

Sommersemester 2021

**Eine Datenverarbeitungsarchitektur zum Erstellen und Verwen-
den 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
25. Juni 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 25. Juni 2021.

A handwritten signature in black ink, appearing to read "Lars Grespan". It is written in a cursive style with a prominent loop on the left and a straight line extending from the right side.

Zusammenfassung

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 sowohl die Erstellung der Modelle als auch die Verwendung der Modelle in Form der Fehlervorhersagen eingebunden sein. Die Erstellung der Modelle und deren Verwendung in Form von Fehlervorhersagen benötigen eine vorangeschaltete Datenverarbeitung. Zentraler Forschungsgegenstand dieser Thesis ist dabei die Entwicklung einer Architektur, die eine einheitliche Datenverarbeitung für die Modellerstellung und die Modellverwendung realisiert.

Abstract

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. To establish 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

Zusammenfassung.....	II
Abstract.....	III
Abbildungsverzeichnis	VI
Tabellenverzeichnis	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	16
2.2.1 Ziel des Prototyps	16
2.2.2 Systemfunktionalität	17
2.2.3 Use-Case Übersicht	17
2.2.4 Nichtfunktionale Anforderungen	27
2.2.5 Lieferumfang.....	27
3 Grundlagen.....	28
3.1 Data Processing	28
3.1.1 Big Data.....	28
3.1.2 Datenverarbeitung vs. Big Data Verarbeitung	30
3.1.3 Big Data Schichten	31
3.1.4 Big Data Technologien	34
3.1.5 Big Data Processing	38
3.2 Big Data Processing Architekturen	39
3.2.1 Lambda-Architektur	40
3.2.2 Kappa-Architektur.....	41
3.3 Maschinelles Lernen	42
3.3.1 Definition.....	42
3.3.2 Supervised Learning und Unsupervised Learning	43
3.3.3 Speichern und Laden von ML-Modellen.....	44
4 Implementierung des Prototyps.....	45
4.1 Aufbau der Daten und der Produktionsumgebung	46
4.2 System Context Diagram.....	47
4.2.1 REST Implementation	47
4.2.2 Change Streams Implementation	49
4.3 Event Generation.....	50
4.4 Model Training.....	52
4.5 Event Processing und Data Model	54
4.5.1 Batch-Schicht	55
4.5.2 Speed-Schicht	57
4.6 Simulate Test Stations.....	58
4.6.1 Simulate Teststations (REST)	59
4.6.2 Simulate Test Stations (Change Streams)	59

4.7	Data Processing Server	60
4.7.1	Processing Server (REST)	60
4.7.2	Result Presentation & Change Streams Service (Change Streams)	61
4.8	Vergleich der Implementierungen	62
4.8.1	Performancevergleich.....	62
4.8.2	Qualitativer Vergleich	65
5	Limitationen und Resümee.....	66
5.1	Resümee und Bewertung der Forschungsfragen	66
5.2	Stärken und Limitationen der Arbeit	68
5.3	Ausblick auf zukünftige Arbeiten	68
Glossar		IX
Abkürzungsverzeichnis		XI
Anhang		XII
Literaturverzeichnis		XIII

Abbildungsverzeichnis

Abbildung 1 - Big Data Schichten	32
Abbildung 2 - Kafka-Architektur	35
Abbildung 3 - MapReduce Funktion	37
Abbildung 4 - Spark Bibliotheken	37
Abbildung 5 - Big Data Processing Architektur	38
Abbildung 6 - Lambda-Architektur	40
Abbildung 7 - Lambda-Architektur und dazugehörige Tools	41
Abbildung 8 - Kappa-Architektur	41
Abbildung 9 - Event Processing System	47
Abbildung 10 - REST Implementation	48
Abbildung 11 - Change Streams Implementation	50
Abbildung 12 - Event Generation	51
Abbildung 13 - Model training	53
Abbildung 14 - Event Processing	54
Abbildung 15 - Simulate Test Stations (REST)	59
Abbildung 16 - Simulate Test Stations (Change Streams)	60
Abbildung 17 - Processing Server	61
Abbildung 18 - Change Streams Service	61
Abbildung 19 - Result Presentation	62

Tabellenverzeichnis

Tabelle 1 - Erstellen Datensatz Teststation 1.....	18
Tabelle 2 - Erstellen Datensatz Teststation 2.....	18
Tabelle 3 - Erstellen Datensatz Teststation 3.....	19
Tabelle 4 - Speichern der Datensätze	19
Tabelle 5 - Simulieren von Live-Events	20
Tabelle 6 - Erstellen des Datenmodells	21
Tabelle 7 - Bereinigung der Events	21
Tabelle 8 - Erstellung abgeleiteter Attribute.....	21
Tabelle 9 - Speichern der bearbeiteten Events.....	22
Tabelle 10 - Verbinden von Events	22
Tabelle 11 - Laden der Events	23
Tabelle 12 - Laden der Labels	23
Tabelle 13 - Trainieren der ML-Modelle.....	24
Tabelle 14 - Speichern der ML-Modelle	25
Tabelle 15 - Erstellen einer Service Schnittstelle	25
Tabelle 16 - Laden der ML-Modelle	26
Tabelle 17 - Erstellen einer Vorhersage	26
Tabelle 18 - Vorhersage speichern	26
Tabelle 19 - Ergebnisse anzeigen	26
Tabelle 20 - Traditional Data vs. Big Data.....	31
Tabelle 21 - Performancevergleich REST - Change Streams.....	63

1 Einleitung

Der Mensch ist perfekt darin neue Werkzeuge zu erschaffen, oder ein bestehendes Werkzeug für einen neuen Verwendungszweck anzupassen [1]. Die Erfindungen, die vom Menschen im Laufe der Geschichte entwickelt wurden, sind unzählig. Viele sind dabei von kleiner Bedeutung und einige hatten das Potenzial die Welt zu verändern – zum Beispiel die Erfindung des Rads oder des Buchdrucks.

Neben vielen anderen Bereichen hat der technologische Fortschritt auch Einfluss auf die Industrie. Der Industriezweig ist laut Lasi et. al. [2] jener Wirtschaftsbereich, der 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 [2]. Die erste industrielle Revolution entspringt aus der Erfindung der Dampfmaschine, durch die 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. [2] 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 dessen Kontext der Begriff Industrie 4.0 (die vierte industrielle Revolution) verwendet wird [2]. Technologischer Fortschritt gilt daher auch hier als Treiber der vierten industriellen Revolution.

Der breitgefächerte Industriezweig bietet für die Anwendung von 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 denen 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 [3].

Resultierend aus der intelligenten Produktion ergeben sich für das Qualitätsmanagement diverse Optimierungspotenziale. Eines dieser Optimierungspotenziale ist das Verwenden von Modellen Maschinellen Lernens (ML) zur Fehlervorhersage in der Produktion. Fehlerhafte Teile in der Produktion können dadurch frühzeitig aussortiert werden, was sich positiv auf die Kosten auswirkt. Voraussetzung für den Einsatz von ML-Modellen in der Produktion sind dabei untereinander vernetzte Objekte innerhalb der Fertigung. Im Qualitätsmanagement sind diese Objekte verschiedene Stationen,

die Qualitätsprüfungen an den Produkten durchführen. Die erfassten Daten werden dann jeweils von den ML-Modellen zur Fehlervorhersage benutzt.

Zur Umsetzung der Fehlervorhersage mittels ML-Modellen sind zum einen die Erstellung der Modelle notwendig und zum anderen die Verwendung der Modelle. Sowohl für die Erstellung als auch für die Verwendung 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, für 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, die für die Erstellung von ML-Modellen und die Fehlervorhersage durch die ML-Modelle verwendet werden kann. Gleichzeitig wird geprüft, wie eine Architektur zur Umsetzung dieser einheitlichen Datenverarbeitung aussehen kann. Zusätzlich müssen heutige Datenverarbeitungssysteme eine immer größer werdende Anzahl an Daten prozessieren. Um dieser Problematik entgegen zu wirken, werden zur Implementierung einer solchen Datenverarbeitungsarchitektur verschiedene Verarbeitungstools gegenübergestellt und geprüft. Aus der oben genannten Problembeschreibung ergeben sich für diese Forschungsarbeit deshalb folgende Forschungsfragen (RQ):

RQ1:

Inwieweit lässt sich eine einheitliche Datenverarbeitung in Form von gleichem Source Code für die Erstellung der ML-Modelle und 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, das für die ML-Modellerstellung und für die Fehlervorhersage durch die ML-Verwendung eingesetzt werden kann. Im Rahmen der einheitlichen Datenverarbeitung wird eine entsprechende Architektur unter Berücksichtigung der Forschungsfragen erstellt und implementiert. Die Forschungsarbeit dient daher zur Validierung 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 eine reale Produktionsumgebung dem Autor zur Verfügung stehen mit Rechenressourcen, die dem 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 an die Forschungen dieses Projekts anzuschließen, wird im PREFERML Kapitel der Kontext dazu gegeben. Daraus abgeleitet wird eine Anforderungsanalyse zur Erstellung einer Datenverarbeitungsarchitektur durchgeführt. Dabei liegt der Schwerpunkt auf der Beantwortung der Forschungsfragen.

Nachdem 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 dem 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 Limitationen der Forschungsarbeit dargelegt und ein Ausblick auf daran anknüpfende Forschungsarbeiten gegeben.

2 Forschungsprojekt PREFERML

Mit dem zunehmenden Einsatz von Künstlicher Intelligenz in der Wirtschaft laufen synchron dazu die Erfolge, die 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% [4]. 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 [5]. Vor allem in Produktionsumgebungen sind die Einsatzgebiete für diese Technologie vielseitig. Beispiele für den Einsatz in der Produktion sind Instandhaltung, Logistik, Qualitätskontrolle, Prozessentwicklung, Prozessoptimierung, Automatisierung, usw. [6].

Schlüssel zum Erstellen erfolgreicher ML-Modelle sind an erster Stelle Daten, die in den Prozessen eines Unternehmens generiert werden. Sind nur wenige oder nicht aussagekräftige Daten verfügbar, so lassen sich nur schlecht Muster darin erkennen und es folgt, dass das Erstellen und die Adaption von ML-Modellen in solchen Fällen nicht sinnvoll ist [7]. Durch die zunehmende Digitalisierung der Unternehmen und deren Geschäftsprozesse ist die Problematik des Nichtvorhandenseins von Daten heutzutage in den Hintergrund geraten. Daten werden innerhalb von Geschäftsprozessen generiert und in Datenbanken abgelegt. Laut dem Index für die digitale Wirtschaft und Gesellschaft (DESI) liegt der Grad der Digitalisierung in Deutschland bei 56,1% [8]. Fokus liegt daher zunehmend auf dem sinnvollen Einbinden der generierten KI-Modelle in die bereits vorhandenen Geschäftsprozesse [7].

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, das im folgenden Abschnitt 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 [9].

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

- Aussortieren oder Nachbearbeiten von Teilen nach frühzeitigem 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:

- 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?“ [9]
- 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?“ [9]
- 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 aktiv am Prozess beteiligt sind?“ [9]

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 [10] [11] [12] [13] [14].

Fokus der vorliegenden Arbeit ist das Erstellen einer Architektur (im folgenden auch Prototyp genannt), die die verschiedenen Aspekte der 2. Fragestellung berücksichtigt. Im Rahmen der bereits absolvierten Forschung sind eine große Anzahl von ML-Modellen entstanden, die nun in eine Automatisierungsumgebung eingebunden werden. Zum Erstellen eines solchen Prototyps gilt es verschiedene Problematiken zu berücksichtigen, die im folgenden Abschnitt genauer erläutert werden. Abgeleitet von den verschiedenen Problematiken wird anschließend eine Anforderungsanalyse durchgeführt, die 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 einzugrenzen, werden die Daten und das Produktionsumfeld nur zum kleinen Teil modelliert und abgebildet. Für eine Einbindung in das im Forschungsprojekt vorliegende 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 ihrer 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 eine hohe Qualität der Produkte zu gewährleisten. Die Qualitätssicherung wird mit Hilfe von Qualitätsprüfungen am Ende des jeweiligen Produktionsschritts durchgeführt. Moderne Produktionsumgebungen führen Qualitätsprüfungen nicht mehr nur am Ende des Produktionsprozesses durch, sondern auch zwischen einzelnen Fertigungsschritten. Mangelhafte Teile können so frühzeitig aussortiert oder nachjustiert werden. Diese Art der Qualitätsprüfung wird von Peter Hohenberger in „Qualitätsmanagement in der Produktion“ als die klassische, produktorientierte Qualitätsprüfung beschrieben [15]. Bei dieser Qualitätsprüfung werden „qualitätsrelevante Merkmale“ [15] gemessen. Messmittel sind zumeist Sensoren. Durch die Verarbeitung der Messdaten und die Integration in unternehmensinterne Informationssysteme können die Messdaten noch weiter verarbeitet werden [15].

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 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 [15].

In mehrstufigen Produktionslinien, bei denen zwischen einzelnen Fertigungsschritten immer wieder Qualitätsprüfungen durchgeführt werden, wird ein mangelhaftes Teil genau an der Stelle aus dem Prozess entfernt, an der 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, das 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 der dritten Qualitätsmessung identifiziert und aussortiert bzw. nachbearbeitet werden. In einer Fließbandproduktion ist das Erkennen von solchen Zusammenhängen durch manuelle Datenauswertung schlachtweg 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 [13].

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 müssen. Ein mehrmaliges Durchführen desselben Produktionsschrittes führt zu Kosten, die ebenfalls eingespart werden können [16].

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 werden.

