

Sommersemester 2021

„Erstellen einer Datenverarbeitungsarchitektur zum Erstellen und Verwenden von Modellen Maschinellen Lernens im Qualitätsmanagement.“

Thesis
zur Erlangung des Grades
Bachelor of Science
im Studiengang Wirtschaftsinformatik
an der Fakultät Wirtschaftsinformatik
der Hochschule Furtwangen University

vorgelegt von

Lars Grespan

Erstgutachter:
Zweitgutachter:
Eingereicht am:

Prof. Dr. Ulf Schreier
Prof. Dr. Holger Ziekow
10. June 2021

Eidesstattliche Erklärung

Ich, Lars Grespan erkläre hiermit an Eides statt, dass ich die vorliegende Bachelorthesis selbstständig und ohne unzulässige fremde Hilfe angefertigt habe.

Die verwendeten Quellen sind vollständig zitiert.

Furtwangen, den 10. June 2021

A handwritten signature in black ink, appearing to read "Lars Grespan", is placed over a horizontal line. The signature is fluid and cursive, with a prominent loop on the left and a straight line extending to the right.

Abstract (deutsch)

Die vorliegende Forschungsarbeit beschäftigt sich mit der Erstellung einer Datenverarbeitungsarchitektur im Qualitätsmanagementbereich. Dabei sollen mit Hilfe von Modellen Maschinellen Lernens Fehlervorhersagen von Produkten in einer Produktionsumgebung gemacht werden. Input für die Modelle Maschinellen Lernens sind dabei Daten, die an verschiedenen Qualitätsprüfungsstationen innerhalb der Produktion generiert werden. Für die Erstellung der Architektur muss zum einen die Erstellung der Modelle als auch die Verwendung der Modelle in Form der Fehlervorhersagen inkludiert sein. Sowohl die Erstellung der Modelle als auch deren Verwendung durch die zu erstellenden Fehlervorhersagen benötigen eine vorangeschaltete Datenverarbeitung. Zentraler Forschungsgegenstand dieser Thesis ist dabei die Entwicklung einer Architektur, die eine einheitliche Datenverarbeitung sowohl für die Modellerstellung als auch die Modellverwendung realisiert.

Abstract (english)

This research work deals with the creation of a data processing architecture in the field of quality management. Machine learning models are used to predict product defects in a production environment. The input for the machine learning models is data generated at various quality testing stations within the production process. For the creation of the architecture, the creation of the models as well as the actual usage of the models in form of defect predictions must be included. Both the creation of the models and their usage by the error predictions require upstream data processing. The central research subject of this paper is the development of an architecture that implements a unified data processing for both the model creation and the model usage.

Inhaltsverzeichnis

<i>Abstract (deutsch)</i>	II
<i>Abstract (english)</i>	III
<i>Abbildungsverzeichnis</i>	VI
<i>Tabellenverzeichnis</i>	VII
1 Einleitung	1
1.1 Problem- und Fragestellung	1
1.2 Zielsetzung und Abgrenzung	2
1.3 Aufbau der Arbeit	3
2 Forschungsprojekt PREFERML	4
2.1 Projektbeschreibung	4
2.1.1 Maschinelles Lernen im Qualitätsmanagement	5
2.1.2 Probleme und Herausforderungen bei Datenanalysen im Produktionsumfeld	7
2.2 Anforderungsanalyse	15
2.2.1 Ziel des Prototyps	15
2.2.2 Systemfunktionalität.....	16
2.2.3 Use-Case Übersicht.....	16
2.2.4 Nichtfunktionale Anforderungen.....	25
2.2.5 Lieferumfang	26
3 Grundlagen	27
3.1 Data Processing.....	27
3.1.1 Big Data	27
3.1.2 Data Processing vs. Big Data Processing.....	29
3.1.3 Big Data Schichten.....	30
3.1.4 Big Data Technologien.....	32
3.1.5 Big Data Processing.....	36
3.2 Big Data Processing Architekturen	38
3.2.1 Lambda Architektur.....	38
3.2.2 Kappa Architektur.....	40
3.3 Maschinelles Lernen	40
3.3.1 Definition	41
3.3.2 Supervised Learning und Unsupervised Learning	41
3.3.3 Speichern und Laden von ML-Modellen	42
4 Implementierung des Prototyps	44
4.1 Aufbau der Daten und der Produktionsumgebung	45
4.2 System Context Diagram	46
4.2.1 REST Implementation	46
4.2.2 Change Streams Implementation	48
4.3 Event Generation	49
4.4 Model Training	50
4.5 Event Processing and Data Model	52
4.5.1 Batch Schicht	53
4.5.2 Speed Schicht	55
4.6 Simulate Test Stations	56
4.6.1 Simulate Teststations (REST).....	56
4.6.2 Simulate Test Stations (Change Streams).....	57

4.7	Data Processing Server	58
4.7.1	Webservice (REST)	58
4.7.2	Webservice (Change Streams) & Change Streams Service.....	58
4.8	Vergleich der Implementierungen.....	60
4.8.1	Performancevergleich	60
4.8.2	Qualitativer Vergleich	62
5	Limitationen und Resümee	63
5.1	Resümee und Bewertung der Forschungsfragen	63
5.2	Stärken und Limitationen der Arbeit.....	64
5.3	Ausblick auf zukünftige Arbeiten.....	65
Glossar.....		VIII
Abkürzungsverzeichnis.....		X
Annex.....		XI
Literaturverzeichnis.....		XII

Abbildungsverzeichnis

Abbildung 1 - Big Data Schichten	30
Abbildung 2 - Kafka Architektur	34
Abbildung 3 - MapReduce Funktion.....	35
Abbildung 4 - Spark Bibliotheken.....	36
Abbildung 5 - Big Data Processing Architektur	37
Abbildung 6 - Lambda Architektur	39
Abbildung 7 - Lambda Architektur und entsprechende Tools	39
Abbildung 8 - Kappa Architektur.....	40
Abbildung 9 - Event Processing System	46
Abbildung 10 - REST Implementation	47
Abbildung 11 - Change Streams Implementation	48
Abbildung 12 - Event Generation	49
Abbildung 13 - Model training.....	51
Abbildung 14 - Event Processing.....	52
Abbildung 15 - Simulate Test Stations (REST)	57
Abbildung 16 - Simulate Test Stations (Change Streams).....	57
Abbildung 17 - Processing Server.....	58
Abbildung 18 - Change Streams Service	59
Abbildung 19 - Result Presentation.....	59

Tabellenverzeichnis

Tabelle 1 - Erstellen Datensatz Teststation 1	17
Tabelle 2 - Erstellen Datensatz Teststation 2	17
Tabelle 3 - Erstellen Datensatz Teststation 3	18
Tabelle 4 - Speichern der Datensätze	18
Tabelle 5 - Simulieren von Live-Events.....	19
Tabelle 6 - Erstellen des Data Models.....	20
Tabelle 7 - Bereinigung der Events.....	20
Tabelle 8 - Erstellung abgeleiteter Attribute	20
Tabelle 9 - Speichern der bearbeiteten Events	21
Tabelle 10 - Verbinden von Events.....	21
Tabelle 11 - Laden der Events.....	22
Tabelle 12 - Laden der Label	22
Tabelle 13 - Trainieren der ML-Modelle	23
Tabelle 14 - Speichern der ML-Modelle	23
Tabelle 15 - Erstellen einer Service Schnittstelle.....	24
Tabelle 16 - Laden der ML-Modelle	24
Tabelle 17 - Erstellen einer Vorhersage	24
Tabelle 18 - Vorhersage speichern.....	25
Tabelle 19 - Ergebnisse anzeigen.....	25
Tabelle 20 – Traditional Data vs. Big Data.....	29
Tabelle 21 - Performancevergleich REST - Change Streams	61

1 Einleitung

Der Mensch durchlief in der Evolution verschiedene Entwicklungsstadien. Die Entwicklungsstadien des Menschen werden unter verschiedenen Homoarten festgehalten. Zu diesen Homoarten gehören beispielsweise der Homo Sapiens (Mensch, der Denken kann), der Homo Erectus (Mensch, der aufrecht geht), der Homo Ludens (Mensch, der Zeit mit Spaß verbringt) oder der Homo Faber (Mensch, der Werkzeuge herstellt) [1].

Letzterer ist Ursache für technologischen Fortschritt. Der Mensch ist perfekt darin neue Werkzeuge zu erschaffen, oder ein bestehendes Werkzeug für einen neuen Verwendungszweck anzupassen [2]. Die Inventionen, die vom Menschen im Laufe der Geschichte entwickelt wurden, sind unzählig. Viele Erfindungen sind dabei von kleiner Bedeutung und einige Erfindungen hatten das Potenzial die Welt zu verändern – zum Beispiel die Erfindung des Rads oder der Buchdruck.

Neben vielen anderen Bereichen hat der technologische Fortschritt auch Einfluss auf die Industrie. Der Industriezweig ist laut Lasi et. al. [3] jener Wirtschaftsbereich, welcher für die Erstellung physischer Produkte durch Mechanisierung und Automatisierung verantwortlich ist. Durch technologische Durchbrüche erfolgten in der Industrie immer wieder große Umstrukturierungen. Diese Umstrukturierungen wurden im nachhinein als „Revolutionen“ bezeichnet [3]. Die erste industrielle Revolution entspringt aus der Erfindung der Dampfmaschine, durch welche die Mechanisierung vorangetrieben wurde. Die zweite industrielle Revolution bezieht sich auf die Nutzung elektrischer Energie. Mit der Entwicklung des Internets, fanden entsprechende Technologien Anwendung in der Industrie. In diesem Kontext verweist Lasi et. al. [3] auf die Digitalisierung der Unternehmen. Diese wird von den Autoren auch als die dritte industrielle Revolution bezeichnet. Durch die Digitalisierung der Industrie bilden sich Anwendungspotenziale moderner Zukunftstechnologien wie Künstliche Intelligenz (KI), Internet of Things (IoT), etc. Die Zusammenführung der Digitalisierung mit den verschiedenen Zukunftstechnologien kann zu einem erneuten Paradigmenwechsel führen, in wessen Kontext der Begriff Industrie 4.0 (die vierte industrielle Revolution) verwendet wird [3]. Technologischer Fortschritt gilt daher als Treiber der vierten industriellen Revolution.

Der breitgefächerte Industriezweig bietet für die Anwendung Zukunftstechnologien viele Ansatzmöglichkeiten. Begriffe die in diesem Kontext verwendet werden sind „Smart Manufacturing“ oder „Smart Factory“. Diese Begriffe stehen exemplarisch für den Paradigmenwechsel in der Industrie und bilden die Ausgangslage für die Forschungsfragen, auf welchen die vorliegende Arbeit aufgebaut ist.

1.1 Problem- und Fragestellung

Smart Manufacturing ist eine sich entwickelnde Form der Produktion, die Produktionsanlagen von heute und morgen mit Sensoren, Computerplattformen, Kommunikationstechnologie, Steuerung, Simulation, datenintensiver Modellierung und Vorhersagetechnik integriert. Sie nutzt die Konzepte der cyber-physischen Systeme, die durch das Internet der Dinge, Cloud Computing, serviceorientiertes Computing, künstliche Intelligenz und Data Science vorangetrieben werden [4].

Resultierend aus der intelligenten Produktion ergeben sich für das Qualitätsmanagement diverse Optimierungspotenziale. Eines dieser Optimierungspotenziale ist das Verwenden von Machine Learning (ML) Modellen zur Fehlervorhersage in der Produktion. Fehlerhafte Teile in der Produktion können dadurch frühzeitig aussortiert werden, was sich positiv auf die Kosten auswirkt.

Voraussetzung für das Verwenden von ML-Modellen in der Produktion sind dabei untereinander vernetzte Objekte innerhalb der Fertigung. Im Qualitätsmanagement sind diese Objekte verschiedene Stationen, welche Qualitätsprüfungen an den Produkten durchführen. Die aufkommenden Daten werden dann jeweils von den ML-Modellen zur Fehlervorhersage benutzt.

Zur Umsetzung der Fehlervorhersage mittels ML-Modellen sind zum einen die Erstellung der ML-Modelle notwendig und zum anderen die Verwendung der ML-Modelle. Sowohl für die Erstellung als auch für die Verwendung der ML-Modelle müssen die Daten der Qualitätsprüfungsstationen auf verschiedene Weisen bearbeitet werden. Mitunter kann die Datenverarbeitung in Abhängigkeit der Produktionsumgebung sehr umfangreich sein. Um die Komplexität zu reduzieren bietet es sich an, die Datenverarbeitung für beide Fälle (ML-Modell Erstellung und ML-Verwendung) eine einheitliche Lösung anzubieten.

In dieser Forschungsarbeit wird untersucht, wie eine Datenverarbeitung angeboten werden kann, welche für die Erstellung von ML-Modellen und die Fehlervorhersage durch die ML-Modelle verwendet werden kann. Gleichzeitig wird geprüft, wie eine Architektur zur Umsetzung einer einheitlichen Datenverarbeitung für die ML-Modell Erstellung und die ML-Modell Verwendung aussehen kann. Zusätzlich müssen heutige Datenverarbeitungssysteme eine immer größer werdende Anzahl an Daten prozessieren. Um dieser Problematik entgegenzuwirken werden zur Implementierung einer solchen Datenverarbeitungsarchitektur verschiedene Verarbeitungstools gegenübergestellt und geprüft. Aus der oben genannten Problembeschreibung ergeben sich für diese Forschungsarbeit deshalb folgende Forschungsfragen:

RQ1:

Inwieweit lässt sich eine einheitliche Datenverarbeitung in Form von gleichem Source Code für die Erstellung der ML-Modelle als auch für die Verwendung der ML-Modelle zur Fehlervorhersage erstellen?

RQ2:

Wie kann eine Datenverarbeitungsarchitektur aussehen, bei der für die Erstellung der ML-Modelle dieselbe Datenverarbeitung durchlaufen wird, wie für die Verwendung der ML-Modelle zur Fehlervorhersage?

RQ3:

Ist eine solche Architektur umsetzbar?

RQ4:

Welche Technologien eignen sich hinsichtlich der Performance für eine solche Datenverarbeitungsarchitektur am besten?

1.2 Zielsetzung und Abgrenzung

Das Ziel dieser Forschungsarbeit ist es, eine Datenverarbeitung anzubieten, welches für die ML-Modell Erstellung und für die Fehlervorhersage durch die ML-Verwendung eingesetzt werden kann. Im Rahmen der einheitlichen Datenverarbeitung wird eine entsprechende Architektur unter Berücksichtig der Forschungsfragen erstellt und implementiert. Die Forschungsarbeit dient daher zum Validieren der Frage, ob eine solche Architektur erstellt werden kann. Zur Überprüfung der

Frage wird eine simulierte Produktionsumgebung erstellt, in der die Datenverarbeitungsarchitektur implementiert werden kann.

Die erstellte Architektur kann als Basis für den Einsatz unter realen Produktionsumgebungen dienen, Forschungsgegenstand dieser Forschungsarbeit ist jedoch die Umsetzbarkeit einer solchen Architektur und nicht die Validierung der Frage, ob die erstellte Architektur unter realen Produktionsbedingungen zum Einsatz kommen kann. Zur Überprüfung dieser Forschungsfrage, müsste zum einen eine reale Produktionsumgebung dem Autor zur Verfügung stehen als auch Rechenressourcen die den Bedarf einer echten Produktionsumgebung gerecht werden.

1.3 Aufbau der Arbeit

Durch den Schwerpunkt des Forschungsgegenstandes ist die vorliegende Forschungsarbeit neben der Einleitung in vier verschiedene Abschnitte gegliedert: Forschungsprojekt PREFERML, Grundlagen, Implementierung sowie Limitationen und Resümee.

Die Forschungsarbeit ist auf der Grundlage eines weiteren Forschungsprojektes aufgebaut. Dieses Forschungsprojekt wird unter dem Akronym PREFERML aufgeführt. Um auf den Forschungen dieses Forschungsprojektes anzuschließen, wird im Forschungsprojekt PREFERML Kapitel der Kontext dazugegeben. Daraus abgeleitet wird eine Anforderungsanalyse zur Erstellung einer Datenverarbeitungsarchitektur durchgeführt. Dabei liegt der Schwerpunkt auf die Einhaltung der Forschungsfragen.

Nachdem thematikfernen Lesern ein Überblick über das Forschungsthema gegeben wurde, werden die Grundlagen im Bereich der Datenverarbeitung erläutert. Durch die Aktualität der Forschungsarbeit wird im Grundlagenkapitel auf State-of-the-Art Datenverarbeitungstechnologien und -architekturen verwiesen. Diese bilden das Fundament auf welchem die Implementierung aufbaut.

Folgend auf das Grundlagenkapitel wird in Abschnitt vier die Implementierung anhand von Architekturbildern beschrieben. Um auf die Forschungsfrage RQ4 eine aussagekräftige Antwort zu geben werden zusätzlich verschiedene Implementierungsvarianten gegenübergestellt und geprüft.

Der letzte Abschnitt – Limitationen und Resümee – dient der Zusammenfassung der erlangten Erkenntnisse und deren Bewertung im Hinblick auf die in der Einleitung definierten Forschungsfragen. Außerdem werden die Limitation der Forschungsarbeit dargelegt sowie der Ausblick auf anknüpfende Forschungsarbeiten.

2 Forschungsprojekt PREFERML

Mit dem zunehmenden Einsatz von Künstlicher Intelligenz in der Wirtschaft, laufen synchron dazu die Erfolge, welche die Technologie mit sich bringt. Beispielsweise stieg laut einer Publikation des Bundesministeriums für Wirtschaft und Energie (BMWi) die Umsatzrendite im Jahr 2018 bei Unternehmen die KI verwendeten um 1,3% an [5]. Betrachtet man den globalen KI-Markt, so steigt auch dort der Einsatz der Technologie. Seit 2014 erhöhen sich die Investitionen in KI-Unternehmen jährlich um ca. 55%. Vorreiter in diesem Gebiet sind überwiegend China und die USA [6]. Vor allem in einer Produktionsumgebung sind die Einsatzgebiete für diese Technologie vielseitig. Beispiele für den Einsatz in der Produktion sind Instandhaltung, Logistik, Qualitätskontrolle, Prozessentwicklung, Prozessoptimierung, Automatisierung, usw. [7].

Schlüssel zum Erstellen erfolgreicher KI-Modelle sind an erster Stelle die Daten. Sind nur wenig oder nicht aussagekräftige Daten verfügbar, so lassen sich nur schlecht Muster in den Daten erkennen. Daraus folgt, dass das Erstellen und die Adaption von ML-Modellen nicht sinnvoll [8]. Durch die zunehmende Digitalisierung der Unternehmen und deren Geschäftsprozesse ist die Problematik mit dem Vorhandensein von Daten heutzutage in den Hintergrund geraten. Daten werden innerhalb von Geschäftsprozessen generiert und auf Datenbanken abgelegt. Laut dem Index für die digitale Wirtschaft und Gesellschaft (DESI) liegt der Grad der Digitalisierung in Deutschland bei 56,1% [9]. Fokus liegt daher zunehmend auf dem sinnvollen Einbinden von den generierten KI-Modellen in die bereits vorhandenen Geschäftsprozesse [8].

Im Rahmen dieser Arbeit liegt das Einsatzgebiet von KI in der Prozessoptimierung innerhalb einer Produktionsumgebung. Ausgangslage dieser Thesis ist ein Forschungsprojekt der Hochschule Furtwangen University, welches im folgenden Kapitel beschrieben wird.

2.1 Projektbeschreibung

Zusammen mit dem Sensorproduktionsunternehmen Sick AG arbeitet die Hochschule Furtwangen University gemeinsam an einem Forschungsprojekt. Titel des Forschungsprojektes ist „Proaktive Fehlervermeidung in der Produktion durch Maschinelles Lernen“ (PREFERML). Das Forschungsprojekt wird im Rahmen der Förderlinie FHprofUnt vom Bundesministerium für Bildung und Forschung (BMBF) gefördert (Förderkennzeichen: 13FH249PX6). Das Ziel dieser Studie ist es, herauszufinden, ob maschinelles Lernen bei der Qualitätskontrolle in der Fertigung sinnvoll eingesetzt werden kann. Mit Hilfe von künstlicher Intelligenz wird untersucht, wie Ausgabedaten aus der Fertigung analysiert werden können, um Fehlerprognosen zu erstellen und Hinweise auf die Ursachen von Defekten zu geben [10].

Im Hinblick auf das Forschungsprojekt PREFERML handelt es sich deshalb um eine Prozessoptimierung im Produktionsumfeld durch Daten, welche aus dem Qualitätsmanagement stammen. Angesammelte Daten aus dem Qualitätsmanagement, in denen sich ggf. Muster erkennen lassen, sind daher die Grundlage für die Fehlerprognosen. Durch die daraus resultierenden Einsichten kann der Prozess daraufhin optimiert werden. Konkret bedeutet dies:

- Aussortieren oder Nachbearbeiten von Teilen durch das frühzeitige Erkennen von Mängeln
- Ursachenbehebung von Defekten durch das Erkennen der Fehlerquelle

Zentrale Bestandteile und Fragen, mit denen sich das Forschungsteam auseinandersetzt, sind laut dem Forschungsbericht 2020/2021 der Hochschule Furtwangen University wie folgt:

- 1) „Nur wenn beispielsweise Abläufe, Produkteigenschaften und Zusammenhänge in der Produktion bekannt sind, kann ein Algorithmus nützliche Ergebnisse liefern. Wie kann gewährleistet werden, dass ein maschinelles Modell den Kontext bzw. die Problemstellung richtig erfasst?“ [10]
- 2) „Ein Lösungskonzept muss in der Lage sein, viele Daten für unterschiedliche Problemfälle ohne große Anpassung und manuellen Aufwand zu verarbeiten. Wie lässt sich solch ein automatisierter Ablauf in der Praxis umsetzen?“ [10]
- 3) „Die späteren User der Systeme sind in der Regel im Qualitätsmanagement aktiv. Wie kann ein Konzept entwickelt werden, das den dauerhaften, nützlichen Betrieb erlaubt, ohne dass KI-Fachleute aktive am Prozess beteiligt sind?“ [10]

Die erste und die letzte Fragestellung werden in anderen Arbeiten und Veröffentlichungen thematisiert. Konkret geht es dort um das Einbinden von Domain Knowledge in die ML Algorithmen und die Erklärbarkeit der Ergebnisse (Explainable AI). Weiterführende Literatur ist hierzu [11] [12] [13] [14] [15].

Fokus der vorliegenden Arbeit ist das Erstellen einer Architektur (im folgenden auch Prototyp genannt), welche die verschiedenen Aspekte der 2. Fragestellung berücksichtigt. Im Rahmen der bereits absolvierten Forschung, sind eine große Anzahl von ML-Modellen entstanden, welche nun in eine Automatisierungsumgebung eingebunden werden. Zum Erstellen eines solchen Prototyps gilt es verschiedene Problematiken zu berücksichtigen, welche in dem folgenden Kapitel genauer erläutert werden. Abgeleitet von den verschiedenen Problematiken wird anschließend eine Anforderungsanalyse durchgeführt, welche die Grundlage für die Implementierung des Prototyps ist.

Wichtig ist zu beachten, dass es sich bei dieser Arbeit um einen Prototyp handelt, der an das Szenario des Forschungsprojektes angelehnt ist. Die Daten und das Produktionsumfeld werden in diesem Kontext deshalb simuliert. Um den Umfang dieser Arbeit einzuschränken, werden die Daten und das Produktionsumfeld nur zum kleinen Teil modelliert und abgebildet. Für eine Einbindung in das im Forschungsprojekt vorliegenden Produktionsumfeld, müssen Produktionsspezifische Merkmale berücksichtigt werden, was bedeutet, dass der Prototyp angepasst und erweitert werden muss.

2.1.1 Maschinelles Lernen im Qualitätsmanagement

Durch Prozessoptimierungen in der Produktion innerhalb eines Unternehmens, werden deren Produkte hinsichtlich Qualität stetig verbessert. Nichtsdestotrotz bedeutet das nicht, dass keine Fehler bzw. Abweichungen auftreten können. Selbst in einem nahezu perfekten Produktionsprozess treten Fehler und Abweichungen in den produzierten Teilen auf. Ziel des Qualitätsmanagements ist es, diese Fehler zu minimieren und somit die Qualität der Produkte zu gewährleisten. Die Qualitätssicherung wird mit Hilfe von Qualitätsprüfungen am Ende des Produktionsprozesses durchgeführt. Moderne Produktionsumgebungen führen Qualitätsprüfungen nicht mehr nur am Ende des Produktionsprozesses durch, sondern auch zwischen einzelnen Fertigungsschritten. Mangelhafte Teile können frühzeitig aussortiert oder nachjustiert werden. Diese Art der Qualitätsprüfung wird von Peter Hehenberger in „Qualitätsmanagement in der Produktion“ als die klassische, produktorientierte Qualitätsprüfung beschrieben [16]. Bei dieser Qualitätsprüfung werden „qualitätsrelevante Merkmale“ [16] gemessen. Messmittel sind zumeist Sensoren. Durch die Verarbeitung der Messdaten und die Integration in unternehmensinterne Informationssysteme können die Messdaten noch weiterbearbeitet werden [16].

Durch die von Peter Hohenberger angeführte klassische Qualitätsprüfung können produzierte Teile bei Abweichungen vom Soll-Wert bereits im Fertigungsprozess aussortiert werden. Gegebenenfalls können diese Teile nachbearbeitet werden und wieder in den Fertigungsprozess eingeschleust werden. Dadurch wird zum einen der Ausschuss reduziert und zum anderen die Qualität der Produkte erhöht. Infolgedessen werden die Kosten gesenkt und die Kundenzufriedenheit erhöht [16].

In mehrstufigen Produktionslinien, bei denen zwischen einzelnen Fertigungsschritten immer wieder Qualitätsprüfungen durchgeführt werden, wird ein mangelhaftes Teil genau an dieser Stelle aus dem Prozess entfernt, an dem die Abweichung gemessen wurde. Unter der Prämisse, dass diese Abweichung als Folge von vorherigen Fertigungsschritten resultiert, kann dieses Teil gegebenenfalls noch früher aus dem Produktionsprozess aussortiert und, nach Bedarf, nachbearbeitet werden. Beispielsweise gibt es in einem Produktionsprozess für ein Teil genau drei Fertigungsschritte. Nach jedem Fertigungsschritt wird eine Qualitätsmessung auf verschiedene Qualitätsmerkmale durchgeführt. Die erste Qualitätsmessung misst das Teil auf die Länge, die Zweite auf die Breite und die Dritte auf die Höhe. Das Teil, welches den Produktionsprozess durchläuft, gelangt nur dann zum nächsten Fertigungsschritt, wenn die Qualitätsmessung innerhalb des Toleranzbereiches liegt. Tritt ein Fehler an der dritten Qualitätsmessung auf, so kann anhand der Daten überprüft werden, ob ein Zusammenhang zu den vorherigen Qualitätsmessungen besteht. Zum Beispiel tritt an der dritten Qualitätsmessung immer dann eine Abweichung auf, wenn die erste Qualitätsmessung im oberen Toleranzbereich liegt oder die Zweite im unteren Toleranzbereich. Ebenfalls kann eine Kombination von bestimmten Messwerten der ersten Qualitätsmessung und der zweiten Qualitätsmessung Ursache für die Abweichung in der dritten Qualitätsmessung sein. Falls solche Zusammenhänge erkannt werden, können potentiell fehlerhafte Teile schon vor dritten Qualitätsmessung identifiziert und aussortiert bzw. nachbearbeitet werden. In einer Fließbandproduktion ist das Erkennen von solchen Zusammenhängen durch manuelle Datenauswertung schlichtweg nicht effizient. Zusätzlich ist es für einen Menschen nicht einfach komplexe Muster in einem Datensatz zu erkennen. An dieser Stelle können nun, durch zunehmende Rechenleistung der Informationssysteme, moderne Data-Science Methoden zum Einsatz kommen [14].

Mit dem Einsatz von ML-Modellen können komplexe Zusammenhänge in den Daten schnell erkannt werden. Durch die gewonnenen Einblicke kann die Qualität der produzierten Teile zusätzlich gesteigert werden und der Ausschuss weiter reduziert werden. Zudem wird vermieden, dass einzelne Fertigungsschritte mehrfach für ein Teil durchgeführt werden muss. Ein mehrmaliges Durchführen desselben Produktionsschrittes führt zu Kosten, welche ebenfalls eingespart werden können [17].

Zur Realisierung dienen die gemessenen Merkmale bei den Qualitätsprüfungen als Input für das ML-Modell. Der Output aus dem ML-Modell gibt an, wie hoch die Wahrscheinlichkeit ist, dass das entsprechende Produkt fehlerhaft ist und deshalb an einer späteren Qualitätsprüfung aussortiert werden würde. Damit in den Fertigungsprozess eingegriffen werden kann, muss diese Information zusätzlich noch an das Qualitätsmanagement übermittelt.

In einer echten Produktion gibt es allerdings nicht nur eine Fertigungsline mit drei Qualitätsmessungen, sondern viele Fertigungslien mit jeweils mehreren Qualitätsmessungen. Zum Einbinden von ML muss nach jeder Qualitätsmessung eine Vorhersage (ob das Produkt aussortiert wird oder nicht) gemacht werden. Bei den unterschiedlichen Qualitätsmessungen werden unterschiedliche qualitätsspezifische Merkmale gemessen. Des Weiteren gibt es sowohl verschiedene Produkte, die in den Fertigungslien produziert werden, als auch verschiedene

Fehlerarten, welche bei den Qualitätsmessungen auftreten können. Durch die Varianz in den Produkten und den Messungen entsteht zusätzliche Komplexität [15]. Insgesamt sammelt sich eine Vielzahl von Problemen und Herausforderungen bei Datenanalysen in Fertigungsarbeiten an, welche beim Implementieren des Prototyps wichtig sind. Diese Probleme und Herausforderungen werden im nächsten Kapitel behandelt.

2.1.2 Probleme und Herausforderungen bei Datenanalysen im Produktionsumfeld

Bevor die Anforderungsanalyse, an den zu erstellenden Prototypen, durchgeführt wird, gilt es noch einige wichtige Probleme zu adressieren und berücksichtigen. Im Rahmen des Forschungsprojektes PREFERML wurden bereits wichtige Herausforderungen im Bereich der Datenanalytik herauserarbeitet. Diese referenzieren sich auf den Bericht “Data Science Approaches to Quality Control in Manufacturing: A Review of Problems, Challenges and Architecture” [14]. Der Bericht bezieht sich auf typische Schwierigkeiten bei datengesteuerten Ansätzen zur Qualitätskontrolle in Fertigungsumgebungen und deckt bereits ein breites Spektrum an verschiedenen Quellen ab. Die in diesem Bericht beschriebenen Komplexitäten bilden die Grundlage für die Erstellung des Prototyps. Die Herausforderungen werden demzufolge im Prototyp berücksichtigt.

Grundsätzlich gibt es vier verschiedene Bereiche, in welche sich datenanalysebeeinflussende Faktoren in der Fertigungsumgebung kategorisieren lassen [14]:

- Fertigungsdomäne
- Merkmale der zugrunde liegenden Daten und Datenquellen
- Eigenschaften der Analysen
- Anwendung und deren Schnittstellen

Diese vier Kategorien werden in den nächsten Unterkapiteln erklärt und die beeinflussenden Faktoren beschrieben. Die Schwerpunkte des Berichtes sind dabei ausschließlich Anforderungen im fachlichen und funktionalen Bereich. Nichtfunktionale Anforderungen wie Sicherheit und Performance werden nicht behandelt. Da die Performance Grundlage einer der Forschungsfragen ist, werden in einem fünften Unterkapitel performancebeeinflussende Faktoren beschrieben, welche bei der Implementierung des Prototyps ebenfalls zu berücksichtigen sind.

2.1.2.1 Herausforderungen der Fertigungsdomäne

Diese Domäne umfasst Probleme und Anliegen der Datenanalyse, die sich aus der Konzeption der Fertigungsumgebung und den üblichen domänenspezifischen Merkmalen ergeben. Folgende Herausforderungen werden im Rahmen dieser Kategorie beschrieben:

- Heterogene Produktfamilien und Betriebsmittel
- Unterschiedliche Fehlerarten
- Unausgewogene Datenverteilungen
- Nichtlineare Produktionsprozesse und Selbstschleifen
- Konzeptdrift
- Kostensensitive Modellierung

Heterogene Produktfamilien und Betriebsmittel

Fertigungsumgebungen sind häufig durch mehrere Produktvarianten und unterschiedliche Versionen von Betriebsmitteln gekennzeichnet, z. B. Maschinen, Werkzeuge oder

Werkstattausrüstung. Die genannten Varianten und Versionen der Produkte und Betriebsmittel unterscheiden sich zum einen in ihren Zusammensetzungen und zum anderen in ihren physikalischen Eigenschaften. Mit zunehmender Vielfalt an Produkten und Betriebsmittel steigt synchron dazu die korrespondierenden Daten. Daraus ergeben sich komplexe Abhängigkeiten, eine hohe Anzahl von Dimensionen und eine schwere Interpretierbarkeit der Daten. So unterscheiden sich z. B. die Wertebereiche, die ein bestimmtes Klassenmuster repräsentieren, oft zwischen einzelnen Produktvarianten oder in verschiedenen Versionen von Betriebsmitteln. Dies macht es ML-Modellen schwer, klar unterscheidbare Muster zu erkennen [14] [18] [19] [20] [21].

