Mahbouba Gharbi • Holger Klus
Holger Tiemeyer • Meng Zhang

Praxiswissen Softwarearchitektur für KI-Systeme



Aus- und Weiterbildung nach iSAQB®-Standard
zum Certified Professional for Software Architecture
Advanced Level – Modul SWARC4AI

iSAQB®
ITECH PROGRESS
dpunkt.verlag

5 Datenmanagement und Datenverarbeitung für KI-Systeme

Daten sind digitale Abbilder von Ereignissen und Zuständen, die durch Messungen oder Erfassungen in Prozessen und Domänen entstehen. Sie bilden die Grundlage jeder KI-Anwendung. Da leistungsfähige KI-Systeme in der Regel große Datenmengen benötigen, kommt einem effizienten Datenmanagement und einer strukturierten Datenverarbeitung zentrale Bedeutung zu. Nur durch gezielte und qualitätsgesicherte Umsetzung dieser Prozesse lassen sich KI-Modelle mit vertretbarem Aufwand, hoher Modellgüte und reproduzierbarer Leistung entwickeln.

Dieses Kapitel bietet einen systematischen Überblick über wesentliche Methoden, Techniken und Strategien im Datenmanagement und in der Datenverarbeitung für KI-Systeme. Im Fokus stehen Verfahren zur Datenerfassung und -annotation (Labeling), der Entwurf und die Realisierung moderner Datenpipelines, Speicherlösungen sowie Ansätze zur Sicherstellung von Datenqualität, Konsistenz und Datensicherheit. Ergänzend werden etablierte Werkzeuge und Plattformen des Data Engineerings vorgestellt, die den Aufbau skalierbarer und robuster Dateninfrastrukturen unterstützen und damit eine grundlegende Basis für zuverlässige KI-Anwendungen schaffen.

5.1 Architektonische Einordnung dieses Kapitels

Datenmanagement und Datenverarbeitung müssen architektonisch explizit als zentrale Bestandteile von KI-Systemen gestaltet werden. Sie bestimmen maßgeblich die Qualität der Modelle, die Reproduzierbarkeit von Trainingsläufen sowie die Stabilität und Wartbarkeit des Gesamtsystems. Fehlende Strukturierung, unzureichende Versionierung oder inkonsistente Datenpipelines führen häufig zu nicht nachvollziehbaren Ergebnissen, erhöhten Betriebsrisiken und eingeschränkter Auditierbarkeit.

Für Softwarearchitekten ist insbesondere relevant, dass Daten nicht nur als Eingabe für Modelle dienen, sondern zugleich Trainingsgrundlage, Laufzeitabhängigkeit und Compliance-Artefakt darstellen. Daraus ergeben sich enge Abhängigkeiten zwischen Datenarchitektur, MLOps-Prozessen, Governance-Strukturen und Betriebsmodellen.

Die Architektur muss daher sicherstellen, dass Datenflüsse, Speicherstrukturen und Transformationsprozesse konsistent, versionierbar und transparent gestaltet

sind. Gleichzeitig sind Anforderungen an Datenschutz, Datensicherheit und regulatorische Nachweisbarkeit systematisch in die Datenpipelines zu integrieren.

Dieses Kapitel zeigt, wie Datenmanagement und Datenverarbeitung als tragende Architekturelemente konzipiert werden können, um Skalierbarkeit, Nachvollziehbarkeit und langfristige Wartbarkeit von KI-Systemen zu gewährleisten.

5.2 Methoden der Datenakquisition und Datenannotation

5.2.1 Datenakquisition

Da die Entwicklung von KI-Anwendungen in hohem Maße von der Verfügbarkeit und Qualität geeigneter Daten abhängt, spielt die Datenakquisition eine zentrale Rolle im KI-Engineering. Obwohl dieser Prozess in vielen Entwicklungsprojekten häufig unterschätzt wird, handelt es sich um eine besonders arbeits- und ressourcenintensive Aufgabe. Sie bildet die Grundlage für alle nachgelagerten Verarbeitungsschritte und beeinflusst maßgeblich die Effektivität und Leistungsfähigkeit des nachfolgenden Trainingsprozesses. Ein sorgfältig geplanter und durchgeführter Akquisitionsprozess wirkt dabei wie eine präzise Vorbereitung, ganz im Sinne des Sprichworts: »Gut vorbedacht ist halb gemacht.«

Je nach Erhebungsmethode lässt sich die Datenakquisition in verschiedene Kategorien einteilen. Eine klassische Form ist die manuelle Datenerhebung, bei dem Daten gezielt durch menschliche Akteure gesammelt oder erstellt werden. Diese Methode ermöglicht eine sehr genaue Anpassung an spezifische Fragestellungen und zeichnet sich durch hohe inhaltliche Kontrolle aus. Ihr Nachteil liegt jedoch im beträchtlichen Zeit- und Kostenaufwand sowie in der eingeschränkten Skalierbarkeit. Neben der direkten Erfassung von Rohdaten, etwa Bildern oder Tonaufnahmen, kommen häufig auch strukturierte Verfahren wie Interviews oder Umfragen zum Einsatz.