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, die bei den Qualitätsmessungen auftreten können. Durch die Varianz in den Produkten und den Messungen entsteht zusätzliche Komplexität [14]. Insgesamt sammelt sich eine Vielzahl von Problemen und Herausforderungen bei Datenanalysen in Fertigungsarbeiten an, die beim Implementieren des Prototyps wichtig sind. Diese werden im nächsten Abschnitt behandelt.

2.1.2 Probleme und Herausforderungen bei Datenanalysen im Produktionsumfeld

Bevor die Analyse der Anforderungen an den zu erstellenden Prototypen durchgeführt wird, gilt es noch einige wichtige Probleme zu adressieren und zu berücksichtigen. Im Rahmen des Forschungsprojekts PREFERML wurden bereits wichtige Herausforderungen im Bereich der Datenanalytik herausgearbeitet. Diese referenzieren auf den Bericht “Data Science Approaches to Quality Control in Manufacturing: A Review of Problems, Challenges and Architecture” [13]. Der Bericht bezieht sich auf typische Schwierigkeiten bei datengesteuerten Ansätzen zur Qualitätskontrolle in Fertigungs-umgebungen 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 die sich die Datenanalyse beeinflussende Faktoren in der Fertigungs-umgebung kategorisieren lassen [13]:

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

Diese vier Kategorien werden in den nächsten Unterabschnitten erklärt und die beeinflussenden Faktoren beschrieben. Die Schwerpunkte des Berichts 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 Unterabschnitt performancebeeinflussende Faktoren beschrieben, die 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 Fehlertypen
- Unausgewogene Datenverteilungen
- Nichtlineare Produktionsprozesse und Produktionsschleifen
- Konzeptdrift
- Kostensensitive Modellierung

Heterogene Produktfamilien und Betriebsmittel

Fertigungsumgebungen sind häufig durch mehrere Produktvarianten und unterschiedliche Versionen von Betriebsmitteln gekennzeichnet, z. B. Maschinen, Werkzeuge oder Werkstattausstattung. 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 Betriebsmitteln steigt synchron dazu die Menge der 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 [13] [17] [18] [19] [20].

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 Labels 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 Labels mehr als nur zwei Werte annehmen. Im Produktionsumfeld können diese Labels 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 [13] [20] [21] [22].

Uunausgewogene 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 Produktrvielfalt ungleichmäßig verteilt sind. So kann es vorkommen, dass diese Klassen oder Produktvarianten unterrepräsentiert sind, was sich negativ auf die ML-Performance auswirken kann [13] [20] [21] [19] [22] [18] [23].

Nichtlineare Produktionsprozesse und Produktionsschleifen

In einer typischen Produktionslinie, wie in Abschnitt 2.1.3 Maschinelles Lernen im Qualitätsmanagement beschrieben, durchläuft ein Produkt verschiedene Produktionsstufen. Nach den einzelnen Produktionsschritten werden jeweils Qualitätsprüfungen durchgeführt [15] [13]. Im Idealfall durchläuft ein Teil die Produktionslinie, ohne dabei bei den Qualitätsprüfungen Abweichungen aufzuweisen. In der Datenerfassung einer solchen Produktionslinie spiegelt sich die lineare Bearbeitungsreihenfolge wieder, 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 vorhergehende 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 Produktions-schleifen gesprochen. In Bezug auf die Produktionsdaten bedeutet ein Wiedereinsetzen von Teilen an vorherigen Produktionsschritten - oder am gleichen Produktionsschritt – ein Wiederauftreten des entsprechenden Teils in den Daten. Der Datensatz enthält Bezüge auf dieselben Teile am gleichen Produktionsschritt mit unterschiedlichen Zeitstempeln [13] [24].

Konzeptdrift

ML-Modelle werden darauf trainiert Muster in Daten zu erkennen - unabhängig davon, ob alle Produktvarianten und alle Fehlertypen in den Daten ausreichend vertreten sind. Input für das Training der ML-Modelle ist ein Datensatz bestehend aus Features mit den zugehörigen Labels. Genaue Details zum Trainieren von ML-Modellen werden in [Kapitel 3 - Grundlagen](#) 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 stark variieren oder sich im Laufe der Zeit ändern, dann wird das 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 Konzeptdrift eigentlich immer auftritt. 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 ein Werkstück ein Qualitätskontrolltor passieren kann oder nicht. Um auf Konzeptdrifts zu reagieren wird ein System benötigt, das 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 redundantes Anpassen von Modellen möglichst 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 [13] [25] [26] [24].

Kostensensitive Modellierung

Damit ML-Modelle im Qualitätsmanagement eingesetzt werden können, 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 darin, 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 dann erst 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 [13] [20] [27] [28] [29].

2.1.2.2 Herausforderungen zu den Merkmalen der zugrunde liegenden Daten und Datenquellen

In diesem Abschnitt 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 Produktions-umgebung auf und sind deshalb nicht zu vernachlässigen. Hierzu 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 reversible 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 „Multimodel Data Fusion: An Overview of Methods, Challenges and Prospects“ von Lahat et al. [30] als Modalität bezeichnet. Zur Differenzierung muss jede Modalität eine Information liefern, die 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 kann diesen Hindernissen entgegengewirkt werden [13] [30] [24] [19] [31].

Fehlende Primary Key und Foreign Key Beziehungen

Bei der Datenerfassung über verschiedene Prozessstufen in Fertigungsumgebungen muss sichergestellt werden, dass die gesammelten Daten eindeutig und genau den relevanten Elementen zugeordnet werden. Dies erfordert eine eindeutige Primary Key und Foreign Key Beziehung zwischen einem Werkstück und allen verschiedenen dazugehörigen Informationen, die während der einzelnen Prozessstufen erfasst werden. Vielen Datenstrukturen und Datenformaten, die in realen Produktionsanwendungen verwendet werden, fehlen dagegen die notwendigen Primary Key und Foreign Key Beziehungen [13] [20] [19].

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 muss, besonders bei Echtzeit-Auswertungen, berücksichtigt werden. Viele der Datenanalysen benötigen diese zeitliche Abfolge als Anforderung für deren Umsetzung. Eine Lösung zum Synchronisieren der Systemzeit der Teststationen bietet das Network Time Protocol (NTP). Falls dieses Protokoll nicht unterstützt wird, müssen andere Methoden zum Zeitabgleich verwendet werden [13].

Doppelte und ungültige Datenpunkte

Wie bereits in [Abschnitt 2.1.2.1 – Herausforderungen der Fertigungsdomäne](#) beschrieben, können Produkte aufgrund von nicht-linearen Fertigungsverläufen und durch Produktionsschleifen 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 und ungültige Datenpunkte – müssen vor dem Trainieren der ML-Modelle entfernt werden [13] [20] [19].

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 der Bedeutung von Variablen und deren Beziehungen untereinander. 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 [13] [20].

Unzureichende Datenerfassung

Fertigungsprozesse folgen typischerweise zeitkontinuierlichen physikalischen Vorgängen. Dementsprechend nehmen die zur Qualitätsprüfung herangezogenen Messmittel Informationen über die Zeitpunkte auf. Die Aufnahme dieser Zeitpunkte durch entsprechende Signale ist an dieser Stelle 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 Abtastfrequenz verdichtet, kann es im schlimmsten Fall dazu führen, dass keine informationsreichen Merkmale abgeleitet werden können [13] [20].

Nicht reversible 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 das zugrunde liegende Datenbankschema angepasst werden muss. Eine Möglichkeit zur Behebung des Problems ist eine schemalose Datenbank. Allerdings müssten die Applikationen, die auf die Datenbank zugreifen, selbst interpretieren, um welche Datentypen es sich bei den Attributen handelt [13].

2.1.2.3 Herausforderungen zu den Analysen

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

- Fehlendes Vertrauen in die ML-Modelle und die Analyseergebnisse
- 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 [2.1.2.1 Herausforderungen in der Fertigungsdomäne](#) unter Abschnitt „Kostensensitive Modellierung“ beschrieben, und auch von Wilhelm et. al. [13] 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 die schwierige Interpretierbarkeit der Vorhersagen kann die Implementierung der ML-Modelle bei den Endbenutzern auf Widerstände treffen [13].

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 passende Werkstück 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 Werkstücks. Für die ersten Informationen 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. [10]. 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 Varianten 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 [13] [32] [33] [20].

Verwaltung von verschiedenen ML-Modellen

In den vorhergehenden Abschnitten wurden bereits Herausforderungen genannt, die ML-Modelle bewältigen müssen, um in der Praxis Anwendung zu finden. Durch die in Abschnitt [2.1.2.1 Herausforderungen der Fertigungsdomäne](#) beschriebenen Probleme bezüglich der heterogenen Produktfamilien und Betriebsmittel, als auch der 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, Entscheidungsbaum, usw.), Input-Features, Klassenlabel, Auswertungen und Datenquellen. Resultierend aus der großen Anzahl von ML-Modellen und deren Ausprägungen ergibt sich ein Verwaltungsproblem, das zum Einbinden der Modelle in der Praxis gelöst werden muss. Ein erster Ansatz zur Lösung des Verwaltungsproblems ist in Form einer Datenbank, die die Modelle verwaltet, bereits erarbeitet worden (ModelDB) [34]. 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 die Verknüpfung von Metadaten der ML-Modelle in Verbindung mit Informationen über die Produkte [13] [35].

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 zeitaufwändige 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 [13].

2.1.2.4 Herausforderungen der Software

Auch im Zusammenhang mit der Software gilt es einige Herausforderungen in Bezug auf die 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, die während des Produktionsprozesses entstehen, werden über Schnittstellen von den Maschinen oder Baugruppen übertragen. Über eine Produktionsumgebung hinweg gibt es viele verschiedene Maschinen und Baugruppen, was zu einem breiten Spektrum von 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 finden über Exportformate statt. Für eine Implementierung im Kontext der datengesteuerten Ansätze zur Qualitätskontrolle ist durch die veraltete Schnittstellentechnik ein zum Teil erheblicher Übertragungsaufwand zu bewältigen [13] [36].

Individuelle Lösungsskripte

Eine typische Vorgehensweise bei datenwissenschaftlichen Projekten in diesem Rahmen ist der Cross-industry standard process for data mining (CRISP-DM) Methodology [37]. Skriptsprachen wie R oder Python bieten eine Vielzahl von Bibliotheken, die 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 nächsten 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. Eine weitere Lösungsidee bietet sich durch den Einsatz des im vorherigen Abschnitts beschriebenen AutoML an [13].

2.1.2.5 Herausforderungen an die Performance

Aus den bisher aufgezählten Aspekten ergibt sich eine Komplexität von verschiedenen Fällen, die für eine Automatisierung abgedeckt werden müssen. Je größer die Anzahl der Modalitäten, desto größer ist die Anzahl der Daten. Um das Automatisierungsproblem mit der damit verbundenen hohen Datenmenge zu lösen, ist eine Vorverarbeitung der Daten notwendig. Um jedoch die notwendige Zuverlässigkeit zu gewährleisten, reichen herkömmliche Methoden bei einer solchen Menge an Daten nicht aus [38]. Werden Werkstücke von den ML-Modellen als mangelhaft deklariert, so muss das korrespondierende Werkstück sofort aus dem Fertigungsprozess aussortiert werden. Wird das Werkstück nicht sofort aussortiert, fallen durch unnötig ausgeführte Fertigungsschritte Kosten an den nachfolgenden Fertigungsstationen an. In diesem Kontext ist deshalb eine Datenauswertung mit geringer Latenzzeit notwendig. Für die Datenverarbeitung in Echtzeit gibt es bereits verschiedene Tools und Technologien, die diesen Herausforderungen mit ihren Funktionalitäten entgegenwirken. Zu diesen Technologien und Tools zählen das Big Data Framework Apache Spark, Datenuntersuchungsplattformen wie Hadoop oder MapReduce, Datenbanken wie Cassandra und MongoDB, sowie Messaging Dienste wie Kafka Streams [39]. 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 schließt mehrere Datenbankzugriffe ein. Durch die hohe Anzahl der Datensätze können die Datenbankzugriffe zu einem Performanceproblem führen. Mit jeder Datenbankabfrage steigt die Latenzzeit im Gesamtsystem. Vorzugsweise können Daten durch sogenannte Sessions, nach einmaligem 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 [40]. Die Auswahl der passenden Datenbank kann also dazu beitragen, die Performance des Gesamtsystems zu verbessern. Zusätzlich können die Zugriffe mit Optimierungsmethoden, wie beispielsweise Indexen, performanter gestaltet werden [41].

Parallelisierung der Datenverarbeitung

Eine zusätzliche Optimierung der Performance ist die Verteilung und Parallelisierung der Datenverarbeitung [42]. Eine Parallelisierung der Datenverarbeitung bedeutet einerseits eine deutlich verbesserte Performance, anderseits ist mit ihr eine Verteilung der Datenverarbeitung auf verschiedene Prozessoren notwendig. Um diese Verteilung konsistent zu halten, sind Synchronisations- und Verwaltungsaufgaben erforderlich, was zu weiterer Komplexität und weiteren Herausforderungen führt [43].

Verwendung performanter Bibliotheken

Wie im vorherigen Abschnitt „Individuelle Lösungsskripte“ 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, die das Manipulieren der Daten stark vereinfachen. In Python sind diese Bibliotheken beispielsweise pandas oder numpy. Zudem vereinfachen 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 drastisch sinkt. Deshalb ist es sinnvoll, diese Bibliotheken zu verwenden, und nicht nur auf die eingebundenen Funktionalitäten der Skriptsprache zurückzugreifen [44].

2.2 Anforderungsanalyse

Basierend auf der vorhergehenden Problembeschreibung wird im Rahmen dieser Abschlussarbeit ein Prototyp entwickelt, der 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 die Anforderungsanalyse aufgenommen.