Unterschiedliche Fehlertypen

ML-Modelle können entweder durch binäre Klassifikation oder durch Multi-Label Klassifikation erstellt werden. Die Grundlagen zum Trainieren von ML-Modellen werden in [Kapitel 3 – Grundlagen](#) genauer erläutert. Grundsätzlich können die Label bei der binären Klassifikation zwei Werte annehmen. Im Rahmen einer Fehleranalyse eines produzierten Produkts kann dies zum Beispiel „OK“ oder „NOT OK“ sein (auch „0“ oder „1“). Im Gegensatz dazu können bei der Multi-Label Klassifikation die Label mehr als nur zwei Werte annehmen. Im Produktionsumfeld können diese Label für unterschiedliche Fehlertypen stehen. Durch die oben genannten heterogenen Produktfamilien und Betriebsmittel, treten durch die Produktvielfalt eine hohe Anzahl von verschiedenen Fehlertypen auf. Eine hohe Anzahl von verschiedenen Fehlertypen erhöht die Komplexität des zu lösenden Klassifikationsproblems und kann sich negativ auf die Performance der ML Modelle auswirken [14] [21] [22] [23].

Unausgewogene Datenverteilung

Im Produktionsumfeld sind die Produktionsdaten üblicherweise nicht ausgewogen verteilt. In Bezug auf die binäre Klassifikation weisen die meisten Datenpunkte das Label „OK“ bzw. „0“ auf. Nur wenige Teile weisen Fehler auf und haben deshalb das Label „NOT OK“ bzw. „1“. Nicht nur bei der binären Klassifikation können die Daten unausgewogen verteilt sein, sondern auch bei der Multi-Label Klassifikation sind die Daten häufig unterschiedlich verteilt. Während einige Klassen eine hohe Anzahl von Datenpunkten aufweisen können, kann es auch Klassen geben, bei denen nur ein paar wenige Datenpunkte vorhanden sind. Durch die heterogenen Produktfamilien und Betriebsmittel können diese Klassen zusätzlich noch über diese Produktvielfalt ungleichmäßig verteilt sind. So kann es vorkommen, dass die Klassen oder Produktvarianten unterrepräsentiert sind, was sich negativ auf die ML Performance auswirken kann [14] [21] [22] [20] [23] [19] [24].

Nichtlineare Produktionsprozesse und Selbstschleifen

In einer typischen Produktionslinie, wie in Kapitel 2.1.3 Maschinelles Lernen im Qualitätsmanagement beschrieben, durchläuft ein Produkt verschiedene Produktionsschritte. Nach den einzelnen Produktionsschritten werden jeweils Qualitätsprüfungen durchgeführt [16] [14]. Im Idealfall durchläuft ein Teil die Produktionslinie, ohne dabei bei den Qualitätsprüfungen Abweichungen aufzuweisen. In der Datenerfassung einer solchen Produktionslinie spiegelt sich diese lineare Bearbeitungsreihenfolge wider, d.h. die Prozess- und Prüfdaten eines Produkts oder Werkstücks erscheinen genau einmal im Datensatz. Unter Realbedingungen treten bei der Qualitätsprüfung Abweichungen auf, was dazu führen kann, dass das entsprechende Teil aus der Produktionslinie entfernt wird. Die vorhergesehene Reihenfolge wird unter diesem Aspekt nicht mehr berücksichtigt, weshalb der Produktionsprozess nicht-linear wird. Üblicherweise werden die fehlerhaften Teile in vorherige Produktionsschritte, oder sogar in den gleichen Produktionsschritt, wieder eingesetzt. In diesem Kontext wird dann von Schleifen oder Selbstschleifen gesprochen. In Bezug

auf die Produktionsdaten bedeutet ein Wiedereinsetzen von Teilen an vorherigen Produktionsstufen - oder am gleichen Produktionsschritt – ein Wiederauftreten des entsprechenden Teils in den Daten. Der Datensatz enthält also dieselben Teile, am gleichen Produktionsschritt mit unterschiedlichen Zeitstempeln [14] [25].

Konzeptdrift

ML Modelle werden darauf trainiert Muster in Daten zu erkennen - unabhängig davon, ob alle Produktvarianten und Fehlertypen in den Daten ausreichend vertreten sind. Input für das Training der ML Modelle ist ein Datensatz bestehend aus Features mit den zugehörigen Label. Genaue Details zum Trainieren von ML Modelle werden in Kapitel 3 erklärt. Die Aufgabe des Trainings von ML-Modellen ist das Erlernen einer Beschreibung eines Konzepts, die alle in den Trainingsmustern festgestellten relevanten Muster repräsentiert. Die Informationen über das im Datensatz gelernte Konzept werden im ML-Modell festgehalten. Die Schwachstelle des gelerten Konzepts ist der Datensatz selbst. Wenn die Daten in der Produktion variieren oder sich im Laufe der Zeit ändern, dann wird das gelernte Konzept des ML-Modells diesen Änderungen nicht gerecht. D.h. die Konzepte können in ihren Bedeutungen verschoben werden. Diese Verschiebungen werden als Konzeptdrift bezeichnet. Fertigungsumgebungen sowie die zugrundeliegenden Daten sind im Wesentlichen nicht stationär und weisen wechselnde und unvorhersehbare Umstände auf, so dass ein Konzeptdrift normal ist. So können sich beispielsweise die mechanischen Eigenschaften von Maschinen oder Werkzeugen, die in einzelnen Prozessphasen eingesetzt werden, im Laufe der Zeit ändern. Dadurch können sich auch die Schwellenwerte der Sensorsignale ändern, die angeben, ob eine Ware ein Qualitätskontrolltor passieren kann oder nicht. Um auf Konzeptdrifts zu reagieren wird ein System benötigt, welches wechselnde Umstände wahrnimmt und die ML-Modelle an diese Umstände anpasst. Wichtig dabei ist, dass das System zwischen Rauschen (Noise) und wechselnden Umständen unterscheiden kann. Um möglichst redundantes Anpassen von Modellen zu vermeiden, ist es zusätzlich sinnvoll, dass das System wiederkehrende Konzeptdrifts erkennt. Erst wenn ein solches System diese genannten Anforderungen erfüllt kann eine hohe Genauigkeit der ML-Modelle gewährleistet werden [14] [26] [27] [25].

Kostensensitive Modellierung

Damit ML-Modelle im Qualitätsmanagement eingesetzt werden, müssen diese Modelle versprechen einen Mehrwert zu schaffen. In der Qualitätskontrolle ist der Mehrwert der ML-Modelle die Kosteneinsparung. Bei der kostensensitiven Modellierung von ML-Modellen werden die Kosten für eine Fehlklassifikation berechnet. Mit dem Wissen über die Kosten einer Fehlklassifikation kann nun eine Mindestanforderung für die Genauigkeit der ML-Modelle vorgegeben werden. Das heißt, dass das Modell so lange trainiert und getestet wird, bis eine bestimmte Genauigkeit bei den Vorhersagen erzielt wird. Bei der kostensensitiven Modellierung von ML-Modellen besteht die Schwierigkeit dabei, die Kosten für eine Fehlklassifikation zu bestimmen. Zusätzlich muss noch zwischen verschiedenen Fehlklassifikationen bei der Berechnung der Kosten differenziert werden. Im Rahmen einer Fehleranalyse wäre eine negative Klassifikation eine bestandene Qualitätsprüfung und eine positive Klassifikation eine nicht bestandene Qualitätsprüfung. Bei beiden Klassifikationen können Fehler auftreten. In diesem Fall wird dann von Falsch-Positiven und Falsch-Negativen Klassifikationen gesprochen. Eine Falsch-Positive Klassifikation bedeutet, dass ein Teil so klassifiziert wird, dass es die Qualitätskontrolle nicht besteht, obwohl das Teil keine Qualitätsprobleme aufweist. Falsch-Negative Klassifikationen führen dazu, dass eigentlich positive Teile als negative Teile klassifiziert werden. Im Schlimmsten Fall wird das Teil vom Kunden wieder zurückgeschickt. Die anfallenden Kosten durch die eigentlich nicht nötigen

Reparaturen von Falsch-Positiven Teilen sind grundsätzlich deutlich geringer als die Kosten für Falsch-Negative Teile [14] [21] [28] [29] [30].

2.1.2.2 Herausforderungen zu den Merkmalen der zugrunde liegenden Daten und Datenquellen

In diesem Kapitel werden die Probleme und Herausforderungen beschrieben, die sich aus den Merkmalen der zugrunde liegenden Daten und Datenquellen ergeben. Bei der Implementierung des Prototyps wird einigen hier genannten Problemen und Herausforderungen eine geringere Priorität zugemessen, da durch die Simulation der Produktionsumgebung die Herausforderungen nicht abgebildet werden können. Dennoch treten diese Komplexitäten in einer echten Produktionsumgebung auf und sind deshalb nicht zu vernachlässigen. Zu den Herausforderungen und Problemen gehören:

- Komplexe Datenfusion aus multimodalen Datenquellen
- Fehlende Primary Key und Foreign Key Beziehungen
- Nicht synchronisierte Zeitstempel über die Produktionsschritte hinweg
- Doppelte und ungültige Datenpunkte
- Fehlende Beschreibungen und Erklärungen der Daten
- Unzureichende Datenerfassung
- Nicht rückführbare Datenschemaänderungen

Komplexe Datenfusion und multimodale Datenquellen

Jede Produktionsumgebung besteht typischerweise aus verschiedenen Arten von Maschinen und Montageeinheiten. Abgeleitet daraus gibt es jeweils verschiedene Messtechniken oder Sensoren, die die Teile kontrollieren und die entsprechenden Daten aufnehmen. Die Kombination aus Datenquelle und Messdatenerfassung werden laut dem Bericht „Multimodal Data Fusion: An Overview of Methods, Challenges and Prospects“ von Lahat et al. [31] als Modalität bezeichnet. Zur Differenzierung muss jede Modalität eine Information liefern, welche von anderen Modalitäten nicht berücksichtigt wird. Um auf alle verfügbaren Informationen der Modalitäten zugreifen zu können, müssten diese zu einem großen Informationsdatensatz integriert werden. Oftmals weisen die Modalitäten eine große Heterogenität auf, was zu Schwierigkeiten bei der Integration der Informationen führen kann. Mittels Datenfusionstechniken können diesen Hindernissen entgegengewirkt werden [14] [31] [25] [20] [32].

Fehlende Primary Key und Foreign Key Beziehungen

Bei der Datenerfassung über verschiedene Prozessstufen in Fertigungsumgebungen muss sicher gestellt werden, dass die gesammelten Daten eindeutig und genau den relevanten Elementen zugeordnet werden. Dies erfordert eine eindeutige und spezielle Primary Key und Foreign Key Beziehung zwischen einer Ware und allen verschiedenen Informationen, die während der verschiedenen Prozessstufen erfasst werden. Vielen Datenstrukturen und Datenformaten, die in realen Produktionsanwendungen verwendet werden, fehlen dagegen bestimmte spezielle Primary Key und Foreign Key Beziehungen [14] [21] [20].

Nicht synchronisierte Zeitstempel über die Produktionsschritte hinweg

Viele Montagelinien haben eine große Anzahl von Verfahrens- und Prüfschritten, die in einer bestimmten Reihenfolge durchlaufen werden müssen. Diese Reihenfolge von Verfahrens- und Prüfschritten müssen, besonders bei Echtzeit-Auswertungen, berücksichtigt werden. Viele der

Datenanalysen benötigen diese zeitliche Abfolge als Anforderung für deren Umsetzung. Eine Lösung zum Synchronisieren bietet das Network Time Protocol (NTP). Falls dieses Protokoll nicht unterstützt wird, müssen andere Methoden zum Zeitabgleich verwendet werden [14].

Doppelte und ungültige Datenpunkte

Wie bereits in Kapitel 2.1.2.1 – Herausforderungen der Fertigungsdomäne können Produkte aufgrund von nicht-linearen Fertigungsverläufen und durch Selbstschleifen in Form von gleichen Datenpunkten mehrmals im Datensatz auftreten. Zusätzlich können während des Datenaufnahmeprozesses Fehler entstehen, die zu invaliden Datenpunkten führen. Beispielsweise können gestörte Sensoren bei der Qualitätskontrolle falsche Daten übermitteln. Beide Fälle – doppelte als auch ungültige Datenpunkte – müssen vor dem Trainieren der ML-Modelle entfernt werden [14] [21] [20].

Fehlende Beschreibungen und Erklärungen der Daten

Datenwissenschaftler müssen vor der Verwendung eines Algorithmus für Maschinelles Lernen zum Trainieren eines Modells zunächst bestimmen, welche Variablen oder Funktionen in einem Datensatz für das Lernen der entsprechenden Zieldefinition wichtig sind. Dazu benötigen die Datenwissenschaftler ein Vorverständnis in Bezug auf die Bedeutung von Variablen und ihren Beziehungen. In Form von Datenschemata und Datenbeschreibungen kann ein solches Verständnis kodiert werden. In vielen realen Anwendungsfällen sind allerdings solche Datenschemata und Datenbeschreibungen nur selten oder nur teilweise zugänglich [14] [21].

Unzureichende Datenerfassung

Fertigungsprozesse folgen typischerweise zeitkontinuierlichen physikalischen Vorgängen. Dementsprechend nehmen die zur Qualitätsprüfung herangezogenen Messmittel Informationen über die Zeitpunkte auf. Die Aufnahme der Zeit durch entsprechende Signale sind an dieser kritisch zu betrachten, da die Signale mit einer bestimmten Frequenz diskretisiert werden. Dies kann dazu führen, dass der Informationsgehalt sinkt. Sind die Darstellungen durch eine niedrige Frequenz verdichtet, kann es im schlimmsten Fall dazu führen, dass keine informationsreichen Merkmale abgeleitet werden können [14] [21].

Nicht rückführbare Datenschemaänderungen

Üblicherweise kommt es bei den Messmitteln zur Qualitätsprüfung oder am Fertigungsprozess selbst über einen längeren Zeitraum hinweg zu Updates oder Änderungen. Beispielsweise wird ein Sensor durch eine neue Version ersetzt. Bei der neuen Version des Sensors werden zusätzliche Attribute bei der Datenaufnahme erfasst. Dies hat zur Folge, dass zugrunde liegende Datenbankschema angepasst werden muss. Eine Möglichkeit zur Behebung des Problems ist eine schemalose Datenbank. Allerdings müssten die Applikationen, welche auf die Datenbank zugreifen, selbst interpretieren um welche Datentypen es sich bei den Attributen handelt [14].

2.1.2.3 Herausforderungen zu den Analysen

Durch den Einsatz von ML-Modellen in einem Produktionsumfeld ergeben sich Probleme, welche in den folgenden Abschnitten genauer erläutert werden, und welche bei der Implementierung des Prototyps berücksichtigt werden müssen. Zu den Herausforderungen gehören:

- Fehlendes Vertrauen in die ML-Modelle und den Analyseergebnissen
- Rückverfolgung der Analyseergebnisse und Hauptursachen
- Verwaltung von verschiedenen ML-Modellen
- Feature Engineering, Auswahl von Modellen und Parametern

Fehlendes Vertrauen in die ML-Modelle und den Analyseergebnissen

ML Modelle, wie z. B. ein Entscheidungsbaum, stellen abstrakte Repräsentationen der zugrunde liegenden gelehrteten Konzepte dar (Genaueres im [Kapitel 3 – Grundlagen](#)). Problematisch an den ML-Modellen ist, dass es für Nicht-Datenwissenschaftler nicht nachvollziehbar ist, wie die Modelle erstellt wurden. Diese Problematik kann die Ursache für eine Skepsis gegenüber den Vorhersagen der ML-Modelle sein. Wie bereits in [Kapitel 2.1.2.1 Herausforderungen in der Fertigungsdomäne](#) unter Abschnitt „Kostensensitive Modellierung“ beschrieben, und auch von Wilhelm et. al. [14] erwähnt, ist es wichtig zu wissen, ob es sich um Falsch-Positive oder Falsch-Negative Vorhersagen handelt. Diese genannten Fehlertypen haben direkten Einfluss auf die Kosten. In diesem Sinne ist die Interpretierbarkeit solcher Vorhersagen für Nicht-Datenwissenschaftler enorm wichtig. Durch die Skepsis gegenüber den ML-Modellen und der schwierigen Interpretierbarkeit der Vorhersagen, kann die Implementierung der ML-Modelle bei den Endbenutzern auf Widerstände treffen [14].

Rückverfolgung der Analyseergebnisse und Hauptursachen

Neben der schweren Interpretierbarkeit der Vorhersagen der ML-Modelle, ist es ebenso schwierig die Bedeutung der Analyseergebnisse zurückzuführen. Das heißt, dass die Zurückführung eines Fehlers auf das entsprechende Teil an der entsprechenden Stelle im Produktionsprozess mitunter auf Hindernisse stoßen kann. Grundsätzlich sind bei einer positiven Vorhersage – also ein fehlerhaftes Teil – zwei Informationen interessant. Zum einen die Rückführung des Auftretens der Hauptursache auf einen bestimmten Produktionsschritt und zum anderen die Identifizierung der Ursache innerhalb des Teils. Für die ersten Information können Standard-Merkmalbedeutungsmaße oder neuere Methoden wie SHAP-Werte (SHapley Additive exPlanations) weiterführend sein. Weiterführende Literatur innerhalb des Forschungsprojektes PREFERML zum Thema SHAP-Werte ist der Bericht von Ziekow et al. [11]. Die Gewinnung der Information zur Identifizierung der Hauptursache hängt davon ab, wie das ML-Modell trainiert wurde. Bei binär trainierten ML-Modellen sind nur zwei Möglichkeiten möglich. Entweder tritt ein Fehler auf („NOT OK“ bzw. „1“) oder es tritt kein Fehler auf („OK“ bzw. „0“). Die Information über den spezifischen Fehler lässt sich aus der binären Klassifikation nicht gewinnen. Bei Multi-Label-Modellen kann es hingegen für jeden binären Fehlertyp eine eigene Klasse geben. Dadurch lassen sich nach der Vorhersage des ML-Modells Rückschlüsse auf die Ursache des Fehlers ziehen [14] [33] [34] [21].

Verwaltung von verschiedenen ML-Modellen

In den vorhergehenden Abschnitten und Unterkapiteln wurden bereits Herausforderungen genannt, die ML-Modelle bewältigen müssen, um in der Praxis Anwendung zu finden. Durch die in [Kapitel 2.1.2.1 Herausforderungen der Fertigungsdomäne](#) beschriebenen Probleme über die heterogenen Produktfamilien und Betriebsmittel, als auch über die unterschiedlichen Fehlertypen, ergeben sich eine vielfältige Anzahl an ML-Modellen, die in der Praxis zu implementieren sind. Zusätzlich unterscheiden sich die ML-Modelle in Typ (Neuronales Netz (NN), Entscheidungsbaum, usw.), Input-Features, Klassenlabel, Auswertungen und Datenquellen. Resultierend aus der großen Anzahl von ML-Modellen und deren Ausprägungen ergibt sich ein Verwaltungsproblem, welches zum Einbinden der Modelle in der Praxis gelöst werden muss. Ein erster Ansatz

zur Lösung des Verwaltungsproblems ist in Form einer Datenbank, welche die Modelle verwaltet, bereits erarbeitet worden (ModelDB) [35]. Für eine Verwendung in der Fertigungsumgebung ist dieser Ansatz allerdings noch nicht ausgereift genug, da zu wenig Fälle abgedeckt werden. Eine weitere Methode zur Lösung des Verwaltungsproblems ist Verknüpfung von Metadaten der ML-Modelle in Verbindung mit Informationen über die Produkte, Fertigung als auch die Untersuchungsziele können [14] [36].

Feature Engineering, Auswahl von Modellen und Parametern

Für das Trainieren der ML-Modelle sind oft nicht alle verfügbaren Attribute notwendig. Oft reicht schon ein kleiner Teil der Attribute für die Fehlervorhersage aus. Die Attributauswahl wird im Fachjargon auch als Feature Selection bezeichnet. Zusätzlich können mit Hilfe von mathematischen Methoden Attribute zusammengefasst werden. In diesem Kontext spricht man von Feature Engineering. Sowohl die Feature Selection als auch das Feature Engineering bieten ein breites Spektrum an verfügbaren Methoden. Des Weiteren müssen Fragen bezüglich des geeigneten Modelltyps (NN, Entscheidungsbaum, usw.), des geeigneten ML-Algorithmus und der Hyperparameter für den ML-Algorithmus geklärt werden. Ursprünglich wurde die beste Lösungskombination manuell gesucht und gefunden, was unter Umständen eine zeitaufwendige Aufgabe sein kann. Automated Machine Learning (AutoML) ist eine Methode zur Automatisierung dieses Prozesses, indem maschinelle Lernprogramme erstellt werden, ohne dass ein menschliches Eingreifen erforderlich ist [14].

2.1.2.4 Herausforderungen der Software

Auch im Zusammenhang mit der Software gilt es einige Herausforderungen in Bezug der datengesteuerten Ansätze zur Qualitätskontrolle in Fertigungsumgebungen zu bewältigen. In diesem Kontext müssen dazu folgende Probleme gelöst werden:

- Fehlende moderne Datenzugriffsschnittstellen
- Individuelle Lösungsskripte

Fehlende moderne Datenzugriffsschnittstellen

Gespeicherte Informationen, welche, während dem Produktionsprozess entstehen, werden über Schnittstellen von den Maschinen oder Baugruppen übertragen. Über eine Produktionsumgebung hinweg gibt es eine Vielzahl verschiedener Maschinen und Baugruppen, was zu einem breiten Spektrum von verschiedenen Schnittstellen führt. Durch den Einsatz von modernen Kommunikationslösungen, wie beispielsweise die Open Platform Communication Unified Architecture (OPC UA), können einheitliche Datenzugriffe umgesetzt werden. In der Praxis hingegen sieht die Realität oft anders aus – Viele Unternehmen bieten ihre eigenen proprietären Schnittstellen an. Die Datenübertragung findet über Exportformate statt. Für eine Implementierung im Kontext der datengesteuerten Ansätze zur Qualitätskontrolle ist durch die veraltete Schnittstellentechnik ein Übertragungsaufwand zu bewältigen [14] [37].

Individuelle Lösungsskripte

Datenwissenschaftliche Projekte werden üblicherweise mit einem systematischen Vorgehen angegangen. Eine typische Vorgehensweise in diesem Rahmen ist die Cross-industry standard process for data mining (CRISP-DM) Methodology [38]. Skriptsprachen wie R oder Python bieten eine Vielzahl von Bibliotheken, welche in den einzelnen Phasen der genannten CRISP-DM Methodology zum Einsatz kommen können. Als Antwort auf die hohe Anzahl an Modalitäten werden von vielen Datenwissenschaftlern spezifische Skriptlösungen geschrieben. Dies kann zu

schwer zu pflegenden Quellcodes führen. Zudem macht es die Skriptimplementierungen weniger wiederverwendbar, was zu einem ähnlich hohen Implementierungsaufwand für die nächste Datenwissenschaftler führt. Infolgedessen gewinnen auch Überlegungen zu Software-Engineering und Architektur an Bedeutung. Zur Lösung des Problems kann ein domänenorientiertes Metamodell bzw. Datenmodell beitragen, was bei der Orchestrierung der einzelnen Komponenten hilft. Zudem ist eine weitere Lösungsidee das Verwenden, von dem im vorherigen Kapitel beschrieben AutoML [14].

2.1.2.5 Herausforderungen an die Performance

Durch die einhergehenden Beschreibungen der Herausforderungen in den vorherigen Kapiteln ergibt sich eine Komplexität von verschiedenen Fällen, welche für eine Automatisierung abgedeckt werden muss. Zusätzlich entstehen in einem modernem Produktionsumfeld, durch die Modalitäten, eine extrem hohe Anzahl an Daten [39]. Je größer die Anzahl der Modalitäten, desto größer die Anzahl der Daten. Um das Automatisierungsproblem mit der damit verbundenen hohen Datenmenge zu lösen, ist eine Datenverarbeitung notwendig. Um jedoch eine gewisse Zuverlässigkeit in der Datenverarbeitung zu gewährleisten, reichen herkömmliche Datenverarbeitungsmethoden bei einer solchen Menge an Daten nicht aus [39]. Werden Produkte von den ML-Modellen als mangelhaft deklariert, so muss das entsprechende Produkt sofort aus dem Fertigungsprozess aussortiert werden. Wird das Produkt nicht sofort aussortiert fallen durch unnötig ausgeführte Fertigungsschritte entsprechende Kosten an der nachfolgenden Fertigungsstationen innerhalb einer Fertigungsline. In diesem Kontext ist deshalb eine Datenauswertung mit geringer Latenzzeit notwendig. Für die Datenverarbeitung in Echtzeit gibt es bereits verschiedene Tools und Technologien, welche dieser Herausforderungen mit entsprechenden Funktionalitäten entgegenwirken. Zu diesen Technologien und Tools zählt das Big Data Framework Apache Spark, Datenuntersuchungsplattformen wie Hadoop oder MapReduce, Datenbanken wie Cassandra und MongoDB, sowie Messaging Dienste wie Kafka Streams [40]. Um die Performance der Automatisierungslösung zu steigern können deshalb unterschiedliche Maßnahmen getroffen werden. Zu diesen gehören:

- Minimierung und Optimierung der Datenbankzugriffe
- Parallelisierung der Datenverarbeitung
- Verwendung performanter Bibliotheken

Minimierung und Optimierung der Datenbankzugriffe

Innerhalb der Datenverarbeitung werden die Daten unterschiedlich oft geladen, bearbeitet, gespeichert, und miteinander verbunden. Dies inkludiert mehrere Datenbankzugriffe. Durch die hohe Anzahl der Daten können die Datenbankzugriffe zu einem Performanceproblem führen. Mit jeder Datenbankabfrage steigt die Latenzzeit im Gesamtsystem. Vorzugsweise können Daten durch sogenannte Sessions, nach einmaligen Laden, gehalten werden. Dadurch können einige Datenbankzugriffe vermieden werden. Für eine weitere Reduzierung der Latenzzeit durch die Datenbankzugriffe spielt die verwendete Datenbank eine Rolle. Je nach Anwendungsfall und Datentypen variiert die Performance der Datenbanktypen stark [41]. Die Auswahl der passenden Datenbank kann also dazu beitragen die Performance des Gesamtsystems zu verbessern. Zusätzlich können die Zugriffe mit Optimierungsmethoden performanter, wie beispielsweise Indexe, gestaltet werden [42].

Parallelisierung der Datenverarbeitung

Eine weitere Möglichkeit zur Optimierung der Performance ist eine Verteilung und Parallelisierung der Datenverarbeitung [43]. Eine Parallelisierung der Datenverarbeitung bedeutet einerseits eine deutlich verbesserte Performance, andererseits ist mit der Parallelisierung eine Verteilung der Datenverarbeitung auf verschiedene Prozessoren notwendig. Um diese Verteilung konsistent zu halten sind gewisse Verwaltungsaufgaben erforderlich, was zu weiterer Komplexität und weiteren Herausforderungen führt [44].

Verwendung performanter Bibliotheken

Wie im Abschnitt „Individuelle Lösungsskripte“ im vorherigen Kapitel erwähnt, werden im Datenwissenschaftlichen Bereich typischerweise Skriptsprachen wie Python oder R verwendet. Bei der Verwendung dieser Skriptsprachen gibt es eine Reihe von Open-Source Bibliotheken, welche das Manipulieren der Daten stark vereinfacht. In Python sind diese Bibliotheken beispielsweise, pandas oder numpy. Allerdings vereinfacht diese Bibliotheken nicht nur die Handhabung der Daten, sondern sie sind auch auf Performance ausgelegt. So kann die Verwendung solcher Bibliotheken dazu führen, dass die Verarbeitungsgeschwindigkeit von mehreren tausend Zeilen Daten drastisch sinkt. Deshalb ist es sinnvoll, diese Bibliotheken zu verwenden, und nicht nur auf die eingebundenen Funktionalitäten der Skriptsprache zurückzugreifen [45].

2.2 Anforderungsanalyse

Basierend auf der einhergehenden Problembeschreibung wird im Rahmen dieser Abschlussarbeit ein Prototyp entwickelt, welcher die beschriebenen Herausforderungen adressiert und entsprechende Lösungsansätze liefert. Zentraler Bestandteil ist dabei die Datenverarbeitung. (Im folgenden auch Event Processing genannt. Ein Event ist mit dem Datenpunkt einer Qualitätsprüfung gleichzusetzen. Die Durchführungen von Qualitätsprüfungen innerhalb der Produktion werden unter dem Begriff Teststation weitergeführt.) Die Vorhersagen der ML-Modelle sollen anschließend auf einer Web-Oberfläche angezeigt werden. Somit sollen mangelhafte Teile aus den Produktionslinien frühzeitig erkannt und aussortiert werden. Voraussetzung hierfür ist die Verarbeitung der Daten der Qualitätsprüfungen mit geringer Verzögerungszeit bzw. nahe Echtzeit, da eine gewisse Latenz aufgrund der Übertragung über ein Netzwerk nicht vermeidbar ist.

Zusätzlich müssen zu einer sinnvollen Erstellung des Prototyps verschiedene Komponenten zur Simulation des Produktionsumfeldes generiert werden. Diese Simulationen der Produktionskomponenten werden ebenfalls in der Anforderungsanalyse inkludiert.

Die in den folgenden Kapiteln aufgeführten Anforderungen bilden daher den Rahmen des Prototyps. Ziel ist es diese Anforderungen umzusetzen. Aufgrund multipler Umsetzungsmöglichkeiten des Event Processings sollen zudem verschiedene Ansätze implementiert und auf Performance evaluiert werden.

2.2.1 Ziel des Prototyps

Für die Implementierung einer Lösung der vorherig genannten Anforderungen in eine Echtzeitumgebung, sind hoch performante Methoden zum Verarbeiten der Daten von essentieller Bedeutung. Bei der Implementierung sollen verschiedene Technologien zur Umsetzung der genannten Methoden in Betracht gezogen werden und anschließend verglichen werden. Ziel des Prototyps ist es am Ende der Arbeit eine vollfunktionsfähige Lösung zu haben, welche zusätzlich die Herausforderungen der effizienten Datenverarbeitung mit geeigneten Methoden und Technologien bewältigt.

2.2.2 Systemfunktionalität

Im Folgenden werden die Systemfunktionalitäten der Lösung spezifiziert. Die Funktionalitäten werden, aus Gründen der Übersichtlichkeit und der klaren Abgrenzung, aufgeteilt in folgende Kategorien:

- 1) Anforderungen zur Simulation des Produktionsumfeldes
- 2) Anforderungen an den Prototypen

Die konkrete Umsetzung der Anforderungen wird in [Kapitel 4 – Implementierung des Prototyps](#) aufgeführt. Spezifische Architekturbilder können dort ebenso entnommen werden.

2.2.3 Use-Case Übersicht

Die Anforderungen an die Systemfunktionalität werden in Form von Use-Cases beschrieben. Folgende Use-Cases gilt es in der Implementierung zu berücksichtigen und umzusetzen.

Simulation der Produktionsumgebung:

- Erstellen Datensatz Teststation 1
- Erstellen Datensatz Teststation 2
- Erstellen Datensatz Teststation 3
- Speicher der Datensätze
- Simulieren von Live-Events

Prototyp für die Datenverarbeitungsarchitektur:

- Datenverarbeitung
 - Erstellen des Daten Modells
 - Bereinigung der Events
 - Erstellung abgeleiteter Attribute
 - Speichern der bearbeiteten Events
 - Verbinden von Events
- Batch-Schicht
 - Laden der Events
 - Laden der Labels
 - Trainieren der ML-Modelle
 - Speichern der ML-Modelle
- Speed-Schicht
 - Erstellen einer Service Schnittstelle
 - Laden der ML-Modelle
 - Erstellen einer Vorhersage
 - Vorhersage speichern
 - Ergebnisse anzeigen

2.2.3.1 Simulation des Produktionsumfeldes

In einer realen Produktionsumgebung werden die Daten der Teststationen in eine zentrale Datenbank geschrieben und archiviert. Die Erstellung von historischen Datensätzen pro Teststation muss zur Implementierung eines Prototyps deshalb simuliert werden. Im Kontext von Big Data ist anzunehmen, dass es in einem realen Produktionsumfeld beliebig viele Teststationen geben. Aufgrund begrenzter Ressourcen beschränkt sich der Autor auf die Simulation von drei Teststationen.

