Implementierung in Abhängigkeit der Länge der Zeitreihen	90
9.10 RAM Auslastung des nativen CGAN Modells im Vergleich zur tsgm Implementierung in Abhängigkeit der Länge der Zeitreihen	90
9.11 Laufzeit der einzelnen Modelle im Vergleich zur Anzahl der Iterationen	91
9.12 CPU Auslastung der Django API	92
9.13 Analyse des RNN Modells	93
9.14 Analyse des LSTM Modells	93
9.15 Analyse des Convolutional LSTM Modells	93
9.16 Analyse des CNN LSMT Modells	93
9.17 Analyse des GAN Modells	94
9.18 Analyse des TGAN Modells	94
9.19 Analyse des TCGAN Modells	95
9.20 Analyse des CGAN tsgm Modells	96
9.21 Analyse des CGAN Keras Modells	96
9.22 Verteilungen des jeweiligen einzelnen Trainingsdaten zu ihren synthetischen Varianten	97
9.23 Vergleich der einzelnen Trainingdaten zu ihren synthetischen Varianten	98
9.24 Analyse des EMD Algorihtmus	99
9.25 Analyse des SSA Algorihtmus	99
9.26 Verteilung der originalen Daten mit den synthetischen Daten des SSA Algorihtmus	99
9.27 Übereinstimmung des SSA Algorihtmus mit den originalen Daten	99
9.28 Forcast der jeweiligen rekursiven Modelle	100
9.29 Vektordistanz zwischen den Lösungen und dem synthetischen Daten, Trainingsdaten: corona.csv	101
9.30 Vektordistanz zwischen den Lösungen und dem synthetischen Daten, Trainingsdaten: sinus.csv	101

10.1	Grafische Darstellung der Ergebnisse des NASA-TLX und SUS-Fragebogens in Abhängigkeit des Alters und der technischen Begabung	106
10.2	Auswertung des NASA-TLX Fragebogens	107
10.3	Auswertung des SUS Fragebogens	107

Tabellenverzeichnis

9.1	Autokorrelation der Datensätze	86
9.2	Trainingszeiten der TSA Algorithmen	90
A.1	Teilnehmerbewertungen: Der SUS-Score sowie der TLX-Score reichen bis 100. Ein hoher SUS-Score und ein niedriger TLX-Score sind erstrebenswert. Die technische Erfahrung wurde auf einer Skala von 0 (kein technisches Verständnis) bis 9 (sehr gutes technisches Verständnis) bewertet.	i

Codeverzeichnis

7.1	Abstrakte Basisklasse der DataSets	38
7.2	Basisklasse der DTO Objekte	39
7.3	Mapper Klasse für DataSets, basierend auf den dataType, ein separates enum, fügen die Datentype eigenen mapper ihre Felder hinzu, bevor die global gültigen Felder gefüllt werden.	40
7.4	Neuer Umgang mit JSONB-Elementen in Java	42
7.5	Error Handling für den Fall, dass Ids nicht gefunden werden	43
7.6	Annotation um einen Advice Annotation zu signalisieren	45
7.7	Annotation basiertes Anbinden eines Nutzermodelles an eine Methode	51
7.8	Websocket Konsummer	52
7.9	Exception Handler	52
7.10	Algorithmus zum Auffüllen von Lücken	54
7.11	Grundklasse zum Trainieren der Machine Learning Modelle, sie stellt Grundmethoden und geteilte Funktionalitäten bereit	56
8.1	Methode um Statistiken aus den generierten Daten zu erstellen	76
8.2	Training von Modellen im Testframework	77