Die in den folgenden Abschnitten aufgeführten Anforderungen bilden den Rahmen des Prototyps. Ziel ist es diese Anforderungen umzusetzen. Aufgrund multipler Umsetzungsmöglichkeiten des Event Processings sollen zudem verschiedene Ansätze implementiert und auf ihre 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 voll funktionsfähige Lösung zu haben, die 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
- Speichern der Datensätze
- Simulieren von Live-Events

Prototyp für die Datenverarbeitungsarchitektur:

- Datenverarbeitung
 - Erstellen des Datenmodells
 - 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 gibt. Aufgrund begrenzter Ressourcen beschränkt sich der Autor hier auf die Simulation von drei Teststationen, da dies ausreichend ist, um die verwendeten Konzepte zu beschreiben und zu beurteilen.

Name	Erstellen Datensatz Teststation 1
Kurzbeschreibung:	Zu generieren ist ein .csv-Datensatz. In die Werte der Spalten werden künstlich Fehler eingebaut, sodass eine spätere Daten-bereinigung 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 tauchen 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 Produktions-umgebung aussortiert werden). Da die <i>Teststation2</i> in der realen Welt die Messungen erst nach der <i>Teststation1</i> macht, muss dies auch in der Simulation in den Spalten Date und Time zu erkennen sein (<i>Teststation1</i> : Date + Time < <i>Teststation2</i> : 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 tauchen auch im <i>Teststation3</i> 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 Produktions-umgebung aussortiert werden). Da die <i>Teststation3</i> in der realen Welt die Messungen erst nach der <i>Teststation1</i> macht, muss dies auch in der Simulation in den Spalten Date und Time zu erkennen sein (<i>Teststation2</i> : Date + Time < <i>Teststation3</i> : 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 eigene 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, sondern einfach die vorhandenen Datensätze, die bereits in der Datenbank (EventDB) gespeichert wurden, pro Teststation durchiteriert werden. Für jede Reihe soll ein Service aufgerufen werden, der als Trigger für die weitere Verarbeitung der Daten dient.
Ablauf	<ol style="list-style-type: none"> 1. Iterieren der Datensätze <i>Teststation1/2/3</i> 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, die nur 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, als Teil der zu prüfenden Forschungsfrage RQ1, auf der Batch-Schicht und auf der Speed-Schicht 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 kompletter Datensatz von Teststation1/2/3. Output kann je nach Input dann ebenfalls ein einzelnes Event sein oder ein kompletter Datensatz. Zur Vereinfachung wird in den Beschreibungen für ein einzelnes Event oder für einen Datensatz der Begriff **Event** verwendet.

Name	Erstellen des Datenmodells
Kurzbeschreibung:	Erstellen eines Datenmodells. Das Datenmodell 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 die ein One-Hot-Encoding ausgeführt werden soll 3. Erstellen eines Dictionaries mit den kategorischen Spaltennamen und deren Wertausprägungen

Tabelle 6 - Erstellen des Datenmodells

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 einer REGEX 4. Löschen von Textspalten (oder Spalten, die immer den gleichen Wert haben). Informationen über diese Spalten werden aus dem Datenmodell 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.
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, die im Datenmodell 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, die zusammengefügt werden sollen, geladen werden können. Dazu werden die prozessierten Events in der EventDB gespeichert.
Ablauf	1. Speichern der prozessierten Events in 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: 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 <i>Teststation1</i> 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 <i>Teststation3</i>) kein Verbinden der Events notwendig, da bei der letzten Teststation keine Vorhersage gemacht wird.
Ablauf	<ol style="list-style-type: none"> Überprüfen, von welcher Teststation die Events stammen. Laden der Events von vorhergehenden Teststationen Verbinden der Events über die Component_ID unter Ausschluss der oben genannten Spalten

Tabelle 10 - Verbinden von Events

Der Output der Datenverarbeitung kann, abhängig davon, von welcher Teststation das Event stammt, variieren. Die unten stehende 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 *Teststationen1/2/3* gearbeitet.

Name	Laden der Events
Kurzbeschreibung:	Gespeicherte Events werden aus der Event DB geladen und 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 Labels
Kurzbeschreibung:	Zum Trainieren der Modelle werden zusätzlich zu den bereits vorverarbeiteten Events die zugehörigen Labels benötigt. Das Label zum Trainieren der Modelle ist jeweils die Spalte „Error_Message“ der nachfolgenden Teststation. Das heißt: für die Events der <i>Teststation1</i> muss die Spalte „Error_Message“ der Daten von <i>Teststation2</i> 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 der 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 Labels

Das Hinzufügen der Labels 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 der Modelle stehen 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 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 Namen der Exportdatei muss für den späteren Zugriff die Teststation_ID, der Product_Type als auch eine Information, mit welchen Daten von welchen Teststationen das Modell trainiert wurde, enthalten sein.
Ablauf	<ol style="list-style-type: none"> 1. Generieren des Pfades und des Namens der Datei 2. Ablegen des Modells

Tabelle 14 - Speichern der ML-Modelle

Speed-Schicht

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

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, das 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 Werkstück 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 Werkstücke aus dem Produktionsprozess auszusortieren, sollen die Vorhersagen auf einem Web Interface angezeigt werden. Werkstücke, die aussortiert werden müssen, sollen rot markiert sein und Werkstücke, die 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 Abschnitt spezifizierten Systemfunktionalitäten hinaus muss die zu entwickelnde Lösung zusätzlich noch weitere, nichtfunktionale Anforderungen erfüllen. Eine dieser Anforderungen ist die Verwendung von ausschließlich frei zugänglicher 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 und CSS implementiert werden. Die Umsetzung der Service-Schnittstelle ist Teil der Forschungsfrage, weshalb im Rahmen dieses Use-Cases mehrere Technologien implementiert werden können. Gleiches gilt für 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 endgültigen Lösung verwendet werden, hängt von der Performance der jeweiligen Technologien ab. Aufgrund des Big Data Kontexts, in dem sich diese Arbeit und der Prototyp bewegt, muss die endgültige Implementierung im Stande sein hohe Datenmengen möglichst schnell und unter geringen Performanceeinbußen in Echtzeit zu bearbeiten. Die Anforderungen an die Performance in Echtzeit sind allerdings nur in der Speed-Schicht von Bedeutung. Das heißt, dass insbesondere dort die Datenbankzugriffe minimiert werden müssen.

Aufgrund von limitierten Ressourcen beschränkt 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 Skalieren auf weitere Teststationen (und den damit verbundenen 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 abgegrenztem Code und Effizienz bzw. Performance, so muss abgewogen werden, welches Ziel die höhere Priorität an der jeweiligen Stelle hat.

2.2.5 Lieferumfang

Resultat der Forschungsarbeit ist ein voll funktionsfähiger 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, das ebenfalls einen 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 Implementierung zu gewährleisten. Die zugrundeliegenden Konzepte werden deshalb in diesem Kapitel erklärt.

3.1 Data Processing

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 [45]. Nationale Bemühungen wie Industrie 4.0 (Deutschland), Made-in China 2025 (China) oder Advanced Manufacturing Partnership (Vereinigten 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 [46] [47]. Dem Bericht von Ismail et. al. zu Folge [45] 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 um daraus resultierende 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, die im Abschnitt [2.1.2 Probleme und Herausforderungen bei Datenanalysen im Produktionsumfeld](#) beschrieben werden. Damit diese Herausforderungen bewältigt werden können, ist eine Vorverarbeitung der Daten notwendig. Durch die großen Mengen zu verarbeiteten Datenmengen hat sich aus der klassischen Datenverarbeitung inzwischen die Big Data Verarbeitung herausgebildet [46] [48] [49]. 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 [50]. Das Konzept Big Data tritt als Resultat von immer größer werdenden Datenmengen auf, die gegebenenfalls versteckte Informationen beinhalten und somit einen Mehrwert beitragen können. Damit diese Informationen erfolgreich extrahiert werden können, wurden Tools und Technologien entwickelt, die dem Problem mit den großen Datenmengen entgegenwirken [51]. Als die Geburtsstunde von Big Data gilt laut Lee der „Advent des E-Commerce“ von 1994-2004 [50]. Dort begannen Entwickler die Online Aktivitäten der User mittels Web-Mining Methoden zu analysieren. Über die Jahre hinweg bildeten sich durch die zunehmende Rechenleistung der Computer, kohärent zum Mooreschen Gesetz [52], immer neue Methoden und Technologien. Die Anwendungsmöglichkeiten des Konzepts Big Data haben deshalb längst auch andere Bereiche, wie beispielsweise Fertigungsumgebungen, erschlossen [46].

In der Literatur findet sich in Bezug auf Big Data ein breit gefächertes Vorgehen zur Definition des Konzeptes Big Data. De Mauro et. al. [53] zu Folge ist eine überzeugende Definition eines Konzepts die Voraussetzung für dessen wissenschaftliche Entwicklung. Ohne dass eine solche Definition bestand, hat sich das relativ junge Konzept Big Data rasant entwickelt. Dies führte zu vielen Definitionsversuchen, die später von den Autor:innen ignoriert, ergänzt oder verändert wurden [53]. In den meisten Fällen jedoch, stimmt die Vorgehensweise zur Begriffserklärung überein. Die Autor: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 [54] [50] [55] [56].

Übereinstimmend sind bei den verschiedenen Autor: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 dafür, dass derzeit ca. alle 2 Tage so viele Informationen entstehen, wie von Beginn der Informationserfassung bis 2003 [54]. 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 [50]. Im Vergleich zu alten Datenträgern sind mittels neuer Datenträger und Cloudtechnologien Speicherungen einer solchen Datenmenge heutzutage kein Problem mehr. Mit älteren Datenträgern wie der Compact Disc (CD) oder der Digital Versatile Disc (DVD) würden für die Speicherung von einem Terabyte eine große Zahl von Datenträgern benötigt werden (beispielsweise 1500 CDs oder 220 DVDs), heute passt eine solche Datenmenge, dank SSD-Speichertechnologie, auf eine nur wenige Zentimeter große Speicherkarte [54].

Velocity

Velocity bezieht sich bei Big Data auf die Geschwindigkeit der einfließenden Daten. Konkret versteht man unter dieser Geschwindigkeit, wie schnell die Daten aus den verschiedenen Quellen einströmen [57]. Ein Beispiel für Velocity liefert Facebook mit ihrer Social Media Plattform. Benutzer laden täglich ca. 900 Millionen Bilder in die Plattform hoch. Facebook muss diese Bilder aufnehmen, verarbeiten, ablegen und für die Benutzer verfügbar machen [58]. Ein klassisch batch-orientierter Ansatz zur Datenverarbeitung reicht bei dieser Geschwindigkeit der Daten nicht aus. Zusätzlich nimmt die Geschwindigkeit der Daten weiter zu [54] [57]. Für die Datenverarbeitung sind deshalb Lösungen notwendig, die die Herausforderung der wachsenden Datengeschwindigkeit meistert [57].

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 [54]. Beispiele für halbstrukturierte und unstrukturierte Daten sind

Text-, Foto-, Audio, Video- und Sensordaten [54] [57]. Durch die Breite an verschiedenen Datentypen sind neue Analysesysteme notwendig. Diese Analysesysteme müssen in der Lage sein, die verschiedenen Datentypen zu behandeln und zu verarbeiten [54].

Eine Definition, die auf Basis der Eigenschaften Volume, Velocity und Variety gebildet ist, liefern De Mauro et. al. [53]: “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.” [53]. Gleichzeitig berücksichtigt diese Definition die bereichsunabhängige Anwendbarkeit des Konzepts, weshalb diese Begriffserklärung auch in Fertigungsumgebungen angewandt werden kann.

Die in dieser genannten Definition enthaltenen Begriffe Big Data Technologien und Big Data Methoden sollen dazu beitragen, dass aus den massiven Datensätzen relevante Informationen gewonnen werden können. Synchron zur 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 [51]. Eigenschaften wie Velocity und Variety beziehen sich in diesem Rahmen auch überwiegend auf die Datenverarbeitung. In den folgenden Abschnitten werden deshalb die Technologien und Methoden beschrieben, die die Datenverarbeitung adressieren. Zuerst allerdings werden die Unterschiede zwischen herkömmlicher Datenverarbeitung und 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 [49] [56], wird in diesem Abschnitt erklärt. Als Ausgangslage dient eine Tabelle, die aus dem Buch Big Data Analytics und Preprocessing entnommen wurde [51]. Tabelle 20 ist eine eigene Darstellung basierend auf der Tabelle im Buch. Die Tabelle zeigt die Unterschiede zwischen 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. [51] setzen die traditionellen Daten bei Gigabytes (GBs) an, während Big Data ein Volumen von Terabytes (TBs), Petabytes (PB) oder Zettabytes (ZBs) annehmen kann. Dementsprechend ist die Datengenerierungsrate bei Big Data um mehrere Größenordnungen höher als bei traditionellen Daten.

Wie bereits in der Sektion Variety im vorherigen Abschnitt 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 hingegen vom Konzept her vollständig verteilt. Abgeleitet aus den Datenstruktur- und Datenquellmerkmalen bildet sich das nächste Differenzierungsmerkmal: die Datenintegration. Da alle traditionellen 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 (verteilte) 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, die auf die Verwendung handelsüblicher Hardware ausgelegt ist [59]. Diese und weitere Big Data Technologien, Methoden und Tools werden im nächsten Abschnitt in den entsprechenden Big Data Schichten erläutert.

3.1.3 Big Data Schichten

Es gibt eine große Anzahl an verschiedenen Technologien, Methoden und Tools, die 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 Abschnitt 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 deren Eigenschaften. In der Literatur lassen sich oft noch weitere Schichten finden [61] [49]. Die Schichten überlappen dabei in ihren Aufgabengebieten 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 Bedeutung sind. Schichten wie Monitoring oder Sicherheit von Erraissi und Belangour [61] werden in diesem Kapitel deshalb nicht berücksichtigt.