Name	Erstellen Datensatz Teststation 1
Kurzbeschreibung:	Zu generieren ist ein .csv-Datensatz. In die Werte der Spalten werden Fehler eingebaut, sodass eine spätere Datenbereinigung notwendig ist.
Struktur	Component_ID: Integer (Unique values) Teststation_ID: Integer (1) Feature_1: Float (Range: 1.2-1.4) Feature_2: Integer (Range: 50-60) Date: Datetime Time: Datetime Category: String (Red, Green, Blue, Yellow) Product_Type: String (PT1, PT2) Error_Message: String (Ok, TemperatureTooHigh, TemperatureTooLow)

Tabelle 1 - Erstellen Datensatz Teststation 1

Name	Erstellen Datensatz Teststation 2
Kurzbeschreibung:	Zu generieren ist ein .csv-Datensatz. Dieser Datensatz baut auf dem Teststation1.csv Datensatz auf. Das heißt, die Component_IDs taucht wieder auf. Des Weiteren, müssen gemeinsame Attribute (wie Product_Type) pro Component_ID über die Datensätze hinweg übereinstimmen. Falls in der Teststation1.csv die Spalte Error_Message den Wert „Ok“ nicht aufweisen kann, so darf die entsprechende Reihe nicht mehr in der zu generierenden Teststation2.csv auftauchen (Entsprechende Teile würden in einer realen Produktionsumgebung aussortiert werden). Da die Teststation2 in der realen Welt die Messungen erst nach der Teststation1 macht, muss dies auch in der Simulation in den Spalten Date und Time zu erkennen sein (Teststation1: Date + Time < Teststation2: Date + Time). In die Werte der Spalten werden Fehler eingebaut, sodass eine spätere Datenbereinigung notwendig ist.
Struktur	Component_ID: Integer (Unique values) Teststation_ID: Integer (2) Feature_3: Float (Range: 90.0-95.6) Feature_4: Integer (Range: 12-13) Date: Datetime Time: Datetime Product_Type: String (PT1, PT2) Error_Message: String (Ok, TemperatureTooHigh, TemperatureTooLow)

Tabelle 2 - Erstellen Datensatz Teststation 2

Name	Erstellen Datensatz Teststation 3
Kurzbeschreibung:	Zu generieren ist ein .csv-Datensatz. Dieser Datensatz baut auf dem Teststation2.csv Datensatz auf. Das heißt, die Component_IDs taucht auch im Teststation3 Datensatz auf. Außerdem müssen gemeinsame Attribute (wie Product_Type) pro Component_ID über die Datensätze hinweg übereinstimmen. Falls in der Teststation2.csv die Spalte Error_Message den Wert „Ok“ nicht aufweisen kann, so darf die entsprechende Reihe nicht mehr in der zu generierenden Teststation3.csv auftauchen (Entsprechende Teile würden in einer realen Produktionsumgebung aussortiert werden). Da die Teststation3 in der realen Welt die Messungen erst nach der Teststation1 macht, muss dies auch in der Simulation in den Spalten Date und Time zu erkennen sein (Teststation2: Date + Time < Teststation3: Date + Time). In die Werte der Spalten werden Fehler eingebaut, sodass eine spätere Datenbereinigung notwendig ist.
Struktur	Component_ID: Integer (Unique values) Teststation_ID: Integer (3) Date: Datetime Time: Datetime Product_Type: String (PT1, PT2) Error_Message: String (Ok, TemperatureTooHigh, TemperatureTooLow)

Tabelle 3 - Erstellen Datensatz Teststation 3

Name	Speichern der Datensätze
Kurzbeschreibung:	Die zuvor erstellten Datensätze Teststation1.csv, Teststation2.csv und Teststation3.csv werden in eine MongoDB namens EventDB geladen. Die einzelnen Datensätze werden jeweils in der gleichen Collection gespeichert.
Ablauf	<ol style="list-style-type: none"> 1. Laden der Datensätze Teststation1.csv, Teststation2.csv und Teststation3.csv 2. Speichern der Datensätze in die EventDB. Pro csv-Datei soll eine andere Collection benutzt werden

Tabelle 4 - Speichern der Datensätze

Name	Simulieren von Live-Events
Kurzbeschreibung:	Nach der Erstellung von historischen Datensätzen müssen zu einer realitätsnahen Simulation de facto ebenfalls „Live-Events“ von den Teststationen gesendet werden. Zur Vereinfachung werden in dieser Simulation allerdings keine neuen Daten erstellt werden, sondern einfach die vorhandenen Datensätze, welche bereits in der Datenbank (EventDB) gespeichert wurden, pro Teststation durchiteriert werden. Für jede Reihe soll ein Service aufgerufen werden, welcher als Trigger für die weitere Verarbeitung der Daten dient.
Ablauf	<ol style="list-style-type: none"> 1. Iterieren der Datensätze Teststation 1/2/3 2. Jede Reihe wird zur Verarbeitung an einen Schnittstellen-Service gesendet

Tabelle 5 - Simulieren von Live-Events

2.2.3.2 Prototyp für die Datenverarbeitungsarchitektur

Nach dem Erstellen der Teststation-Datensätze und durch das Durchiterieren der Datensätze als Simulation zum Senden von Live-Events der Teststationen, ist die Simulation der Produktionsumgebung, im Rahmen dieser Forschungsarbeit, ausreichend nah an einer realen Produktionsumgebung abgebildet. Nachfolgend werden die zu implementierenden Use-Cases beschrieben, welche nun den Prototypen betreffen. Zur Übersichtlichkeit wird zwischen zwei verschiedenen Ebenen differenziert – Batch- und Speed-Schicht. Die Batch-Schicht repräsentiert den Aufgabenbereich für die ML-Modellerstellung und die Speed-Schicht ist für das Laden der erstellten ML-Modelle zum Fehlervorhersagen verantwortlich. Diese Ebenen werden im [Kapitel 3 Grundlagen](#) im Detail erörtert. Außerdem wird die Datenverarbeitung der Events als weiteres Differenzierungsmerkmal, zusätzlich zu den zwei Ebenen (Batch. und Speed-Schicht), eingeführt. Hintergrund hierfür ist, dass die Datenverarbeitung, aus Teil der zu prüfenden Forschungsfrage RQ1, auf der Batch Ebene als auch auf der Speed Ebene gleich sein soll. Die Spezifikation der Datenverarbeitung wird im Folgenden deshalb nur einmal unter **Datenverarbeitung** aufgeführt.

Datenverarbeitung

Die Datenverarbeitung soll auf beiden Ebenen, sowohl der Batch-Schicht als auch der Speed-Schicht, gleich sein. Das bedeutet, dass zwischen einem einzelnen Event zur Fehlervorhersage und einem Datensatz zum ML-Modelltraining nicht differenziert wird. Input für die gesamte Datenverarbeitung ist deshalb entweder ein einzelnes Event von Teststation1/2/3 oder ein ganzer Datensatz von Teststation1/2/3. Output kann je nach Input dann ebenfalls ein einzelnes Event sein oder ein ganzer Datensatz. Zur Vereinfachung wird in den Beschreibungen für ein einzelnes Event oder für einen Datensatz der Begriff **Events** verwendet.

Name	Erstellen des Daten Modelles
Kurzbeschreibung:	Erstellen eines Datenmodells. Das Daten Model dient als Informationsquelle zum Bearbeiten der Events

Ablauf	<ol style="list-style-type: none"> 1. Erstellen einer Drop-Liste 2. Erstellen einer Liste mit allen kategorischen Spaltennamen, auf welche ein OneHot-Encoding ausgeführt werden soll 3. Erstellen eines Dictionary mit den kategorischen Spaltennamen und deren Wertausprägungen
---------------	--

Tabelle 6 - Erstellen des Daten Modelles

Name	Bereinigung der Events
Kurzbeschreibung:	Die Events werden bereinigt
Ablauf	<ol style="list-style-type: none"> 1. Löschen von Reihen mit NAN-Werten 2. Umwandeln von Kommas zu Punkten 3. Anwenden eines REGEX 4. Löschen von Textspalten (oder Spalten die immer den gleichen Wert haben). Informationen über entsprechende Spalten werden aus dem Daten Modell entnommen 5. Zuweisen von verschiedenen Ausprägungen eines Wertes zu einem Wert (OK -> 1; ok ->1)

Tabelle 7 - Bereinigung der Events

Name	Erstellung abgeleiteter Attribute
Kurzbeschreibung:	An dieser Stelle der Datenverarbeitung werden neue Attribute aus den bereits existierenden Attributen erstellt werden.
Ablauf	<ol style="list-style-type: none"> 1. Erstellen einer Spalte „DateTime“ aus den Attributen „Date“ und „Time“ 2. Erstellen einer Prüfsumme aus dem Attribut „Time“ 3. Löschen der Spalten „Date“ und „Time“ 4. Basierend auf der Spalte „Date“ soll ein OneHot-Encoding für die Wochentage angewendet werden. 5. Erstellung einer weiteren Spalte für die Kalenderwoche (Ordinal Encoding) 6. Anwenden eines OneHot-Encoding bei kategorischen Spalten, welche im Date Modell enthalten sind

Tabelle 8 - Erstellung abgeleiteter Attribute

Name	Speichern der bearbeiteten Events
Kurzbeschreibung:	Für den nächsten Use-Case Verbinden von Events werden die bearbeiteten Events zusammengefügt. Damit dies gelingt müssen die Events, welche zusammengefügt werden

	sollen, geladen werden können. Dazu werden die prozessierten Events auf der EventDB gespeichert.
Ablauf	1. Speichern der prozessierten Events auf der EventDB

Tabelle 9 - Speichern der bearbeiteten Events

Name	Verbinden von Events
Kurzbeschreibung:	Ausgehend von welcher Teststation Events stammen, werden Events mit Events von vorherigen Stationen über die „Component_ID“ verbunden. Beim Verbinden der Events werden folgende Spalten der vorherigen Events ausgeschlossen werden: Teststation_ID, Date, Time, Product_Type, Error_Message sowie die abgeleiteten Spalten DateTime, Checksum, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday und Week . Bei Events der Teststation1 ist kein Verbinden der Daten mit vorherigen Stationen möglich, weshalb Events von dieser Teststation von diesem Use-Case ausgeschlossen sind. Ebenfalls ist bei der letzten Teststation (im Rahmen dieser Arbeit Teststation3) kein Verbinden der Events notwendig, da bei der letzten Teststation keine Vorhersage gemacht wird.
Ablauf	<ol style="list-style-type: none"> 1. Überprüfen von welcher Teststation die Events stammen. 2. Laden der Events von vorhergehenden Teststationen 3. Verbinden der Events über die Component_ID unter Ausschluss der oben genannten Spalten

Tabelle 10 - Verbinden von Events

Der Output der Datenverarbeitung kann, basierend auf welcher Teststation das Event stammt, variieren. Die untenstehende Aufstellung zeigt, welche Spalten die möglichen Outputs der Datenverarbeitung haben:

Teststation1:

Component_ID, Teststation_ID, Feature_1, Feature_2, DateTime, Checksum, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday, Week, Green, Yellow, Red, Blue, Product_Type, Error_Message

Teststation2:

Component_ID, Teststation_ID, Feature_3, Feature_4, DateTime, Checksum, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday, Week, Product_Type, Error_Message, Feature_1, Feature_2, Green, Yellow, Red, Blue

Batch Schicht

In der Batch-Schicht wird mit den erstellten Datensätzen der Teststationen 1, 2 und 3 gearbeitet.

Name	Laden der Events
Kurzbeschreibung:	Gespeicherte Events werden aus der Event DB geladen und in einem Dataframe zur weiteren Verarbeitung zugewiesen. Der Data Scientist hat die Möglichkeit zu entscheiden, welche Events geladen werden sollen
Ablauf	1. Abhängig vom Input des Data Scientist, laden der entsprechenden Events aus der EventDB

Tabelle 11 - Laden der Events

Die geladenen Events werden anschließend in der Datenverarbeitung, wie oben beschrieben, bearbeitet. Der Output wird darauffolgend zum Trainieren der ML-Modelle verwendet.

Name	Laden der Label
Kurzbeschreibung:	Zum Trainieren der Modelle werden zusätzlich zu den bereits vorverarbeiteten Events die zugehörigen Label benötigt. Die Label zum Trainieren der Modelle ist jeweils die Spalte „Error_Message“ der nachliegenden Teststation. Das heißt für die Events der Teststation1 muss die Spalte „Error_Message“ der Daten von Teststation2 geladen und verbunden werden. Verbindungselement zwischen beiden Events ist wieder die Component_ID.
Ablauf	<ol style="list-style-type: none"> Überprüfen von welcher Teststation die Events stammen. Laden der Events von nachfolgenden Teststation Hinzufügen der Spalte „Error_Message“ der nachfolgenden Teststation, bei übereinstimmender Component_ID. Name der hinzugefügten Spalte ist „Future_Error_Message“

Tabelle 12 - Laden der Label

Das Hinzufügen der Label zum Dataframe wird ausschließlich in der Batch-Schicht benötigt, weshalb dieser Use-Case nicht Bestandteil der Datenverarbeitung ist. Die Struktur sieht wie folgt aus.

Teststation1

Component_ID, Teststation_ID, Feature_1, Feature_2, DateTime, Checksum, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday, Week, Green, Yellow, Red, Blue, Product_Type, Error_Message, Future_Error_Message

Teststation2

Component_ID, Teststation_ID, Feature_3, Feature_4, DateTime, Checksum, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday, Week, Product_Type, Error_Message, Feature_1, Feature_2, Green, Yellow, Red, Blue, Future_Error_Message

Zum Trainieren Modelle liegen die bearbeiteten Events anschließend zur Verfügung. Für jedes Dataframe wird der Funktionsaufruf zum Trainieren der ML Modelle aufgerufen.

Name	Trainieren der ML Modelle
Kurzbeschreibung:	Vor dem Training der ML Modelle muss das Input Dataframe zuerst nach der Spalte „Product_Type“ gefiltert werden. Nach dem Filtern muss es genauso viele Dataframes geben, wie es Produkttypen gibt. Darauffolgend muss die „Teststation_ID“ überprüft werden. Für jede Teststation werden verschiedene Modelle erzeugt. Input für das Training bei sind alle Attribute (außer ID-Attribute). Das Label für das Training ist die Spalte „Future_Error_Message“. Für jeden Fehlertyp der Spalte „Future_Error_Message“ muss ein eigenes Modell erstellt werden. Wird ein Modell mit einem Fehlertyp trainiert, so muss dieser Fehlertyp auf den Wert 1 zugewiesen werden und alle anderen Fehlertypen auf den Wert 0 gesetzt werden. Anschließend werden die Daten in Test und Trainingsdaten gesplittet und mit der Spalte „Future_Error_Message“ als Label trainiert.
Ablauf	<ol style="list-style-type: none"> 1. Filtern des Dataframes nach „Product_Type“ 2. Überprüfen, um welche Teststation es sich bei den Events handelt 3. Pro Fehlertyp ein entsprechendes Modell trainieren

Tabelle 13 - Trainieren der ML-Modelle

Name	Speichern der ML Modelle
Kurzbeschreibung:	Nachdem die ML Modelle trainiert wurden, werden diese exportiert. Im Verzeichnispfad und dem der Exportdatei muss für den späteren Zugriff die Teststation_ID, der Product_Type als auch eine Information mit welchen Daten von welchen Teststations das Modell trainiert wurde, enthalten sein.
Ablauf	<ol style="list-style-type: none"> 1. Generieren des Pfades und des Names der Datei 2. Ablegen des Modells

Tabelle 14 - Speichern der ML-Modelle

Speed Schicht

In der Speed-Schicht werden die Events verarbeitet, welche über das simulierte Senden von Events von verschiedenen Teststationen an die Service Schnittstelle gesendet wird.

Name	Erstellen einer Service Schnittstelle
Kurzbeschreibung:	Damit die Simulation des Sendens von Events von Teststationen tatsächlich Events senden kann, muss diese Service Schnittstelle erstellt werden. Zusätzlich muss die Schnittstelle für die Simulation verfügbar gemacht werden. Dazu muss diese Schnittstelle im Hintergrund laufen. Der Data Scientist hat die Möglichkeit Fehlervorhersagen für nur bestimmte Teststationen und Produkttypen generieren zu lassen
Ablauf	<ol style="list-style-type: none"> 1. Erstellen der Service Schnittstelle 2. Starten der Service Schnittstelle

Tabelle 15 - Erstellen einer Service Schnittstelle

Als ersten Punkt der Implementierung des Service werden die Events, wie unter Datenverarbeitung beschrieben, verarbeitet. Output ist ein einzeiliges Dataframe. Im Folgenden werden die weiteren Use-Cases des Service spezifiziert.

Name	Laden der ML Modelle
Kurzbeschreibung:	Um für das Event eine Vorhersage machen zu können, muss das entsprechende Modell geladen werden, welches in der Batch-Schicht erstellt und gespeichert wurde. Dazu müssen Informationen aus dem Dataframe extrahiert werden. Zu diesen Informationen gehören Teststation_ID und Product_Type.
Ablauf	<ol style="list-style-type: none"> 1. Extrahieren der Informationen zu Teststation_ID und Product_Type 2. Überprüfen, ob Modelle im Namen dieselben Informationen enthalten 3. Alle Modelle, die im Namen dieselben Informationen beinhalten, werden geladen

Tabelle 16 - Laden der ML-Modelle

Name	Erstellen einer Vorhersage
Kurzbeschreibung:	Die ML Modelle erhalten als Input die Attribute des Events. Mit diesem Input klassifizieren die Modelle das Event und erstellen Vorhersagen, ob das Messteil in Zukunft aussortiert werden könnte.
Ablauf	<ol style="list-style-type: none"> 1. Übergeben der Attribute des Events an die ML Modelle 2. ML Modelle treffen Vorhersagen

Tabelle 17 - Erstellen einer Vorhersage

Name	Vorhersage speichern
Kurzbeschreibung:	Die ML Vorhersagen werden in einer DB (ResultDB) abgelegt. Zusätzlich sollen, jeweils zum Ergebnis, Informationen über das verwendete ML-Modell gespeichert werden.
Ablauf	<ol style="list-style-type: none"> 1. Extrahieren von Informationen über das verwendete ML Modell 2. Ablegen des Vorhersageergebnisses und der ML Modell Informationen in eine Datenbank

Tabelle 18 - Vorhersage speichern

Name	Ergebnisse anzeigen
Kurzbeschreibung:	Um rechtzeitig die fehlerhaften Messteile aus dem Produktionsprozess auszusortieren, sollen die Vorhersagen auf einem Web Interface angezeigt werden. Messteile, die aussortiert werden müssen, sollen rot markiert sein und Messeile, welche nicht aussortiert werden müssen, sollen grün hinterlegt sein.
Ablauf	<ol style="list-style-type: none"> 1. Erstellen einer Web-Schnittstelle 2. Laden der Ergebnisse aus der ResultDB 3. Anzeigen der Werte in tabellenform

Tabelle 19 - Ergebnisse anzeigen

2.2.4 Nichtfunktionale Anforderungen

Über die im vorherigen Kapiteln spezifizierten Systemfunktionalitäten, muss die zu entwickelnde Lösung zusätzlich noch weitere, nichtfunktionale Anforderungen erfüllen. Eine dieser nichtfunktionalen ist die Verwendung von ausschließlich frei zugänglichen Software Technologie. Grundlage des Prototyps ist die Skriptsprache Python. Zur Archivierung der Daten wird der Community-Server von MongoDB verwendet. Die Web-Schnittstelle soll mit Hilfe von HTML, CSS implementiert werden. Die Umsetzung der Service-Schnittstelle ist Teil der Forschungsfrage, weshalb im Rahmen dieses Use-Cases mehrere Technologien implementiert werden können. Gleichermaßen gilt das Exportieren und Speichern der ML Modelle. Einzig die freie Verfügbarkeit (Open Source) der Technologie muss gewährleistet sein.

Welche Technologien bei der Implementierung der Service Schnittstelle oder beim Exportieren und Speichern von ML Modell in der finalen Lösung verwendet werden, hängt von der Performance der jeweiligen Technologien ab. Aufgrund des Big Data Kontextes, im welchen sich diese Arbeit und der Prototyp befindet, muss die finale Lösung im Stande hohen Datenmengen möglichst schnell und unter geringer Performanceeinbußen in Echtzeit zu bearbeiten. Die Anforderungen an die Performance in Real-Time sind allerdings nur in der Speed-Schicht von Bedeutung. Das heißt, dass dort die Datenbankzugriffe minimiert werden müssen..

Aufgrund von beschränkten Ressourcen limitiert sich diese Lösung auf die Simulation von wenigen Teststationen. Unter dem Aspekt, dass der Prototyp später in einer echten Produktionsumgebung zum Einsatz kommen könnte, muss der Code so generisch wie möglich

gestaltet sein, sodass das Aufskalieren auf weitere Teststation (und den damit verbunden Anstieg von zu erstellenden ML-Modellen) ohne große Anpassung möglich ist.

Zusätzlich zur Skalierbarkeit der Lösung muss der Source-Code nach den spezifizierten Use-Cases abgetrennt sein, unter der Prämisse, dass die Performance keine negativen Auswirkungen zu tragen hat. Kommt es zu einem Zielkonflikt zwischen nach Use-Case abgegrenzten Code und Effizienz bzw. Performance, so muss abgewogen werden, welches Ziel die höhere Priorität an der entsprechenden Stelle hat.

2.2.5 Lieferumfang

Resultat der Forschungsarbeit ist ein vollfunktionsfähiger Prototyp, der in der simulierten Produktionsumgebung zum Einsatz kommt. Für den Einsatz in einer echten Produktionsumgebung ist die Architektur bereits auf Skalierbarkeit der Teststationen und Performance geprüft und vorbereitet, ohne große Anpassungen im Quellcode vornehmen zu müssen. Grundlage für die Skalierbarkeit und Performance ist das im Rahmen dieser Arbeit erstellte Datenmodell, welches ebenfalls Teil der Lösung darstellt.

3 Grundlagen

Auf Basis der vorangegangenen Beschreibungen über die Herausforderungen im Forschungsprojekt PREFERML und der daraus resultierenden Anforderungsanalyse zur Implementierung eines Prototyps, muss zunächst ein Fundament von Grundlagen geschaffen werden, um eine erfolgreiche und sinnhafte Implementierung zu gewährleisten. Die zugrundeliegenden Konzepte werden deshalb in diesem Kapitel erklärt.

3.1 Datenverarbeitung

Vorangetrieben von der zunehmenden Digitalisierung der Unternehmen und deren Geschäfts- und Produktionsprozessen, steigt der Bedarf an intelligenten Strategien und Lösungen, um aus den extrem großen Datenmengen wertvolle Informationen und Einblicke zu gewinnen [46]. Nationale Bemühungen wie Industrie 4.0 (Deutschland), Made-in China 2025 (China) oder AMP (Vereinigte Staaten von Amerika) sind als Folge der Digitalisierung und der damit verbundenen Informationsgewinnungspotenziale entstanden. Die Herausbildung des Konzepts von Cyber Physical System (CPS) und Big Data führen zu einer intelligenteren und wettbewerbsfähigeren Fertigung über die nationalen Grenzen hinweg [47] [48]. Dem Bericht von Ismail et. Al. zu Folge [46] geht es bei Big Data primär um die notwendigen Änderungen in der Architektur und den Systemen, um die gewaltigen Datenmengen zu verwalten, und nicht um das Auftreten von größeren Datenmengen und daraus resultierenden Performanceanforderungen.

Auch im Forschungsprojekt PREFERML geht es im weitesten Sinne darum, den Qualitätsprozess durch die anfallenden Sensor- und Systemdaten intelligenter zu gestalten. Um Konzepte wie Maschinelles Lernen in den Qualitätsmanagementprozess sinnvoll einbinden zu können, sind diverse Herausforderungen zu bewältigen, welche im Kapitel [2.1.2 Probleme und Herausforderungen bei Datenanalysen im Produktionsumfeld](#) beschrieben werden. Damit diese Herausforderungen bewältigt werden können, ist eine Datenverarbeitung notwendig. Durch die großen Mengen zu verarbeiteten Datenmengen hat sich aus der klassischen Datenverarbeitung inzwischen die Big Data Verarbeitung herausgebildet [47] [49] [50]. Zur Einordnung der Datenverarbeitung bzw. der Big Data Verarbeitung wird im Folgenden zunächst der Begriff Big Data erläutert.

3.1.1 Big Data

Big Data gilt als einer der wichtigsten Bereiche der zukünftigen Informationstechnologien und entwickelt sich in einem rasanten Tempo weiter [51]. Das Konzept Big Data tritt als Resultat von immer größer werdenden Datenmengen auf, welche gegebenenfalls versteckte Informationen beinhalten und somit einen Mehrwert beitragen können. Damit diese Informationen erfolgreich extrahiert werden können wurden Tools und Technologien entwickelt, welche dem Problem mit den großen Datenmengen entgegenwirkt [52]. Als die Geburtsstunde von Big Data gilt laut Lee der „Advent des E-Commerce“ von 1994-2004 [51]. Dort begannen Entwickler die Online Aktivitäten mittels Web-Mining Methoden der User zu analysieren. Über die Jahre hinweg bildeten sich durch die zunehmende Rechenleistung der Computer, kohärent zum Moorschen Gesetz [53], immer neue Methoden und Technologien. Die Anwendungsmöglichkeiten des Konzepts Big Data haben deshalb längst auch andere Bereiche erschlossen, wie beispielsweise Fertigungsumgebungen [47].

In der Literatur finden sich in Bezug auf Big Data ein breit gefächertes Vorgehen zur Definition des Konzeptes Big Data. De Mauro et. al. [54] zu Folge ist eine überzeugende Definition eines Konzepts die Voraussetzung für dessen wissenschaftliche Entwicklung. Ohne dass ein Dasein einer solche Definition bestand, hat sich das relativ jung Konzept Big Data rasant entwickelt.

Dies führte zu vielen Definitionsversuchen, welche später von den Autoren*innen ignoriert, ergänzt oder verändert wurden [54]. In den meisten Fällen jedoch, stimmt die Vorgehensweise zur Begriffserklärung überein. Die Autoren*innen versuchen dabei über die Kerneigenschaften das Konzept Big Data zu definieren. Der Anfangsbuchstabe der Eigenschaften beginnt häufig mit dem Buchstaben „V“, weshalb die Merkmale zur Begriffserklärung gerne als die „V“s des Big Data gruppiert werden. Je nach Autor*in variiert allerdings die Anzahl der „V“s [55] [51] [56] [57].

Übereinstimmend sind bei den verschiedenen Autoren*innen allerdings drei verschiedene „V“s die im Folgenden beschrieben werden:

Volume

Soziale Medien, E-Commerce, Internet of Things (IoT) oder Sensoren sind die Treiber von Big Data. Sie sind verantwortlich, dass ca. alle 2 Tage so viele Informationen entstehen, wie von Beginn der Informationserfassung bis 2003 [55]. In diesem Sinne bezieht sich die Eigenschaft Volume auf die Menge an Daten, die von einer Instanz generiert und gesammelt werden. Lee setzt dabei die Grenze der Mindestmenge an Daten – um als Big Data eingestuft zu werden – bei einem Terabyte [51]. Im Vergleich zu alten Datenträgern sind mittels neuer Datenträger und Cloudtechnologien Speicherungen einer solchen Datenmenge heutzutage kein Problem mehr. Mit älteren Datenträgern wie die Compact Disc (CD) oder die Digital Video Disc (DVD) würden für die Speicherung für einen Terabyte mehrere Datenträger benötigt werden (beispielsweise 1500 CDs oder 220 DVDs) [55].

Velocity

Unter Velocity versteht man bei Big Data die Geschwindigkeit der Daten, die aus verschiedenen Quellen stammen. Hierbei handelt es sich nicht nur um die Geschwindigkeit der eingehenden Daten, sondern auch um die Geschwindigkeit, mit der die Daten fließen [58]. Ein klassisch batch-orientierter Ansatz zur Datenverarbeitung reicht bei der Geschwindigkeit der aktuellen Datengenerierung nicht aus. Zusätzlich nimmt die Geschwindigkeit der Geschwindigkeit der Daten weiter zu [55] [58]. Für die Datenverarbeitung sind deshalb Lösungen notwendig, welche die Herausforderung der wachsenden Datengeschwindigkeit meistert. Der Bedarf an Echtzeitverarbeitungsapplikation steigt [58].

Variety

Variety im Big Data Kontext bedeutet, dass verschiedene Datentypen generiert, gespeichert und verarbeitet werden müssen. Gesammelt werden nicht nur strukturierte Daten, wie in herkömmlichen Applikationen, sondern auch halbstrukturierte und unstrukturierte Daten [55]. Beispiele für halbstrukturierte und unstrukturierte Daten sind Text-, Foto-, Audio, Video- und Sensordaten [55] [58]. Durch die Breite an verschiedenen Datentypen sind neue Analysesysteme notwendig. Dieses Analysesystem muss in der Lage sein die verschiedenen Datentypen zu behandeln und zu verarbeiten [55].

Eine Definition, welche auf Basis der Eigenschaften Volume, Velocity und Variety gebildet ist, liefert De Mauro et. al. [54]: “Big Data represents the Information assets characterized by such a High Volume, Velocity and Variety to require specific Technology and Analytical Methods for its transformation into Value.” [54]. Gleichzeitig berücksichtigt diese Definition die bereichsunabhängige Anwendbarkeit des Konzepts, weshalb diese Bergrifferklärung auch in Fertigungsumgebungen angewandt werden kann.

Die in dieser genannten Definition enthaltenen Begriffe Big Data Technologien und Big Data Methoden, welche dazu beitragen sollen, dass aus den massiven Datensätzen Informationen

gewonnen werden können. Synchron zu der Informationsgewinnung müssen die Technologien und Methoden die Herausforderungen bezüglich der Varietät und der Geschwindigkeit überwinden. Im Vordergrund dieser Forschungsarbeit steht die Datenverarbeitung, was als elementarer Bestandteil des Konzepts von Big Data angesehen werden kann [52]. Eigenschaften wie Velocity und Variety beziehen sich in diesem Rahmen auch überwiegend auf die Datenverarbeitung. In den folgenden Kapiteln werden deshalb die Technologien und Methoden beschrieben, welches die Datenverarbeitung adressieren. Zuerst allerdings werden die Unterschiede von herkömmlichen Datenverarbeitung zu Big Data Verarbeitung aufgezeigt.

3.1.2 Datenverarbeitung vs. Big Data Verarbeitung

Was für Auswirkungen die Eigenschaften Volume, Velocity und Variety auf die Datenverarbeitung haben und warum herkömmliche Datenverarbeitungstechnologien bei Big Data nicht ausreichen [50] [57], wird in diesem Kapitel erklärt. Als Ausgangslage dient eine Tabelle, die aus dem Buch Big Data Analytics und Preprocessing entnommen wurde [52]. Tabelle 20 ist eine eigene Darstellung basierend auf der Tabelle im Buch. Die Tabelle zeigt die Unterschiede von traditionellen Daten und Big Data anhand von Differenzierungsmerkmalen. Die Differenzierungsmerkmale sind hierbei das Volumen, die Datengenerierungsrate, die Datenstruktur, die Datenquelle, die Datenintegration, die Speichertechnologie und der Zugriff. Das Volumen der traditionellen Daten ist im Gegensatz zu Big Data um ein Vielfaches geringer. Shehab et. al. [52] setzen die traditionellen Daten bei Giga Bytes (GBs) an, während Big Data ein Volumen von Tera Bytes (TBs), Peta Bytes (PB) oder Zetta Bytes (ZBs) annehmen kann. Dementsprechend ist die Datengenerierungsrate bei Big Data schneller als bei traditionellen Daten. Wie bereits in der Sektion Variety im vorherigen Kapitel angesprochen, zeichnen sich Big Data durch unterschiedliche Datenstrukturen aus. Sie können strukturiert, unstrukturiert oder auch halbstrukturiert sein. Traditionelle Daten sind im Gegensatz dazu immer strukturiert. Die traditionellen Daten liegen oft an einem zentralen Ablageort, können jedoch auch an verteilten Ablageorten liegen. Big Data ist vollständig verteilt. Abgeleitet aus den Datenstruktur- und Datenquellmerkmal bildet sich das nächste Differenzierungsmerkmal – die Datenintegration. Da alle traditionelle Daten immer strukturiert und oft zentral abgelegt sind, können diese recht leicht integriert werden. Große Datenmengen hingegen, mit ihren unterschiedlichen Ausprägungen in der Struktur und ihrer Verteilung auf mehrere Speicherorte, lassen sich dagegen schwerer integrieren. Ebenso ergeben sich aus den unterschiedlichen Datenstrukturen und Datenquellen unterschiedliche Speichertechnologien. Während traditionelle Daten mit relationalen Datenbank Management Systemen (RDBMS) arbeiten, verwenden Big Data Konzepte das Hadoop Distributed File System (HDFS) und Not only SQL (NoSQL) Systeme. Als letztes Differenzierungsmerkmal wird in der Tabelle 20 der Zugriff auf die Daten genannt. Dieser geschieht bei traditionellen Daten interaktiv und bei Big Data batch-orientiert in Echtzeit.

Comparison	Traditional data	Big data
Volume	In GBs	TBs, PBs, and ZBs
Data generation rate	Per hour/day	Faster than traditional data
Data structure	Structured	Structured, unstructured, semi-structured
Data source	Centralized/distributed	Fully distributed
Data integration	Easy	Difficult
Storage technology	RDBMS	HDFS + NoSQL
Access	Interactive	Batch/real-time

Tabelle 20 – Traditional Data vs. Big Data

Durch die genannten Unterschiede von Big Data zu traditionellen Daten, müssen neue Technologien und Methoden verwendet werden, um den Anforderungen gerecht zu werden. Die erste

Technologie in Bezug auf die Speicherung der Daten wird in der Tabelle bereits gezeigt. HDFS ist eine Technologie zur Speicherung der Daten auf verteilten Dateisystemen, welche auf die Verwendung handelsüblicher Hardware ausgelegt ist [59]. Diese und weitere Big Data Technologien, Methode und Tools werden im nächsten Kapitel in den entsprechenden Big Data Schichten Kapiteln erläutert.

3.1.3 Big Data Schichten

Es gibt eine große Anzahl an verschiedenen Technologien, Methoden und Tools welche im Big Data Umfeld zum Einsatz kommen. Um all diese Technologien, Methoden und Tools vorzustellen könnte eine eigene, weitere Forschungsarbeit durchgeführt werden. Zur Vereinfachung und zum Eingrenzen der Komplexität werden in diesem Kapitel zunächst die verschiedenen Big Data Schichten erläutert. Darauffolgend werden für die Schichten entsprechende Technologien, Methoden und Tools beschrieben und vorgestellt. Die Big Data Schichten wurden aus Manikandan und Abirami [60] entnommen. Sie setzen sich zusammen aus:

- Big Data Sources
- Data Acquisition
- Data Storage
- Data Analysis

Abbildung 1 zeigt die genauen Schichten und dessen Eigenschaften. In der Literatur lassen sich oft noch weitere Schichten finden [61] [50]. Die Schichten überlappen sich dabei in ihren Aufgabenbereichen oder sind oft unterschiedlich benannt. Im Rahmen dieser Forschungsarbeit liegt der Fokus auf der Datenverarbeitung, weshalb nur die Schichten von Manikandan und Abirami [60] von Bedarf sind. Schichten wie Monitoring oder Sicherheit von Erraissi und Belangour [61] werden in diesem Kapitel deshalb nicht berücksichtigt.