Demgegenüber erlauben automatisierte Verfahren eine effizientere und kontinuierliche Datenerhebung. Besonders verbreitet ist die sensorgestützte Datenerfassung, bei der physische oder virtuelle Sensoren, beispielsweise Kameras, LiDAR-Systeme oder Temperatursensoren, die Umgebungsdaten automatisch erfassen. Diese Methode bietet eine hohe zeitliche Auflösung und Echtzeitfähigkeit, erfordert jedoch geeignete Hardware und ist anfällig für Störungen durch Datenrauschen oder Messfehler.

Darüber hinaus werden zunehmend netzbasierte Erhebungsverfahren eingesetzt, etwa durch Web Scraping oder Web Crawling. Sie ermöglichen den Zugriff auf große und vielfältige Datenmengen, wie sie beispielsweise für das Training großer Sprachmodelle benötigt werden. Da solche Datenquellen jedoch heterogen und häufig unstrukturiert sind, ist eine nachgelagerte Datenbereinigung unverzichtbar. Zudem müssen rechtliche Rahmenbedingungen, insbesondere Urheberrecht und Datenschutz, sorgfältig beachtet werden.

Neben der aktiven Erhebung kann auch auf bestehende Datenquellen zurückgegriffen werden, wie Log- und Protokolldaten, Datenbanken oder frei verfügbare

Open-Data-Portale. Eine detaillierte Betrachtung dieser Quellen folgt später in Abschnitt 5.2.3; an dieser Stelle sei lediglich hervorgehoben, dass sie den Aufwand der Datenerhebung deutlich reduzieren können, zugleich aber hinsichtlich Aktualität und Qualität kritisch zu prüfen sind.

Zunehmend gewinnen auch synthetische und simulierte Daten an Bedeutung. Sie stammen nicht aus der realen Welt, sondern werden durch Simulationen oder generative Modelle künstlich erzeugt. Beispiele sind Verkehrssimulationen zur Erzeugung von Trainingsdaten für automatisierte Fahrsysteme oder der Einsatz generativer Ansätze wie GANs (Generative Adversarial Networks, siehe Abschnitt 2.3.5) zur Erzeugung realistischer Bilddaten. Diese Verfahren ermöglichen eine kontrollierte und vielfältige Datengenerierung, bergen jedoch das Risiko von Verzerrungen oder Domänenabweichungen, da synthetische Daten nicht immer alle Eigenschaften realer Phänomene widerspiegeln.

Unabhängig von der Quelle ist im Anschluss an die Erhebung eine sorgfältige Datenkuratierung erforderlich, um Qualität, Konsistenz und Validität sicherzustellen. Dazu gehören die Bereinigung fehlerhafter Einträge, die Harmonisierung unterschiedlicher Formate und die Dokumentation von Herkunft und Struktur der Daten. Ebenso sind datenschutzrechtliche und ethische Anforderungen zu beachten. Der Umgang mit personenbezogenen oder sensiblen Daten erfordert die Einhaltung rechtlicher Vorgaben, etwa der DSGVO, sowie eine transparente und verantwortungsbewusste Datenverarbeitung im Sinne anerkannter ethischer Leitlinien.

Unabhängig von der Quelle ist im Anschluss an die Erhebung eine sorgfältige Datenkuratierung erforderlich, um Qualität, Konsistenz und Validität sicherzustellen. Hierzu zählen die Bereinigung fehlerhafter Einträge, die Harmonisierung unterschiedlicher Formate sowie die Dokumentation von Herkunft und Struktur der Daten. Ebenso sind rechtliche Anforderungen zu beachten. Der Umgang mit personenbezogenen oder sensiblen Daten unterliegt der Datenschutz-Grundverordnung (DSGVO) [eu-dsgvo]; zusätzlich sind u. a. Urheber- und Nutzungsrechte (einschließlich Lizenzbedingungen), Vertrags- und Wettbewerbsrecht sowie branchenspezifische Vorgaben relevant. Transparente Prozesse und eine nachvollziehbare Dokumentation unterstützen Compliance und Nachweisführung.