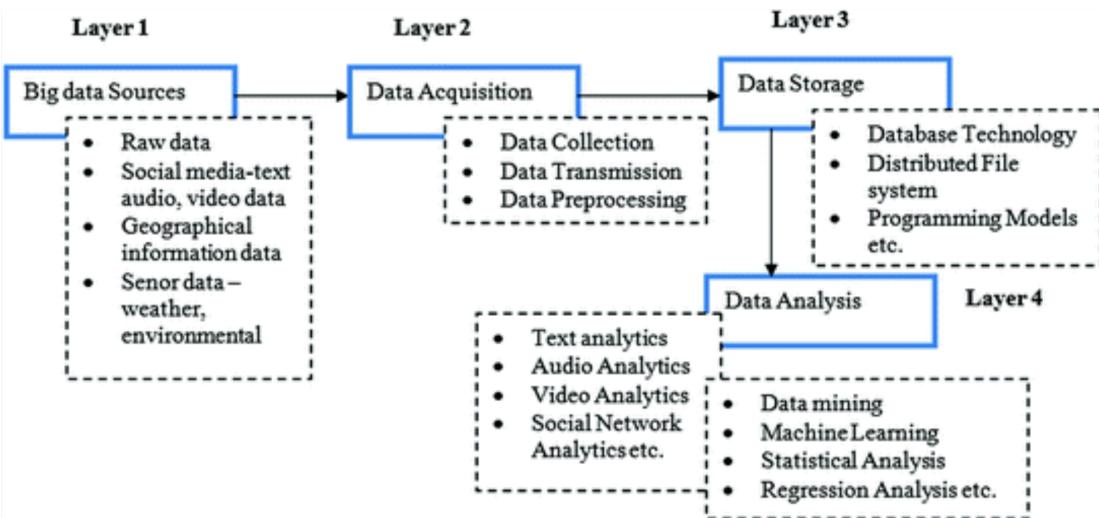


Abbildung 1 - Big Data Schichten

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 Generierung der Daten. Üblicherweise kommen die Daten aus 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 [14].

Die im [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 diese Schicht die zukünftige Speicherung und Analyse der Daten [60].

Resultierend aus der Anforderung, die 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 dem die Rohdaten in eine Speicherinfrastruktur übertragen werden [60]. Für die Speicherung der Daten gibt es spezielle Speichertechnologien, die 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, die für das Verarbeiten von Big Data ausgelegt sind [62].

Data Preprocessing

Die 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 [51]. 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 den verwendeten Analyseverfahren in der Data Analysis Schicht und dem Aufbau der Rohdaten sind verschiedene Verarbeitungsschritte von Bedeutung [51] [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] [51]. 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 verrauchte Daten zu identifizieren und zu bereinigen [51].

Die Schwierigkeit im Big Data Umfeld besteht darin, 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 Speicherlösungskonzepte werden deshalb in den

folgenden Abschnitten nur 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 möglicherweise bis dahin nicht explizit vorhandene 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].

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

3.1.4 Big Data Technologien

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

3.1.4.1 MongoDB

Ursprünglich dominierten bei der Verwendung von Speicherlösungen die Relationalen Datenbankmanagementsysteme wie Structured Query Language (SQL) Datenbanken [65]. Diese Art der Datenbanken stoßen durch die Skalierung von großen Datensätzen an ihre Grenzen [49] [65]. Um den Herausforderungen von Big Data entgegen zu wirken, haben sich deshalb NoSQL-Datenbanken entwickelt, die auf die ACID-Transaktions-Eigenschaften (Atomicity, Consistency, Isolation und Durability) der relationalen Datenbanken verzichten [65].

MongoDB ist eine dokumentenorientierte NoSQL Datenbank, die 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 schemilos, sodass die Dokumente eine beliebige Anzahl von Feldern einnehmen können. 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 man MapReduce-ähnliche Operationen auf den Dokumente ausführen kann (das Konzept MapReduce wird in den folgenden Abschnitten 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 es sich beispielsweise um Sensordaten von vielen unterschiedlichen Sensoren (wie im Forschungsprojekt PREFERML), die ihre Messungen in eine replizierte Datenbank 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, sowie 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 verteilten Übertragungsprotokolls 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 dafür konzipiert große Mengen an Daten mit geringer Latenz zu sammeln und zu versenden [69].

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, die 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].

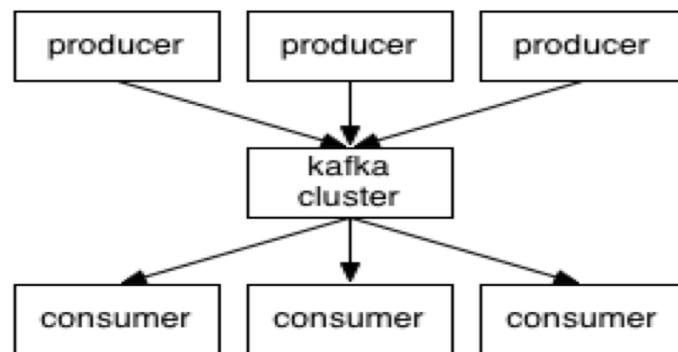


Abbildung 2 - Kafka-Architektur

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, dass 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 unabhängig voneinander verteilt erstellt wurden. Mit Hilfe des Kafka Clusters können diese Events 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 werden. 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] [56].

Das Hadoop Framework adressiert mit seinen umfassenden Modulen fast alle Herausforderungen, die 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 des Java File Systems 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. HDFS hat eine Master/Slave-Konstellation, bei der ein oder mehrere Slave-Maschinen von einer Master-Maschine gesteuert werden [59] [56].

Map Reduce

Eine Hadoop MapReduce-Aufgabe besteht im Allgemeinen aus zwei benutzerdefinierten Funktionen, die *map* und *reduce* genannt werden. 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] [56] [74].

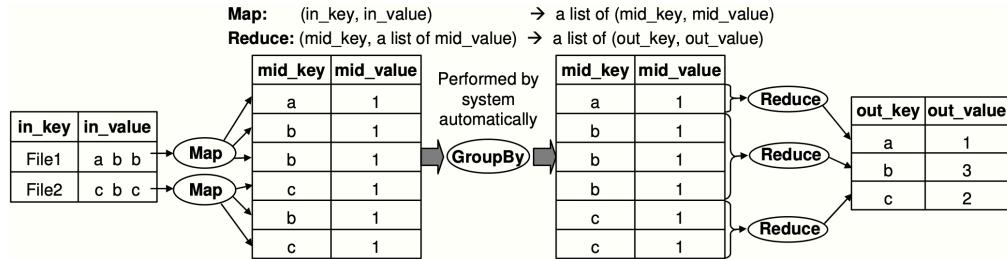


Abbildung 3 - MapReduce Funktion

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 bei 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, 79, 79].

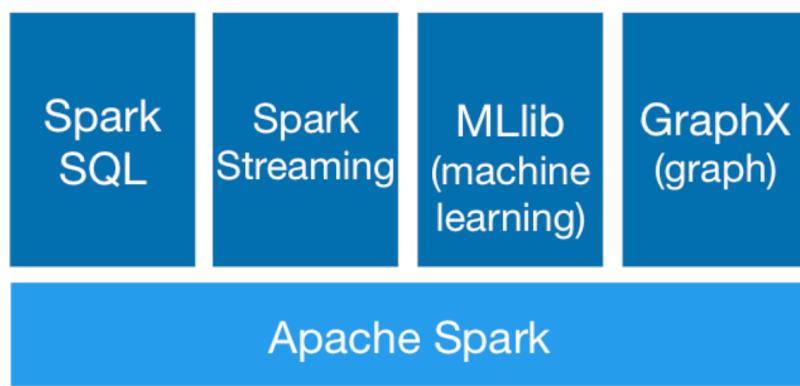


Abbildung 4 - Spark Bibliotheken

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) [80]. 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 sie nicht direkt ausgeschlossen ist, wird eine Hybridlösung in der Abbildung nicht extra dargestellt

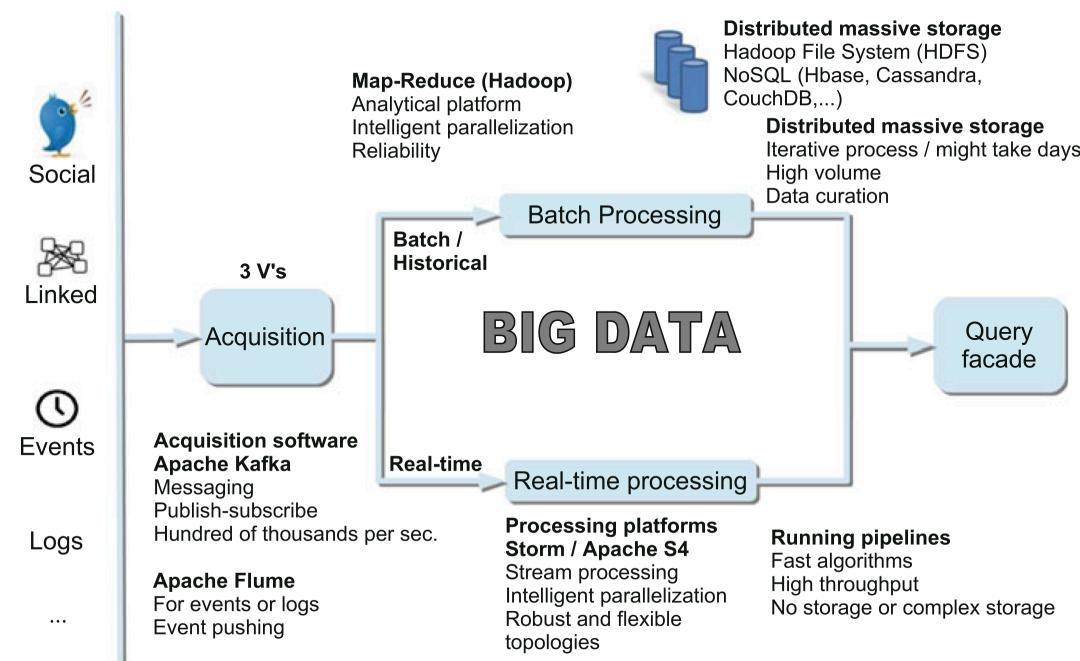


Abbildung 5 - Big Data Processing Architektur

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 [80].

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 [80]. Bei Real-Time werden sich die Eigenschaften zu Nutze gemacht, die die Tools wie Apache Spark so performant machen. Anstatt die Daten im FileSystem abzulegen (wie zum Beispiel HDFS), werden sie „in memory“ transformiert [76].

Im Rahmen dieser Forschungsarbeit sollen die Daten in Real-Time prozessiert werden. Mit den Analyseergebnissen (Vorhersagen der ML-Modelle) sollen fehlerhafte Werkstücke so schnell wie möglich aussortiert werden. Die niedrige Latenzzeit der Analyseergebnisse schlägt sich jedoch auf die Qualität der Ergebnisse nieder. Um diese Schwierigkeit zu überwinden, kann ein hybrider Verarbeitungsansatz als neues Paradigma eingesetzt werden [80].

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 Dateneingabe, -verarbeitung, -analyse und -reporting komplexere Prozesse, die eine gewisse Orchestrierung erfordern [80].

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 Eigenheiten 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 [81].

3.2.1 Lambda-Architektur

Die Lambda-Architektur hat ihren Namen von Nathan Marz [82] 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 [82] [81]. 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 [83] [81].

Abbildung 6 [84] zeigt die Lambda-Architektur. Die Architektur setzt sich zusammen aus drei verschiedenen Schichten. Die Schichten werden Speed-Schicht, Batch-Schicht und Serving-Schicht genannt [84] [81]. Den Schichten ist jeweils 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 [81]. Die Lambda-Architektur repräsentiert eine hybride Big Data Verarbeitungslösung wie in [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.

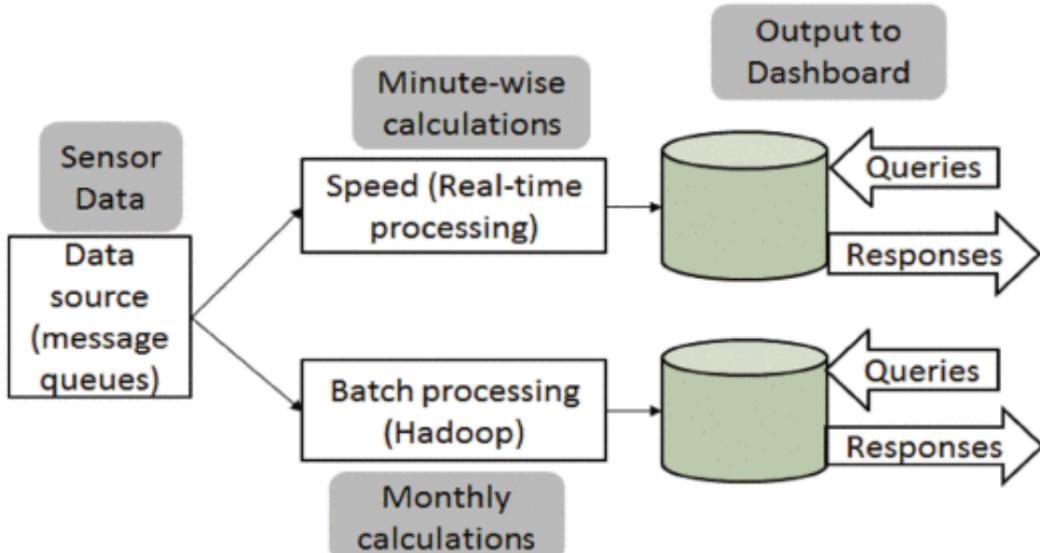


Abbildung 6 - Lambda-Architektur

Im Hinblick auf die verwendeten Technologien in den Schichten Batch und Speed, werden die Technologien verwendet, die in [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. [81] 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.