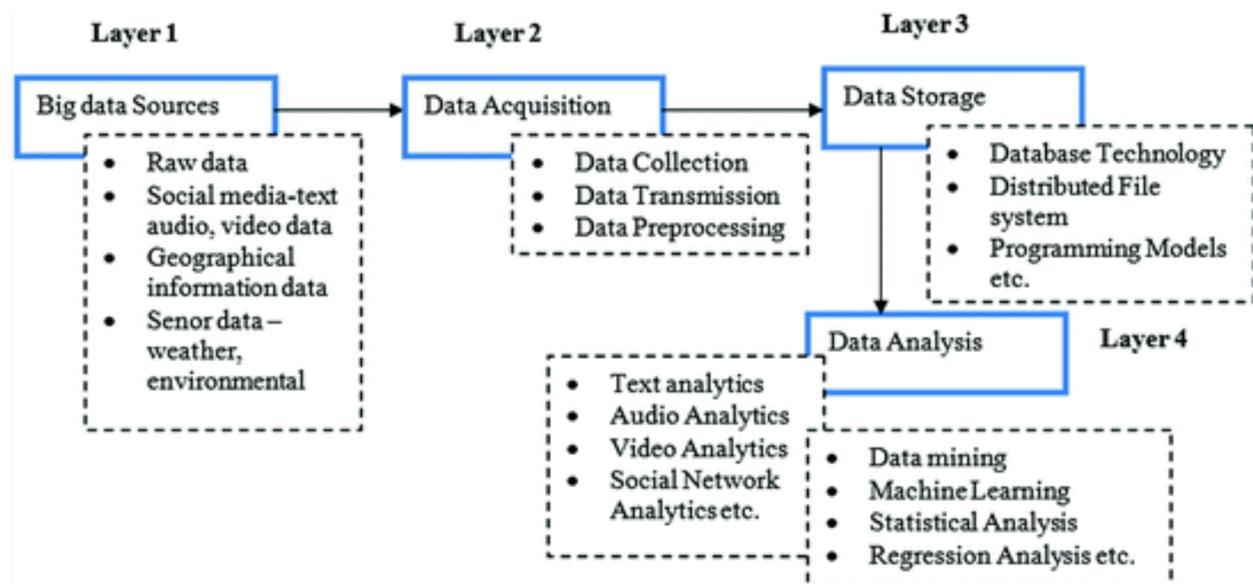


Abbildung 1 - Big Data Schichten

3.1.3.1 Big Data Sources

Die erste Schicht in Bezug auf Big Data ist die Big Data Sources Schicht. Diese Schicht beschäftigt sich mit der Generation der Daten. Üblicherweise kommen die Daten von verschiedenen Quellen in unterschiedlichen Formaten (strukturiert, unstrukturiert und halbstrukturiert) [60].

Im Forschungsprojekt PREFERML findet sich diese Schicht in Form der Vielzahl an Teststationen wieder. Jede Teststation misst mittels Sensormessungen verschiedene Eigenschaften der Fertigungsteile und nimmt diese Daten auf. Hierbei können die Daten verschiedene Strukturen annehmen, wobei die Daten in diesem Fall überwiegend strukturiert sind [15].

Die im [Kapitel 2.2.1.2 – Herausforderungen zu den Merkmalen der zugrunde liegenden Daten und Datenquellen](#) beschriebenen Herausforderungen sind auf die Generierung der Daten in dieser Schicht zurückzuführen. Beispiel hierfür ist die anstehende Datenfusion in der nächsten Schicht (Data Acquisition) aufgrund der multimodalen Datenquellen.

3.1.3.2 Data Acquisition

Die zweite Schicht in einem Big Data System ist die Data Acquisition. Dort werden die Daten der verschiedenen Datenquellen gesammelt und aggregiert. Dadurch ermöglicht die Schicht die zukünftige Speicherung und Analyse der Daten [60].

Resultierend aus der Anforderung, welche die Data Acquisition bewältigen muss, ergeben sich drei konkrete Aufgaben, bevor die Daten in eine Speicherlösung gelangen [60] [62]:

- Data Collection
- Data Transmission
- Data Preprocessing

Data Collection

Data Collection ist die erste Stufe bei der Data Acquisition. Dort werden die existierenden Rohdaten aus der heterogenen Systemlandschaft gesammelt. Nach der Data Collection folgt eine Übertragung der Rohdaten in ein Speichersystem für weitere Analysen [60]. Im Prozess des Forschungsprojekts PREFERML bestehen in dieser Schicht zwei Möglichkeiten zur Umsetzung. Entweder werden die Daten der Teststation über eine Schnittstelle übermittelt, wie zum Beispiel über eine Representational State Transfer (REST) Schnittstelle, und an einer zentralen Stelle gespeichert, oder die Daten werden direkt von der Teststation auf einer replizierten Datenbank gespeichert.

Data Transmission

Data Transmission ist der Schritt in der Data Acquisition Schicht, bei der die Rohdaten in eine Speicherinfrastruktur übertragen werden [60]. Für die Speicherung der Daten gibt es spezielle Speichertechnologien, welche für die Verarbeitung von Big Data entwickelt wurden [62]. Bei der Datenübertragung wird differenziert, ob die Übertragung von einer externen Quelle zur Speicherlösung transferiert wird, oder ob die Daten innerhalb einer verteilten Speicherlösung übertragen werden [60]. Auch für die Übertragung der Daten gibt es spezielle Technologien, Standards und Protokolle, welche für das Verarbeiten von Big Data ausgelegt sind [62].

Data Preprocessing

Voraussetzung für das Verwenden der Daten in der Data Analysis Schicht ist die Sicherstellung der Datenqualität. Werden die Rohdaten ohne vorheriges Data Preprocessing in der Data

Analysis Schicht verwendet, so besteht die Gefahr, dass die Analyseergebnisse falsch sind [60]. Das Data Preprocessing ist somit ein entscheidender Erfolgsfaktor bei der Implementierung eines Big Data Systems [52]. Damit die Datenqualität für die Verwendung von Datenanalyse-Methoden in der Data Analysis Schicht gewährleistet werden kann, müssen die Rohdaten gewisse Bearbeitungsschritte durchlaufen. Im Idealfall geschieht dies, bevor die Rohdaten in die Speicherlösung transferiert werden [60]. Abhängig von der verwendeten Analyseverfahren in der Data Analysis Schicht und dem Aufbau der Rohdaten sind verschiedene Verarbeitungsschritte von Bedeutung [52] [60]. Beispielsweise ist bei hochdimensionalen Daten eine Dimensionsreduktion erforderlich, um performante Ergebnisse erzielen zu können (hierfür werden Methoden, wie Feature Selection oder Information Gain benötigt) [63] [52]. Grundsätzlich lässt sich der Data Preprocessing Schritt in zwei Teile aufteilen – die Data Preparation und die Data Reduction. Die Data Preparation gilt als notwendiger Schritt und umfasst Aufgaben wie die Integration, Normalisierung, Bereinigung und Transformation. Die Data Reduction hat die Aufgabe redundante und verrauschte Daten zu identifizieren und zu bereinigen [52].

Die Schwierigkeit im Big Data Umfeld ist einen einheitlichen Data Preprocessing Schritt zur Verfügung zu stellen. Aufgrund des Merkmals „Variety“ müssen unterschiedlich viele Fälle abgedeckt werden. Im Rahmen des Forschungsprojektes PREFERML kann es so zum Beispiel sein, dass das Data Cleaning für die Daten einer Teststation anders aussieht, als für die Daten der nächsten Teststation.

3.1.3.3 Data Storage

Nachdem die Rohdaten gesammelt, aggregiert und verarbeitet wurden, werden sie von der Data Acquisition Schicht in die Data Storage Schicht in eine Speicherlösung transferiert [60]. Die Speicherlösung hängt dabei von der Struktur der Daten und der Systemumgebung ab. Gängige Speicherlösungen im Big Data Umfeld sind Hadoop oder HBase [64]. Im Rahmen dieser Forschungsarbeit liegt der Hauptfokus allerdings auf der Datenverarbeitung. Die Speicherlösungs-konzepte werden deshalb in den folgenden Kapiteln kurz vorgestellt, jedoch nicht ausgeführt. Im Allgemeinen gilt es die Herausforderung der großen Datenmengen (Volume) zu lösen und den schnellen Zugriff auf die passenden Daten zu gewährleisten.

3.1.3.4 Data Analysis

Die letzte Schicht im Big Data Model von Manikandan und Abirami [60] ist die Data Analysis Schicht. Dort gilt es Informationen aus den Datenmengen zu extrahieren. Dabei wird auf die zuvor aufbereiteten und abgelegten Daten der Data Storage Schicht zugegriffen. Basierend auf den Analyseergebnissen können dann Entscheidungen getroffen werden. Analytische Methoden zur Informationsgewinnung sind Konzepte wie Maschinelles Lernen, statistische Analyse oder Regressionsanalyse [60].

In der Forschungsprojektes PREFERML und der vorliegenden Arbeit kommen zur Datenanalyse Modelle des Maschinellen Lernens zum Einsatz. Die Merkmale und Besonderheiten dieses Konzepts wird im [Kapitel 3.3 – Maschinelles Lernen](#) genauer ausgeführt.

3.1.4 Big Data Technologien

In den Ausführungen dieses Kapiteln werden gängige Technologien, Methoden und Werkzeuge vorgestellt, welche in den zuvor vorgestellten Big Data Schichten Anwendung finden. Zur Ein-grenzung werden jedoch nur die Technologien, Methoden und Werkzeuge vorgestellt, welche im Rahmen der Implementierung des Prototyps eingesetzt oder in Betracht gezogen werden.

3.1.4.1 MongoDB

Ursprünglich dominierten bei der Verwendung von Speicherlösung die Relationalen Datenbank-managementsysteme wie Structured Query Language (SQL) Datenbanken [65]. Diese Art der Datenbanken stoßen durch die Skalierung von großen Datensätzen ihren Grenzen [50] [65]. Um den Herausforderungen von Big Data entgegen zu wirken, haben sich deshalb NoSQL-Datenbanken entwickeln, welche auf den ACID-Transaktions-Eigenschaften (Atomicity, Consistency, Isolation und Durability) der relationalen Datenbanken verzichten [65].

MongoDB ist eine dokumentorientierte NoSQL Datenbank, welche im Rahmen eines Open-Source Projekts entstanden ist. Die Datenbank ist in C++ entwickelt worden und wird von der Softwarefirma 10gen unterstützt [65]. Die Dokumente basieren in MongoDB auf dem JSON Format und werden vom BSON Format unterstützt [66]. Wird ein neues Dokument in die Datenbank eingepflegt, so wird automatisch vom System eine eindeutige Identifizierung (Primary Key) hinterlegt (wird vom Benutzer eine eigene Identifizierung mitgeliefert, so verzichtet das System auf eine weitere Identifizierung). Die MongoDB ist schemalos, sodass die Dokumente können eine beliebige Anzahl von Feldern einnehmen. Zusätzlich können in den einzelnen Feldern weitere Dokumente oder Arrays eingebettet sein. Aufgrund der JSON-Eigenschaften der Datenbank können die Dokumente mittels JSON-Manipulationen gesucht, geändert und gelöscht werden. Zudem werden Sortierungen, Iterationen und Projektionen von dem System unterstützt. Vorteil der MongoDB ist außerdem, dass MapReduce-ähnliche Operationen auf die Dokumente ausführen kann (das Konzept MapReduce wird in den folgenden Kapiteln genauer erläutert). MongoDB bietet für die Herausforderung der großen Datenmengen zudem eine Indizes Funktion. Die Indizes Funktion indiziert die abgelegten Dokumente, sodass die Suchanfragen beschleunigt werden [65] [66].

Eine besondere Eigenschaft von MongoDB ist MongoDB Change Streams. Seit der Version MongoDB 3.6 ermöglicht die Change Streams Funktionalität das Streamen von Datenänderungen in Echtzeit, indem die zugrunde liegenden Replikationsfunktionen von MongoDB genutzt werden [67]. Diese Funktionalität kann je nach der Systemlandschaft von wertvoller Bedeutung sein. Handelt sich beispielsweise um Sensordaten von vielen unterschiedlichen Sensoren (wie im Forschungsprojekt PREFERML), die in eine replizierte Datenbank ihre Messungen eintragen (Insert Operation), kann die Change Stream Funktionalität als Trigger in Echtzeit für die Datenverarbeitung fungieren. MongoDB Change Streams unterscheidet dabei unterschiedliche Arten von Events. Es können als Folge daraus nicht nur Inserts einen Change Stream auslösen, sondern auch Deletes, Updates, etc. [68]. Voraussetzung für das Implementieren der Funktionalität ist eine replizierte MongoDB Datenbank der Version 3.6 oder neuer, als auch das Konfigurieren und Aktivieren des Change Streams selbst [68].

3.1.4.2 Apache Kafka

Apache Kafka ist ein Publish-Subscribe-Messaging Dienst, der in Form eines verteiltes Übertragungsprotokoll implementiert ist und sich sowohl für den Offline- als auch für den Online-Messaging-Konsum eignet. Ursprünglich wurde der Messagingdienst von LinkedIn entwickelt und wurde darauf konzipiert große Mengen an Daten mit geringer Latenz zu sammeln und zu versenden.

Message-Publishing ist ein Mechanismus zur Verbindung verschiedener Anwendungen mit Hilfe von Nachrichten, die beispielsweise von einem Message-Broker wie Kafka untereinander geroutet werden. Dabei handelt es sich um eine Art "Write-Ahead-Log", das Nachrichten in einem persistenten Speicher aufzeichnet und es den Konsumenten ermöglicht, diese Änderungen zu lesen und in einem systemgerechten Zeitrahmen in ihre eigenen Speicher zu übernehmen. Zu den

üblichen Konsumenten gehören Live-Dienste oder andere Verarbeitungsströme, welche Streams durchführen, sowie Hadoop- und Data-Warehousing-Pipelines, die alle Feeds zur stapelorientierten Verarbeitung laden [69]. [Abbildung 2](#) zeigt eine typische Kafka-Architektur, die von Thein aus dem Bericht „Apache Kafka: Next Generation Distributed Messaging System“ übernommen wurde [69].

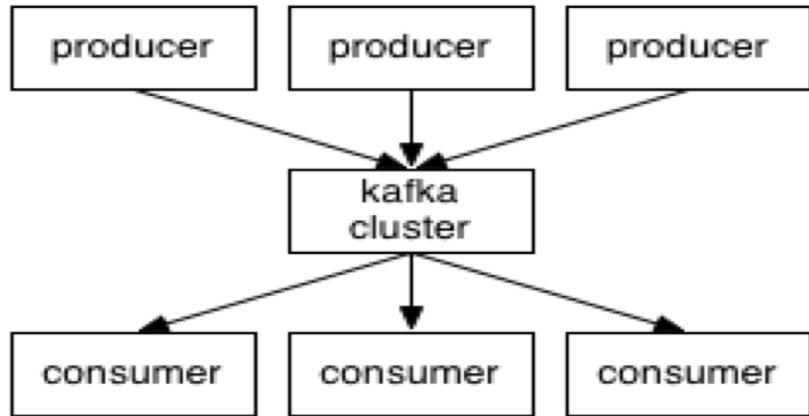


Abbildung 2 - Kafka Architektur

Die Funktionsweise der Architektur reagiert im Allgemeinen auf die Generierung von Events. Producer sind die Client-Anwendungen, die Events in Kafka veröffentlichen (schreiben), und Consumer sind diejenigen, die diese Ereignisse konsumieren. In Kafka sind Producer und Consumer vollständig voneinander entkoppelt und agnostisch, was ein wichtiges Designelement ist, um die hohe Skalierbarkeit zu erreichen. Beispielsweise müssen Producer nie auf einen Consumer warten. Sie können das Event einfach an das Kafka Cluster übergeben. Sobald ein Consumer frei ist, kann dieser auf das Event im Cluster zugreifen. Außerdem bietet Kafka verschiedene Funktionalitäten wie die Fähigkeit, das Events immer nur genau einmal verarbeitet werden [70].

Aufgrund der voneinander abgekoppelten Producer und Consumer bietet sich Kafka auch in der vorliegenden Forschungsarbeit an. Durch die verschiedenen Teststationen wird eine hohe Anzahl von Events erzeugt, die alle voneinander verteilt erstellt wurden. Mit Hilfe des Kafka Clusters können diese Events über das verteilte Kafka Cluster parallel verarbeitet werden.

3.1.4.3 Hadoop Framework

Das Hadoop Framework ist ein Open-Source Softwareprojekt für zuverlässige, skalierbare, verteilte Datenverarbeitung [71]. Die im Framework enthaltenen Bibliotheken ermöglichen die verteilte Verarbeitung von großen Datensätzen über Computercluster hinweg. Dazu werden einfache Programmiermodelle verwendet. Das Framework ermöglicht außerdem das Skalieren auf bis zu tausenden Maschinen, bei denen jeweils die lokale Rechenleistung und der lokale Speicher ausgenutzt wird. Das Framework ist zudem darauf konzipiert, dass Ausfälle von Anwendungen vom System selbst erkannt und behandelt werden. Durch diese Funktionalität wird die Hochverfügbarkeit gewährleistet [71] [57].

Das Hadoop Framework enthält folgende Module:

- Hadoop Distributed File System
- Map Reduce

- HBase
- Pig
- Hive
- Sqoop
- Zookeeper
- Avro
- Cassandra
- Mahout
- Tez
- Spark
- Flume

Das Hadoop Framework adressiert mit seinen umfassenden Modulen fast alle Herausforderungen, welche im Big Data Kontext zu bewältigen sind. Zur Eingrenzung dieser Forschungsarbeit werden allerdings nur die notwendigen Module vorgestellt. Zu diesen Modulen gehört HDFS, Map Reduce und Spark.

Hadoop Distributed File System

HDFS wurde auf Basis von einem Java File System entwickelt und ist ein verteiltes Dateisystem, das den effizienten Zugriff auf Anwendungsdaten ermöglicht. Es ist ein fehlertolerantes Dateisystem, das bei der sicheren Speicherung großer Datenmengen hilft. Der Master-Knoten ist für die Verwaltung der Cluster-Metadaten zuständig. Es hat eine Master/Slave-Konstellation, bei der ein oder mehrere Slave-Maschinen von einer Master-Maschine gesteuert werden [59] [57].

Map Reduce

Eine Hadoop MapReduce-Aufgabe besteht im Allgemeinen aus zwei benutzerdefinierten Funktionen: *map* und *reduce*. Ein Hadoop-MapReduce-Auftrag erhält eine Reihe von Key-Value-Paaren (k, v) als Eingabe. Für jedes Paar wird die map-Funktion aufgerufen. Die map-Funktion gibt einen Satz von Zwischen-Key-Value-Paaren (k', v') mit null oder mehr Werten zurück. Das Hadoop MapReduce-Framework gruppiert dann diese Zwischen-Key-Value-Paare nach dem Zwischen-Key k' und ruft die reduce-Funktion für jede dieser Gruppen auf. Schließlich erzeugt die reduce-Funktion eine Anzahl von aggregierten Werten, die null oder mehr sein können [72]. Abbildung 3 zeigt eine, aus dem Report von Chen und Schlosser entnommene [73] MapReduce Funktion.

Anwender müssen bei Hadoop MapReduce normalerweise nur die Map- und Reduce-Funktionen beschreiben. Alles weitere, wie z. B. die Parallelisierung, wird vom System selbst erledigt. Zum Lesen und Schreiben von Dateien verwendet die Hadoop-MapReduce-Architektur ein verteiltes Dateisystem. Da üblicherweise hierfür HDFS verwendet wird, hat dies einen erheblichen Einfluss auf die Eingabe/Ausgabe-Effizienz von Hadoop-MapReduce-Jobs [72] [57] [74].

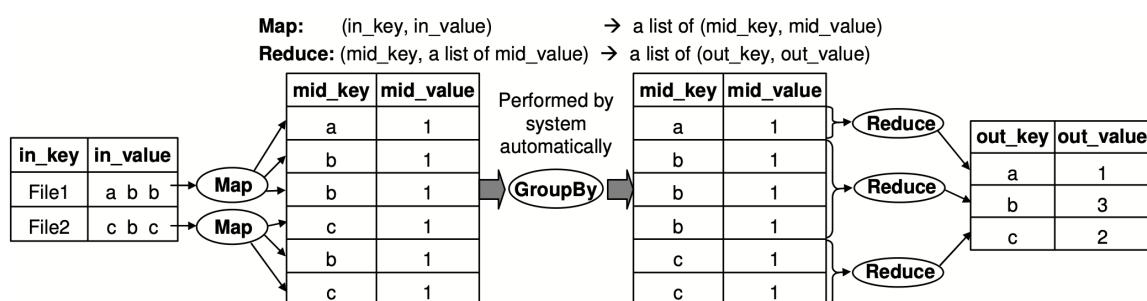


Abbildung 3 - MapReduce Funktion

Spark

Spark wurde als Reaktion auf die Einschränkungen des MapReduce-Paradigmas entwickelt, das verteilte Programme dazu zwingt, in einem linearen und grob definierten Datenfluss als eine Kette von verbundenen Map- und Reduce-Funktionen geschrieben zu werden [75]. Apache Spark ist ein verteiltes Open-Source-Systeme, das für Big Data-Analysen verwendet wird. Spark ist in der Lage, Aufgaben wesentlich schneller als andere Big-Data-Tools (z. B. MapReduce) abzuschließen, da es über ein In-Memory-Caching und eine optimierte Query-Ausführung verfügt [76]. Die dahinterliegende Technik wird Resilient Distributed Dataset (RDD) genannt. RDDs sind eine Sammlung von Elementen, die über die Knoten des Clusters verteilt sind und parallel verarbeitet werden können. Zudem können RDDs automatisch von Knotenausfällen wiederhergestellt werden [77]. Spark bietet APIs in Python, Java, Scala und R. Zusätzlich zum Haupt-Computing-Framework bietet Spark Bibliotheken für maschinelles Lernen, SQL, Graphenanalyse und Streaming [76]. Abbildung 4 zeigt die verschiedenen Spark Bibliotheken. Die Abbildung wurde von der offiziellen Spark Webseite übernommen [78].

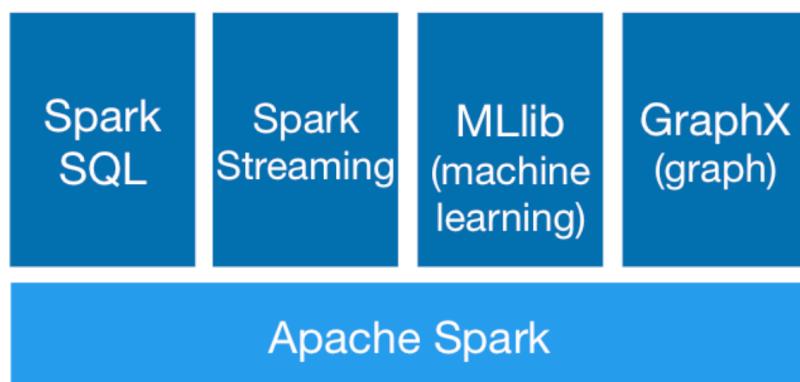


Abbildung 4 - Spark Bibliotheken

Da im vorliegenden Forschungsprojekt die Verwendung von ML-Bibliotheken für die Datenanalyse implementiert werden, bietet sich eine Verwendung des Spark Stacks mit der hoch performanten RDD Technologie, und der ML Bibliothek an. Die Verwendung von Spark für die Datenverarbeitung in Verbindung mit den in Spark enthaltenen ML-Bibliotheken können das Fundament für die Weiterentwicklung des Prototyps sein.

3.1.5 Big Data Processing

Big Data Verarbeitung kann in drei verschiedene Kategorien gruppiert werden. Die Kategorien setzen sich zusammen aus Batch-Processing, Real-Time-Processing und einer Hybrid Kategorie (Batch- und Real-Time-Processing) [79]. Abbildung 5 zeigt eine Big Data Verarbeitungsarchitektur, die von Lyko et. al. [62] entnommen wurde. Den einzelnen Schritten werden entsprechende Technologien, Methoden und Werkzeuge zugeordnet. Für die Acquisition eignet sich zum Beispiel der Messaging Service Kafka. Bei der Datenverarbeitung wird wie von Lyko et. al. [62] zwischen Real-Time Processing und Batch-Processing unterschieden. Auch wenn nicht direkt ausgeschlossen, wird eine Hybridlösung in der Abbildung nicht extra dargestellt

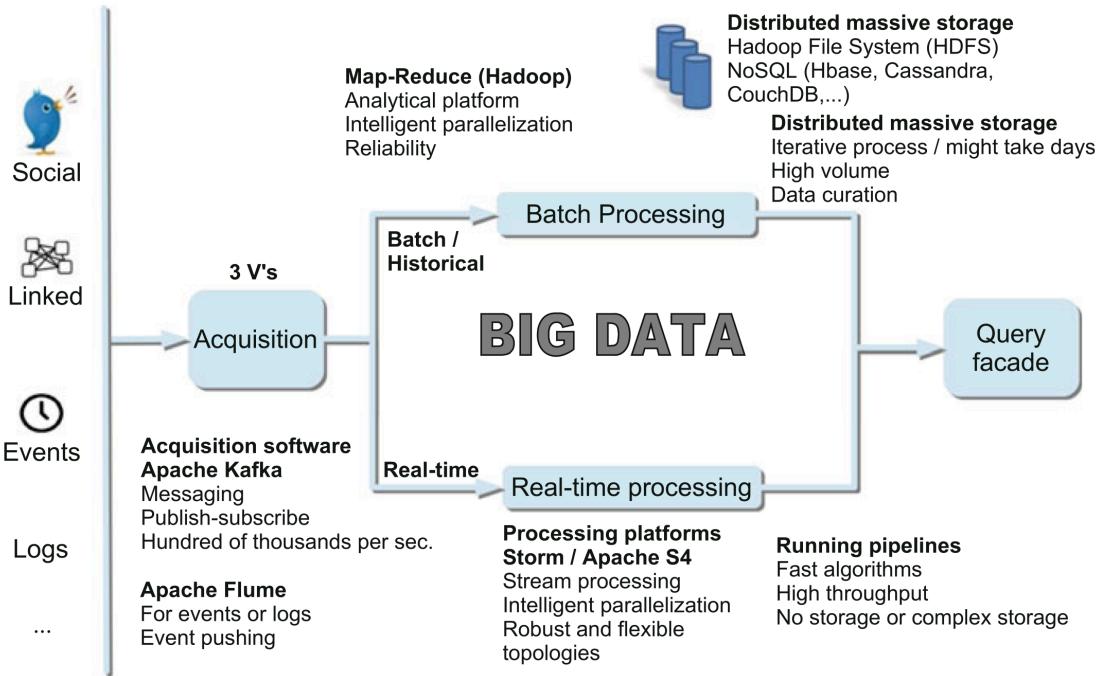


Abbildung 5 - Big Data Processing Architektur

Batch Processing

Das Batch-Processing ist eine Technik zur Verarbeitung großer Datenmengen. Die Transaktionen in diesem Paradigma werden innerhalb einer bestimmten Zeitspanne durchgeführt. MapReduce und HDFS des Hadoop Frameworks sind die am häufigsten verwendeten Tools für das Batch Processing. Das Batch-Processing umfasst verschiedene Transaktionen für Ingestion, Processing, Analyse und Reporting. In diesem Paradigma werden die Daten aufgenommen, in den Datenbanken gespeichert und verarbeitet. Die Batch-Ergebnisse werden analysiert und anschließend präsentiert. Im Vergleich zum Real-Time Processing ist das Batch-Processing nicht zeitlich begrenzt, daher ist es möglich, umfassendere Analysen durchzuführen, um effektivere Ergebnisse zu erzielen. Dieses Paradigma ist jedoch nicht für Anwendungen geeignet, die eine geringe Reaktionszeit erfordern. In diesem Fall müssen Real-Time-Anwendungen mit einer geringen Reaktionszeit ausgeführt werden [79].

Real-Time Processing

Zahlreiche Big-Data-Anwendungen erfordern ein Realtime Processing von Big Data (sogenannte Streams). Das Real-Time Processing besteht aus der kontinuierlichen Eingabe, Verarbeitung sowie Analyse und Ausgabe von Daten. Dieses Verarbeitungsparadigma zielt auf die niedrigste Latenz während des Prozesses ab. Ein passendes Framework zu Verarbeitung von Big Data in Echtzeit ist Apache Spark [79]. Bei Real-Time werden sich die Eigenschaften zu Nutzen gemacht, welche die Tools wie Apache Spark so performant machen. Anstatt die Daten auf den Disk-Speicher zu legen (wie zum Beispiel HDFS), werden die Daten im Memory-Speicher transformiert [76].

Im Rahmen dieser Forschungsarbeit sollen die Daten in Real-Time prozessiert werden. Mit den Analyseergebnissen (Vorhersagen der ML-Modelle) sollen fehlerhafte so schnell wie möglich aussortiert werden. Die niedrige Latenzzeit der Analyseergebnisse schlägt sich jedoch auf die Qualität der Ergebnisse aus. Um diese Schwierigkeit zu überwinden, kann ein hybrider Verarbeitungsansatz als neues Paradigma eingesetzt werden [79].

Batch- und Real-Time Processing

Die Hybrid-Lösung der beiden Verarbeitungsparadigmen (Batch- und Real-Time-Processing) nehmen die jeweiligen Vorteile ihres eigenen Konzepts mit. Die durch die gemeinsame Verwendung von Batch- und Real-Time-Processing erzielten Ergebnisse, werden analysiert und abgefragt, um in diesem Paradigma die gewünschten Ergebnisse zu erzielen. Die Ergebnisse werden dann miteinander kombiniert, harmonisiert und ausgewertet. In diesem Paradigma sind Daten-eingabe, -verarbeitung, -analyse und -reporting komplexere Prozesse, welche eine gewisse Orchestrierung erfordern [79].

3.2 Big Data Processing Architekturen

Die Implementierung des Batch-Processings, des Real-Time-Processings oder der hybriden Lösung hängt von der Systemlandschaft und dem Einsatzbereich des Big-Data Systems ab. Die Implementierung sollte jeweils an die Besonderheiten und Gegebenheiten des Umfelds angepasst sein. Die Big Data Systemlösungen sind deshalb oft individuell angepasst. In diesem Kontext wurden verschiedene Datenverarbeitungs-Architekturen für Big Data vorgeschlagen, um den unterschiedlichen Eigenschaften von Big Data gerecht zu werden [62].

Die Menge der Daten und die Bedeutung einfacher, skalierbarer und fehlertoleranter Architekturen zur Verarbeitung der Daten nimmt stetig zu. Da Big Data ein sehr einflussreiches Thema in zahlreichen Unternehmen ist, hat sich ein umfassendes Interesse an diesen Daten entwickelt. Sowohl die Lambda- als auch die Kappa-Architektur stellen modernste Echtzeit-Datenverarbeitungsarchitekturen zur Bewältigung von massiven Datenströmen dar und werden deshalb im Folgenden vorgestellt [80].

3.2.1 Lambda Architektur

Die Lambda-Architektur hat ihren Namen von Nathan Marz [81] erhalten und beschreibt eine generische, skalierbare und fehlertolerante Real-Time-Processing Architektur. Sie bietet einen universellen Ansatz, um eine beliebige Funktion auf eine beliebige Datenmenge anzuwenden [81] [80]. Marz definiert eine allgemeingültige Funktion als eine Funktion, die alle vorhandenen Daten als Eingabe nimmt und ihre Ergebnisse mit geringer Latenzzeit zurückgibt. Die Berechnung von Ergebnissen zu Ad-hoc-Abfragen unter Verwendung des gesamten Datensatzes ist jedoch recht rechenintensiv. Daher verwendet die Lambda-Architektur vorausberechnete Ergebnisse (Views), die mit geringer Latenz abgerufen werden können [82] [80].

Abbildung 6 [83] zeigt die Lambda Architektur. Die Architektur setzt sich zusammen aus drei verschiedenen Schichten. Die Schichten werden Speed-Schicht, Batch-Schicht und Serving-Schicht genannt [83] [80]. Den Schichten ist eine Datenquelle vorgeschaltet. In der Abbildung 6 wird die Datenquelle durch Sensoren angegeben. Es sind allerdings auch andere Datenquellen möglich. Die Übertragung der Daten zur Speed- oder Batch-Schicht kann beispielsweise mit dem Messaging-Service Kafka realisiert werden [80]. Die Lambda Architektur repräsentiert eine hybride Big Data Verarbeitungslösung wie in [Kapitel 3.1.5 Big Data Processing](#) beschrieben. Sowohl ein Batch-Processing als auch ein Real-Time-Processing sind innerhalb der Lambda Architektur realisiert. Konkret findet sich das Batch-Processing in der Batch Schicht wieder, und das Real-Time Processing in der Speed-Schicht.