g

Liste an Formeln

Literatur

- [14] Spring Boot 1.0 GA Released. <https://spring.io/blog/2014/04/01/spring-boot-1-0-ga-released>. (Accessed on 11/08/2023). Apr. 2014 (siehe S. 29).
- [Ago20] John Mark Agosta. The PCA Trick with Time-Series. PCA can be used to reject cyclic... | by John Mark Agosta | Towards Data Science. <https://towardsdatascience.com/the-pca-trick-with-time-series-d40d48c69d28>. (Accessed on 01/06/2024). Dez. 2020 (siehe S. 79).
- [ASS20] A H Aziira, N A Setiawan und I Soesanti. "Generation of Synthetic Continuous Numerical Data Using Generative Adversarial Networks". In: *Journal of Physics: Conference Series* 1577.1 (Juli 2020), S. 012027. DOI: 10.1088/1742-6596/1577/1/012027. URL: <https://dx.doi.org/10.1088/1742-6596/1577/1/012027> (siehe S. 4).
- [Aut] Prometheus Authors. Overview | Prometheus. <https://prometheus.io/docs/introduction/overview/>. (Accessed on 11/09/2023) (siehe S. 32).
- [Beh+19] R. Behjati, E. Arisholm, M. Bedregal und C. Tan. "Synthetic Test Data Generation Using Recurrent Neural Networks: A Position Paper". In: 2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE). Los Alamitos, CA, USA: IEEE Computer Society, Mai 2019, S. 22–27. DOI: 10.1109/RAISE.2019.00012. URL: <https://doi.ieeecomputersociety.org/10.1109/RAISE.2019.00012> (siehe S. 4).
- [BON+14] PIETRO BONIZZI, JOËL M. H. KAREL, OLIVIER MESTE und RALF L. M. PEETERS. "Singular Spectrum Decomposition: a New Method for Time Series Decomposition". In: *Advances in Adaptive Data Analysis* 06.04 (2014), S. 1450011. DOI: 10.1142/S1793536914500113. eprint: <https://doi.org/10.1142/S1793536914500113>. URL: <https://doi.org/10.1142/S1793536914500113> (siehe S. 6).
- [Bro95] John Brooke. "SUS: A quick and dirty usability scale". In: *Usability Eval. Ind.* 189 (Nov. 1995) (siehe S. 103).
- [DDM15] Hamid Dadkhahi, Marco F. Duarte und Benjamin Marlin. "Isomap out-of-sample extension for noisy time series data". In: 2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP). 2015, S. 1–6. DOI: 10.1109/MLSP.2015.7324314 (siehe S. 79).
- [DJJ03] Huang Daji, Zhao Jinping und Su Jilan. "Practical implementation of Hilbert-Huang transform algorithm". In: ACTA OCEANOLOGICA SINICA-ENGLISH EDITION- 22.1(2003), S. 1–14 (siehe S. 6).
- [Eva04] Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004 (siehe S. 36).
- [Fow04] Martin Fowler. *Fail Fast*. <https://www.martinfowler.com/ieeeSoftware/failFast.pdf>. (Accessed on 11/23/2023). Sep. 2004 (siehe S. 43).