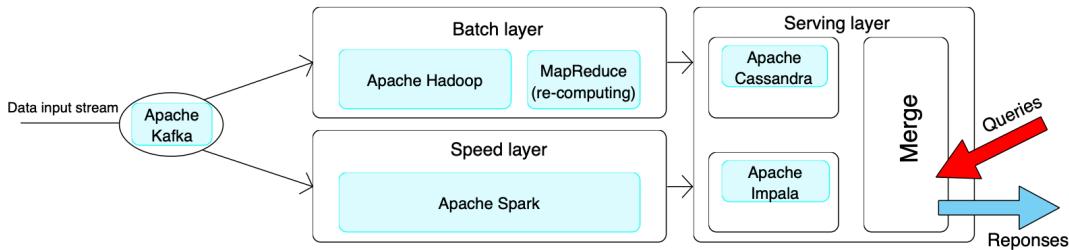


Abbildung 7 - Lambda-Architektur und dazugehörige Tools

Die Batch-Schicht hat im Wesentlichen zwei Funktionen: Sie speichert zum einen den 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 [83] [81]. 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 [81].

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 hier auch „Data Streams“ genannt). Aus der Sicht der Entwickler:innen bedeutet dies mehr Aufwand und Komplexität [81]. Kurz nachdem die Lambda-Architektur 2011 von Marz [82] 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 [85]. Die von Kreps vorgeschlagene Architektur zielt ebenfalls auf die Verarbeitung von Big Data ab, verfügt aber nicht mehr über eine Batch-Schicht. Bei Kreps wird lediglich ein Stream verarbeitet. Kreps, der zu dieser Zeit bei LinkedIn tätig war und maßgeblich an der Messaging-Technologie 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.“ [86].

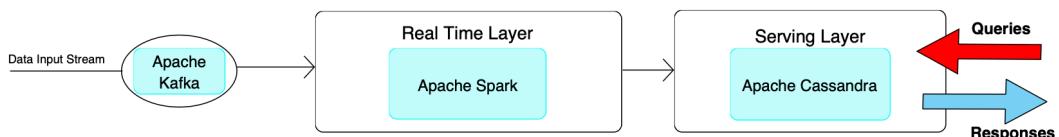


Abbildung 8 - Kappa-Architektur

Abbildung 8 von Feick et. al. [81] zeigt die Kappa-Architektur. Im Gegensatz zur Lambda-Architektur sind hier nur noch die Real-Time-Schicht und die Serving-Schicht vorhanden. Da sich ansonsten die Architekturbilder nicht weiter unterscheiden, sind dieselben Technologien der Lambda-Architektur verwendbar. Wie bei der Lambda-Architektur ist eine Datenquelle vorangeschaltet, die über ein Tool (in diesem Fall Kafka) die Daten zur Real-Time-Schicht transferiert. Der Aufgabenbereich 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 [81]. 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 hier zu einem Problem führen [87].

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 Werkstück 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 Abschnitt [3.1.3 – Big Data Schichten](#)). In diesem Abschnitt werden die Grundlagen für das Maschinelle Lernen gelegt. Zur Implementierung des Prototyps sind die Feinheiten des Trainierens und Verwendens von ML-Modellen ebenso von nachrangiger Bedeutung wie die unterschiedlichen Arten von ML-Modellen und wann welches Modell am besten zum Einsatz kommen sollte. Im Vordergrund steht das Einbinden der ML-Modelle in den Datenverarbeitungs-Workflow des Prototyps. Im Mittelpunkt dieses Abschnitts 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 war allerdings eher theoretischer Natur, da die Ressourcen (in Form von Daten oder Rechenleistung) begrenzt waren [88].

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 [89]. Eine der ersten Definitionen zum Thema Maschinelles Lernen ist auf Arthur Samuel [90] im Jahre 1959 zurückzuführen: „Field of study that gives computers the ability to learn without being explicitly programmed.“. Aufbauend darauf setzt Tom Mitchell [91] 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 [92]. 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 Maschinellem Lernen wird zwischen Supervised Learning und Unsupervised Learning unterschieden [93] [94]. Bei Supervised Learning wird ein Algorithmus darauf trainiert aus Daten zu lernen, bei denen 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 bisher ungesehene Daten vorherzusagen [95]. Im Fachjargon werden die Ausgabeveriablen auch Label genannt. Das Vorkommen solcher Labels 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) [96]. Jeder Algorithmus hat seine Vor- und Nachteile und kommt bei unterschiedlichen Anwendungen zum Einsatz. Typischerweise werden bei der Entwicklung mehrere dieser Algorithmen implementiert und dann verglichen. Zum Beispiel eignet sich bei Textklassifikation im Kontext von Natural Language Processing (NLP) der NB-Algorithmus besser [97].

Zum Trainieren der Algorithmen werden die Datensätze in einen Trainings- und einen Testdatensatz aufgeteilt. Zunächst 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. Nach dem 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 zufriedenstellend, 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 Ergebnis des Preprocessing ist der endgültige Trainingsatz [98]. Das Preprocessing für ML-Modelle findet in der in Abschnitt [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 Abschnitt [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 [87], 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 [79].

Der im Rahmen dieser Forschungsarbeit entwickelte Prototyp implementiert die Datenverarbeitung im ersten Schritt zunächst 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 in der Datenbank abzulegen. Ebenso ist es möglich diese Formate wieder vom lokalen Speicher oder aus 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 linearen Bytestrom umgewandelt wird, und Unpickling ist der umgekehrte Vorgang, bei dem ein Bytestrom wieder in eine Objekthierarchie umgewandelt wird. Pickling 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 ist [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 Abschnitt [2.2 – Anforderungsanalyse](#) auf. Ausgehend von den Forschungsfragen ist das Ziel der Implementierung die Erstellung einer Architektur und deren Umsetzung, bis eine einheitliche Datenverarbeitung angeboten werden kann, unabhängig davon, wie viele Schichten die Architektur besitzt. Für die Implementierung der Datenverarbeitungsarchitektur kommen beide – in Abschnitt [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 vornherein auf gegeben. Die Komplexität liegt bei der Kappa-Architektur auf der Umsetzung und Implementierung. Gibt es nur einen einzigen Datenstrom, so müssten 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 von der Kappa-Architektur im Kontext dieser Forschungsarbeit entgegenzuwirken, wird die Lambda-Architektur 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 [Abschnitt 3.2.1](#) beschrieben, ist die Verarbeitung der verschiedenen Datenströme oft verschieden. Dies führt zu einer größeren Komplexität des Systems und erfordert einen Verwaltungsaufwand. Um diesen Nachteil auszugleichen wird im Folgenden eine Lambda-Architektur vorgestellt, bei der 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 Auslösungereignis beim Speichern eines Events wird mit MongoDB Change Streams umgesetzt. Die Datenverarbeitung selbst wird aus Einfachheitsgründen ohne die in den Grundlagen vorgestellten Technologien in Tools umgesetzt. Die Tools und Technologien dienen allerdings als Grundlage für weitere Forschungsarbeiten.

Die in den folgenden Abschnitten enthaltenen Architekturdiagramme zeigen, wie der Prototyp implementiert wurde. Die Architekturbilder sind mit der C4-Modellierung umgesetzt [102]. Zur Übersicht wird dabei zuerst das System Context Diagramm gezeigt und beschrieben. Darauf folgend werden die Detaillierungen der Software sowohl mittels Container Diagramm, als auch mit dem Component Diagramm 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, die verschiedene Messungen an den zu produzierenden Teilen bzw. Produkten durchführen. Um mangelhafte Teile zu identifizieren werden die Messdaten der Teststation vom System bearbeitet [12]. Das Durchführen von Messungen an den Teststationen ist somit mit den Qualitätsprüfungen von Peter Hohenberger in „Qualitätsmanagement in der Produktion“ gleichzusetzen [15]. 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 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 [12]. Das bedeutet, dass beim späteren Arbeiten mit den Messdaten der Teststationen eine eindeutige Identifizierung der Teststationen erforderlich ist. 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 Abschnitt [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 unzulässigerweise 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 Werkstück gehören, ist der eindeutige Schlüssel für das Zusammenfügen die ID des Werkstücks. Diese wird zur Realisierung der Implementierung „Component_ID“ genannt.

Jedes Werkstück, das 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 hinzugemommen. Zum einen „Product_Type“ (ist bei Events jeder Teststation enthalten) und zum anderen „Category“ (ist nur bei Events der *Teststation1* enthalten). Zudem wird beim Messen jeweils ein Zeitstempel und eine Fehlerspalte hinzugefügt, die sich in den Merkmalen „Date“, „Time“ und „Error_Message“ widerspiegeln. 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 die Modelle nur auf einen einzelnen Fehler zu trainieren [11].

4.2 System Context Diagram

Abbildung 9 zeigt das System Context Diagramm des Prototyps. Das System Context Diagramm 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 Teststationen 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, die der zweite Benutzer (Quality Engineer) entgegennehmen kann. Dem Quality Engineer obliegt es dann, bei Abweichungen in den Produktionsprozess einzutreten.

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 Diagrammen 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 Re-albedingungen ebenfalls vorkommt:

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, die 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 zum Event processing sind in Abschnitt [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, die nach Ermessen des Data Scientist ausgeführt werden kann, kann auf die Konzeptdrifts in den Daten reagiert werden ([2.1.2 Probleme und Herausforderungen bei Datenanalysen im Produktionsumfeld](#)).

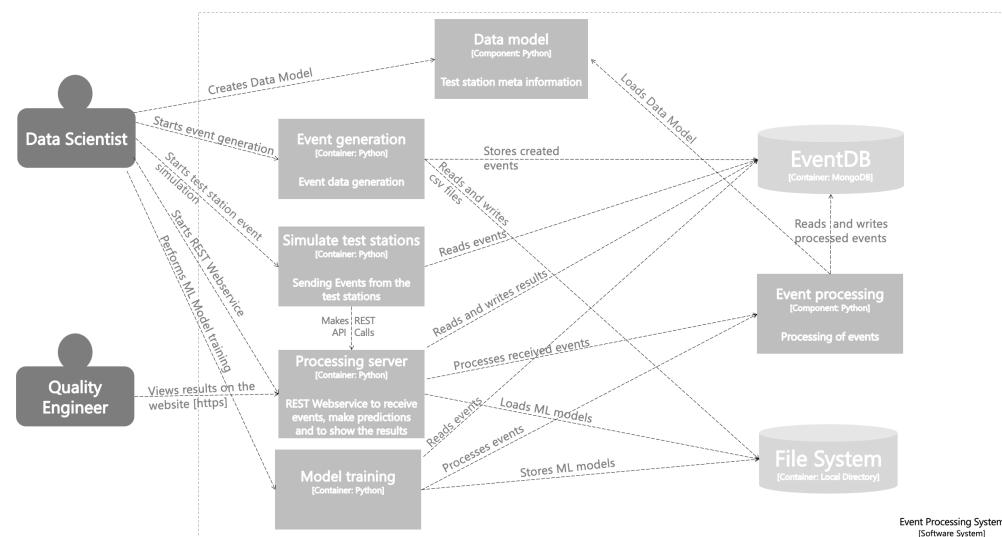


Abbildung 10 - REST Implementation

3. 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 gleiche 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 vom File System geladen. Die Ergebnisse der Vorhersagen werden in der EventDB gespeichert.

Die Datenverarbeitung durch den REST-Server ist im Kontext der Lambda-Architektur der Speed-Schicht zuzuordnen. Bei jedem Event, das 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.

4. 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** Containers werden die gespeicherten Daten der Event generation aus der EventDB geladen und mit einer Schleife über einen REST Aufruf an den REST Server gesendet.
5. 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 dem vorherigen Abschnitt 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 erfolgt über das erneute Abspeichern der Events in der EventDB.

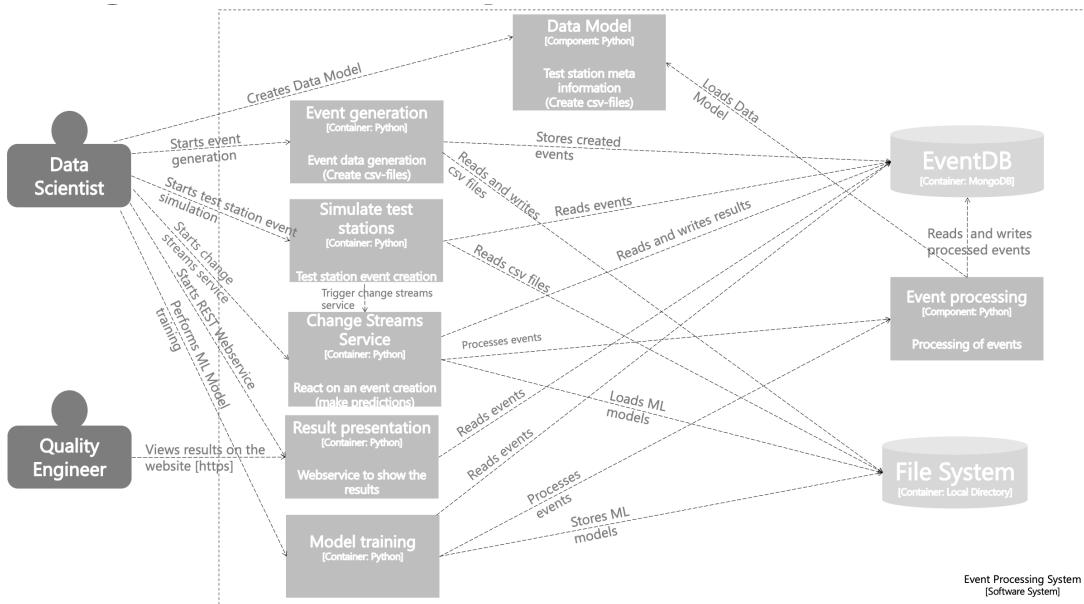


Abbildung 11 - Change Streams Implementation

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 in 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 Website ebenfalls die Vorhersageergebnisse der ML-Modelle anzeigen lassen.