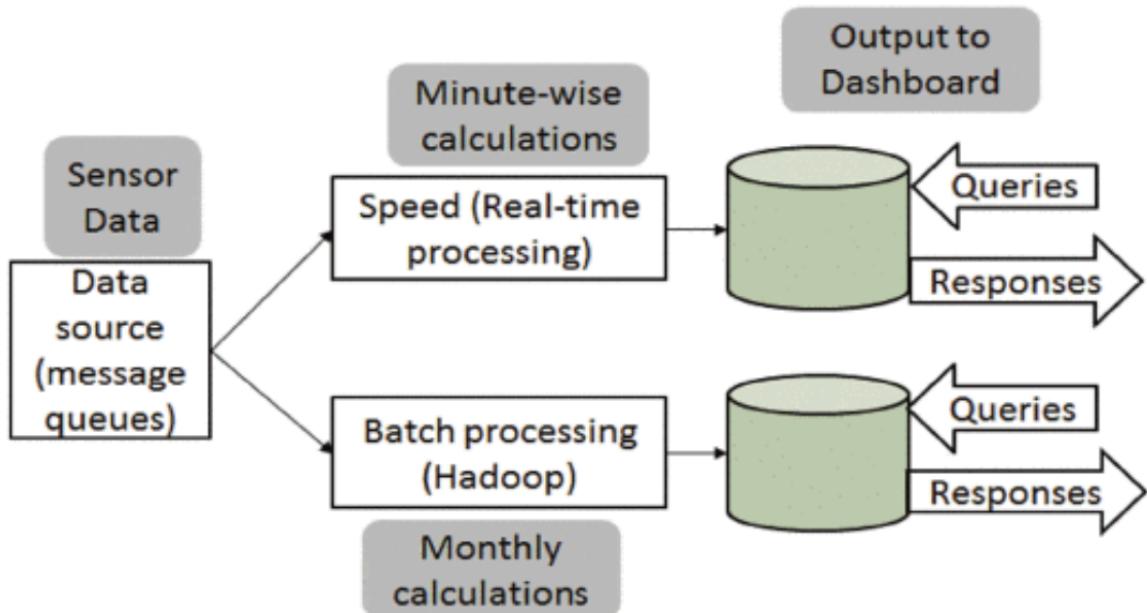


Abbildung 6 - Lambda Architektur

Im Hinblick auf die verwendeten Technologien in den Schichten Batch und Speed, werden die Technologien verwendet, welche in [Kapitel 3.1.5 – Big Data Processing](#) für die jeweiligen Schichten ebenfalls vorgeschlagen werden (es können jedoch auch andere Technologien verwendet werden). Abbildung 7 von Feick et. al. [80] zeigt deren Sichtweise auf die Lambda Architektur mit entsprechenden Technologien. Für die Batch Schicht werden die Tools HDFS und MapReduce verwendet und für die Speed Schicht die Technologie Spark.

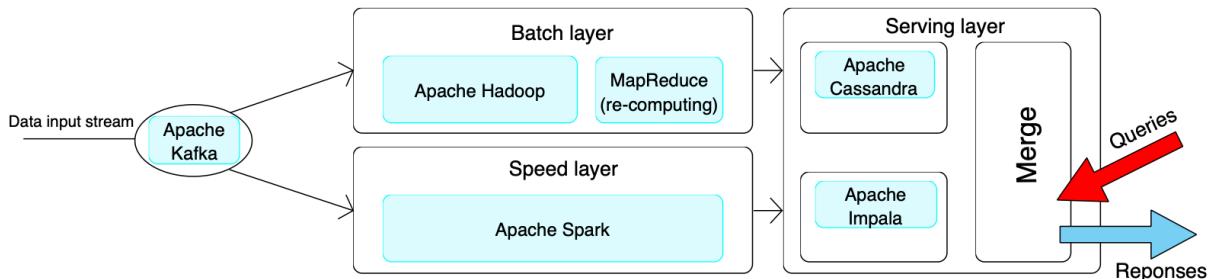


Abbildung 7 - Lambda Architektur und entsprechende Tools

Die Batch-Schicht hat im Wesentlichen zwei Funktionen: Sie speichert zum einen der unveränderlichen Stammdatensatz (sogenannter Master Dataset) und zum anderen ist sie für die Vorberechnung der Batch-Views auf Basis dieses Datensatzes zuständig. Die Speed-Schicht ist für die Indizierung der Real-Time-Views zuständig und kompensiert damit die hohe Latenz der Batch-Schicht. Insbesondere dauert es aufgrund der großen Datenmengen in der Batch-Schicht lange, bis die neuesten Views der Batch-Schicht berechnet sind, was zu einer mangelnden Verfügbarkeit führt. Die Speed-Schicht wird verwendet, um diese Lücke zu schließen, indem sie eine effiziente Möglichkeit für die Abfrage der aktuellsten Daten bietet [82] [80]. Sobald die Batch-Schicht ihre Views neu berechnet hat, verwirft die Speed-Schicht die redundanten Daten, sodass diese von den Views der Batch-Schicht bereitgestellt werden. Außerdem gibt es Abfragen, bei denen die aktuellsten Daten (aus der Speed-Schicht) und die Daten aus der Batch-Schicht benötigt werden. Daher führt die Serving-Schicht die Ergebnisse aus den Views der Batch- und der Speed-Schicht zusammen. Außerdem kümmert sich die Serving-Schicht um die

Indizierung und die Bereitstellung der zusammengeführten Views, um dem Benutzer einen einfachen Zugriff zu ermöglichen [80].

3.2.2 Kappa Architektur

Das Problem bei der Lambda-Architektur ist, dass sie Daten auf zwei verschiedenen Ebenen (Batch- und Speed-Schicht) verarbeitet (die Datenverarbeitung wird auch Data Streams genannt). Aus der Sicht der Entwickler*innen bedeutet dies mehr Aufwand und Komplexität [80]. Kurz nachdem die Lambda-Architektur 2011 von Marz [81] in seinem Blogeintrag "How to beat the CAP theorem" vorgestellt wurde, reagierte Kreps auf diese Architektur und schlug eine verbesserte Version im Jahr 2014 vor [84]. Die von Kreps vorgeschlagene Architektur zielt ebenfalls auf die Verarbeitung von Big Data ab, aber verfügt nicht mehr über eine Batch-Schicht. Bei Kreps wird lediglich ein Stream verarbeitet. Kreps, der zu dieser Zeit bei LinkedIn tätig war und maßgeblich am Messaging-Technologien Kafka beteiligt war, begründet den Namen seiner Architektur wie folgt: „Maybe we could call this the Kappa Architecture, though it may be too simple of an idea to merit a Greek letter.“ [85].

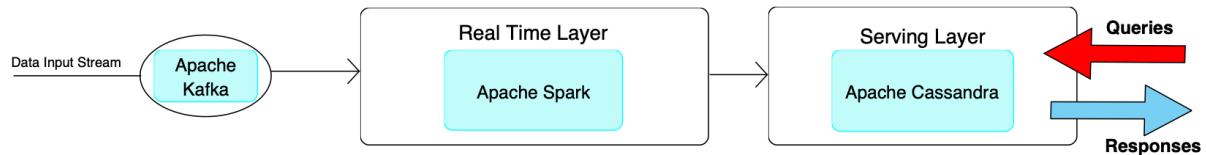


Abbildung 8 - Kappa Architektur

Abbildung 8 von Feick et. al. [80] zeigt die Kappa Architektur. Im Gegensatz zu der Lambda Architektur sind hier nur noch die Real-Time-Schicht und die Serving-Schicht vorhanden. Da ansonsten sich die Architekturbilder nicht weiter unterscheiden, sind dieselben Technologien der Lambda Architektur verwendbar. Wie bei der Lambda Architektur ist eine Datenquelle vorangeschaltet, welche über ein Tool (in diesem Fall Kafka) die Daten zu der Real-Time-Schicht transferiert. Der Aufgabebereich der Real-Time-Schicht erweitert sich durch das Fehlen der Batch-Schicht. Diese Schicht ist für die Ausführung der Stream-Processing-Jobs und die Bereitstellung des Real-Time-Processings verantwortlich [80]. Der Nachteil der Kappa-Architektur ist, dass die eigentlichen Batch-Schicht Aufgaben in der Speed-Schicht verarbeitet werden. Rechenintensive Anwendungen, wie z. B. das Training großer ML-Modelle, können zu einem Problem führen [86].

3.3 Maschinelles Lernen

Zum Analysieren der zuvor prozessierten Daten werden ML-Modelle herangezogen. Die von den Teststationen gesendeten Daten sollen als Input für das ML-Modell dienen. Das ML-Modell selbst hat die Aufgabe zu prognostizieren, ob das durch Sensoren gemessene Teil an der Teststation fehlerhaft ist oder nicht. Die Verwendung von ML-Modellen findet sich aufgrund der Analyse-Eigenschaften der Modelle in der Data Analysis Schicht wieder (siehe [Kapitel 3.1.3 – Big Data Schichten](#)). In diesem Kapitel werden die Grundlagen für das Maschinelle Lernen gelegt. Zur Verfügung gestellt. Zur Implementierung des Prototyps sind die Feinheiten des Trainierens und Verwendens von ML-Modellen von nachrangiger Bedeutung. Ebenso die unterschiedlichen Arten von ML-Modellen und wann welches Modell am besten zum Einsatz kommen sollte. Im Vordergrund steht das Einbinden von ML-Modellen in den Datenverarbeitungs-Workflow des Prototyps. Im Mittelpunkt dieses Kapitels stehen deshalb die Möglichkeiten, wie ML-Modelle

effizient abgelegt und geladen werden können. Bevor die Möglichkeiten zum Ablegen und Laden der ML-Modelle aufgezeigt werden, gilt es den Begriff Maschinelles Lernen abzugrenzen.

3.3.1 Definition

Das Konzept des Maschinellen Lernens geht auf bis nach dem 2. Weltkrieg zurück. Wissenschaftler*innen erforschten dort die ersten Ansätze für Maschinelles Lernen. Beispielsweise geht der Entwurf des ersten Perceptrons auf Rosenblatt im Jahr 1958 zurück. Dies bildet heute die Grundlage für die künstlichen neuronalen Netze (ANN). Die Forschung in dieser Zeit waren allerdings eher theoretischer Natur, da die Ressourcen (in Form von Daten oder Rechenleistung) begrenzt waren [87].

Durch neue Algorithmen und Techniken (zum Beispiel Deep Learning), große Mengen an Daten (Big Data) und parallelisiertes Verarbeiten von Daten (Graphics processing unit (GPU)) hat sich im ersten Jahrzehnt des neuen Jahrtausends das Konzept des Maschinellen Lernens durchgesetzt [88]. Eine der ersten Definitionen zum Thema Maschinelles Lernen ist auf Arthur Samuel [89] im Jahre 1959 zurückzuführen: „Field of study that gives computers the ability to learn without being explicitly programmed.“. Aufbauend darauf setzt Tom Mitchell [90] eine breit gefächerte Definition des Begriffs Maschinelles Lernen mit: “A Computer program is said to learn from an experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.“. In den Computerwissenschaften repräsentiert der Begriff “Experience” Daten [91]. Ziel ist also einem Computer beizubringen, eine Aufgabe auszuführen durch das automatische Lernen aus Daten. Die Daten stellen dementsprechend die Grundlage für das Lernen des Computers.

3.3.2 Supervised Learning und Unsupervised Learning

Bei Maschinellen Lernen wird zwischen Supervised Learning und Unsupervised Learning unterschieden [92] [93]. Bei Supervised Learning wird ein Algorithmus darauf trainiert aus Daten zu lernen, bei dem die Lösung bekannt ist. Beim Supervised Learning geht es darum, eine Abbildung zwischen einem Satz von Eingabeveriablen X und einer Ausgabeveriablen Y zu lernen und diese Abbildung anzuwenden, um die Ausgaben für ungewohnte Daten vorherzusagen [94]. Im Fachjargon werden die Ausgabeveriablen auch Label genannt. Das Präsenssein solcher Label unterscheidet Supervised Learning von Unsupervised Learning. Beim Unsupervised Learning werden Muster in den Eingabeveriablen X gesucht, ohne dass die Ausgabeveriablen Y vorhanden sind. Im Rahmen des Forschungsprojektes PREFERML sollen ML-Modelle auf bereits bekannte Fehler trainiert werden. Das heißt, dass der Datensatz gelabelt ist und deshalb das Supervised Learning Anwendung findet.

Für das Supervised Machine Learning gibt es eine große Anzahl an bereits existierenden Algorithmen. Dazu gehören: Support-Vector-Machine (SVM), ANN, Logistic Regression (LOGREG), Naive Bayes (NB), K-Nearest-Neighbour (KNN), Random Forest (RF) und Decision Tree (DT) [95]. Jeder Algorithmus hat seine Vor- und Nachteile und kommt bei unterschiedlichen Anwendungen zum Einsatz. Typischerweise werden bei der Entwicklung mehrere Algorithmen implementiert und dann verglichen. Zum Beispiel eignet sich bei Textklassifikation im Kontext von Natural Language Processing (NLP) der NB-Algorithmus besser [96].

Zum Trainieren der Algorithmen werden die Datensätze in einen Test- und einen Trainingsdatensatz aufgeteilt. Als erstes dienen die Trainingsdaten als Input für den Algorithmus. Da die Label bekannt sind und mit den Trainingsdaten mitgeliefert werden, kann der Algorithmus die Zusammenhänge zwischen den Features und dem entsprechenden Label lernen. Nachdem Training

kann das ML-Modell mit dem Testdatensatz validiert werden. Dabei wird der Testdatensatz aufgeteilt in Features und das Label. Zuerst werden die Features in das ML-Modell geladen und es werden Vorhersagen gemacht. Diese Vorhersagen werden dann mit dem entsprechenden Label abgeglichen. Das Ergebnis ist eine Genauigkeit (oder Accuracy) des ML-Modells. Ist die Genauigkeit nicht zufriedenstellen, so können die Parameter des ML-Modell nachjustiert werden.

Für den Erfolg der ML-Modelle spielen verschiedene Faktoren eine Rolle. An erster Stelle steht die Darstellung und Qualität der Daten. Damit die Qualität gewährleistet werden kann, müssen verschiedene Schritte im Voraus ausgeführt werden. Die Gesamtheit der Schritte wird unter dem Begriff Preprocessing zusammenfasst. Das Preprocessing umfasst Datenbereinigung, Normalisierung, Transformation, Merkmalsextraktion und -auswahl usw. Das Produkt des Preprocessing ist der endgültige Trainingssatz [97]. Das Preprocessing für ML-Modelle findet in der in Kapitel 3.1.3. – Big Data Schichten genannten Schicht Data Acquisition statt. Das Preprocessing ist somit Teil des Data Preprocessing und muss bei der Implementierung berücksichtigt werden.

3.3.3 Speichern und Laden von ML-Modellen

Wie in [Kapitel 2 - Forschungsprojekt PREFERML](#) erwähnt, resultieren aus den dort beschriebenen verschiedenen Herausforderungen (Im Speziellen [Kapitel 2.1.2.1 Herausforderungen der Fertigungsdomäne](#)) eine vielfältige Anzahl von ML-Modellen. Die ML-Modelle unterscheiden sich also in Typ (SVM, ANN, LOGREG, NB, KNN, RF oder DT), Input-Features, Fehlerkategorien (Label), Auswertungen und Datenquellen. Da die Modellerstellungen rechenintensiv sind, und diese bei Echtzeitverarbeitungen zu langsam sind [86], müssen diese im Voraus erstellt werden. Konkret bedeutet dies, dass die ML-Modelle nach dem Erstellen abgespeichert und beim Verwenden geladen werden müssen. Tools wie Spark bieten durch ihre vorhandenen ML-Bibliotheken bereits Methoden für das schnelle Speichern und Laden von ML-Modellen [98].

Der im Rahmen dieser Forschungsarbeit entwickelten Prototyp implementiert die DAtenverarbeitung im ersten Schritt ohne ein Tool wie Apache Spark, weshalb für das Speichern und Laden andere Ansätze in Betracht gezogen werden müssen. Die ML-Modelle können grundsätzlich auf verschiedene Arten exportiert und geladen werden. Durch die Exportformate ist es möglich die Daten aus dem lokalen Speicher zu laden oder auf der Datenbank abzulegen. Ebenso ist es möglich diese Formate wieder vom lokalen Speicher oder von der Datenbank zu laden. Der Prototyp wird wie in der Anforderungsanalyse beschrieben in Python programmiert. Die ML-Modelle werden mit der Library scikit-learn kreiert. Die ML-Modelle von scikit-learn lassen sich durch folgende Formate Exportieren und Laden [99]:

- Pickle
- Joblib

Pickle

Das pickle-Modul implementiert Binärprotokolle zum Serialisieren und Deserialisieren einer Python-Objektstruktur. Pickling ist der Prozess, bei dem eine Python-Objekthierarchie in einen Bytestrom umgewandelt wird, und Unpickling ist der umgekehrte Vorgang, bei dem ein Bytestrom wieder in eine Objekthierarchie umgewandelt wird. Pickling (und Unpickling) wird alternativ auch als Serialisierung, Marshalling oder Flattening bezeichnet [100].

Joblib

Im speziellen Fall von scikit-learn kann es besser sein, den joblib-Ersatz von pickle (dump & load) zu verwenden, der bei Objekten, die intern große Numpy-Arrays tragen, wie es bei angepassten scikit-learn Modellen oft der Fall ist, effizienter sind [99].

Neben Pickle und Joblib gibt es auch weitere Ansätze zum Exportieren, wie zum Beispiel mit dem Format JSON, allerdings werden diese von scikit-learn nicht unterstützt und sind hinsichtlich der Sicherheit weniger zuverlässig [101].

4 Implementierung des Prototyps

Die Implementierung des Prototyps baut auf den erarbeiteten Use-Cases in [Kapitel 2.2 – Anforderungsanalyse](#) auf. Ausgehend von den Forschungsfragen, ist das Ziel der Implementierung die Erstellung einer Architektur und deren Umsetzung, bei einer einheitlichen Datenverarbeitung angeboten werden kann, unabhängig davon, wie viele Schichten die Architektur besitzt. Für die Implementierung der Datenverarbeitungsarchitektur kommen beide – in [Kapitel 3.2 Big Data Processing Architekturen](#) vorgestellten Architekturen (Kappa und Lambda Architektur) – in Frage. Durch die Singularität des Datenstroms in der Kappa Architektur ist das Vorhandensein einer einheitlichen Datenverarbeitung von Grund auf gegeben. Die Komplexität liegt bei der Kappa Architektur auf der Umsetzung und Implementierung. Gibt es nur einen einzigen Datenstrom, so müsste für jede Fehlervorhersage einer Messung einer Teststation, ein oder mehrere ML Modelle trainiert und generiert werden. Abhängig von der Anzahl an Teststationen und der Produktionsauslastung steigt die Anzahl der zu verarbeiteten Fehlervorhersagen. Die resultierende Rechenauslastung kann zu einer Überlastung des Rechenzentrums der Produktionsumgebung führen. Des Weiteren entsteht durch die hohen Rechenleistungen eine längere Vorbereitungszeit zum Erstellen von Vorhersagen. Der Aspekt der Echtzeitverarbeitung geht deshalb verloren. Um den hohen Ressourcenbedarf der Kappa im Kontext dieser Forschungsarbeit entgegenzuwirken, wird die Lambda in Betracht gezogen. Mit zwei Verarbeitungsströmen (Batch- und Speed-Schicht) können ML Training und Fehlervorhersagen voneinander getrennt werden. Der Ressourcenbedarf sinkt und es ist möglich die Ergebnisse der Fehlervorhersagen in Echtzeit anzuzeigen. Durch die Trennung der Datenströme, muss die Datenverarbeitung auf beide Datenströme angewendet werden. Wie bei der Lambda Architektur in Kapitel 3.2.1 beschrieben, ist die Verarbeitung der verschiedenen Datenströme oft verschieden. Dies führt zu einer größeren Komplexität des Systems und erfordert einen Verwaltungsaufwand. Zum Unterminieren dieses Nachteils wird im Folgenden eine Lambda Architektur vorgestellt, bei dem die Datenverarbeitung auf beiden Schichten (beiden Datenströmen) gleich abläuft.

Neben der einheitlichen Datenverarbeitung auf beiden Schichten (Speed und Batch) sollen verschiedene Implementierungsvarianten in Bezug auf Performance getestet werden. Zum einen gibt es eine Implementierungsvariante, bei der die Fehlervorhersagen durch REST Aufrufe direkt von der Teststation initiiert werden. Zum anderen gibt es eine Implementierungsvariante, bei der das Speichern eines gemessenen Events einer Teststation auf einer replizierten Datenbank als Auslöser für die Fehlervorhersage dient. Das Auslösungsereignis beim Speichern eines Events wird mit MongoDB Change Streams umgesetzt. Die Datenverarbeitung selbst wird aus Einfachheitsgründen ohne die den Grundlagen vorgestellten Technologien in Tools umgesetzt. Die Tools und Technologien dienen allerdings als Grundlage für weitere Forschungsarbeiten.

Die in den folgenden Kapiteln enthaltenen Architekturdiagramme zeigen auf, wie der Prototyp implementiert wurde. Die Architekturbilder sind mit der C4-Modellierung umgesetzt [102]. Zur Übersicht wird dabei zuerst das System Context Diagram gezeigt und beschrieben. Darauffolgend werden die Detaillierungen der Software mittels dem Container Diagram, als auch mit dem Component Diagram spezifiziert und erläutert. Zum Verständnis der Architekturen wird zuvor ein Überblick über die Struktur der Daten und der Produktionsumgebung geschaffen. Diese dient als Ausgangslage für die folgenden Architekturen und Beschreibungen. Anschließend an die Beschreibungen der Architektur und der Implementierung werden Tests über die verschiedenen Implementierungsvarianten durchgeführt. Die Performance steht bei den Tests im Vordergrund.

4.1 Aufbau der Daten und der Produktionsumgebung

Gemäß dem Bericht „Supporting Quality Assessment in Manufacturing by Machine Learning: First Results of PREFERML Project“ besteht eine mustertypische Produktionslinie aus mehreren Teststationen, welche verschiedene Messungen an den zu produzierenden Teilen bzw. Produkten durchführen. Um mangelhafte Teile zu identifizieren werden die Messdaten der Teststation vom System bearbeitet [13]. Das Durchführen von Messungen an den Teststationen ist somit mit den Qualitätsprüfungen von Peter Hohenberger in „Qualitätsmanagement in der Produktion“ gleichzusetzen [16]. Es wird angenommen, dass die Messungen mit Hilfe von Sensoren durchgeführt werden (wobei andere Messmittel auch in Frage kommen würden). Die Messungen werden an Merkmalen, wie Länge, Höhe oder Breite durchgeführt. Im Folgenden werden diese Merkmale Features genannt. Da jede Teststation unterschiedliche Features misst, werden die Features mit einer numerischen Nummer versehen (Feature1, Feature2, ...). Grundsätzlich können die Features sämtliche Ausprägungen der Datentypen annehmen (Integer, Float, String, Date ...). Bei einer späteren Automatisierung müssen alle Ausprägungen bearbeitet werden können.

Resultierend aus der Gegebenheit, dass jede Teststation eine Summe von verschiedenen Merkmalen misst, ist jede Teststation einzigartig [13]. Das bedeutet, dass beim späteren Arbeiten mit den Messdaten der Teststationen eine eindeutige Identifizierung der Teststationen vorliegt. Diese eindeutige Identifizierung wird in dieser Forschungsarbeit mit der Kennzeichnung „Teststation_ID“ realisiert. Innerhalb einer Produktionslinie sind die Teststationen hintereinandergeschaltet. Diese Reihenfolge muss später ebenfalls in der Teststation_ID ersichtlich sein, da die Daten über verschiedene Teststationen hinweg (innerhalb einer Produktionslinie) zusammengefügt werden müssen. Das Zusammenfügen der Daten ist für das Training der ML-Modelle notwendig, was im [Kapitel 2.2 Anforderungsanalyse](#) genauer erläutert wird. Des Weiteren, muss in der Teststation_ID ersichtlich sein, in welcher Produktionslinie sich die Teststation befindet. Ansonsten besteht die Gefahr, dass Daten von verschiedenen Teststationen über verschiedene Produktionslinien hinweg zusammengefügt werden. Aus Gründen der Einfachheit wird in dieser Forschungsarbeit davon ausgegangen, dass nur eine Produktionslinie vorliegt. Die Berücksichtigung der Produktionslinie in der Teststation_ID ist in diesem Kontext deshalb nicht notwendig. Die Teststation_ID wird deshalb wie folgt umgesetzt: *Teststation1 – Teststation2 – Teststation3 - ...*

Für das Zusammenfügen der Daten über die verschiedenen Teststationen hinweg ist ein eindeutig identifizierender Schlüssel nötig. Da die Messdaten von den Teststationen immer genau zu einem Teil gehören, ist der eindeutige Schlüssel für das Zusammenfügen die ID des Teils. Diese wird zur Realisierung der Implementierung „Component_ID“ genannt.

Jedes Teil, welches die Produktionslinie durchläuft, hat zusätzlich noch weitere, nicht messbare Merkmale bzw. Eigenschaften. Beispiele für diese nicht messbaren Eigenschaften sind Zugehörigkeiten zu Produkttypen, Kategorien, Margen, usw. In der späteren Realisierung werden deshalb exemplarisch zwei Merkmale inkludiert. Zum einen „Product_Type“ (ist bei Events jeder Teststation inkludiert) und zum anderen „Category“ (ist nur bei Events der Teststation1 inkludiert). Zudem wird beim Messen jeweils ein Zeitstempel und eine Fehlerspalte hinzugefügt, welcher sich durch die Merkmale „Date“, „Time“ und „Error_Message“ widerspiegelt. Die Spalte Error_Message kann verschiedene Fehlerarten enthalten. Um beim Trainieren der ML-Modelle eine möglichst hohe Genauigkeit zu erzielen, empfiehlt es sich mehrere ML-Modelle zu erzeugen und nur auf einen einzelnen Fehler die Modelle zu trainieren [12].

4.2 System Context Diagram

Abbildung 9 zeigt das System Context Diagram des Prototyps. Das System Context Diagram ist für beide Implementierungsvarianten (REST und Change Streams) gleich. Der Prototyp hat oberflächlich betrachtet nur zwei Benutzer. Der Data Scientist ist für die Implementierung der Software verantwortlich und ableitend daraus für das Trainieren der ML-Modelle. Sobald das System in Betrieb ist, senden die Teststation in den Produktionslinien Events für die Fehlervorhersage. Für die Fehlervorhersage bearbeitet das Softwaresystem die Daten, lädt die vom Data Scientist erstellten ML-Modelle und macht Vorhersagen. Die Vorhersagen werden dann auf einer Webseite angezeigt, welche der zweite Benutzer (Quality Engineer) entgegennehmen kann. Dem Quality Engineer obliegt es dann bei Abweichungen in den Produktionsprozess einzugreifen.

Die in Abbildung 10 und Abbildung 11 gezeigten Container **Simulate test stations** und **Event generation** können ebenfalls als Akteure angesehen werden. Da diese Akteure allerdings Teil der Implementierung sind und somit in das Event Processing System gehören, zeigt Abbildung 9 diese Akteure nicht an. Sie werden deshalb als Container in den Container und Component Diagrams dargestellt.

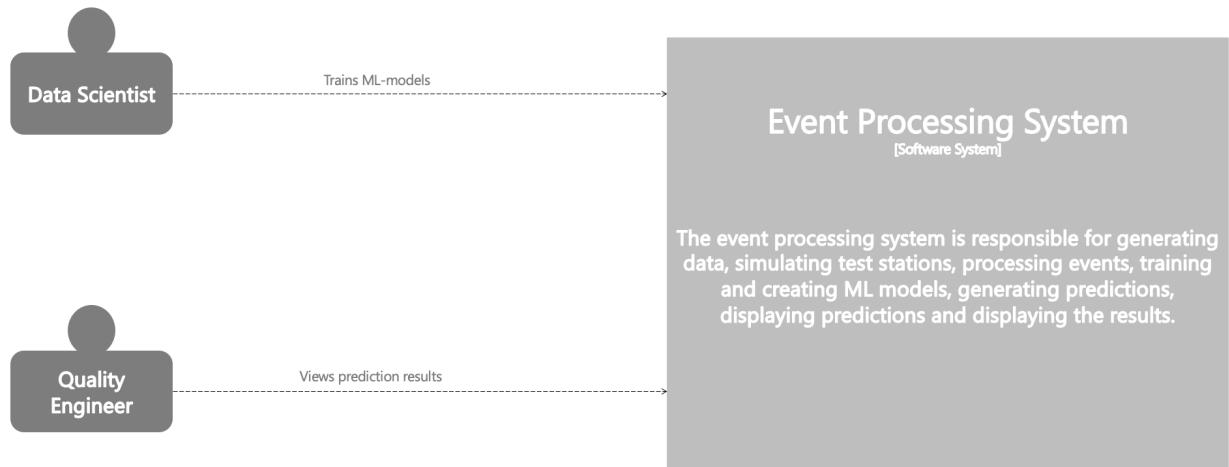


Abbildung 9 - Event Processing System

4.2.1 REST Implementation

Abbildung 10 zeigt das Container Diagramm der REST Implementierung. Die Implementierung zeigt einen Überblick über die einzelnen Container und wie sie miteinander interagieren. Zusätzlich werden die Abhängigkeiten der Container zu den Benutzern dargestellt. Für einen besseren Überblick wird in der folgenden Beschreibung die Architektur chronologisch abgearbeitet, wie es im mustertypischen Ablauf unter Realbedingungen ebenfalls abläuft:

1. Damit das Softwaresystem arbeiten kann müssen Daten vorhanden sein. Dieser Schritt wird künstlich vom Data Scientist ausgeführt, in dem er das „**Event generation**“ Skript aufruft. Das Skript erzeugt csv-Datensätze welche vorerst auf dem lokalen File System gespeichert werden. Später werden diese CSV-Datensätze in die EventDB (MongoDB) geladen.
2. Sobald die Testdaten in der EventDB gespeichert sind, kann der Data Scientist mit dem **Model Training** beginnen. Das **Model Training** lädt die Daten aus der EventDB und initiiert das **Event processing**. Das Event Processing ist verantwortlich für das Cleaning der Events, das Ableiten von weiteren Merkmalen (Features), sowie das Zusammenfügen von Events über verschiedene Teststationen hinweg (Spezifikationen zu dem Event

processing sind in [Kapitel 4.5 – Event Processing](#) beschrieben). Nach dem **Event Processing** trainiert das **Model Training** verschiedene ML-Modelle und legt diese auf dem lokalen File System ab.

In Bezug auf die Lambda Architektur ist das **Model Training** der Batch-Schicht zuzuordnen. In einer echten Produktionsumgebung werden kontinuierlich neue Daten in die EventDB geschrieben. Durch erneutes Training der ML-Modelle – welche nach Erkennen des Data Scientist ausgeführt werden kann – können auf die Konzept Drifts in den Daten reagiert werden ([Kapitel 2.1.2 Probleme und Herausforderungen bei Datenanalysen im Produktionsumfeld](#)).

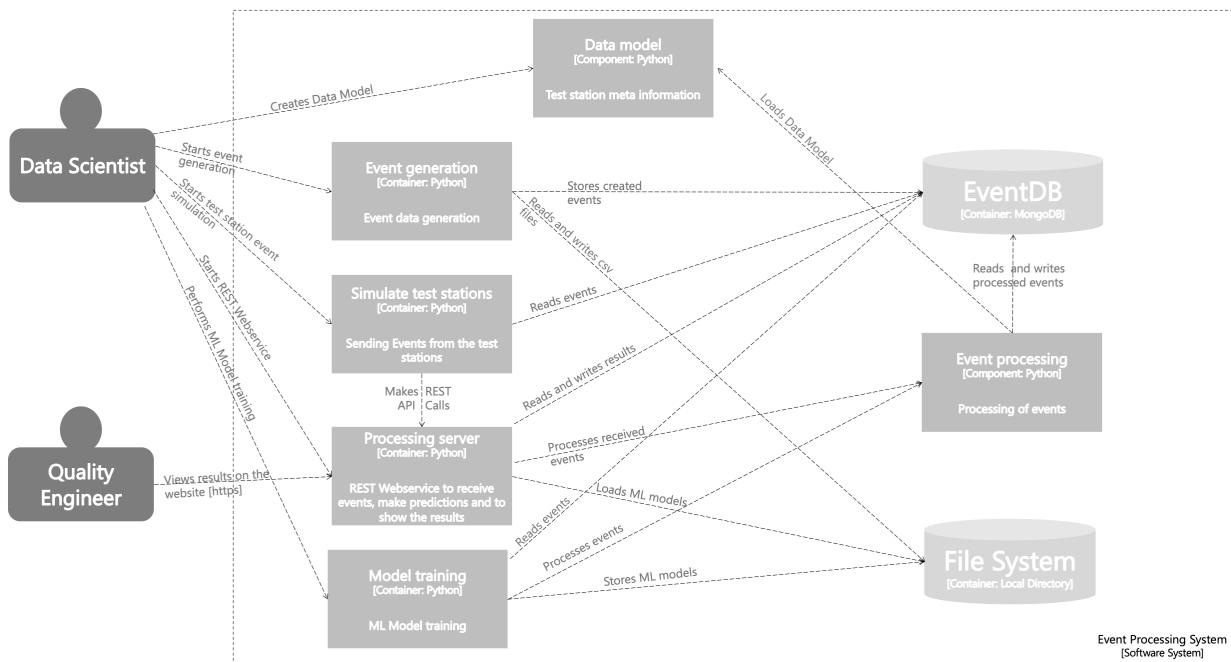


Abbildung 10 - REST Implementation

- Nach dem **Model training** können die Teststationen Events zur Fehlervorhersage senden. Damit die Events über einen REST Aufruf verarbeitet werden muss zunächst der REST-Server gestartet werden. Der REST-Server ist in der REST Implementierung zur Vereinfachung der gleichen Server wie der Web-Server für die Webseite. Zum Starten des Servers ruft der Data Scientist deshalb **Processing Server** auf.

Der REST Server ist für die Verarbeitung von Events verantwortlich. Werden Events von Teststationen gesendet, wird erneut das **Event processing** aufgerufen. Nach der Datenverarbeitung werden für die Fehlervorhersage entsprechende ML-Modelle von File System geladen. Die Ergebnisse der Vorhersagen werden auf der EventDB gespeichert.