- [GBC16] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016 (siehe S. 12).
- [Geh20] Dr. Jan-Philip Gehrcke. *covid-19-germany-gae/cases-rki-by-state.csv at master · jgehrcke/covid-19-germany-gae*. <https://github.com/jgehrcke/covid-19-germany-gae/blob/master/cases-rki-by-state.csv>. (Accessed on 01/10/2024). 2020 (siehe S. 85).
- [giv20] non given. *long_term_forecast – Google Drive*. https://drive.google.com/drive/folders/1vE00NyqPlym2JaaAoEe0XNDR8FS_d322. (Accessed on 10/18/2023). 2020 (siehe S. 86).
- [GMB21] Rohan Gupta, Satya Mudigonda und Pallav Kumar Baruah. “TGANs with Machine Learning Models in Automobile Insurance Fraud Detection and Comparative Study with Other Data Imbalance Techniques”. In: *International Journal of Recent Technology and Engineering* 9 (Feb. 2021), S. 236–244. DOI: [10.35940/ijrte.E5277.019521](https://doi.org/10.35940/ijrte.E5277.019521) (siehe S. 13).
- [GNZ01] Nina Golyandina, Vladimir Nekrutkin und Anatoly Zhigljavsky. *Analysis of Time Series Structure: SSA and Related Techniques*. Bd. 90. Jan. 2001. DOI: [10.1201/9781420035841](https://doi.org/10.1201/9781420035841) (siehe S. 11).
- [Guj+22] Shubham Gujar, Tanishka Shah, Dewen Honawale, Vedant Bhosale, Faizan Khan, Devika Verma und Rakesh Ranjan. “GenEthos: A Synthetic Data Generation System With Bias Detection And Mitigation”. In: *2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS)*. 2022, S. 1–6. DOI: [10.1109/IC3SIS54991.2022.9885653](https://doi.org/10.1109/IC3SIS54991.2022.9885653) (siehe S. 3).
- [Hem16] Wanja A. Hemmerich. *StatistikGuru*. Autokorrelation. Mai 2016. URL: <https://statistikguru.de/lexikon/autokorrelation.html> (besucht am 14. 01. 2024) (siehe S. 73).
- [HH22] Raymond Ho und Kevin Hung. “Empirical Mode Decomposition Method Based on Cardinal Spline and its Application on Electroencephalogram Decomposition”. In: *2022 IEEE 12th Symposium on Computer Applications and Industrial Electronics (ISCAIE)*. 2022, S. 17–21. DOI: [10.1109/ISCAIE54458.2022.9794540](https://doi.org/10.1109/ISCAIE54458.2022.9794540) (siehe S. 10).
- [HS88] Sandra G. Hart und Lowell E. Staveland. “Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research”. In: *Human Mental Workload*. Hrsg. von Peter A. Hancock und Najmedin Meshkati. Bd. 52. *Advances in Psychology*. North-Holland, 1988, S. 139–183. DOI: [https://doi.org/10.1016/S0166-4115\(08\)62386-9](https://doi.org/10.1016/S0166-4115(08)62386-9). URL: <https://www.sciencedirect.com/science/article/pii/S0166411508623869> (siehe S. 103).
- [Hua+98] Norden E. Huang, Zheng Shen, Steven R. Long, Manli C. Wu, Hsing H. Shih, Quanan Zheng, Nai-Chyuan Yen, Chi Chao Tung und Henry H. Liu. “The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis”. In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 454.1971 (1998), S. 903–995. DOI: [10.1098/rspa.1998.0193](https://doi.org/10.1098/rspa.1998.0193). eprint: <https://royalsocietypublishing.org/doi/pdf/10.1098/rspa.1998.0193>. URL: <https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1998.0193> (siehe S. 5).
- [Koh04] Jason Kohles. *Faker | Faker*. <https://fakerjs.dev/>. (Accessed on 06/25/2023). 2004 (siehe S. 3 f.).