Nach den Container Diagrammen werden im Folgenden die einzelnen Container und deren 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, der von den Wissenschaftlichen Mitarbeiter:innen 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 entsprechen den Beschreibungen in der Anforderungsanalyse (Use Cases: Erstellen Datensatz Teststation1, Erstellen Datensatz Teststation2, Erstellen Datensatz Teststation3). Folgender Textabschnitt zeigt einen stark vereinfachten Aufbau der yaml-Datei. Für das Konfigurieren der Datei muss der Spaltenname, der Datentyp, mögliche Werte und die Anzahl der Werte angegeben werden:

```

data:
  FeatureName: <Value>
    type: <Value>
    vals: <Value>
  count: <Value>

```

Nach Erstellung der Konfigurationsdateien startet der Data Scientist die Datengenerierung mittels eines Skripts (**Start Data Generation**). Beim Aufrufen des Skripts 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 angegebenen Anzahl von 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.

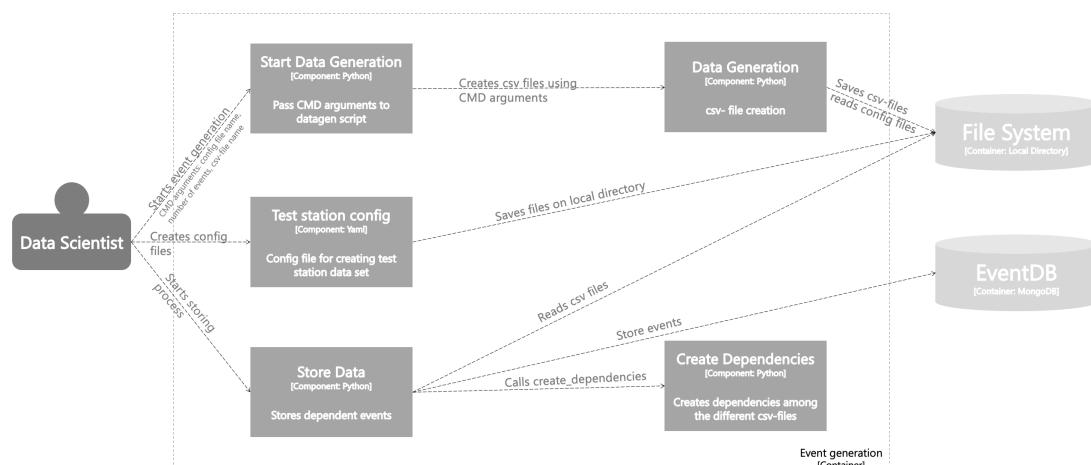


Abbildung 12 - Event Generation

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, das 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, die 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 zur *Teststation2*. Für die Dataframes bedeutet dies, dass, wenn ein Event mit der Component_ID X eine positive Fehlermeldung beim 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*) beim Auftreten von Fehlermeldungen aus den Dataframes nachfolgender Teststationen entfernt werden. Realisiert wird dies, indem eine Liste von Component_IDs angelegt wird, bei der die Fehlermeldungsspalte 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 herangezogen 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 Abschnitt [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 Diagramm für das Model Training. Beim erstmaligen Verwenden des Software-Systems, oder nach Ermessen des Data Scientists zum Reagieren auf Konzeptdrifts, 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 auf einmal zu trainieren. **Start Batch Training** lädt, je nach Terminal Argumenten, alle oder nur spezielle Events aus der EventDB. Anschließend wird das **Event Processing** aufgerufen, bei dem die Daten verarbeitet werden. (Genauere Informationen zum Event Processing sind im nächsten Abschnitt beschrieben.) Zur Datenverarbeitung gehört die Datenbereinigung, das Ableiten von weiteren Features, das Zusammenfügen von Events über Teststationen hinweg und das Speichern der verarbeiteten 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 Labels zu den entsprechenden Events benötigt. Die Labels werden über das **Load Label** Skript geladen und den Events angehängt. Die Labels 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 Labels werden den entsprechenden Events angehängt und dem **Start Batch Training** zurückgegeben. Das

Load Label Skript kann nur auf Events der *Teststationen1/2* angewendet werden. Die *Teststation3* ist in der Simulation des Prototyps die letzte Teststation, weswegen keine Labels der nachfolgenden Teststation geladen werden können.

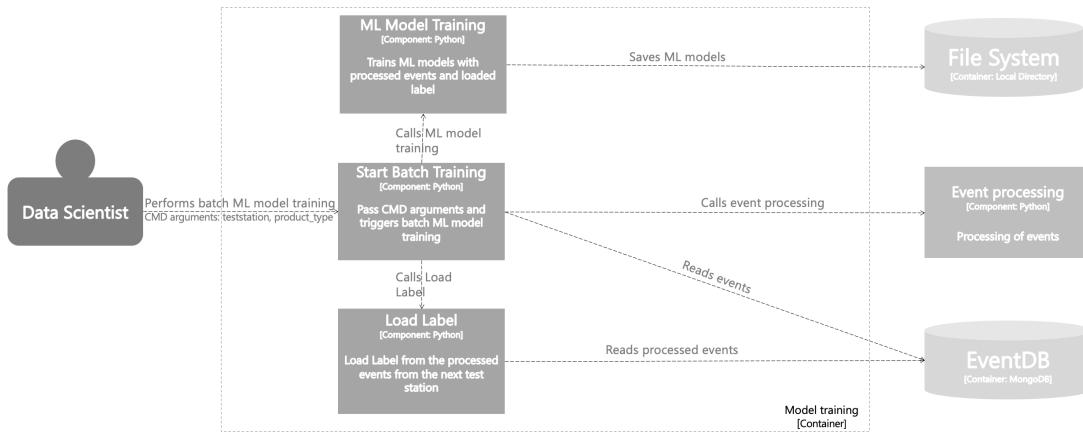


Abbildung 13 - Model training

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 Abschnitt [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 ihre 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 *Teststationen1/2* Vorhersagen gemacht werden, da nur für diese Teststationen Labels 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, das mit Events der *Teststation2* 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 Labels zum Trainieren verwendet. Aus den Daten sollen Muster erkannt werden und im ML-Modell gespeichert werden. Die Testdaten werden zum Validieren des ML-Modells verwendet. Dabei werden die Events als Input für Vorhersagen verwendet. Die Vorhersagen des Modells werden darauffolgend mit den korrespondierenden Labels 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 Modell 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, die 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, die 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 zeigt das **Event Processing** für beide Implementierungsvarianten. Das Event Processing wird in beiden Implementierungen vom Container **Model Training** aufgerufen, der 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 vom Container **Change Stream Service**.

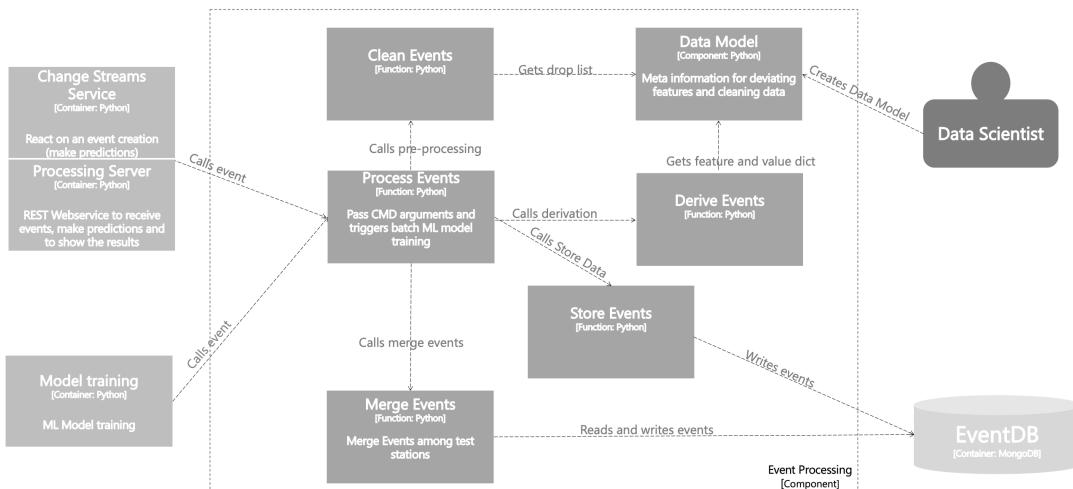


Abbildung 14 - Event Processing

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 komplett 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**, das Informationen über die Daten der Teststationen beinhaltet. Aufgrund des **Data Models** 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 getrennt berücksichtigt werden.

4.5.1 Batch-Schicht

In der Batch-Schicht gelangt ein kompletter 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 ML-Modelle und Vorhersagen nur mit konsistenten Daten erstellt werden. Im Prototyp werden nur einige wenige Datensäuberungsschritte durchgeführt. Abhängig von der Produktionsumgebung und den 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 Events** in der Abbildung 14 repräsentiert diesen Vorgang,
 - a) Damit nur vollständige Events prozessiert werden, werden alle Reihen, in denen 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 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.

Zusätzlich zu der Punktekonvertierung wird eine REGEX-Methode auf die Events im Datensatz ausgeführt. Mit Hilfe einer regular expression, die die zulässigen Zeichen, Zahlen und Buchstaben beschreibt, soll sichergestellt werden, dass nur zulässige Zeichenfolgen in den Spalten der Events vorkommen können.

- d) In einer realen Produktionsumgebung kann es vorkommen, dass die Teststationen Events übermitteln, die 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. Mit Hilfe des **Data Models**, das vom Data Scientist gepflegt wird, kann eine Liste geladen werden, die teststationsübergreifend alle Spaltennamen enthält, die gelöscht werden müssen. Die entsprechende Methode gleicht die übergebenen Events mit der Liste ab und löscht die korrespondierenden Spalten.
- e) Als letzten Bereinigungsschritt werden die Fehlermeldungen gesäubert. Fehlermeldungen sind später die Labels 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.
Derive Events in Abbildung 14 zeigt diese Funktionalität.

- 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 Wochentage aus der zusammengefügten Datums-Zeit-Spalte abgeleitet werden. Anstatt eine einzige Spalte für die Wochentage zu erstellen, wird für jeden Wochentag eine eigene Spalte angelegt. Der Default-Wert der Spalten ist Null. Abhängig vom Wochentag in der Datum-Zeit-Spalte wird der Wert in der korrespondierenden Wochentagsspalte erhöht. Umgesetzt wird dies mit der pandas-Funktionalität `get_dummies()`. Dazu wird zuerst aus der Datum-Zeit-Spalte der Wochentag 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. Darauffolgend wird pro Event aus der Day-Spalte überprüft, welche der neu erstellten Wochentagsspalten 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 Wochentag). Das OneHot-Encoding ist notwendig, da nicht alle ML-Modell Arten (NN, DT, etc.) kategorische Werte verarbeiten können.
- e) Ebenso wie die Wochentage soll die Kalenderwoche aus der Datum-Zeit-Spalte extrahiert werden. Allerdings soll bei der Kalenderwoche kein One-Hot-Encoding angewendet werden, da 52 zusätzlich neue Spalten (Eine Spalte pro Kalenderwoche) den Datensatz unnötig vergrößern würden. Anstatt des OneHot-Encodings wird deshalb ein Ordinal-Encoding angewandt. Beim Ordinal-Encoding wird aus der Datum-Zeit-Spalte die Kalenderwoche errechnet. Zusätzlich wird eine neue Spalte („Week“) angelegt,

die als Werte die errechneten Kalenderwochen-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 zur Vorgehensweise beim OneHot-Encoding der Wo-chentage.
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 nur in einer Collection gespeichert würden, kann es unter Umständen vorkommen, dass die falschen Events zusammengefügt werden, was zu Inkonsistenzen und verfälschten Ergebnissen führt.
4. Nach der Speicherung der Daten werden die Events über die Teststationen hinweg zusammengefügt (**Merge Events** in Abbildung 14) 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 Labels 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, der das **Event Processing** aufruft, hängt von der Implementierungsvariante ab (REST oder Change Streams). Der Ablauf des Event Processing läuft gleich wie in der Batch-Schicht ab. Jedoch unterscheiden sich einige Funktionen in ihrer Arbeitsweise, was in diesem Abschnitt genauer beschrieben

Grundsätzlich ist der Unterschied bei den Events zwischen Speed- und Batch-Schicht die Anzahl der Events. Bei der Batch-Schicht wird ein kompletter 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 One-Hot-Encoding der Wochentage und der kategorischen Werte. Wie im vorherigen Abschnitt 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-Encodings 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 nur 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 Wochentagspalten („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 Wochentagspalte auf ihren Wert überprüft und die korrespondierende Spalte auf Eins erhöht. Kategoriale Spalten tauchen nicht wie das Datum bei jeder Teststation auf. In einer realen Produktionsumgebung können zudem unterschiedliche Teststationen verschiedene kategoriale 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 Dictionary 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 Dictionary über das Data Model extrahieren. Das manuelle Ausführen des OneHot-Encodings funktioniert analog wie das manuelle One-Hot-Encoding der Wochentage.

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 werden, muss die Simulation unterschiedlich 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, 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 die später auch die Ergebnisse auf der Webseite angezeigt werden können.