Die Datenverarbeitung durch den REST-Server ist im Kontext der Lambda Architektur auf die Speed-Schicht zuzuordnen. Bei jedem Event, welches von den Teststationen gesendet wird, wird diese Datenverarbeitung aufgerufen. Das Auftreten einer Fehlervorhersage ist infolgedessen relativ häufig und darf für eine Echtzeitverarbeitung nicht viel Zeit in Anspruch nehmen.

- Aufgrund der künstlichen Produktionsumgebung muss der Data Scientist das Senden von Events der Teststationen zur Fehlervorhersage manuell auslösen. Mit dem Aufrufen des

Simulate test stations Container werden die gespeicherten Daten der Event generation aus der EventDB geladen und mit einer Schleife über einen REST Aufruf an den REST Server gesendet.

- Als letztes kann der Quality Engineer auf die Ergebnisse mit dem Aufrufen einer Webseite zugreifen. Mit dem Aufruf der Webseite werden über den **Processing Server** die Ergebnisse der Fehlervorhersagen aus der EventDB geladen und angezeigt.

4.2.2 Change Streams Implementation

Die Change Streams Implementierung ist der REST Implementierung sehr ähnlich. Abbildung 11 zeigt das Container Diagramm der Change Streams Implementierung. Die Container **Event generation**, und **Model Training** bzw. die Schritte 1 und 2 aus vorherigem Kapitel sind 1:1 auf die Change Streams Implementierung übertragbar. Lediglich die Container **Simulate test stations** und **Result presentation** sind verschieden. Außerdem kommt noch ein weiterer Container **Change Stream Service** hinzu

Im Gegensatz zur REST Implementierung gibt es bei der Change Streams Implementierung keinen REST Server, an den die Events der Teststationen gesendet werden können. Bei der Change Streams Implementierung wird ein Event ausgelöst, sobald eine Teststation ein Event in der EventDB speichert. **Simulate test stations** funktioniert bei der Change Streams Implementierung deshalb nicht mehr mit REST Aufrufen, sondern die Simulation der Sendungen von Events überfolgt über das erneute Abspeichern der Events in die EventDB.

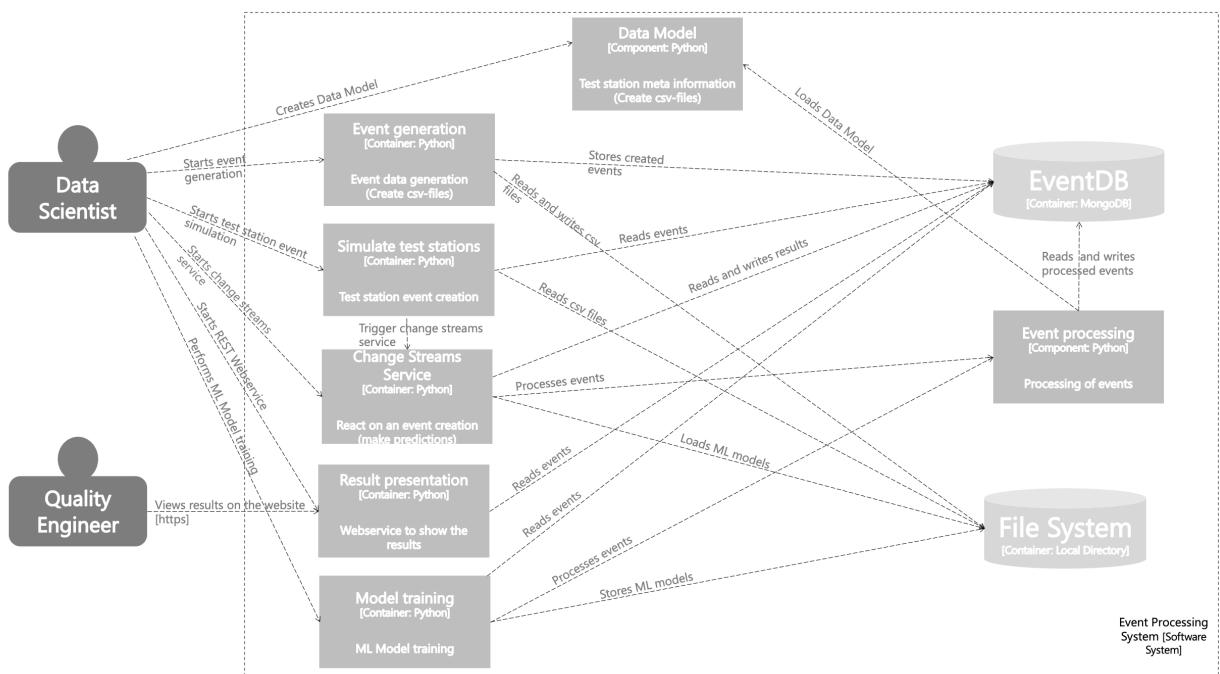


Abbildung 11 - Change Streams Implementation

Als Ersatz für den REST Server zur Datenverarbeitung gibt es in der Change Streams Implementierung einen **Change Streams Service**. Werden Events gespeichert, dann sorgt der Change Streams Service für den Aufruf des **Event processings**, das Laden der entsprechenden ML-Modelle und das Speichern der Ergebnisse auf der EventDB. Der Change Streams Service wird ebenfalls vom Data Scientist ausgeführt.

Da der REST Server in der Change Streams Implementierung nicht existiert, muss die Result Presentation separat implementiert werden. Der Processing Server wird wie bei der REST Implementierung vom Data Scientist gestartet. Der Quality Engineer kann durch das Laden der Webseite, ebenfalls die Vorhersageergebnisse der ML Modelle anzeigen lassen.

Nach den Container Diagrammen werden im Folgenden die einzelnen Container und dessen Komponenten beschrieben.

4.3 Event Generation

Die Event Generation ist in beiden Implementierungsvarianten gleich realisiert, weshalb dieser Container nur einmal beschrieben wird. Für die Datenerstellung wird ein Pre-Data-Generator verwendet, welcher von den Wissenschaftlichen Arbeitern des Forschungsprojektes PREFERML zur Verfügung gestellt wurde. Grundlage für die Datenerstellung sind yaml-Dateien. Der Data Scientist erstellt für jede Teststation eine yaml-Datei. Dieser Schritt wird in Abbildung 12 durch die Komponente **Test station config** dargestellt. Die Datenstrukturen in den Konfigurationen der Teststationsdatensätze entspricht den Beschreibungen in der Anforderungsanalyse (Use Cases: Erstellen Datensatz Teststation 1, Erstellen Datensatz Teststation 2, Erstellen Datensatz Teststation 3).

Nach Erstellung der Konfigurationsdateien startet der Data Scientist die Datengenerierung mittels eines Skriptes (**Start Data Generation**). Beim Aufrufen des Skriptes über das Terminal muss der Data Scientist drei Argumente übergeben. Das erste Argument ist der Name der Konfigurationsdatei. Das zweite Argument ist die Anzahl an zu erstellenden Events. Das dritte und letzte Argument ist der Name des zu erstellenden Datensatzes. **Start Data Generation** übergibt die Argumente an **Data Generation**. **Data Generation** lädt die entsprechende Konfigurationsdatei vom File System und erstellt einen csv-Datensatz mit der angegeben Anzahl an Events (Zeilen). Insgesamt werden bei der Implementierung des Prototyps drei Teststationen simuliert, weshalb der Data Scientist drei Konfigurationsdateien erstellen muss. Für jede Konfigurationsdatei muss der Data Scientist jeweils die Datengenerierung ausführen. Die csv-Dateien werden auf dem lokalen File System abgelegt.

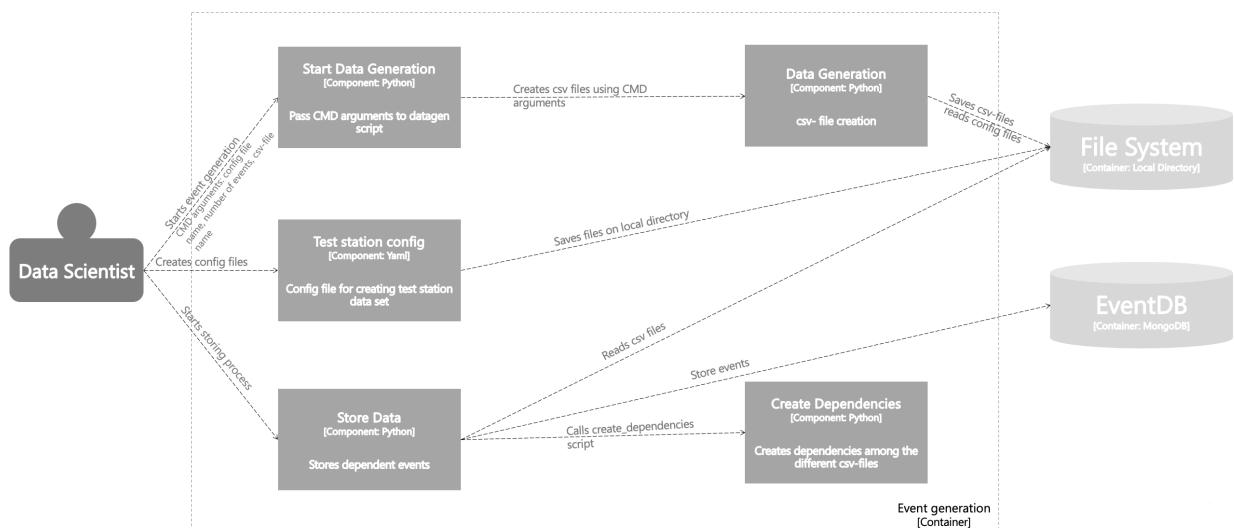


Abbildung 12 - Event Generation

Nach dem erfolgreichen Erstellen und Ablegen der csv-Dateien auf dem lokalen File System, müssen die Daten für eine weitere Datenverarbeitung auf der EventDB abgelegt werden. Dazu ruft der Data Scientist **Store Data** auf, welches die drei csv-Dateien vom lokalen File System in ein pandas-Dataframe lädt. Bevor die Dataframes in die EventDB geladen werden, müssen Abhängigkeiten zwischen den Datensätzen geschaffen werden, welche vom Data-Pre-Generator nicht geschaffen werden können. **Create Dependencies** in Abbildung 12 repräsentiert diese Funktionalität. Tritt in einer realen Produktionsumgebung beispielsweise ein Fehler bei der Teststation1 bei einem Produkt auf, so wird dieses Produkt aussortiert (ggf. nachbearbeitet) und gelangt nicht zu der Teststation2. Für die Dataframes bedeutet dies, dass wenn ein Event mit der Component_ID X eine positive Fehlermeldung bei dem Dataframe der Teststation1 hat, es (das Event mit der Component_ID X) bei den Dataframes der Teststationen2/3 nicht mehr auftreten darf. Analog dazu, müssen Events von anderen Teststationen (Teststationen2/3), bei Auftreten von Fehlermeldungen, von Dataframes nachfolgender Teststationen entfernt werden. Realisiert wird dies, in dem eine Liste von Component_IDs angelegt wird, bei der die Fehlermeldungs-spalte positiv ist. Die Liste wird mit den Dataframes nachfolgender Teststationen abgeglichen und es werden alle Events mit den Component_IDs aus der Liste aus dem Dataframe entfernt. Die Liste von Component_IDs wird für jedes Dataframe der Teststationen angelegt und mit den nachfolgenden Dataframes abgeglichen. Lediglich das Dataframe der letzten Teststation erfordert keine Erstellung einer Component_ID Liste, da keine Events nachfolgender Dataframes mehr gelöscht werden können. Neben dem Entfernen von Events bei positiver Fehlermeldung über die Dataframes der Teststationen hinweg, schafft es der Data-Pre-Generator ebenfalls nicht, die Produkttypen (PT1 oder PT2) pro Component_ID über die Dataframes der Teststationen hinweg konsistent zu halten. Ohne die Herstellung von Abhängigkeiten, kann es vorkommen, dass ein Produkt mit Component_ID 1 im Dataframe der Teststation1 zum Product_Type PT1 zugeordnet ist und im Dataframe der Teststation2 zum Product_Type PT2 zugeordnet ist. Damit die Konsistenz von Product_Type pro Component_ID gewährleistet ist, werden vom Dataframe der Teststation1 zwei Listen mit Component_IDs erzeugt. Die erste Liste enthält alle Component_IDs bei der der Product_Type PT1 ist. Die zweite Liste hingegen enthält alle Component_IDs bei der der Product_Type PT2 ist. Diese zwei Listen werden für alle Dataframes der nachfolgenden Teststationen genommen und abgeglichen.

Erst nach der Herstellung der Abhängigkeiten können die Dataframes in der Event DB abgelegt werden. Zur Einfachheit werden alle Dataframes auf derselben Collection gespeichert.

4.4 Model Training

Das **Model Training** ist wie in Kapitel 4.2.1 – REST Implementation der Batch-Schicht der Lambda Architektur zuzuordnen. Es erfolgt bei der REST Implementation und der Change Streams Implementation gleich, weshalb diese Beschreibung für beide Implementierung gilt. Abbildung 13 zeigt das Container Diagram von Model Training. Beim erstmaligen Verwenden des Software-Systems, oder nach Ermessen des Data Scientist zum Reagieren auf Konzept Drifts, wird das **Model Training** ausgeführt. Der Data Scientist übergibt beim Aufruf des **Start Batch Training** über das Terminal zwei Argumente. Zum einen die Information über die Teststation und zum anderen über den Produkttyp. Dadurch hat der Data Scientist Einfluss darauf, welche Teststationen bzw. welche Produkttypen genau trainiert werden sollen. Es besteht ebenso die Möglichkeit alle Modelle von sämtlichen Teststationen und Produkttypen zu trainieren. **Start Batch Training** lädt, je nach Terminal Argumenten, alle oder nur spezielle Events aus der EventDB. Anschließend wird das **Event Processing** Skript aufgerufen, bei dem die Daten verarbeitet werden. (Genauere Informationen zum Event Processing sind im nächsten Kapitel beschreiben.) Zu der Datenverarbeitung gehört die Datenbereinigung, das Ableiten von weiteren Features, das Zusammenfügen von Events über Teststationen hinweg und das Speichern der

verarbeiten Daten auf separaten Collections in der EventDB. Anschließend werden die verarbeiteten Daten dem **Start Batch Training** zurückgegeben. Damit Vorhersagen mit supervised ML-Modellen gemacht werden können, werden Label zu den entsprechenden Events benötigt. Die Label werden über das **Load Label** Skript geladen und den Events angehängt. Die Label für die Events einer Teststation X sind immer die Fehlermeldungen der entsprechenden Events der Teststation X+1. Beim **Event Processing** werden alle prozessierten Events mit den zugehörigen Fehlermeldungen in der EventDB gespeichert. Das Load Label Skript kann daraus resultierend für Events der Teststation X über die Component_ID die Fehlermeldungen der Events der Teststation X+1 aus der EventDB laden. Die Label werden den entsprechenden Events angehängt und dem **Start Batch Training** zurückgegeben. Das **Load Label** Skript kann nur auf Events der Teststationen 1/2 angewendet werden. Die Teststation 3 ist in der Simulation des Prototyps die letzte Teststation, weswegen keine Label der nachfolgenden Teststation geladen werden können.

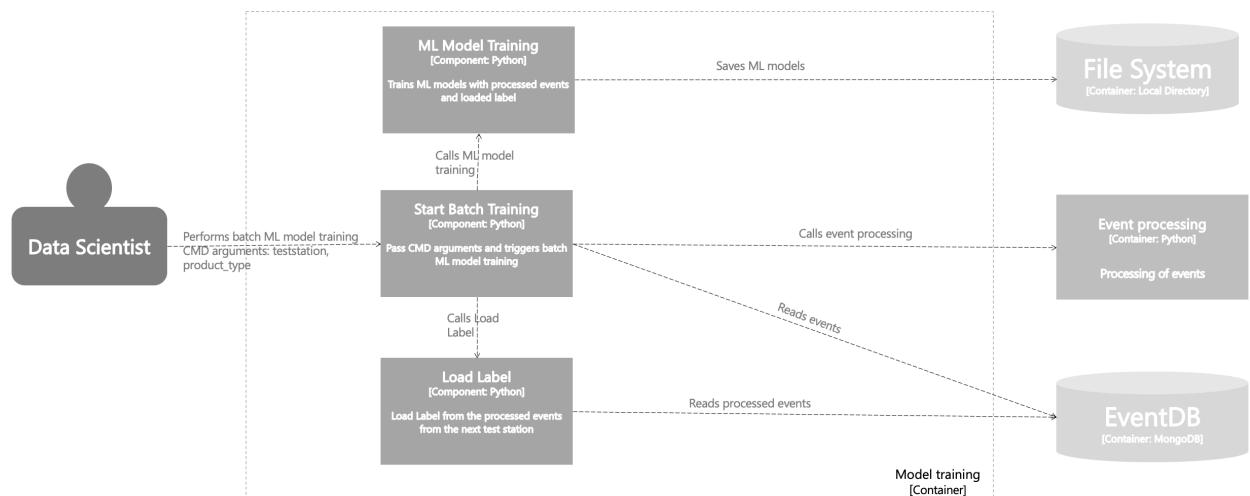


Abbildung 13 - Model training

Für das ML-Model Training sind im Anschluss daran alle Anforderungen erfüllt und das **Start Batch Training** ruft das **ML Model Training** Skript auf. Wie in [Kapitel 2.1.2 Probleme und Herausforderungen bei Datenanalysen im Produktionsumfeld](#) beschrieben, erreichen ML-Modelle die besten Ergebnisse, wenn sie auf nur einen Fehlertyp trainiert werden. Die Events der Teststationen werden deshalb nach Fehlertyp gruppiert. Zusätzlich werden sie darauffolgend in ihren Produkttypen unterteilt. Für jede Teststation gibt es insgesamt vier Datensätze, die als Input für das Training der ML-Modelle erstellt wurden. Aus vier Datensätzen resultieren vier Modelle pro Teststation. In der Simulation des Prototyps können für die Teststationen 1/2 Vorhersagen gemacht werden, da nur für diese Teststationen Label geladen werden konnten. Innerhalb des Prototyps werden deshalb insgesamt acht ML-Modelle erstellt. Für die späteren Vorhersagen in der Speed-Schicht müssen jeweils die entsprechenden ML-Modelle geladen werden. Damit das Laden der korrekten ML-Modelle gelingt, werden aus den Events Informationen extrahiert und in den Dateipfad und den Dateinamen eingearbeitet. Ein ML-Modell welches mit Events der Teststation 2 auf den Produkttyp *PT1* auf den Fehler *Error_Type_3* trainiert wurde, wird beispielsweise unter dem Namen *Error_Type_2.joblib* im Directory *.../Database/Models/Teststation_ID_2/Product_Type_PT1* abgelegt.

Für das Erstellen der ML Modelle werden die verschiedenen Datensätze jeweils in Test- und Trainingsdaten aufgeteilt. Der Trainingsdatensatz wird mit den entsprechenden Label zum Trainieren verwendet. Aus den Daten sollen Muster erkannt werden und im ML-Modell gespeichert werden. Die Testdaten werden zum Validieren des ML-Models verwendet. Dabei werden die Events als Input für Vorhersagen verwendet. Die Vorhersagen des Models werden darauffolgend mit den korrespondierenden Label abgeglichen. Über eine Confusion Matrix wird ausgegeben,

wie viele Vorhersagen richtig und wie viele falsch getroffen wurden. Dabei wird zwischen True Positives, True Negatives, False Positives und False Negatives differenziert. Sind die Ergebnisse zufriedenstellend wird das Model wie oben beschrieben benannt und unter dem entsprechenden Pfad auf dem File System gespeichert.

4.5 Event Processing und Data Model

Das **Event Processing** ist jene Komponente, welcher bei der Implementierung des Prototyps die wichtigste Rolle spielt. Das Event Processing muss sowohl in der Speed-Schicht, als auch in der Batch-Schicht gleich ablaufen. Das Processing kann als Modalität angesehen werden, welche in beiden Schichten gleich eingebunden werden kann. Aufgrund der Modalitäteneigenschaft kann das Event Processing sowohl in der REST Implementierung als auch in der Change Streams Implementierung eingebunden werden. Dahingehend wird diese Komponente im Folgenden für beide Implementierungsvarianten beschrieben. Abbildung 14 zeigen das **Event Processing** für beide Implementierungsvarianten. Das Event Processing wird in beiden Implementierungen vom Container **Model Training** aufgerufen, welches sich in der Batch Schicht befindet. In der Speed-Schicht wird das **Event Processing** in der REST Implementierung vom Container **Processing Server** aufgerufen und in der Change Streams Implementierung von dem Container **Change Stream Service**.

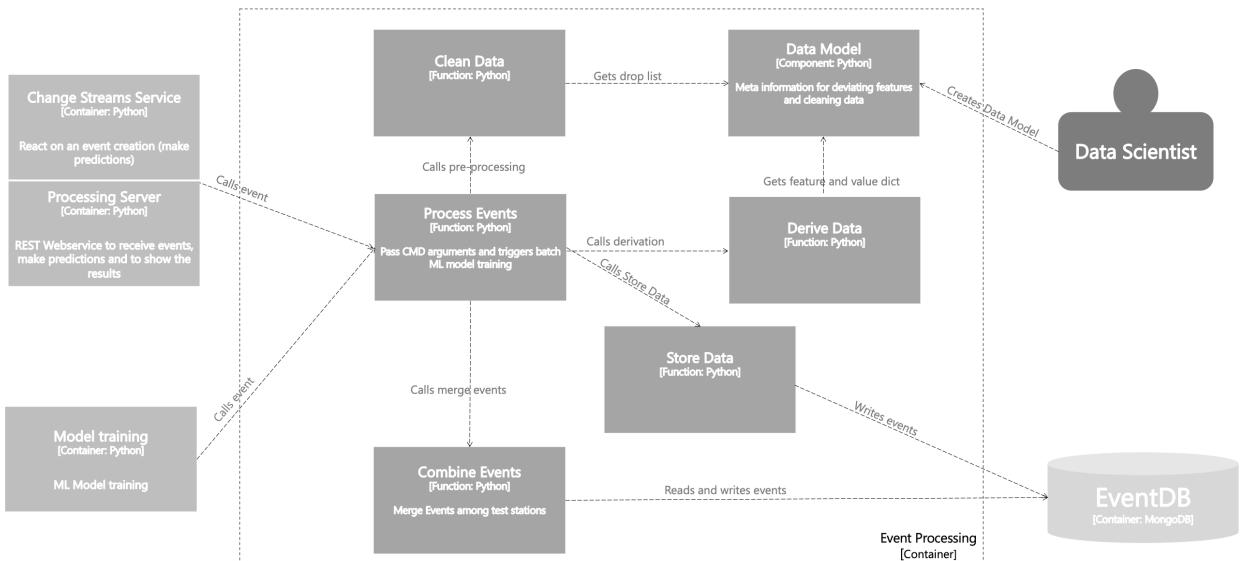


Abbildung 14 - Event Processing

Das Ziel beim Event Processing ist es, dass dieser auf Batch- und Speed-Schicht gleich funktioniert. Hintergrund für das Einsetzen desselben **Event Processing** in den zwei Schichten ist, dass die Events beider Schichten eine Datenverarbeitung durchlaufen müssen. Um die Komplexität und den Verwaltungsaufwand zu reduzieren, empfiehlt es sich das gleiche Processing für beide Ebenen anzubieten. Aus dieser Anforderung resultierend hat sich das **Event Processing** in Abbildung 14 ergeben. Aufgrund der zwei verschiedenen Schichten (Batch und Speed) und deren Merkmalen, ist es innerhalb der Datenverarbeitung nicht ganz möglich, ein einheitliches Vorgehen zu ermöglichen. Der Ablauf des **Event Processings** wird deshalb für die Batch- und Speed-Schicht differenziert erklärt. Als Verbindungsstück der beiden Ebenen erstellt der Data Scientist ein **Data Model**, welches Informationen über die Daten der Teststationen beinhaltet. Aufgrund des **Data Model** sind die Unterschiede beim **Event Processing** von außen nicht sichtbar. Innerhalb des Event Processings erkennt die Implementierungslogik selbstständig, von welcher Schicht die Events an das Event Processing übergeben wurde. Der Aufruf der Datenverarbeitung erfolgt

deshalb immer gleich und es muss vom Anwender des Softwaresystems nicht berücksichtigt werden.

4.5.1 Batch Schicht

In der Batch Schicht gelangt ein ganzer Datensatz (bzw. ein pandas-Dataframe) einer Teststation über das **Model Training** in das **Event Processing**. Anschließend werden mehrere Datenverarbeitungsschritte ausgeführt:

1. Der erste Verarbeitungsschritt ist das Säubern der Daten. Dabei werden mehrere Methoden nacheinander aufgerufen, um verschiedene Säuberungsschritte auszuführen. Dies dient dazu, dass lediglich mit konsistenten Daten ML-Modelle und Vorhersagen erstellt werden. Im Prototyp werden einige wenige Datensäuberungsschritte durchgeführt. Abhängig von der Produktionsumgebung und der Datenstrukturen, muss der Datensäuberungsprozess erweitert und angepasst werden. Die implementierte Datensäuberung ist exemplarisch und nicht vollständig, da es sich bei den Daten um simulierte Daten einer simulierten Produktionsumgebung handelt. Die Funktion **Clean Data** in der Abbildung X repräsentiert diesen Vorgang,
 - a) Damit nur vollständige Events prozessiert werden, werden alle Reihen, in dessen Zeilen Not-a-Number-Werte (NaN) enthalten sind, gelöscht
 - b) Beim Laden der Events aus der EventDB, fügt das MongoDB-System automatisch eine eindeutige Identifizierungsspalte „_id“ hinzu. Da die „Component_ID“ die Schlüsselspalte in den Events einer Teststation ist, ist die „_id“ Spalte nicht notwendig und wird aus diesem Grund gelöscht.
 - c) Während der Datengenerierung der Teststationen können sowohl Kommas als auch Punkte in der Syntax von Datumswerten oder numerischen Werten auftreten. Um konsistent zu bleiben werden deshalb alle Kommas, die im Datensatz auftreten, zu Punkten konvertiert.
 - d) Zusätzlich zu der Punktekonvertierung wird eine Regex-Methode auf die Events im Datensatz ausgeführt. Mit Hilfe der Regex soll sichergestellt werden, dass nur bestimmte Zeichen, Zahlen und Buchstaben in den Spalten der Events vorkommen können.
 - e) In einer realen Produktionsumgebung kann es vorkommen, dass die Teststationen Events übermitteln, welche in gewissen Spalten entweder die gleichen Werte aufweisen, oder längere Texte beinhalten. Für das Training der ML-Modelle bieten beide Varianten keinen Mehrwert und sind hinsichtlich der Performance hinderlich. Aus diesem Grund werden Spalten dieser Varianten vor dem Training gelöscht. Mittels dem **Data Model**, was vom Data Scientist gepflegt wird, kann eine Liste geladen werden, welche teststationsübergreifend alle Spaltennamen enthält, die gelöscht werden müssen. Die entsprechende Methode gleicht die übergebenen Events mit der Liste ab und löscht die korrespondierenden Spalten.
 - f) Als letzten Bereinigungsschritt werden die Fehlermeldungen gesäubert. Fehlermeldungen sind später die Label mit denen die ML-Modelle trainiert werden. Für das Training der ML-Modelle eignen sich numerische Werte besser, weshalb die möglichen Fehlermeldungen zu Zahlen konvertiert werden.

2. Auf die Datensäuberung folgend werden weitere Features von den bestehenden Features in den Event Datensätzen der verschiedenen Teststationen abgeleitet.
 - a) Jede Teststation übermittelt beim Übertragen der Events einen Zeitstempel als Meta-Information. Der Zeitstempel besteht dabei aus zwei verschiedenen Spalten. Zum einen die Datumsspalte und zum anderen die Zeitspalte. Als Input für das Training der ML-Modelle ist allerdings eine einzelne Spalte vorhergesehen. Die Datums- und Zeitspalte werden deshalb zusammengefügt und als neue Spalte in den Events hinterlegt (DateTime). Die ursprüngliche Datumsspalte wird darauffolgend gelöscht. Die Zeitspalte wird im nächsten Ableitungsschritt noch benötigt. Die entsprechende Spalte wird zu einem späteren Zeitpunkt in der Datenableitung gelöscht.
 - b) Für die spätere Verwendung der Events soll für jedes Event eine Prüfsumme aus der Zeitspalte abgeleitet werden. Dazu wird die Zeit in Sekunden aufsummiert und unter einer neuen Spalte („Checksum“) zugeordnet.
 - c) Nachdem die Prüfsumme erstellt wurde, wird die Zeitspalte nicht mehr benötigt und aus den Events gelöscht.
 - d) Als Input für die ML-Modelle sollen die Tage aus der zusammengefügten Datums-Zeit-Spalte abgeleitet werden. Anstatt eine einzige Spalte für die Tage zu erstellen, wird für jeden Tag eine eigene Spalte angelegt. Die Namen der Spalten sind nach den Tagen benannt („Montag“, „Dienstag“, etc.) Der Default-Wert der Spalten ist Null. Abhängig von dem Tag in der Datum-Zeit-Spalte wird der Wert in der korrespondierenden Tagesspalte erhöht. Umgesetzt wird dies mit der pandas-Funktionalität `get_dummies()`. Dazu wird zuerst aus der Datum-Zeit-Spalte der Tag in einer eigenen Spalte extrahiert („Day“). Die `get_dummies()` Funktion extrahiert alle verschiedenen Werte dieser Spalte und erstellt für jeden unterschiedlichen Wert eine neue Spalte. Anschließend werden den Spalten die Default-Werte 0 zugewiesen. Darauf folgend wird pro Event aus der Day-Spalte überprüft, welche der neu erstellten Tagesspalten auf 1 erhöht werden muss. Dieser Prozess wird unter anderem auch One-Hot-Encoding genannt. Da die Spalte „Day“ nach dem OneHot-Encoding keine zusätzlichen Informationen bieten kann, wird diese gelöscht. Insgesamt sind durch das OneHot-Encoding 7 weitere Spalten dem Datensatz zugefügt worden (Eine Spalte für jeden Tag).
 - e) Ebenso wie die Tage, soll die Woche aus der Datum-Zeit-Spalte extrahiert werden. Allerdings soll bei der Woche kein OneHot-Encoding angewendet werden, da 52 zusätzlich neue Spalten (Eine Spalte pro Kalenderwoche) des Datensatzes unnötig vergrößern würden. Anstatt des OneHot-Encoding wird deshalb ein Ordinal-Encoding angewandt. Beim Ordinal-Encoding wird aus der Datum-Zeit-Spalte die Wochen extrahiert. Zusätzlich wird eine neue Spalte („Week“) angelegt, die als Werte die extrahierten Wochen-Werte der Datum-Zeit-Spalte bekommt.
 - f) Als letzten Ableitungsschritt werden alle kategorischen Spalten einem OneHot-Encoding unterzogen. Welche Spalten kategorisch sind wird über eine Liste aus dem **Data Model** geladen. Diese Liste wird mit den Spalten der vorliegenden Events einer Teststation abgeglichen. Gibt es eine oder mehrere Übereinstimmungen werden die entsprechenden Methoden für das OneHot-Encoding der Spalten aufgerufen. Exemplarisch wird diese Funktionalität mit dem Feature „Category“ der Teststation1

ausgeführt. Das Vorgehen ist analog zu der Vorgehensweise beim OneHot-Encoding der Tage.

3. Die verarbeiteten Events werden nach der Datensäuberung und Datenableitung in der EventDB gespeichert (**Store Events**). Zu den Speicherungen werden abhängig von der Teststation unterschiedliche Collections verwendet. Bei einem späteren Zusammenfügen der Daten über die Teststationen hinweg, wird auf diese Events mittels der Component_ID zugegriffen. Ist diese nicht mehr eindeutig, weil Events von verschiedenen Teststationen in einer Collection gespeichert werden, kann es unter Umständen vorkommen, dass die falschen Events zusammengefügt werden.
4. Nach der Speicherung der Daten werden die Events über die Teststationen hinweg zusammengefügt und an den aufrufenden Container (**Train Model, Processing Server** oder **Change Stream Service**) zurückgegeben. Dabei werden immer alle Events der vorliegenden Teststationen über die Component_ID aus den entsprechenden Collections geladen, zusammengefügt und dann zurückgegeben. Zum Beispiel liegt ein Event der Teststation2 mit der Component_ID 1 vor, so wird auf die Collection der prozessierten Events der Teststation1 zugegriffen und über die Component_ID das entsprechende Event geladen und zusammengefügt. Bei Events der Teststation1 wird das Zusammenfügen der Events übersprungen, da keine vorliegenden Teststationen existieren. Diese Events werden sofort an den aufrufenden Container zurückgegeben. Das Zusammenfügen wird ebenso bei den Events der letzten Teststation übersprungen, da diese Events weder Input für die Vorhersagen in der Speed-Schicht sind noch Input für das Training der ML-Modelle. Lediglich die Fehlermeldungen werden für die Label beim **Model Training** über die Komponente **Load Label** benötigt.

4.5.2 Speed Schicht

In der Speed-Schicht kommt das **Event Processing** zum Einsatz, wenn der Container vom Processing Server oder vom Change Stream Service aufgerufen wird. Der spezifische Service, welcher das **Event Processing** aufruft, hängt von der Implementierungsvariante ab (REST oder Change Streams). Der Ablauf des Event Processing läuft gleich wie im Batch-Schicht ab. Jedoch unterscheiden sich einige Funktionen in deren Arbeitsweise, was in diesem Kapitel genauer spezifiziert wird.