5.2.2 Datenannotation (Labeling)

Nach der Datenerhebung sind die gesammelten Informationen häufig nicht unmittelbar für das Training von KI-Modellen verwendbar. Besonders bei überwachten Lernverfahren ist eine explizite Kennzeichnung der Daten erforderlich. So müssen etwa Bilder mit Objektklassen versehen werden, wenn ein neuronales Netz zur Bildklassifikation trainiert werden soll. In solchen Fällen ist ein zusätzlicher Verarbeitungsschritt notwendig: die Datenannotation, auch als Labeling oder Datenbeschriftung bezeichnet.

Traditionell erfolgt die Annotation vollständig manuell durch menschliche Annotatoren. Dabei werden Bildinhalte klassifiziert oder auf Pixelebene segmentiert.

Dieses Verfahren ist jedoch äußerst zeit- und kostenintensiv, insbesondere bei großen Datensätzen. Um den Aufwand zu verringern, wurde die Annotation häufig an externe Dienstleister in Niedriglohnländern ausgelagert. In einigen Fällen wird die manuelle Annotation zudem mit sicherheitsrelevanten Nutzerverifikationen kombiniert, beispielsweise bei CAPTCHA-Systemen, bei denen Nutzer durch das Erkennen bestimmter Bildinhalte zur Datenerhebung beitragen.

Mit dem Fortschritt automatisierter Verfahren wurden in den letzten Jahren zunehmend (teil-)automatisierte Tools entwickelt, die den Annotierungsprozess effizienter gestalten. Bei der halbautomatischen Datenannotation erstellt ein KI-System zunächst Vorhersagen, die anschließend von menschlichen Annotatoren geprüft und gegebenenfalls korrigiert werden. Dies ermöglicht eine signifikante Effizienzsteigerung bei gleichbleibender Datenqualität [SO++08] [AP08] [DT++23]. Vollautomatisierte Verfahren bieten eine hohe Skalierbarkeit und sind besonders für einfache, klar strukturierte Aufgaben geeignet. Bei komplexen oder sicherheitskritischen Anwendungen stößt die vollständige Automatisierung jedoch an ihre Grenzen, da hier eine hohe Präzision und Verlässlichkeit erforderlich sind. In solchen Fällen ist eine manuelle oder zumindest teilautomatisierte Qualitätskontrolle unerlässlich.

Zur Unterstützung des Annotationsprozesses sind eine Reihe leistungsfähiger Werkzeuge entstanden. Ein typisches Beispiel ist CVAT (Computer Vision Annotation Tool) [cvat], das speziell für die Annotation von Bildern und Videos in Computer-Vision-Projekten entwickelt wurde. Plattformen wie Amazon Mechanical Turk [amazon-mechturk] ermöglichen zudem die skalierbare Annotation durch Crowdsourcing mit einer Vielzahl menschlicher Annotatoren. Flexible Open-Source-Tools wie Label Studio [label-studio] unterstützen die Annotation verschiedenster Datentypen und lassen sich an individuelle Projektanforderungen anpassen.

Trotz dieser technologischen Fortschritte reicht die automatisierte Annotation allein bislang nicht aus, um ohne weitere Maßnahmen verlässliche Ergebnisse zu erzielen. Für eine hohe Plausibilität und Validität der Annotationsergebnisse sind zusätzliche Qualitätssicherungsmaßnahmen erforderlich. Dazu zählen Konsistenzprüfungen auf logischer und semantischer Ebene sowie Verfahren wie Mehrfachannotation und Expertenvalidierung. Diese sichern zwar eine hohe Annotationsqualität, führen jedoch zu einem deutlich erhöhten Aufwand und Kosten. In der Praxis ist daher ein geeigneter Trade-off zwischen Effizienz, Kosten und Qualität zu finden.

5.2.3 Plattformen für öffentlich zugängliche Daten

Mit dem rasanten Fortschritt im Bereich der KI-Technologien hat sich auch die globale KI-Community erheblich vergrößert. Da der Erfolg des KI-Engineerings maßgeblich von der Verfügbarkeit qualitativ hochwertiger Trainingsdaten abhängt, wurden zahlreiche Plattformen für öffentlich zugängliche Daten geschaffen. Diese fördern Transparenz, wissenschaftliche Reproduzierbarkeit sowie Innovation in Forschung und Industrie.

Ein Teil dieser Plattformen wird von öffentlichen Institutionen und Regierungen betrieben. So stellt beispielsweise die Plattform data.gov [data-gov] in den USA umfangreiche Regierungsdaten aus Bereichen wie Gesundheit, Energie und Bildung bereit. In Deutschland verfolgt GovData.de [gov-data] ein ähnliches Ziel, indem sie offene Verwaltungsdaten aus Bund, Ländern und Kommunen zentral zugänglich macht. Auf europäischer Ebene dient Eurostat [eurostat] als zentrale Anlaufstelle für statistische Daten zu Wirtschaft, Demografie und Umwelt.