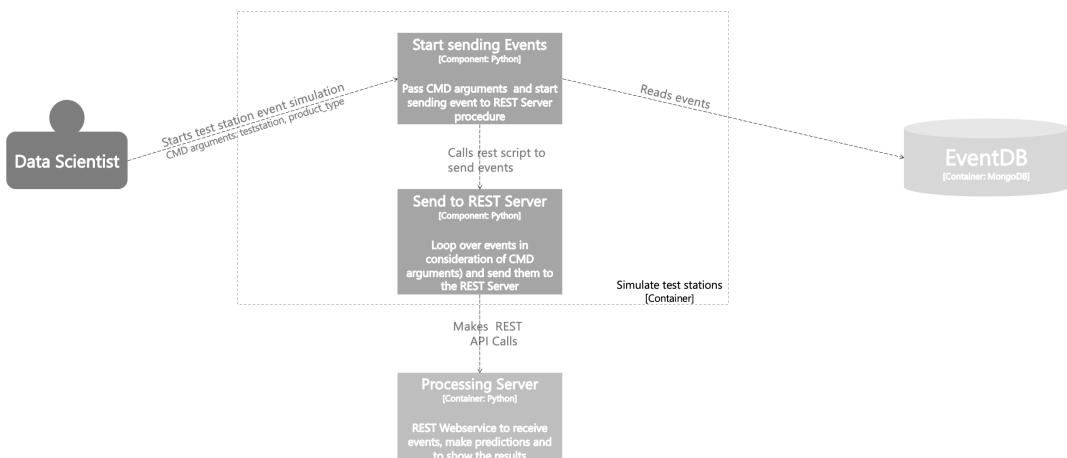


Abbildung 15 - Simulate Test Stations (REST)

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 in Abschnitt [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.

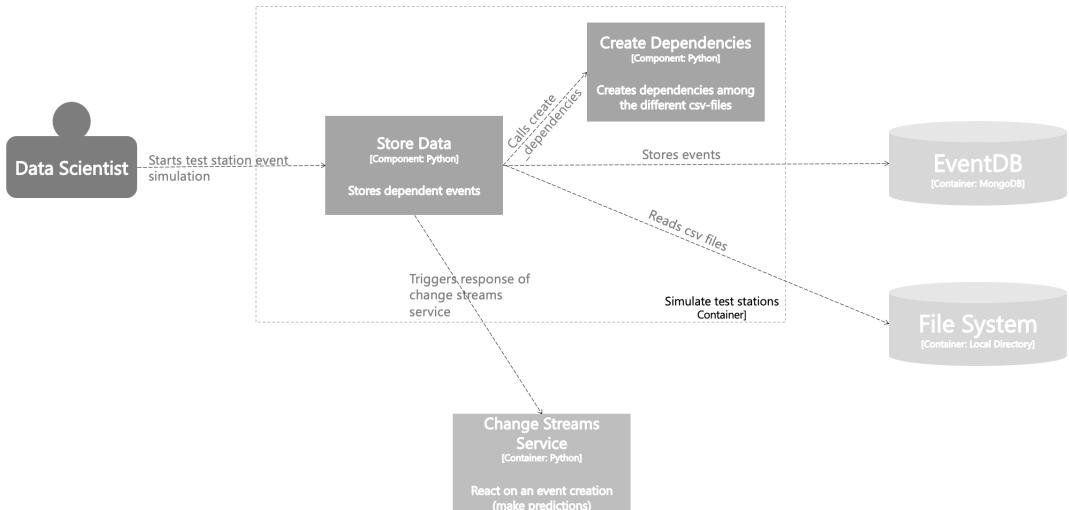


Abbildung 16 - Simulate Test Stations (Change Streams)

4.7 Data Processing Server

Die Datenverarbeitungsserver nehmen die simulierten 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 integriert werden. Die Funktionsweise der zwei verschiedenen Server wird in den nächsten zwei Abschnitten 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**), der wiederum die Services **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 werden 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.

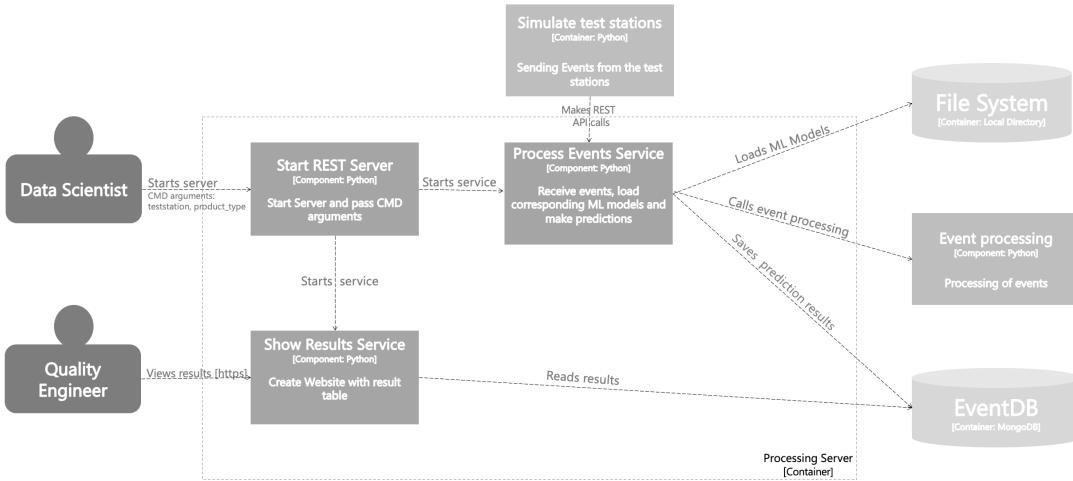


Abbildung 17 - Processing Server

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 Datenverarbeitungsservers (**Change Streams Service**) und Abbildung 19 zeigt die Implementierung des Web-Servers (**Result Presentation**).

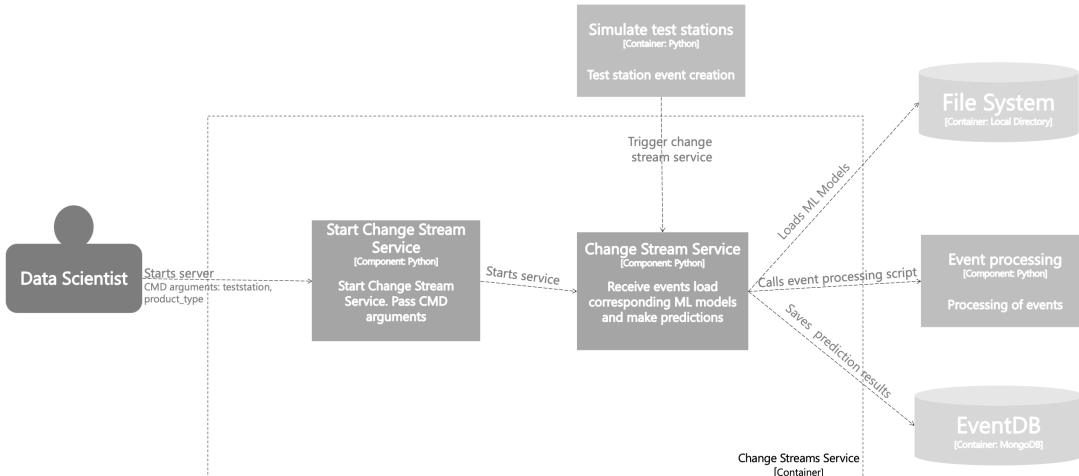


Abbildung 18 - Change Streams Service

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** aus, der die Ergebnisse aus der EventDB lädt und tabellarisch visualisiert.

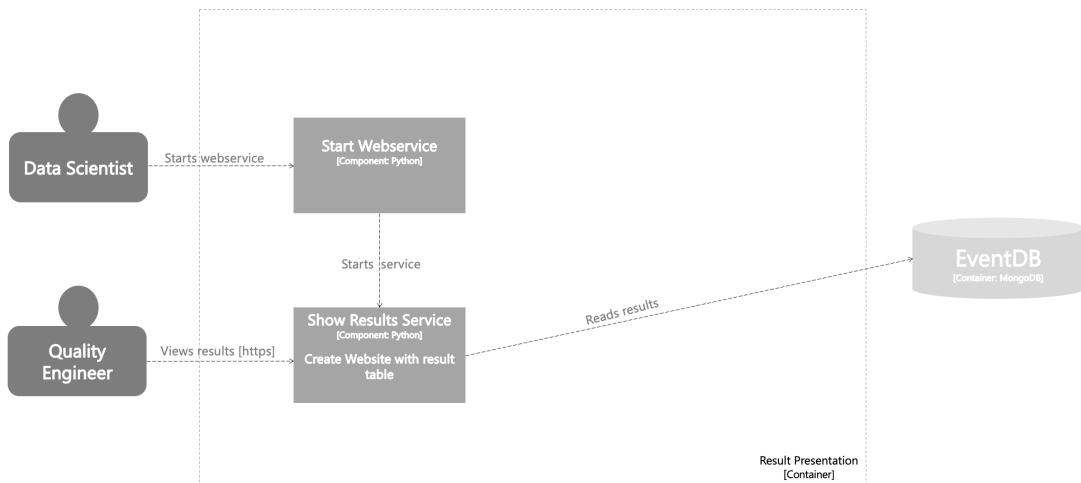


Abbildung 19 - Result Presentation

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, andererseits müssen die ML-Modelle bei den Vorhersagen nur noch geladen werden, was sich positiv auf die Performance auswirkt. Da sich die Genauigkeit bei batchorientiertem Training erst nach einem längeren Zeitraum spürbar verschlechtert, ist die negative Auswirkung des Batch-Trainings auf die Genauigkeit verkraftbar, wenn durch erneutes Training diesem Umstand entgegengewirkt wird (siehe Abschnitt [2.1.2.1 - Herausforderungen der Fertigungsdomäne](#)). Um die performantere Implementierungsvariante zu finden, werden die beiden Implementierungen im nächsten Abschnitt verglichen. Neben dem Performancevergleich werden die beiden Implementierungen im nun folgenden Abschnitt auch 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 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 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 wird jeweils 10 Minuten gewartet, um identische Ausgangssituationen sicherzustellen.

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

Die folgende Tabelle zeigt die Ergebnisse der durchgeföhrten Tests. Die linke Spalte gibt Auskunft über die verwendeten Teststationen und Produkttypen, die in der Messung verwendet wurde. Die mittleren Spalten zeigen die Performance der REST Implementierung und der Change Streams Implementierung für die jeweiligen Tests an. Die rechte Spalte zeigt den Unterschied der REST Implementierung Tests zu den Change Streams Implementierung Tests.

Teststationen/ Produkttypen	REST Implementierung	Change Streams Implementierung	Difference
All / All	5:45 min	6:14 min	8 %
T1 / All	2:21 min	3:03 min	23 %
T1 / PT1	1:13 min	1:35 min	24 %
T1 / PT2	1:11 min	1:33 min	24 %
T2 / PT1	2:56 min	3:15 min	9 %
T2 / PT2	2:51 min	3:05 min	8 %

Tabelle 21 - Performancevergleich REST - Change Streams

Legende:

Teststationen:

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

Produkttypen:

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

4.8.1.4 Evaluierung der Ergebnisse

Die Ergebnisse der Tests zeigen, dass die REST Implementierungsvariante im Durchschnitt 16% Prozent 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 Nöten, die jedoch nicht im Fokus dieser Forschungsarbeit liegen. Als mögliche Ursache könnte allerdings die replizierte Datenbank der Change Streams Implementierung gelten. Durch die notwenige Synchronisierung der verteilten Datenbanken könnte ein Performancenachteil entstehen.

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 *Teststation1* mit dem Produkttyp 1 bei 1:13 Minuten. Mutmaßlich müsste das Ergebnis bei der *Teststation2* mit dem Produkttype 1 bei 2:26 Minuten liegen (Wird auf die *Teststation2* getestet, so werden auch die entsprechenden Events der *Teststation1* übermittelt. Dies hat den Hintergrund, dass die Daten der *Teststation2* mit den Events der *Teststation1* für Fehlervorhersagen zusammengefügt werden müssen. Die benötigte Verarbeitungszeit ist daraus resultierend ca. doppelt so groß). Tatsächlich liegt die benötigte Verarbeitungszeit bei 2:56 Minuten. Die zusätzlich benötigten 30 Sekunden lassen sich auf das Zusammenfügen der Events zurückführen. Events der *Teststation1* werden vom Event Processing nach der Datenverarbeitung direkt zurückgegeben und können für die Fehlervorhersage benutzt werden. Events der *Teststation2* müssen vor der Vorhersage mit den Events der *Teststation1* zusammengefügt werden. Die daraus resultierende zusätzliche Datenbankabfrage lässt auf den Performanceunterschied hindeuten. Der Performanceunterschied zwischen *Teststation1* und *Teststation2* 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 sind auf Unschärfen in den Messungen zurückzuführen. 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 die beiden Produkttypen verteilen.

4.8.2 Qualitativer Vergleich

Beide Implementierungsvarianten haben ihre Vor- und Nachteile. Unabhängig von den Performanceergebnissen 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 hat jede Teststation 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 kann 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 in etwa gleich ist.

5 Limitationen und Resümee

In diesem Kapitel werden die in der Einleitung definierten Forschungsfragen auf Basis der beiden Implementierungsvarianten und der durchgeführten Tests diskutiert. Zusätzlich wird in diesem Zusammenhang ein Ausblick für zukünftige Forschungsarbeiten auf Basis dieser Forschungsarbeit gegeben.