Grundsätzlich ist der Unterschied bei den Events von der Speed- und der Batch-Schicht, die Anzahl der Events. Bei der Batch-Schicht wird ein ganzer Datensatz von Events einer Teststation dem **Event Processing** übergeben. Bei der Speed-Schicht wird jeweils nur ein einzelnes Event einer Teststation an das **Event Processing** übertragen. Die unterschiedliche Anzahl der Events führt zu einem Konflikt beim OneHot-Encoding der Tage und der kategorischen Werte. Wie im vorherigen Kapitel unter 2. d) beschrieben, erstellt das OneHot-Encoding neue Spalten basierend auf verschiedenen Werten in den Zeilen der zugrunde liegenden Spalte. Zum Beispiel hat die Spalte „Day“ die Werte „Montag“, „Dienstag“, etc., und es wird ein OneHot-Encoding darauf ausgeführt, so entstehen sieben neue Spalten mit den Namen der Werte der Spalte „Day“. Voraussetzung für die Ausführung eines OneHot-Encoding auf eine spezifische Spalte, ist deshalb, dass die Spalte in ihren Werten der Zeilen alle möglichen Ausprägungen enthält. In der Speed-Schicht hingegen wird nur ein einzelnes Event übertragen, was zu einer einzigen Ausprägung in den Zeilen der Spalten führt. Ein OneHot-Encoding ist deshalb nicht möglich. Um auf das Problem zu reagieren, überprüft das **Event Processing** vor Ausführung der OneHot-Encoding Methoden die Anzahl der Zeilen des Datensatzes. Ist die Zeilenanzahl gleich eins, so werden alternative Funktionen für das OneHot-Encoding angeboten. Für das Erstellen der Tagesspalten („Montag“, „Dienstag“, etc.) wird das OneHot-Encoding ohne die Funktion `get_dummies()`

durchgeführt. Da die unterschiedlichen Ausprägungen bekannt sind – Montag bis Sonntag – können die Spalten manuell erstellt werden. Die Spalten erhalten jeweils den Default-Wert Null. Anschließend wird die Tagesspalte auf ihren Wert überprüft und die korrespondierende Spalte auf Eins erhöht. Kategorische Spalten tauchen nicht wie das Datum bei jeder Teststation auf. In einer realen Produktionsumgebung können zudem unterschiedliche Teststationen verschiedene kategorische Spalten haben. Um die möglichen Ausprägungen zu bekommen, wird im Event Processing dabei wieder mit dem **Data Model** gearbeitet. Beim Erstellen des Data Models erzeugt der Data Scientist ein Dict mit allen notwendigen Informationen zum Extrahieren der möglichen Ausprägungen. Als Key wird der Spaltenname der Kategorie verwendet und als Value wird eine Liste mit allen möglichen Ausprägungen hinterlegt. Das **Event Processing** kann deshalb über den Spaltennamen die möglichen Werte aus dem Dict über das Data Model extrahieren. Das manuelle Ausführen des OneHot-Encodings funktioniert analog wie das manuelle OneHot-Encoding der Tage.

Neben der unterschiedlichen Bearbeitung des OneHot-Encodings in der Speed- und Batch-Schicht, müssen außerdem die fertig prozessierten Daten in den zwei Schichten in verschiedenen Collections gespeichert werden. In einer realen Produktionsumgebung würden Events, die bereits in der EventDB liegen, nicht nochmals von den Teststationen gesendet werden. Aus Gründen der Einfachheit werden zur Simulation der Teststationen die bereits vorhandenen Daten in der EventDB verwendet. Durch diese Simulation sind die Events nicht mehr eindeutig, was beim Zusammenfügen der Daten zu Konflikten führen kann. Um dieses Problem zu umgehen werden in der Speed- und Batch-Schicht zur Speicherung der prozessierten Daten unterschiedliche Collections verwendet.

4.6 Simulate Test Stations

Der Container **Simulate Test Stations** (Abbildung 15 und Abbildung 16) zeigt die Implementierung der Simulation des Produktionsumfeldes. Damit Fehlervorhersagen mit den im **Model Training** erstellten ML-Modelle gemacht werden können, müssen die verschiedenen Teststationen Events senden. Aufgrund der verschiedenen Technologien (REST und Change Streams) die in den Implementierungen verwendet wird, muss die Simulation unterschiedlichen aufgebaut sein. In den nächsten zwei Unterkapiteln wird die Simulation der Teststationen mit den Technologien REST und Change Streams erläutert.

4.6.1 Simulate Teststations (REST)

Grundlage für das Simulieren von Teststationen mit der REST Schnittstelle sind die bereits in der EventDB liegenden Events. Der Data Scientist muss beim Aufruf der **Start sending Events** Komponente über das Terminal Argumente übergeben. Ähnlich wie beim **Model Training** kann der Data Scientist entscheiden für welche Teststationen und welche Produkttypen betroffen sein sollen. Abhängig von den Argumenten werden die korrespondierenden Events aus der EventDB geladen. **Start sending Events** initiiert die Komponente **Send to REST Server**. Dort wird mittels einer Schleife jedes einzelne der geladenen Events über eine REST-Schnittstelle an den REST-Server für die weitere Verarbeitung gesendet. Der REST-Server wird in Abbildung 15 über den **Processing Server** repräsentiert, über welchen später auch die Ergebnisse auf der Webseite angezeigt werden können.

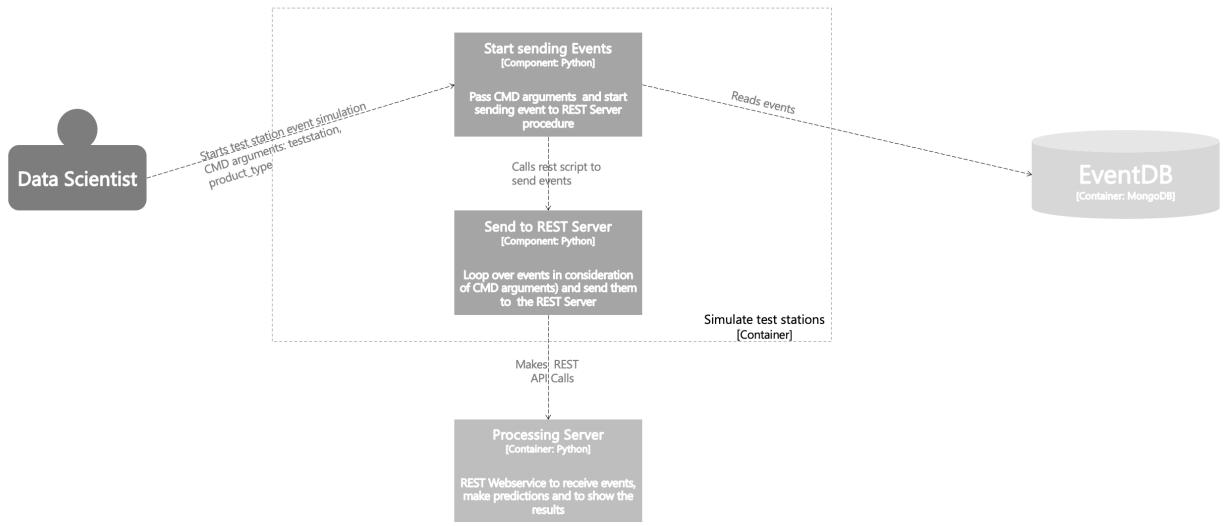


Abbildung 15 - Simulate Test Stations (REST)

4.6.2 Simulate Test Stations (Change Streams)

Anders als bei der REST Implementierung können die Events bei der Change Streams Implementierung nicht über eine Schleife an einen Server gesendet werden. Auslöser für den Change Stream ist die Speicherung der Events. Zur Simulation der Teststationen wird die Speicherung von Events auf der EventDB deshalb künstlich (durch den Data Scientist) erzeugt.

Der Data Scientist führt die **Store Data** (Abbildung 16) Komponente aus der **Event Generation** erneut aus. Die Funktionsweise der Store Data Komponente ist im [Kapitel 4.3 – Event Generation](#) beschrieben. Durch den Speicherungsprozess gelangt eine Benachrichtigung der EventDB an den **Change Streams Service**. Über diese Benachrichtigung wird die weitere Datenverarbeitung initiiert.

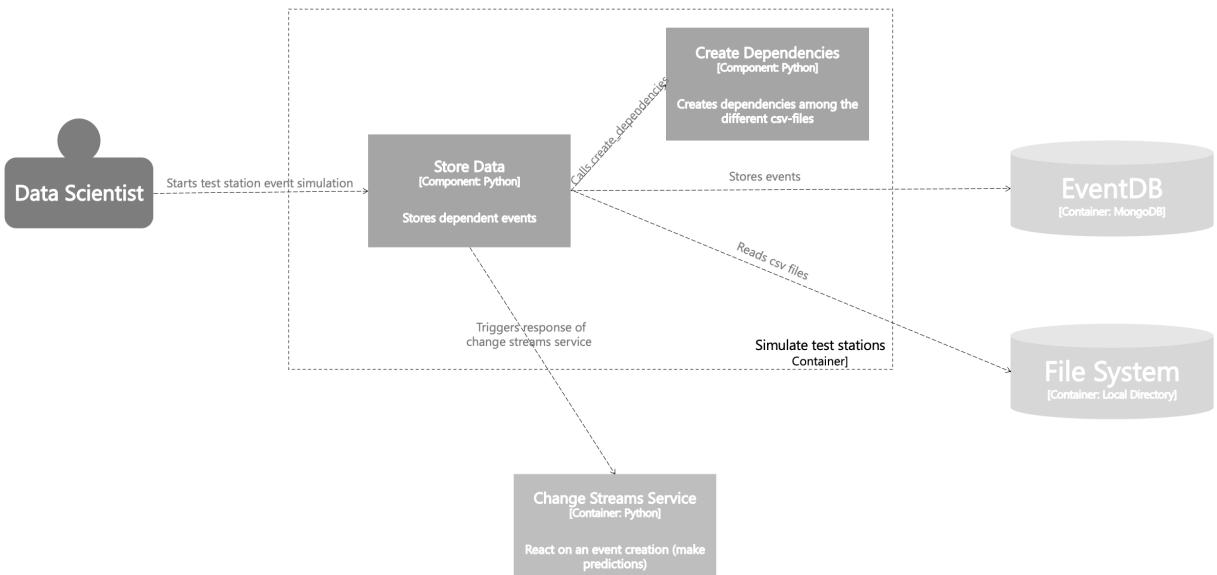


Abbildung 16 - Simulate Test Stations (Change Streams)

4.7 Data Processing Server

Die Datenverarbeitungsserver nehmen die Simulierte Events entgegen, und verarbeiten diese weiter. Wichtig ist, dass die Server vom Data Scientist gestartet wurden, bevor das Senden von Events der Teststationen simuliert wird. Mit den Technologien (REST und Change Stream Service) müssen für beide Implementierungsvarianten unterschiedliche Verarbeitungsserver integrierte werden. Die Funktionsweise der zwei verschiedenen Server werden in den nächsten zwei Kapiteln beschrieben. Zusätzlich wird für jede Implementierungsvariante zusätzlich der Web-Server beschrieben. Dieser ist für das Anzeigen der Vorhersageergebnisse verantwortlich und ist in der REST-Implementierung direkt mit dem REST-Server verbunden.

4.7.1 Processing Server (REST)

Der REST-Server und der Web-Server sind über den Container **Processing Server** abgebildet. Abbildung 17 zeigt die Implementierung des Containers. Zum Empfangen der Events und zum Anzeigen der Vorhersageergebnisse startet der Data Scientist den REST Server (Start **REST Server**), welcher wiederum die Service **Process Events Service** und **Show Results Service** verfügbar macht. Werden Events über die REST-Schnittstelle gesendet, so nimmt der **Process Event Service** die Events entgegen. Zuerst wird das Event über die **Event Processing** Komponente verarbeitet. Anschließend werden die Werte der Spalten „Teststation_ID“ und „Product_Type“ vom **Process Events Service** extrahiert. Mit diesen Informationen lässt sich der richtige Pfad bestimmen, um die passenden ML-Modelle vom File System zu laden. Daraufhin werden die ML-Modelle zur Fehlervorhersage verwendet. Tritt ein Fehler auf, so wird die Vorhersageergebnisse und das Event auf dem Terminal ausgegeben. Die Ergebnisse werden in einer neuen Collection in der EventDB gespeichert. Der **Show Results Service** greift auf die Ergebnisse der Fehlervorhersagen in der EventDB zu und visualisiert diese mittels einer Tabelle auf einer Webseite, auf die der Quality Engineer zugreifen kann.

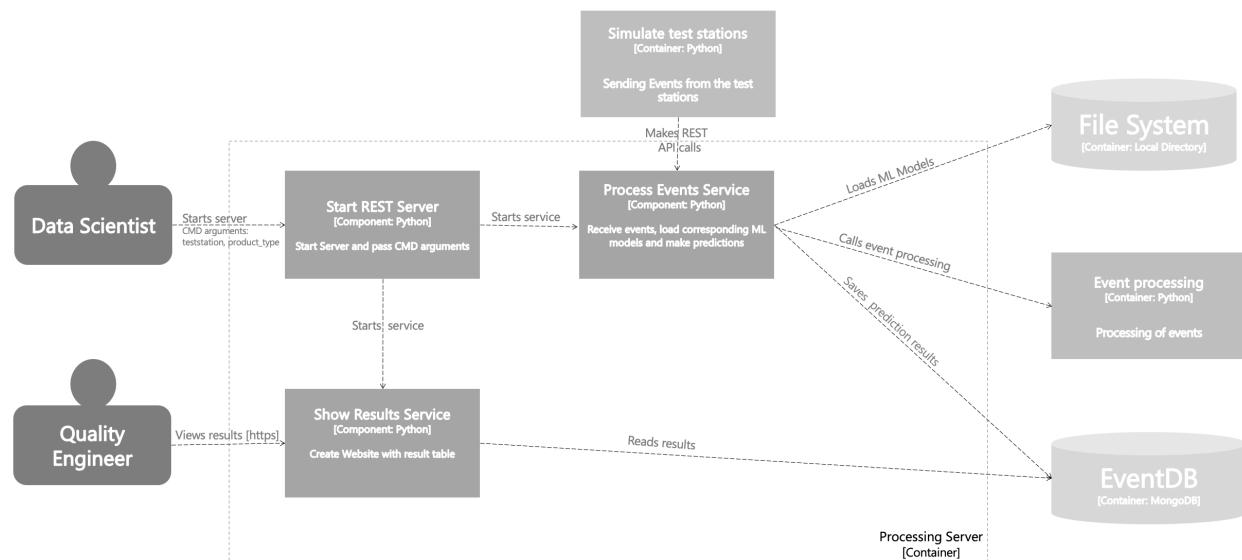


Abbildung 17 - Processing Server

4.7.2 Result Presentation & Change Streams Service (Change Streams)

Bei der Change Stream Implementierung ist der Datenverarbeitungsserver getrennt vom Web-Server, weshalb zwei Modellierungen dargestellt werden. Abbildung 18 zeigt die Implementierung des Datenverarbeitungsserver (**Change Streams Service**) und Abbildung 19 zeigt die Implementierung des Web-Servers (**Result Presentation**).

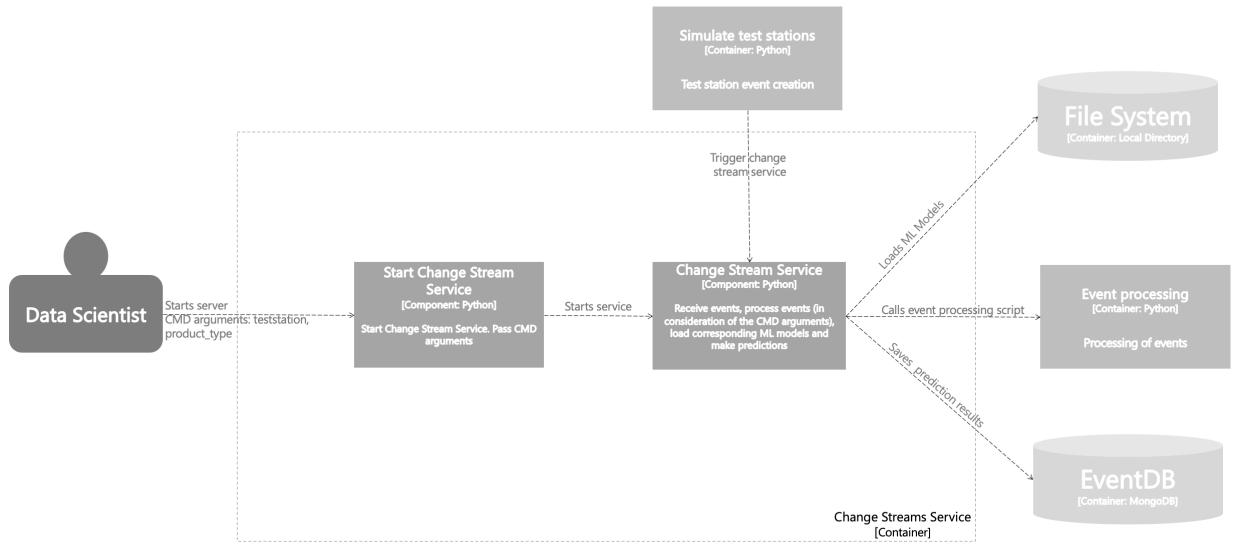


Abbildung 18 - Change Streams Service

Die Funktionsweise der beiden Server ist sehr ähnlich wie bei der REST Implementierung. Der Data Scientist muss die Server (**Start Change Stream Service** und **Start Webservice**) separat starten. Anstatt über die REST Schnittstelle nimmt der Change Stream Service Event Benachrichtigungen vom EventDB System entgegen. Aus der Benachrichtigung wird darauffolgend das Event im **Change Stream Service** extrahiert. Die weitere Datenverarbeitung ist anschließend gleich wie beim **Process Event Service**. Über das **Event Processing** werden die Daten verarbeitet, über extrahierte Informationen vom Event wird das passende ML-Modell geladen und die Fehlervorhersageergebnisse werden in der EventDB gespeichert. Der Quality Engineer kann anschließend auf die Ergebnisse über die Webseite zugreifen. Der Zugriff auf die Webseite löst den **Show Results Service** auf, welcher die Ergebnisse aus der EventDB lädt und tabellarisch visualisiert.

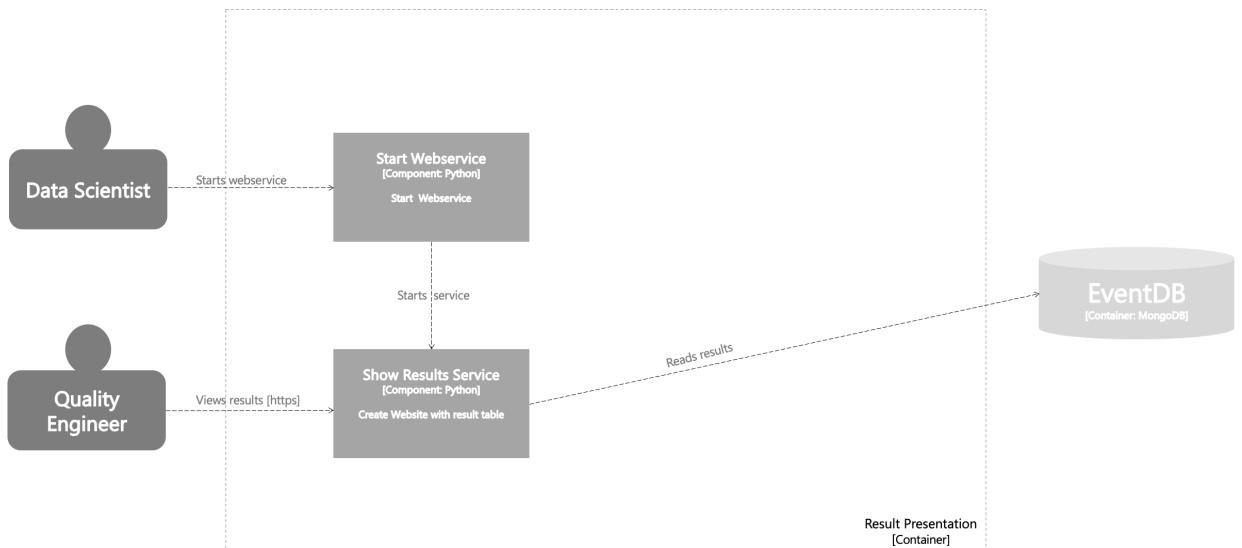


Abbildung 19 - Result Presentation

4.8 Vergleich der Implementierungen

Aufgrund der hohen Anforderungen an die Rechenkapazitäten durch das Trainieren der ML-Modelle wurde die Lambda Architektur mit zwei Data Streams der Kappa Architektur mit einem Data Stream vorgezogen. Durch die zwei Data Streams müssen die ML-Modelle nur noch nach Bedarf trainiert werden (Batch orientiert). Dies kann sich einerseits negativ auf die Genauigkeit der ML-Modelle auswirken, anderseits müssen die ML-Modelle bei den Vorhersagen nur noch geladen werden, was sich positiv auf die Performance auswirkt. Um die performantere Implementierungsvariante zu finden, werden die beiden Implementierungen im nächsten Kapitel verglichen. Neben dem Performancevergleich werden die beiden Implementierungen im darauffolgenden Kapiteln nach qualitativen Punkten verglichen.

4.8.1 Performancevergleich

4.8.1.1 Versuchsaufbau

Zum Performancevergleich der beiden Implementierungsvarianten (REST und Change Streams) werden mehrere Tests durchgeführt. Damit die Testungen sinnvoll miteinander verglichen werden können, werden für die einzelnen Tests die gleichen Bedingungen geschaffen.

Ausgangslage sind für die Testungen sind drei csv-Dateien der Teststationen, die vom Data Scientist im Vorfeld erzeugt und in die EventDB geladen wurden. Es werden insgesamt drei Teststationen simuliert, bei denen jeder zugehörige Datensatz aus 1000 Events besteht. 50% der Events sind dabei vom Produkttyp PT1 und die anderen 50% vom Produkttyp PT2. Ebenso wurden mit den Daten aus der EventDB die ML-Modelle erzeugt. Bevor das Senden von Events von den Teststationen in den jeweiligen Implementierungsvarianten ausgeführt wird, werden die notwendigen Server gestartet (Für die REST Implementierung wird der Processing Server gestartet und für die Change Streams Implementierung der Change Streams Service). Messgegenstand zum Performancevergleich ist die Zeit. Für die Messungen wurden zwei print()-Statements zur Kennzeichnung von Start und Stop eingebaut. Zwischen den Messungen werden jeweils 10 Minuten gewartet, sodass der verwendete Computer Zeit zum Kühlen der Rechenkapazitäten hat.

4.8.1.2 Systeminformationen

Für die Zeitmessung wurde eine digitale Stoppuhr verwendet. Die Experimente wurden in einer VirtualBox VM mit folgenden Spezifikationen durchgeführt:

Hardware

Architecture:	x86_64
Model name:	Intel(R) Core(TM) i5-8275U CPU @ 1.40GHz
CPU op-mode(s):	32-bit, 64-bit
CPU(s):	4
RAM:	8 GB

Distribution

Description:	Ubuntu 20.04.1 LTS
Release:	20.04
Codename:	focal

Kernel

5.4.0-53-generic

4.8.1.3 Versuchsergebnisse

Folgende Tabelle zeigt die Ergebnisse der durchgeführten Tests. Die linke Spalte gibt Auskunft über die verwendeten Teststationen und Produkttypen, welche in der Messung verwendet wurde. Die mittlere Spalte zeigt die Performance der REST Implementierung und die rechte Spalte die Performance der Change Streams Implementierung für die jeweiligen Tests an.

Teststationen/ Produkttypen	REST Implementierung	Change Streams Implementierung
All / All	5 Minuten 45 Sekunden	6 Minuten 14 Sekunden
T1 / All	2 Minuten 21 Sekunden	3 Minuten 3 Sekunden
T1 / PT1	1 Minute 13 Sekunden	1 Minute 35 Sekunden
T1 / PT2	1 Minute 11 Sekunden	1 Minute 33 Sekunden
T2 / PT1	2 Minuten 56 Sekunden	3 Minuten 15 Sekunden
T2 / PT2	2 Minuten 51 Sekunden	3 Minuten 5 Sekunden

Tabelle 21 - Performancevergleich REST - Change Streams

Legende:

Teststationen:

- „All“ – Die Events der Teststationen 1/2 und 3 werden gesendet
- „T1“ – Nur für Events der Teststation 1 gesendet
- „T2“ – Events der Teststation 1 und 2 werden gesendet

Produkttypen:

- „All“ – Die Events von Produkttype PT1 und PT2 werden gesendet
- „PT1“ – Nur für Events vom Produkttyp PT1 werden gesendet
- „PT2“ – Nur für Events vom Produkttyp PT2 werden gesendet

4.8.1.4 Evaluierung der Ergebnisse

Die Ergebnisse der Tests zeigen, dass die REST Implementierungsvariante performanter ist als die Change Streams Implementierung. Für eine fundierte Aussage über die Ursache der Differenzen reichen die Ergebnisse allerdings nicht aus. Weitere Testungen und Analysen sind für die Ergründung der Ursachen von Bedarf, welche jedoch nicht im Fokus dieser Forschungsarbeit liegen. Als Ansatzpunkte könnte allerdings die replizierte Datenbank der Change Streams Implementierung dienen. Durch die Synchronisierung der verteilten Datenbaken könnte ein Performancevorteil entstehen. Insgesamt ist die REST Implementierung bei Verwendung aller Events um 30 Sekunden schneller als die Change Streams Implementierung. Im Verhältnis stimmen die Performanceunterschiede ebenso bei den anderen Testungen überein.

Des Weiteren, ist aus den Ergebnissen der Effekt des Zusammenfügens der Events zu erkennen. Beispielsweise lag die benötigte Verarbeitungszeit bei der REST Implementierung bei Events der Teststationen 1 mit dem Produkttyp 1 bei 1 Minute 13 Sekunden. Mutmaßlich müsste das Ergebnis bei der Teststation 2 mit dem Produkttype 1 bei 2 Minuten 26 Sekunden liegen (Wird auf die Teststation 2 getestet, so werden auch die entsprechenden Events der Teststation 1 übermittelt. Dies hat den Hintergrund, dass die Daten der Teststation 2 mit den Events der Teststation 1 für Fehlervorhersagen zusammengefügt werden müssen. Die benötigte Verarbeitungszeit ist daraus resultierend ca. doppelt so groß). Tatsächlich liegt die benötigte Verarbeitungszeit bei 2 Minuten 56 Sekunden. Die zusätzlich benötigten 30 Sekunden (2 Minuten 56 Sekunden – 2 Minuten 26 Sekunden) lassen sich auf das Zusammenfügen der Events zurückführen. Events der Teststation 1 werden vom Event Processing nach der Datenverarbeitung direkt zurückgegeben und können für die Fehlervorhersage benutzt werden. Events der Teststation 2 müssen vor der

Vorhersage mit den Events der Teststation1 zusammengefügt werden. Die daraus resultierende zusätzliche Datenbankabfrage lässt auf den Performanceunterschied hindeuten. Der Performanceunterschied zwischen Teststation 1 und Teststation 2 aufgrund des Zusammenfügens der Events lässt sich ebenso in den Ergebnissen innerhalb der Change Streams Implementierung erkennen.

Die Unterschiede zwischen den Produkttypen 1 und 2 sowohl in der REST Implementierung als auch in der Change Streams Implementierung auf Ungenauigkeiten in den Messungen zurückführen. Zudem kann die hundertprozentige Gewährleistung von gleichen Bedingungen im Testumfeld nicht garantiert werden. Die Unterschiedliche Auslastung des Systems kann ebenso zu den Unterschieden in den Ergebnissen zwischen den Produkttypen beigetragen haben. Eine weitere Erklärung für die Unterschiede sind die Eventdatensätze. Bei der Datengenerierung werden pro Datensatz eine geringe Anzahl von Daten fehlerhaft erstellt. Dies hat den Sinn, dass die fehlerhaften Daten beim Event Processing bereinigt oder aussortiert werden können. Da die Fehler zufällig von der Event Generierung eingebaut werden, kann es bei der geringen Anzahl von Events vorkommen, dass sich die Fehler ungleich auf einen Produkttyp verteilen.

4.8.2 Qualitativer Vergleich

Beide Implementierungsvarianten haben ihre Vor- und Nachteile. Unabhängig der Performance Ergebnisse, hängt die Wahl der Implementierungsvariante von der Produktionsumgebung ab. Gibt es innerhalb der Produktionsumgebung eine zentrale Datenbank, bietet sich die REST Implementierung an. Durch das Senden der Events über die REST-Schnittstelle können die Daten auf der zentralen Datenbank zeitgleich abgelegt werden. Wird in der Produktionsumgebung eine replizierte Datenbank verwendet und jede Teststation hat die Möglichkeit Daten auf der Datenbank zu speichern, dann bietet sich die Change Streams Implementierung an. Eine Übertragung der Events über eine REST-Schnittstelle ist in diesem Fall zusätzlicher Aufwand und kann deshalb gespart werden. Aus Implementierungstechnischer Sicht können beide Implementierungsvarianten gleich gewertet werden. Der Großteil der Komponenten können 1:1 übertragen werden. Anstelle der REST-Schnittstelle in der REST-Implementierung muss bei der Change Streams Implementierung der Change Stream konfiguriert und eingebunden werden, weshalb der Aufwand für beide Varianten ist in etwa gleich ist.

5 Limitationen und Resümee

In diesem Kapitel wird im Hinblick auf die in der Einleitung definierten Forschungsfragen auf Basis der beiden Implementierungsvarianten und der Durchgeführten Tests diskutiert. Zusätzlich wird in diesem Zusammenhang einen Ausblick für zukünftige Forschungsarbeiten auf Basis dieser Forschungsarbeit gegeben.

5.1 Resümee und Bewertung der Forschungsfragen

Abgeleitet vom dem Forschungsprojekt PREFERML wurde die vorliegende Forschungsarbeit erstellt, um einen Prototyp für die Datenverarbeitung zu erstellen. Der Prototyp dient als Vorlage für die Einbindung in eine reale Produktionsumgebung. Da die Verwendung einer realen Produktionsumgebung nicht möglich war, wurde die Produktionsumgebung für die Implementierung simuliert. Für die Erstellung des Prototyps wurden die Arbeiten des Forschungsprojektes PREFERML herangezogen [11] [12] [13] [14] [15]. Die wichtigsten Merkmale für die Implementierung eines Datenverarbeitungsprototyp wurden dabei herausgearbeitet. Resultierend aus den Merkmalen wurde darauffolgend eine Anforderungsanalyse für die Umsetzung des Prototyps durchgeführt. Zusätzlich wurden die Gesichtspunkte der Forschungsfragen berücksichtigt. Ergebnis der Anforderungsanalyse waren verschiedene Use-Cases, welche bei der Implementierung des Prototyps berücksichtigt und umgesetzt werden sollten. Für eine qualitativ hochwertige Implementierung wurden im Anschluss an die Anforderungsanalyse die theoretischen Grundlagen über die Datenverarbeitung gegeben. Dabei wurden verschiedene Technologien und Architekturen vorgestellt. Für die Implementierung der Use Cases wurde die Lambda Data Processing Architektur herangezogen. Für die konkrete Umsetzung wurden dabei zwei verschiedene Varianten (REST und Change Streams) implementiert. Die Spezifikation zu dem Aufbau der Implementierungsvarianten wurden auf das Grundlagenkapitel folgend beschrieben. Zusätzlich wurden die Implementierungsvarianten hinsichtlich der Performance untereinander verglichen.

Zum Evaluieren über das Erreichen der Forschungsfragen werden die Ergebnisse dieser Forschungsarbeit bezüglich der Forschungsfragen erörtert:

RQ1: Inwieweit lässt sich eine einheitliche Datenverarbeitung in Form von gleichem Source Code für die Erstellung der ML-Modelle als auch für die Verwendung der ML-Modelle zur Fehlervorhersage erstellen?

Hinsichtlich des Forschungsprojekts PREFERML besteht die Herausforderung der RQ1 darin, sowohl für das Training der ML-Modelle, als auch für die Fehlervorhersagen unter Verwendung der ML eine einheitliche Datenverarbeitung anzubieten. Die Datenverarbeitung bei den zwei Ausprägungen unterscheiden sich darin, dass bei der rechenintensiven Ausprägung ein ganzer Datensatz prozessiert werden muss und bei der rechenarmen Ausprägung ein einzelnes Event. Mit dem Event Processing innerhalb des zwei verschiedenen Implementierungsvarianten ist die Implementierung einer einheitlichen Datenverarbeitung gelungen. Das Event Processing kann sowohl für das rechenintensive ML-Model Training eingesetzt werden als auch für das rechenarme Vorhersagen von Fehlern. Der Aufruf des Event Processings erfolgt bei beiden Varianten genau auf dieselbe Weise, weshalb oberflächlich betrachtet derselbe Source Code verwendet wird. Innerhalb des Event Processings muss allerdings auf die Heterogenität der Events (Ganzer Datensatz oder einzelnes Event) eingegangen werden. Beispielsweise kann das OneHot-Encoding bei der Datenableitung für kategoriale Werte bei einzelnen Events nicht angewendet werden. Als Workaround wird mit der Einbindung des Data Models gearbeitet. Es ist deshalb möglich eine Datenverarbeitungs-Komponente für rechenintensive als auch rechenarme Aufgaben zur Verfügung zu stellen. Innerhalb der Datenverarbeitungs-Komponente muss jedoch

zwischen den Aufgaben (rechenintensiv und rechenarm) differenziert werden. Eine Möglichkeit zur Verwendung der Komponente ohne interne Differenzierung liefert das Data Model, in dem auf das OneHot-Encoding bei rechenintensiven Aufgaben verzichtet wird. Diese werden ebenfalls mit der Verwendung des Data Models umgesetzt.