- [Kot+21] Akash Kothare, Shridhara Chaube, Yash Moharir, Gaurav Bajodia und Snehlata Dongre. "SynGen: Synthetic Data Generation". In: 2021 International Conference on Computational Intelligence and Computing Applications (ICCICA). 2021, S. 1–4. DOI: 10.1109/ICCICA52458.2021.9697232 (siehe S. 3).
- [Kri23] Friedhelm Kring. Die Grundsätze der Dialoggestaltung nach ISO 9241-110. <https://www.weka-manager-ce.de/betriebsanleitung/grundsaetze-dialoggestaltung-iso-9241-110/>. 28. April 2023 (siehe S. 22).
- [Kur17] Will Kurt. Kullback-Leibler Divergence Explained – Count Bayesie. <https://www.countbayesie.com/blog/2017/5/9/kullback-leibler-divergence-explained>. (Accessed on 12/22/2023). Mai 2017 (siehe S. 74).
- [LRL22] Guoyao Li, Shahbaz Rezaei und Xin Liu. User-Level Membership Inference Attack against Metric Embedding Learning. 2022. arXiv: 2203.02077 [cs.LG] (siehe S. 76).
- [Nay19] Manish Nayak. An Introduction To Conditional GANs (CGANs) | by Manish Nayak | DataDrivenInvestor. <https://medium.datadriveninvestor.com/an-introduction-to-conditional-gans-cgans-727d1f5bb011>. Mai 2019 (siehe S. 13).
- [NIK23] Alexander Nikitin, Letizia Iannucci und Samuel Kaski. "TSGM: A Flexible Framework for Generative Modeling of Synthetic Time Series". In: *arXiv preprint arXiv:2305.11567* (2023) (siehe S. 76, 80, 87).
- [PBC21] Clément Pealat, Guillaume Bouleux und Vincent Cheutet. "Improved Time-Series Clustering with UMAP dimension reduction method". In: 2020 25th International Conference on Pattern Recognition (ICPR). 2021, S. 5658–5665. DOI: 10.1109/ICPR48806.2021.9412261 (siehe S. 79).
- [Rah23] Moklesur Rahman. Understanding Wasserstein Distance: A Powerful Metric in Machine Learning | by Moklesur Rahman | Medium. <https://rmoklesur.medium.com/understanding-wasserstein-distance-a-powerful-metric-in-machine-learning-100a1ff46b66>. (Accessed on 12/21/2023). Juni 2023 (siehe S. 73).
- [Ric13] Toniann Pitassi Richard Zemel Yu (Ledell) Wu. Kevin Swersky. Learning Fair Representations. <https://www.cs.toronto.edu/~toni/Papers/icml-final.pdf>. (Accessed on 07/07/2023). 2013 (siehe S. 3).
- [Rik22] Teo Yu Siang Rikke Friis Damm. Personas – A Simple Introduction | IxDF. <https://www.interaction-design.org/literature/article/personas-why-and-how-you-should-use-them>. (Accessed on 07/08/2023). 2022 (siehe S. 15).
- [Rob18] George Athanasopoulos Rob J Hyndman. 6.1 Time series components | Forecasting: Principles and Practice (2nd ed). <https://otexts.com/fpp2/components.html>. (Accessed on 06/25/2023). Mai 2018 (siehe S. 9).
- [San20] Natsuki Sano. "Synthetic Data by Principal Component Analysis". In: 2020 International Conference on Data Mining Workshops (ICDMW). 2020, S. 101–105. DOI: 10.1109/ICDMW51313.2020.900023 (siehe S. 6).
- [Shn+16] Ben Schneiderman, Catherine Plaisant, Maxine Cohen, Steven Jacobs, Niklas Elmquist und Nicholas Diakopoulos. Designing the User Interface: Strategies for Effective Human-Computer Interaction. 6th. Pearson, 2016 (siehe S. 103).

- [Sze11] Richard Szeliski. *Computer vision algorithms and applications*. 2011. URL: <http://dx.doi.org/10.1007/978-1-84882-935-0> (siehe S. 74, 78).
- [Tea20] TechAhead Team. *The History of React Native: Facebook's App Development ...* <https://www.techaheadcorp.com/blog/history-of-react-native/>. (Accessed on 11/08/2023). Sep. 2020 (siehe S. 29).
- [WC19] Kwan Yeung Wong und Fu-lai Chung. "Visualizing Time Series Data with Temporal Matching Based t-SNE". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. 2019, S. 1–8. DOI: [10.1109/IJCNN.2019.8851847](https://doi.org/10.1109/IJCNN.2019.8851847) (siehe S. 79).
- [Whi23] Brennan Whitfield. *Principal Component Analysis (PCA) Explained | Built In*. <https://builtin.com/data-science/step-step-explanation-principal-component-analysis>. (Accessed on 07/07/2023). März 2023 (siehe S. 3).
- [Zee23] Amos Zeeberg. *Neural Networks Need Data to Learn. Even If It's Fake.* | Quanta Magazine. <https://www.quantamagazine.org/neural-networks-need-data-to-learn-even-if-its-fake-20230616/>. (Accessed on 08/25/2023). Juni 2023 (siehe S. 1).
- [Zür23] Universität Zürich. *Mann-Whitney-U-Test | Methodenberatung | UZH*. https://www.methodenberatung.uzh.ch/de/datenanalyse_spss/unterschiede/zentral/mann.html. (Accessed on 07/12/2023). 2023 (siehe S. 5).