5.1 Resümee und Bewertung der Forschungsfragen

Abgeleitet vom 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 [10] [11] [12] [13] [14]. Die wichtigsten Merkmale für die Implementierung eines Datenverarbeitungsprototyps 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, die 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 vorgestellt. Dabei wurden verschiedene Technologien und Architekturen präsentiert. 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 zum Aufbau der Implementierungsvarianten wurde im Anschluss an das Grundlagenkapitel beschrieben. Zusätzlich wurden die Implementierungsvarianten hinsichtlich ihrer Performance untereinander verglichen.

Zur Evaluierung werden nun die Ergebnisse dieser Forschungsarbeit mit Blick auf die am Anfang gestellten Forschungsfragen erörtert:

RQ1: Inwieweit lässt sich eine einheitliche Datenverarbeitung in Form von gleichem Source Code für die Erstellung der ML-Modelle und 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 Modelle eine einheitliche Datenverarbeitung anzubieten. Die Datenverarbeitung bei den zwei Ausprägungen unterscheidet sich darin, dass bei der rechenintensiven Ausprägung ein kompletter Datensatz prozessiert werden muss und bei der rechenarmen Ausprägung nur ein einzelnes Event. Mit dem Event Processing innerhalb der 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 bis auf das OneHot-Encoding und das Speichern der Daten 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 kategorische Werte bei einzelnen Events nicht angewendet werden. Als Workaround wird mit der Einbindung des Data Models gearbeitet. Es ist deshalb möglich, nur eine Datenverarbeitungs-Komponente sowohl für rechenintensive als auch für 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 One-Hot-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, rechenintensive (ML-Model Training) Aufgaben von den rechenarmen (Fehlervorhersage) zu trennen und gleichzeitig eine einheitliche Datenverarbeitung zu verwenden. Durch die Implementierung der beiden Varianten für den Prototyp ist es gelungen eine solche Architektur umzusetzen. Außerdem besteht durch die beiden Implementierungsvarianten die Möglichkeit den Prototyp in verschiedenen 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 einen REST und zum anderen MongoDB Change Streams. Abhängig vom Aufbau der realen Produktionsumgebungen eignen sich beide Implementierungsvarianten. Bei der Durchführung der Performance-Tests hat die REST Implementierung eine bessere Performance gegenüber der Change Streams Implementierung gezeigt. Unter Betrachtung der durchgeföhrten

Tests eignet sich die REST Implementierung hinsichtlich der Performance besser. Für eine fundierte Aussage über die tatsächliche Performance müssen allerdings noch Tests unter Produktionsbedingungen durchgeführt werden.

5.2 Stärken und Limitationen der Arbeit

Durch das Fehlen einer realen Produktionsumgebung ist diese Forschungsarbeit auf die vom Autor erstellte 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 einer echten Produktionsumgebung notwendig. Zusätzlich beschränkt sich die Durchführung der Performancetests auf die Rechenressourcen des Autors. Um sinnvolle Ergebnisse zu erzielen beschränkt sich die Anzahl der Events pro Teststation auf 1000. Abhängig von der Produktionsumgebung kann diese Anzahl für aussagekräftige Tests deutlich zu gering sein. Der Zugriff auf Big Data ausgelegte Rechenressourcen ist daher von essentieller Bedeutung.

Die Produktionssimulation eröffnet für den Autor hinsichtlich der verschiedenen Technologien allerdings auch Möglichkeiten. Die Produktionssimulation wird 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 sie 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 Modularitätseigenschaften. Die Integration neuer Technologien, wie zum Beispiel Kafka, lässt sich dadurch 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? Gelten die entstanden Resultate auch bei der Skalierung auf größeren Datenmengen?
- Vollständige Vereinheitlichung des Event Processings durch Verzicht auf One-Hot-Encodings und den Umstieg auf das Data Model
- Erweiterung des Prototyps durch die Einführung weiterer Teststationen und komplexerer Daten.
- Integration 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 würde 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)“ [8].
Dictionary	„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].
pandas	“pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. [107]“

REGEX	„[...] a regex is a string of text that allows you to create patterns that help match, locate, and manage text“ [108].
REST	“REST ist ein Architekturstil für den Entwurf vernetzter Anwendungen” [108]
SHAP	“SHAP is a method to explain individual predictions. SHAP is based on the game theoretically optimal Shapley Values.” [109]
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.“ [110]

Abkürzungsverzeichnis

ANN	Künstliches neuronales Netzwerk
AutoML	Automated Machine Learning
BMBF	Bundesministerium für Bildung und Forschung
BMWi	Bundesministerium für Wirtschaft und Energie
CD	Compact Disc
CPS	Cyber Physical System
CRISP-DM	Cross-industry standard process for data mining
CSV	Comma separated value
DESI	Digital Economy and Society Index
DVD	Digital Versatile Disc
DT	Decision Tree
GBs	Gigabytes
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	Petabytes
PREFERML	Proaktive Fehlervermeidung in der Produktion durch Maschinelles Lernen
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
SSD	Solid state disc
SVM	Support Vector Machine
TBs	Terabytes
ZBs	Zettabytes

Anhang

Anhang I: code_repository.zip

Literaturverzeichnis

- [1] D. E. Nye, *Technology Matters*, Cambridge: MIT Press, 2006.
- [2] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld und M. Hoffmann, „Industrie 4.0,“ *Wirtschaftsinformatik*, Bd. 56, Nr. 4, pp. 261-264, 2014.
- [3] A. Kusiak, „Smart manufacturing,“ *International Journal of Production Research*, Bd. 56, Nr. 1-2, pp. 508-517, 2018.
- [4] C. Rammer, „Auf Künstliche Intelligenz kommt es an,“ Bundesministerium für Wirtschaft und Energie (BMWi), Berlin, 2020.
- [5] N. Brandstetter, R.-M. Dobler und D. J. Ittstein, „Künstliche Intelligenz,“ in *Interdisziplinär*, Tübingen, UVK Verlag, 2020, pp. 70-84.
- [6] B. Hatiboglu, S. Schuler, A. Bildstein und M. Häammerle, „Einsatzfelder von Künstlicher Intelligenz im Produktionsumfeld,“ Fraunhofer IPA, Stuttgart, 2019.
- [7] B. Martens, „The Importance of Data Access Regimes for Artificial Intelligence and Machine Learning,“ JRC Digital Economy Working Paper, Sevilla, 2018.
- [8] E. Commission, „Index für die digitale Wirtschaft und Gesellschaft (DESI) 2020,“ European Commission, 2020.
- [9] C. Seiffer, U. Schreier, H. Ziekow und A. Gerling, „KI für die Produktionsqualität,“ *Forschungsbericht 2020/2021*, Bd. 800, pp. 18-19, 11 Dezember 2020.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] 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.
- [14] 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.

- [15] P. Hehenberger, „Qualitätsmanagement in der Produktion,“ in *Computerunterstützte Produktion: Eine kompakte Einführung*, Berlin, Heidelberg, Springer Berlin Heidelberg, 2020, pp. 197-228.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] 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.
- [21] 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.
- [22] 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.
- [23] C. Nitesh, N. Japkowicz und A. Kołcz, „Editorial: Special Issue on Learning from Imbalanced Data Sets,“ *SIGKDD Explorations*, Bd. 6, pp. 1-6, 2004.
- [24] 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.
- [25] J. Gama, I. Žliobaitė, 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.
- [26] 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.
- [27] 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.
- [28] 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.

- [29] 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.
- [30] 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.
- [31] 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.
- [32] 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.
- [33] 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.
- [34] 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.
- [35] 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.
- [36] 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.
- [37] R. Wirth und J. Hipp, „CRISP-DM: Towards a standard process model for data mining,“ Bd. 1, 2000.
- [38] 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.
- [39] 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.
- [40] 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.
- [41] 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.
- [42] 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.
- [43] 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.
- [44] 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.
 - [45] 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.
 - [46] 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.
 - [47] 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.
 - [48] 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.
 - [49] R. Ranjan, „Streaming Big Data Processing in Datacenter Clouds,“ *IEEE Cloud Computing*, Bd. 1, Nr. 1, pp. 78-83, 2014.
 - [50] I. Lee, „Big data: Dimensions, evolution, impacts, and challenges,“ *Business Horizons*, Bd. 60, Nr. 3, pp. 293-303, 2017.
 - [51] 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.
 - [52] C. A. Mack, „Fifty Years of Moore's Law,“ *IEEE Transactions on Semiconductor Manufacturing*, Bd. 24, Nr. 2, pp. 202-207, 2011.
 - [53] 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.
 - [54] K. Taylor-Sakyi, „Big Data: Understanding Big Data,“ *arXiv preprint arXiv:1601.04602*, 2016.
 - [55] 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.
 - [56] Ishwarappa und J. Anuradha, „A Brief Introduction on Big Data 5Vs Characteristics and Hadoop Technology,“ *Procedia Computer Science*, Bd. 48, pp. 319-324, 2015.
 - [57] A. Katal, M. Wazid und R. H. Goudar, „Big data: Issues, challenges, tools and Good practices,“ *Sixth International Conference on Contemporary Computing (IC3)*, pp. 404-409, 2013.
 - [58] D. Gewitz, „Volume, velocity, and variety: Understanding the three V's of big data,“ ZDNet, 21 März 2018. [Online]. Available: <https://www.zdnet.com/article/volume-velocity-and-variety-understanding-the-three-vs-of-big-data/>. [Zugriff am 18 Juni 2021].
 - [59] hadoop, „HDFS Architecture Guide,“ The Apache Software Foundation, 10 Oktober 2020. [Online]. Available:

- https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. [Zugriff am 7 Mai 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 Mai 2021].
 - [67] R. Londner und A. Cabral, „An Introduction to Change Streams,“ mongoDB, 2018 Juni 18. [Online]. Available: <https://www.mongodb.com/blog/post/an-introduction-to-change-streams>. [Zugriff am 9 Mai 2021].
 - [68] mongoDB, „Change Streams,“ mongoDB, [Online]. Available: <https://docs.mongodb.com/manual/changeStreams/?jmp=blog#watch-collection-database-deployment>. [Zugriff am 9 Mai 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 Mai 2021].
 - [71] hadoop, „Apache Hadoop,“ hadoop, [Online]. Available: <https://hadoop.apache.org>. [Zugriff am 09 Mai 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 Mai 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 Mai 2021].
- [78] Apache Spark, „Lightning-fast unified analytics engine,“ Apache Spark, [Online]. Available: <https://spark.apache.org>. [Zugriff am 10 Mai 2021].
- [79] 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 Mai 2021].
- [80] 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.
- [81] 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.
- [82] N. Marz, „How to beat the CAP theorem,“ 13 Oktober 2011. [Online]. Available: <http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html>. [Zugriff am 11 Mai 2021].
- [83] N. Marz und J. Warren, *Big Data - Principles and best practices of scalable real-time systems*, Shelter Island: Manning Publication, 2015.
- [84] 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.
- [85] J. Lin, „The lambda and the kappa,“ *IEEE Internet Computing*, Bd. 21, Nr. 5, pp. 60-66, 2017.
- [86] J. Kreps, „Questioning the Lambda Architecture,“ O'Reilly, 2 Juli 2014. [Online]. Available: <https://www.oreilly.com/radar/questioning-the-lambda-architecture/>. [Zugriff am 11 Mai 2021].
- [87] S. a. R. A. Zhelev, „Big data processing in the cloud - Challenges and platforms,“ *AIP Conference Proceedings*, Bd. 1910, p. 060013, 2017.
- [88] J. G. Carbonell, R. S. Michalski und T. M. Mitchell, „An overview of machine learning,“ in *Machine learning*, Elsevier, 1983, pp. 3-23.
- [89] A. L. Fradkov, „Early History of Machine Learning,“ *IFAC-PapersOnLin*, Bd. 53, Nr. 2, pp. 1385-1390, 2020.
- [90] 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.
- [91] T. M. Mitchell, *Machine learning*, New York: McGraw-Hill Education Ltd, 1997.
- [92] 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.
- [93] 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.

- [94] M. W. Berry, A. Mohamed und B. W. Yap, *Supervised and unsupervised learning for data science*, Cham: Springer, 2019.
- [95] 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.
- [96] R. Caruana und A. Niculescu-Mizil, *An Empirical Comparison of Supervised Learning Algorithms*, Ney York: Association for Computing Machinery, 2006.
- [97] J. Hartmann, J. Huppertz, C. Schamp und M. Heitmann, „Comparing automated text classification methods,“ *International Journal of Research in Marketing*, Bd. 36, Nr. 1, pp. 20-38, 2019.
- [98] 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.
- [99] scikit -earn, „Model persistence,“ scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/model_persistence.html. [Zugriff am 12 Mai 2021].
- [100] Python, „pickle - Python object serialization,“ Python, [Online]. Available: <https://docs.python.org/3/library/pickle.html>. [Zugriff am 12 Mai 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 Mai 2021].
- [102] S. Brown, „The C4 model for visualising software architecture,“ [Online]. Available: <https://c4model.com>. [Zugriff am 2 Juni 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,“ *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 Juni 2021].
- [105] scikit learn, „sklearn.preprocessing.OneHotEncoder,“ 2020. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html>. [Zugriff am 29 März 2021].
- [106] scikit learn, „sklearn.preprocessing.OrdinalEncoder,“ 2020. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html>. [Zugriff am 29 März 2021].
- [107] pydata, „Intro to data structures,“ pydata, [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/user_guide/dsintro.html. [Zugriff am 24 Juni 2021].
- [108] 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.
- [109] C. Molnar, „Model-Agnostic Methods,“ in *Interpretable machine learning*, Bookdown, 2021.

- [110 PostgreSQL, „Write-Ahead Logging (WAL),“ PostgreSQL, [Online].
] Available: <https://www.postgresql.org/docs/9.1/wal-intro.html>. [Zugriff am
6 Juni 2021].
- [111 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.