RQ2: Wie kann eine Datenverarbeitungsarchitektur aussehen, bei der für die Erstellung der ML-Modelle dieselbe Datenverarbeitung durchlaufen wird, wie für die Verwendung der ML-Modelle zur Fehlervorhersage?

Für die Erstellung der Event Processing Architektur sind die in den Grundlagen beschriebenen Data Streaming Architekturen Lambda und Kappa herangezogen worden. Unterscheidungsmerkmal der beiden Architekturen ist die Anzahl an vorhandenen Data Streams. Für die Umsetzung einer Datenverarbeitung für rechenintensive und rechenarme Aufgaben empfiehlt sich daher die Kappa Architektur mit der Bearbeitung nur eines Data Streams. Durch den Aufbau der Architektur ist die Verwendung derselben Datenverarbeitung automatisch gegeben. Allerdings ist diese Methode aufgrund des hohen Ressourcenbedarfs weniger praktikabel. Mit der Verwendung nur eines Datenstroms, müssen für jede Fehlervorhersage die entsprechenden ML-Modelle trainiert werden. Je mehr Daten vorhanden sind, desto länger können die Fehlervorhersagen dauern. Die Ergebnisse werden deshalb unter Zeitverzögerung angezeigt. Da die Performance ein Aspekt der RQ4 ist, wird daher die Lambda Architektur für die Implementierung des Prototyps verwendet. Durch die zwei Datenströme der Lambda Architektur kann das rechenintensive ML-Model Training von den rechenarmen Fehlervorhersagen getrennt werden. Gemeinsame Komponente beider Datenströme ist das Event Processing.

RQ3: Ist eine solche Architektur umsetzbar?

Mit Hilfe der Lambda Architektur und des Event Processings ist es möglich, die rechenintensiven (ML-Model Training) Aufgaben von den Rechenarmen (Fehlervorhersage) zu trennen und gleichzeitig eine einheitliche Datenverarbeitung zu verwenden. Durch die Implementierung der beiden Implementierungsvarianten für den Prototyp ist es gelungen eine solche Architektur umzusetzen. Außerdem besteht durch die beiden Implementierungsvarianten die Möglichkeit den Prototyp in verschiedene Produktionsumgebungen einzusetzen.

RQ4: Welche Technologien eignen sich hinsichtlich der Performance für eine solche Datenverarbeitungsarchitektur am besten?

Für die Implementierung des Prototyps sind zwei verschiedene Technologien implementiert und untereinander verglichen worden. Zum einem REST und zum anderen MongoDB Change Streams. Abhängig vom Aufbau der realen Produktionsumgebungen eignen sich beide Implementierungsvarianten. Bei der Durchführung der Performancetests hat die REST Implementierung eine bessere Performance gegenüber der Change Streams Implementierung gezeigt. Unter Betrachtung der durchgeführten Tests eignet sich die REST Implementierung hinsichtlich der Performance besser. Für eine fundierte Aussage über die Performance müssen allerdings noch Tests unter realen Bedingungen durchgeführt werden.

5.2 Stärken und Limitationen der Arbeit

Durch das Fehlen einer realen Produktionsumgebung ist diese Forschungsarbeit auf die vom Autor erstellten Simulation der Produktionsumgebung beschränkt. Die durchgeführten Tests beziehen sich deshalb lediglich auf die Performance innerhalb der Produktionssimulation. In einer realen Produktionsumgebung kommen zusätzliche Komplexitäten, wie heterogene

Softwaresysteme, größere Anzahl an Teststationen, komplexere Datenstrukturen, etc. vor. Um die Aussagekraft der durchgeführten Tests zu festigen, ist eine Implementierung der Prototypvarianten in eine echte Produktionsumgebung notwendig. Zusätzlich beschränkt sich die Durchführung der gemachten Performancetests auf die Rechenressourcen des Autors. Um sinnvolle Ergebnisse erzielen zu können, beschränkt sich die Anzahl der Events pro Teststation auf 1000. Abhängig von der Produktionsumgebung kann diese Anzahl deutlich zu gering sein für aussagekräftige Tests. Der Zugriff auf Big Data ausgewählte Rechenressourcen sind daher von essentieller Bedeutung.

Die Produktionssimulation eröffnet für den Autor hinsichtlich der verschiedenen Technologien allerdings auch Möglichkeiten. Die Produktionssimulation kann vom Autor selbst erstellt. Durch die „grüne Wiese“ hat der Verfasser die Möglichkeit verschiedene Implementierungen zu realisieren und zu validieren. Einzige Einschränkung für die Entwicklung der Produktionssimulation und des Prototyps ist die freie Zugänglichkeit der Technologien und Tools (Open Source).

Aufgrund der zwei verschiedenen Implementierungsvarianten sind die verwendeten Komponenten (insbesondere das Event Processing) so konzipiert, dass diese leicht austauschbar sind. Beispielsweise wird die Komponente (Event Processing) sowohl in der REST Implementierung als auch in der Change Streams Implementierung verwendet. Der Aufbau der Komponente ist dabei in beiden Varianten exakt gleich. Resultierend daraus hat diese Komponente Modalitätseigenschaften. Die Implementierungen neuer Technologien – zum Beispiel Kafka – lässt sich durch diese Eigenschaft sehr einfach realisieren.

5.3 Ausblick auf zukünftige Arbeiten

Aufbauend auf der Erstellung des Prototyps für die Datenverarbeitung in der vorliegenden Forschungsarbeit gibt es verschiedene Ansätze für weitere Arbeiten. Die Erkenntnisse und der Prototyp selbst dienen dabei als Grundlage, auf der die weiteren Arbeiten aufbauen können.

- Analyse der durchgeführten Performancetests. Aus welchen Gründen ist REST die performantere Implementierungsvariante?
- Vollständige Vereinheitlichung des Event Processings durch Verzicht des OneHot-Encodings und dem Umstieg auf das Data Model
- Erweiterung des Prototyps durch die Einführung weiterer Teststationen und komplexeren Daten.
- Verwendung weiterer Technologien (Kafka, Spark, etc.) für die Implementierung des Prototyps
- Einbindung des Prototyps in eine echte Produktionsumgebung

Mit Hilfe der weiteren Forschungsarbeiten könnte der Prototyp insgesamt generischer und mächtiger gestaltet werden. Durch die Einbindung in eine echte Produktionsumgebung kann der Prototyp an die Produktionsgegebenheiten angepasst und auf Performance getestet werden. Resultierend aus den Testergebnissen würden gezeigt werden können, welche Tools und Technologien sich am besten für die in dieser Forschungsarbeit erstellte Event Processing Architektur unter realen Bedingungen eignen.

Glossar

AutoML	„With the explosion in the use of machine learning in various domains, the need for an efficient pipeline for the development of machine learning models has never been more critical. However, the task of forming and training models largely remains traditional with a dependency on domain experts and time-consuming data manipulation operations, which impedes the development of machine learning models in both academia as well as industry. This demand advocates the new research era concerned with fitting machine learning models fully automatically i.e., AutoML. Automated Machine Learning(AutoML) is an end-to-end process that aims at automating this model development pipeline without any external assistance.“ [103]
BSON	“BSON simply stands for “Binary JSON,” and that’s exactly what it was invented to be. BSON’s binary structure encodes type and length information, which allows it to be parsed much more quickly.“ [66]
DESI	„Seit 2014 überwacht die Europäische Kommission den Stand der Digitalisierung in den Mitgliedstaaten und dokumentiert die erzielten Fortschritte im Bericht zum Index für die digitale Wirtschaft und Gesellschaft (DESI)“ [9].
Dict	„Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by <i>keys</i> , which can be any immutable type; strings and numbers can always be keys. Tuples can be used as keys if they contain only strings, numbers, or tuples; if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key“ [104].
Domain Knowledge	Wissen über ein bestimmtes Fachgebiet. Im Kontext des Forschungsprojektes PREFERML ist das Domain Knowledge Wissen über den Produktionsprozess oder das Qualitätsmanagement
OneHot Encoding	“Encode categorical features as a one-hot numeric array” [105].
Ordinal Encoding	“Encode categorical features as an integer array” [106].
REST	“REST ist ein Architekturstil für den Entwurf vernetzter Anwendungen” [107]
SHAP	“SHAP is a method to explain individual predictions. SHAP is based on the game theoretically optimal Shapley Values.” [108]
Write-Ahead-Log	„Write-Ahead Logging (WAL) is a standard method for ensuring data integrity. A detailed description can be found in most (if not all) books about transaction processing. Briefly, WAL's central

concept is that changes to data files (where tables and indexes reside) must be written only after those changes have been logged, that is, after log records describing the changes have been flushed to permanent storage.“ [109]

Abkürzungsverzeichnis

ANN	Künstliches neuronales Netzwerk
AutoML	Automated Machine Learning
BMBF	Bundesministerium für Bildung und Forschung
BMWi	Bundesministerium für Wirtschaft und Energie
ca	circa
CD	Compact Disc
CPS	Cyber Physical System
CRISP-DM	Cross-industry standard process for data mining
CSV	Comma separated value
DESI	Index für die digitale Wirtschaft und Gesellschaft
DVD	Digital Video Disc
DT	Decision Tree
etc	et cetera
GBs	Giga Bytes
GPU	Graphics processing unit
HDFS	Hadoop Distributed File System
IoT	Internet of Things
KI	Künstliche Intelligenz
KNN	K-Nearest-Neighbours
LOGREG	Logistic Regression
ML	Maschinelles Lernen
NaN	Not a Number
NLP	Natural Language Processing
NN	Neuronales Netz
NoSQL	Not only SQL
NTP	Network Time Protocol
OPC UA	Open Platform Communication Unified Architecture
PBs	Peta Bytes
PREFERML	Proaktive Fehlervermeidung in der Produktion durch Maschinelles Lernern
RF	Random Forest
RDBMS	Relationale Datenbank Management Systeme
RDD	Resilient distributed dataset
Regex	Regular expression
REST	Representational state transfer
RQ	Research Questions
SHAP	Shapley Additive Explanations
SQL	Structured Query Language
SVM	Support Vector Machine
TBs	Terra Bytes
ZBs	Zetta Bytes

Annex

Annex I: code_repository.zip

Literaturverzeichnis

- [1] R. Oerter, „Der Homo sapiens erobert die Welt,“ in *Der Mensch, das wundersame Wesen: Was Evolution, Kultur und Ontogenese aus uns machen*, Wiesbaden, Springer, 2014, pp. 61-79.
- [2] D. E. Nye, *Technology Matters*, Cambridge: MIT Press, 2006.
- [3] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld und M. Hoffmann, „Industrie 4.0,“ *Wirtschaftsinformatik*, Bd. 56, Nr. 4, pp. 261-264, 2014.
- [4] A. Kusiak, „Smart manufacturing,“ *International Journal of Production Research*, Bd. 56, Nr. 1-2, pp. 508-517, 2018.
- [5] C. Rammer, „Auf Künstliche Intelligenz kommt es an,“ Bundesministerium für Wirtschaft und Energie (BMWi), Berlin, 2020.
- [6] N. Brandstetter, R.-M. Dobler und D. J. Ittstein, „Künstliche Intelligenz,“ in *Interdisziplinär*, Tübingen, UVK Verlag, 2020, pp. 70-84.
- [7] B. Hatiboglu, S. Schuler, A. Bildstein und M. Hämerle, „Einsatzfelder von Künstlicher Intelligenz im Produktionsumfeld,“ Fraunhofer IPA, Stuttgart, 2019.
- [8] B. Martens, „The Importance of Data Access Regimes for Artificial Intelligence and Machine Learning,“ JRC Digital Economy Working Paper, Sevilla, 2018.
- [9] E. Commission, „Index für die digitale Wirtschaft und Gesellschaft (DESI) 2020,“ European Commission, 2020.
- [10] C. Seiffer, U. Schreier, H. Ziekow und A. Gerling, „KI für die Produktionsqualität,“ *Forschungsbericht 2020/2021*, Bd. 800, pp. 18-19, 11 Dezember 2020.
- [11] H. Ziekow, U. Schreier, A. Gerling und A. Saleh, „Technical Report: Interpretable Machine Learning for Quality Engineering,“ *Manufacturing - Importance measures that reveal insights on errors*, p. 12, 2019.
- [12] H. Ziekow, U. Schreier, A. Saleh, C. Rudolph, K. Ketterer, D. Grozinger und A. Gerling, „Proactive Error Prevention in Manufacturing Based on an Adaptable Machine Learning Environment,“ *Artificial Intelligence: From Research to Application: The Upper-Rhine Artificial Intelligence Symposium UR-AI 2019, March 13th, 2019, Offenburg, Germany*, pp. 113-117, 2019.
- [13] A. Gerling, A. Saleh, U. Schreier und H. Ziekow, „Supporting Quality Assessment in Manufacturing by Machine Learning : First Results of PREFERML Project,“ *Artificial Intelligence - Research Impact on Key Industries : The Upper-Rhine Artificial Intelligence Symposium UR-AI 2020; Collection of Accepted Papers of the Canceled Symposium Karlsruhe, 13th May 2020*, Nr. 978-3-943301-29-8, pp. 52-53, 2020.
- [14] Y. Wilhelm, U. Schreier, P. Reimann, B. Mitschang und H. Ziekow, „Data Science Approaches to Quality Control in Manufacturing: A Review of Problems, Challenges and Architecture,“ *Service-Oriented Computing : 14th Symposium and Summer School on Service-Oriented Computing, SummerSOC 2020, Crete, Greece, September 13-19, 2020*, pp. 45 - 65, 2020.
- [15] A. Gerling, U. Schreier, A. Hess, A. Saleh, H. Ziekow und D. O. Abdeslam, „A Reference Process Model for Machine Learning Aided Production Quality Management,“ in *Proceedings of the 22nd International Conference on Enterprise Information Systems, May 5-7, 2020*, 2020.

- [16] P. Hehenberger, „Qualitätsmanagement in der Produktion,“ in *Computerunterstützte Produktion: Eine kompakte Einführung*, Berlin, Heidelberg, Springer Berlin Heidelberg, 2020, pp. 197-228.
- [17] M. Colledani, u. Tolio, A. Fischer, B. Iung, G. Lanza, R. Schmitt und J. Váncza, „Design and management of manufacturing systems for production quality,“ *CIRP Annals*, Bd. 63, Nr. 2, pp. 773-796, 2014.
- [18] S. J. Hu, X. Zhu, H. Wang und Y. Koren, „Product variety and manufacturing complexity in assembly systems and supply chains,“ *CIRP Annals*, Bd. 57, Nr. 1, pp. 45-48, 2008.
- [19] S. Thalmann, H. G. Gursch, J. Suschnigg, M. Gashi, H. Ennsbrunner, A. K. Fuchs, T. Schreck, B. Mutlu, J. Mangler, G. Kappl, C. Huemer und S. Lindstaedt, „Cognitive decision support for industrial product life cycles: A position paper,“ in *Proceedings of the 11th International Conference on Advanced Cognitive Technologies and Applications*, Venice, 2019.
- [20] T. Wuest, D. Weimer, C. Irgens und K.-D. Thoben, „Machine learning in manufacturing: advantages, challenges, and applications, Production & Manufacturing Research,“ *Production & Manufacturing Research*, Bd. 4, Nr. 1, pp. 23-45, 2016.
- [21] V. Hirsch, P. Reimann und B. Mitschang, „Data-Driven Fault Diagnosis in End-of-Line Testing of Complex Products,“ in *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, Washington, D.C., IEEE, 2019, pp. 492-503.
- [22] L. B. Kassner und M. Bernhard, „Exploring Text Classification for Messy Data: An Industry Use Case for Domain-Specific Analytics,“ in *Proceedings of the 19th International Conference on Extending Database Technology (EDBT 2016)*, Bordeaux, 2016.
- [23] L. a. L. A. a. E. C. Leitner, „End-of-line fault detection for combustion engines using one-class classification,“ in *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, Banff, 2016.
- [24] C. Nitesh, N. Japkowicz und A. Kołcz, „Editorial: Special Issue on Learning from Imbalanced Data Sets,“ *SIGKDD Explorations*, Bd. 6, pp. 1-6, 2004.
- [25] Y. Cheng, K. Chen, H. Sun, Y. Zhang und F. Tao, „Data and knowledge mining with big data towards smart production,“ *Journal of Industrial Information Integration*, Bd. 9, pp. 1-13, 2019.
- [26] J. Gama, I. Žliobaité, A. Bifet, M. Pechenizkiy und A. Bouchachia, „A Survey on Concept Drift Adaptation,“ *ACM Comput. Surv.*, Bd. 46, Nr. 4, p. 44:1–44:37, 2014.
- [27] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama und G. Zhang, „Learning under Concept Drift: A Review,“ *IEEE Transactions on Knowledge and Data Engineering*, Bd. 31, Nr. 12, pp. 2346-2363, 2019.
- [28] S. Wang und X. Yao, „Multiclass Imbalance Problems: Analysis and Potential Solutions,“ *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Bd. 42, Nr. 4, pp. 1119-1130, 2012.
- [29] Yanminsun, A. Wong und M. S. Kamel, „Classification of imbalanced data: A review,“ *International Journal of Pattern Recognition and Artificial Intelligence*, Bd. 23, Nr. 4, p. 687–719, 2009.
- [30] N. Thai-Nghe, Z. Gantner und L. Schmidt-Thieme, „Cost-sensitive learning methods for imbalanced data,“ in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, 2010, pp. 1-8.

- [31] D. Lahat, T. Adali und C. Jutten, „Multimodal Data Fusion: An Overview of Methods, Challenges, and Prospects,“ *Proceedings of the IEEE*, Bd. 103, Nr. 9, pp. 1449-1477, 2015.
- [32] B. Khaleghi, A. Khamis, F. O. Karray und S. N. Razavi, „Multisensor data fusion: A review of the state-of-the-art,“ *Information Fusion*, Bd. 14, Nr. 1, pp. 28-44, 2013.
- [33] V. Hirsch, P. Reimann, O. Kirn und B. Mitschang, „Analytical Approach to Support Fault Diagnosis and Quality Control in End-Of-Line Testing,“ *Procedia CIRP*, Bd. 72, p. 1333– 1338, 2018.
- [34] S. Lundberg und S.-I. Lee, „A unified approach to interpreting model predictions,“ *Proceedings of the 31st International Conference on Neural Information Processing Systems*, p. 4765–4774, 2017.
- [35] M. Vartak, H. Subramanyam, W.-E. Lee, S. Viswanathan, S. Husnoo, S. Madden und M. Zaharia, „ModelDB: A System for Machine Learning Model Management,“ *Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA)*, 2016.
- [36] C. Weber, P. Hirmer und P. Reimann, „A Model Management Platform for Industry 4.0 – Enabling Management of Machine Learning Models in Manufacturing Environment,“ in *Business Information Systems - 23rd International Conference*, Bd. 389, Cham, Springer International Publishing, 2020, pp. 403-417.
- [37] M. Wollschlaeger, T. Sauter und J. Jasperneite, „The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0,“ *IEEE Industrial Electronics Magazine*, Bd. 11, Nr. 1, pp. 17-27, 2017.
- [38] R. Wirth und J. Hipp, „CRISP-DM: Towards a standard process model for data mining,“ Bd. 1, 2000.
- [39] J. Yan, Y. Meng, L. Lu und L. Li, „Industrial Big Data in an Industry 4.0 Environment: Challenges, Schemes, and Applications for Predictive Maintenance,“ *IEEE Access*, Bd. 5, pp. 23484-23491, 2017.
- [40] E. Mehmood und T. Anees, „Challenges and Solutions for Processing Real-Time Big Data Stream: A Systematic Literature Review,“ *IEEE Access*, Bd. 8, pp. 119123-119143, 2020.
- [41] D. A. Pereira, W. Ourique de Moraes und E. Pignaton de Freitas, „NoSQL real-time database performance comparison,“ *International Journal of Parallel, Emergent and Distributed Systems*, Bd. 33, Nr. 2, pp. 144-156, 2018.
- [42] S. J. Kamatkar, A. Kamble, A. Viloria, L. Hernandez-Fernandez und E. G. Cali, „Database Performance Tuning and Query Optimization,“ in *Data Mining and Big Dat*, Cham, Springer International Publishing, 2018, pp. 3-11.
- [43] S. G. Ahmad, C. S. Liew, M. M. Rafique und E. U. Munir, „Optimization of data-intensive workflows in stream-based data processing models,“ *The Journal of Supercomputing*, Bd. 73, Nr. 9, pp. 3901-3923, 2017.
- [44] A. Ahmad, A. Paul, S. Din, M. M. Rathore, G. S. Choi und G. Jeon, „Multilevel Data Processing Using Parallel Algorithms for Analyzing Big Data in High-Performance Computing,“ *International Journal of Parallel Programming*, Bd. 46, Nr. 3, pp. 508-527, 2018.
- [45] A. Nagpal und G. Gabrani, „Python for Data Analytics, Scientific and Technical Applications,“ in *2019 Amity International Conference on Artificial Intelligence (AICAI)*, IEEE, 2019, pp. 140-145.
- [46] A. Ismail, H.-L. Truong und W. Kstner, „Manufacturing process data analysis pipelines: a requirements analysis and survey,“ *Journal of Big Data*, Bd. 6, Nr. 1, 2019.

- [47] Y. Cui, S. Kara und K. C. Chan, „Manufacturing big data ecosystem: A systematic literature review,“ *Robotics and Computer-Integrated Manufacturing*, Bd. 62, p. 101861, 2020.
- [48] P. Gölzer, P. Cato und M. Amberg, „Data processing requirements of Industry 4.0 - use cases for Big Data applications,“ *ECIS 2015 Research-in-Progress Papers.*, Bd. 61, 2015.
- [49] C. Ji, Y. Li, W. Qiu, Y. Jin, Y. Xu, U. Awada, K. Li und W. Qu, „Big data processing: Big challenges and opportunities,“ *Journal of Interconnection Networks*, Bd. 13, p. 125009, 2012.
- [50] R. Ranjan, „Streaming Big Data Processing in Datacenter Clouds,“ *IEEE Cloud Computing*, Bd. 1, Nr. 1, pp. 78-83, 2014.
- [51] I. Lee, „Big data: Dimensions, evolution, impacts, and challenges,“ *Business Horizons*, Bd. 60, Nr. 3, pp. 293-303, 2017.
- [52] N. Shehab, M. Badawy und H. Arafat, „Big Data Analytics and Preprocessing,“ in *Machine Learning and Big Data Analytics Paradigms: Analysis, Applications and Challenges*, Cham, Springer International Publishing, 2021, pp. 25-43.
- [53] C. A. Mack, „Fifty Years of Moore's Law,“ *IEEE Transactions on Semiconductor Manufacturing*, Bd. 24, Nr. 2, pp. 202-207, 2011.
- [54] A. De mauro, M. Greco und M. Grimaldi, „What is big data? A consensual definition and a review of key research topics,“ *AIP Conference Proceedings*, Bd. 1644, pp. 97-104, 2015.
- [55] K. Taylor-Sakyi, „Big Data: Understanding Big Data,“ *arXiv preprint arXiv:1601.04602*, 2016.
- [56] M. A.-u.-d. Khan und M. F. N. Uddin, „Seven V's of Big Data understanding Big Data to extract value,“ *Proceedings of the 2014 Zone 1 Conference of the American Society for Engineering Education*, pp. 1-5, 2014.
- [57] Ishwarappa und J. Anuradha, „A Brief Introduction on Big Data 5Vs Characteristics and Hadoop Technology,“ *Procedia Computer Science*, Bd. 48, pp. 319-324, 2015.
- [58] A. Katal, M. Wazid und R. H. Goudar, „Big data: Issues, challenges, tools and Good practices,“ *2013 Sixth International Conference on Contemporary Computing (IC3)*, pp. 404-409, 2013.
- [59] hadoop, „HDFS Architecture Guide,“ The Apache Software Foundation , 10 10 2020. [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. [Zugriff am 7 May 2021].
- [60] G. Manikandan und S. Abirami, „Big Data Layers and Analytics: A Survey,“ in *Computer Communication, Networking and Internet Security*, Singapore, Springer Singapore, 2017, pp. 383-393.
- [61] A. Erraissi und A. Belangour, „Data sources and ingestion big data layers: meta-modeling of key concepts and features,“ *International Journal of Engineering & Technology*, Bd. 7, Nr. 4, pp. 3607-3612, 2018.
- [62] K. M. N. a. A.-C. N. N. Lyko, „Big data acquisition,“ in *New Horizons for a Data-Driven Economy*, Cham, Springer, 2016, pp. 39-61.
- [63] V. Bolón-Canedo, N. Sánchez-Marcano und A. Alonso-Betanzo, „Recent advances and emerging challenges of feature selection in the context of big data,“ *Knowledge-Based Systems*, Bd. 86, pp. 33-45, 2015.

- [64] D. Geng, C. Zhang, C. Xia, X. Xia, Q. Liu und X. Fu, „Big Data-Based Improved Data Acquisition and Storage System for Designing Industrial Data Platform,“ *IEEE Access*, Bd. 7, pp. 44574-44582, 2019.
- [65] D. Wu, S. Sakr und L. Zhu, „Big Data Storage and Data Models,“ in *Handbook of Big Data Technologies*, Cham, Springer International Publishing, 2017, pp. 3-29.
- [66] mongoDB, „JSON and BSON,“ mongoDB, [Online]. Available: <https://www.mongodb.com/json-and-bson>. [Zugriff am 5 May 2021].
- [67] R. Londner und A. Cabral, „An Introduction to Change Streams,“ mongoDB, 2018 June 18. [Online]. Available: <https://www.mongodb.com/blog/post/an-introduction-to-change-streams>. [Zugriff am 9 May 2021].
- [68] mongoDB, „Change Streams,“ mongoDB, [Online]. Available: <https://docs.mongodb.com/manual/changeStreams/?jmp=blog#watch-collection-database-deployment>. [Zugriff am 9 May 2021].
- [69] K. M. M. Thein, „Apache kafka: Next generation distributed messaging system,“ *International Journal of Scientific Engineering and Technology Research*, Bd. 3, Nr. 47, pp. 9478-9483, 2014.
- [70] Apache Kafka, „Introduction,“ Apache Kafka, [Online]. Available: <https://kafka.apache.org/intro>. [Zugriff am 10 May 2021].
- [71] hadoop, „Apache Hadoop,“ hadoop, [Online]. Available: <https://hadoop.apache.org>. [Zugriff am 09 May 2021].
- [72] J. Dittrich und J.-A. Quiane-Ruiz, „Efficient Big Data Processing in Hadoop MapReduce,“ *Proceedings of the VLDB Endowment*, Bd. 5, Nr. 12, pp. 2014-2015, 2012.
- [73] S. Chen und S. W. Schlosser, „Map-reduce meets wider varieties of applications,“ Intel, Pittsburgh, 2008.
- [74] hadoop, „Map Reduce Tutorial,“ hadoop, [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html#Purpose. [Zugriff am 09 May 2021].
- [75] D. Wu, S. Sakr und L. Zhu, „Big Data Programming Models,“ in *Handbook of Big Data Technologies*, Cham, Springer International Publishing, 2017, pp. 31-64.
- [76] Spark, Apache, „Apache spark,“ *Math 403*, Bd. 3, 2018.
- [77] Apache Spark, „RDD Programming Guide,“ Apache Spark, [Online]. Available: <https://spark.apache.org/docs/latest/rdd-programming-guide.html#overview>. [Zugriff am 10 May 2021].
- [78] Apache Spark, „Lightning-fast unified analytics engine,“ Apache Spark, [Online]. Available: <https://spark.apache.org>. [Zugriff am 10 May 2021].
- [79] F. Gürcan und M. Berigel, „Real-Time Processing of Big Data Streams: Lifecycle, Tools, Tasks, and Challenges,“ *2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, pp. 1-6, 2018.
- [80] M. Feick, N. Kleer und M. Kohn, „Fundamentals of Real-Time Data Processing Architectures Lambda and Kappa,“ in *SKILL 2018 - Studierendenkonferenz Informatik*, Bonn, Gesellschaft für Informatik e.V, 2018, pp. 55-66.
- [81] N. Marz, „How to beat the CAP theorem,“ 13 October 2011. [Online]. Available: <http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html>. [Zugriff am 11 May 2021].
- [82] N. Marz und J. Warren, *Big Data - Principles and best practices of scalable real-time systems*, Shelter Island: Manning Publication, 2015.

- [83] M. Kiran, P. Murphy, I. Monga, J. Dugan und S. S. Baveja, „Lambda architecture for cost-effective batch and speed big data processing,“ in *2015 IEEE International Conference on Big Data (Big Data)*, IEEE Access, 2015, pp. 2785-2792.
- [84] J. Lin, „The lambda and the kappa,“ *IEEE Internet Computing*, Bd. 21, Nr. 5, pp. 60-66, 2017.
- [85] J. Kreps, „Questioning the Lambda Architecture,“ O'Reilly, 2 July 2014. [Online]. Available: <https://www.oreilly.com/radar/questioning-the-lambda-architecture/>. [Zugriff am 11 May 2021].
- [86] S. a. R. A. Zhelev, „Big data processing in the cloud - Challenges and platforms,“ *AIP Conference Proceedings*, Bd. 1910, p. 060013, 2017.
- [87] J. G. Carbonell, R. S. Michalski und T. M. Mitchell, „An overview of machine learning,“ in *Machine learning*, Elsevier, 1983, pp. 3-23.
- [88] A. L. Fradkov, „Early History of Machine Learning,“ *IFAC-PapersOnLin*, Bd. 53, Nr. 2, pp. 1385-1390, 2020.
- [89] A. L. Samuel, „Some studies in machine learning using the game of checkers,“ *IBM Journal of R&D*, Bd. 3, Nr. 3, pp. 210-229, 1959.
- [90] T. M. Mitchell, *Machine learning*, New York: McGraw-Hill Education Ltd, 1997.
- [91] U. S. Shanthamallu, A. Spanias, C. Tepedelenlioglu und M. Stanley, „A Brief Survey of Machine Learning Methods and their Sensor and IoT Applications,“ *2017 8th International Conference on Information, Intelligence, Systems Applications (IISA)*, pp. 1-8, 2017.
- [92] R. Sathya und A. Abraham, „Comparison of supervised and unsupervised learning algorithms for pattern classification,“ *International Journal of Advanced Research in Artificial Intelligence*, Bd. 2, Nr. 2, pp. 34-38, 2013.
- [93] M. W. Berry, A. Mohamed und B. W. Yap, *Supervised and unsupervised learning for data science*, Cham: Springer, 2019.
- [94] P. Cunningham, M. Cord und S. J. Delany, „Supervised Learning,“ in *Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval*, Berlin, Heidelberg, Springer Berlin Heidelberg, 2008, pp. 21-49.
- [95] R. Caruana und A. Niculescu-Mizil, *An Empirical Comparison of Supervised Learning Algorithms*, Ney York: Association for Computing Machinery, 2006.
- [96] J. Hartmann, J. Huppertz, C. Schamp und M. Heitmann, „Comparing automated text classification methods,“ *nternational Journal of Research in Marketing*, Bd. 36, Nr. 1, pp. 20-38, 2019.
- [97] S. B. Kotsiantis, D. Kanellopoulos und P. E. intelas, „Data preprocessing for supervised leaning,“ *International Journal of Computer Science*, Bd. 1, Nr. 2, pp. 111-117, 2006.
- [98] Apache Spark, „ML Pipelines,“ Apache Spark, [Online]. Available: <https://spark.apache.org/docs/latest/ml-pipeline.html#ml-persistence-saving-and-loading-pipelines>. [Zugriff am 12 May 2021].
- [99] scikit -earn, „Model persistence,“ scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/model_persistence.html. [Zugriff am 12 May 2021].
- [100] Python, „pickle - Python object serialization,“ Python, [Online]. Available: <https://docs.python.org/3/library/pickle.html>. [Zugriff am 12 May 2021].
- [101] M. Pavloski, „scikit-learn: Save and Restore Models,“ Stack Abuse, [Online]. Available: <https://stackabuse.com/scikit-learn-save-and-restore-models/>. [Zugriff am 12 May 2021].

- [102] S. Brown, „The C4 model for visualising software architecture,“ -, [Online]. Available: <https://c4model.com>. [Zugriff am 2 June 2021].
- [103] K. Chauhan, S. Jani, D. Thakkar, R. Dave, J. Bhatia, S. Tanwar und M. S. Obaidat, „Automated Machine Learning: The New Wave of Machine Learning,“ *2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, pp. 205-212, 2020.
- [104] P. S. Foundation, „Data Structures,“ Python Software Foundation, [Online]. Available: <https://docs.python.org/3/tutorial/datastructures.html#dictionaries>. [Zugriff am 05 June 2021].
- [105] scikit learn, „sklearn.preprocessing.OneHotEncoder,“ 2020. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html>. [Zugriff am 29 March 2021].
- [106] scikit learn, „sklearn.preprocessing.OrdinalEncoder,“ 2020. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html>. [Zugriff am 29 March 2021].
- [107] W. Zhou, L. Li, M. Luo und W. Chou, „REST API Design Patterns for SDN Northbound API,“ *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, pp. 358-365, 2014.
- [108] C. Molnar, „Model-Agnostic Methods,“ in *Interpretable machine learning*, Bookdown, 2021.
- [109] PostgreSQL, „Write-Ahead Logging (WAL),“ PostgreSQL, [Online]. Available: <https://www.postgresql.org/docs/9.1/wal-intro.html>. [Zugriff am 6 June 2021].