Auch wissenschaftliche Einrichtungen und Universitäten tragen wesentlich zur Bereitstellung offener Datensätze bei. Da reproduzierbare Forschung im Bereich des Maschinellen Lernens ohne offene Daten kaum möglich ist, veröffentlichen viele Forschungsprojekte ihre Datensätze im Anschluss an Publikationen in Konferenzen oder Fachzeitschriften. Bekannte Plattformen in diesem Kontext sind das UCI Machine Learning Repository [DG17], Kaggle [kaggle] und OpenML [VR++14].

Darüber hinaus engagieren sich auch Unternehmen, insbesondere große Technologiekonzerne wie Google oder Microsoft. Im Rahmen ihrer Corporate-Social-Responsibility(CSR)-Initiativen stellen sie Datensätze bereit, um Innovationen durch externe Entwickler und Forschungseinrichtungen zu ermöglichen. Beispiele hierfür sind Google Dataset Search [BBN++19] [dataset-search] und Microsoft Research Open Data [ms-opendata].

Zudem existieren zahlreiche Community- und Crowdsourcing-basierte Datenplattformen, bei denen Datensätze durch freiwillige Beiträge von Einzelpersonen oder Gruppen entstehen. Zu den bekanntesten zählen Common Crawl [commoncrawl], eine Sammlung frei verfügbarer Webdaten, sowie Projekte wie Zooniverse [zooniverse] oder OpenStreetMap [osm], das geobasierte Daten durch eine globale Nutzergemeinschaft sammelt.

Obwohl diese Plattformen den Zugang zu wertvollen Trainingsdaten erleichtern und das KI-Engineering erheblich unterstützen, bedeutet ihre Nutzung nicht zwangsläufig eine bedenkenlose Übernahme der bereitgestellten Inhalte. In der Praxis müssen Kriterien wie Datenqualität und Formatkompatibilität sowie semantische Konsistenz sorgfältig geprüft werden. Ein weiterer entscheidender Aspekt betrifft die Art der Lizenzierung (siehe Abschnitt 3.3.6). Während viele Datensätze für nichtkommerzielle Nutzung freigegeben sind, kann der Einsatz in kommerziellen KI-Anwendungen Einschränkungen unterliegen. Entsprechend sind die Lizenzbedingungen im Vorfeld einer Verwendung kritisch zu prüfen, insbesondere wenn ein trainiertes Modell produktiv oder wirtschaftlich eingesetzt werden soll.

5.2.4 Architektonische Einordnung: Datenakquisition und Datenannotation

Datenakquisition und Datenannotation müssen architektonisch als integrale Bestandteile von KI-Systemen berücksichtigt werden. Unstrukturierte Erhebungsprozesse, fehlende Dokumentation der Datenherkunft und inkonsistente Labeling-

Verfahren führen häufig zu verzerrten Trainingsdaten, eingeschränkter Nachvollziehbarkeit und erhöhten regulatorischen Risiken.

Für Softwarearchitekten ist insbesondere relevant, dass Akquisitions- und Annotationsprozesse langfristige strukturelle Abhängigkeiten erzeugen. Entscheidungen über Datenquellen, Erhebungsmethoden und Labeling-Regeln beeinflussen die Modellqualität, die Wartbarkeit sowie die Möglichkeit zur späteren Reproduktion und Auditierung.

Die Architektur muss daher sicherstellen, dass Datenerhebung und Annotation nicht als einmalige Vorarbeiten, sondern als dauerhafte, versionierbare und qualitätsgesicherte Prozesse gestaltet werden. Gleichzeitig sind Datenschutz-, Lizenz- und Governance-Anforderungen systematisch in diese Prozesse zu integrieren.

Für die Architektur ergeben sich insbesondere folgende Anforderungen:

- Unterstützung versionierter Erhebungs- und Labeling-Pipelines
- Etablierung nachvollziehbarer Dokumentation zur Datenherkunft und Annotation
- Integration automatisierter und manueller Qualitätskontrollen
- Trennung von Rohdaten, annotierten Daten und Validierungsdaten
- Absicherung rechtlicher und regulatorischer Vorgaben in der Datennutzung
- Berücksichtigung von Feedback-Schleifen aus Training und Betrieb

Aspekt	**Bedeutung für die Architektur**
Datenquellen	Definierte Schnittstellen und Verträge
Annotation	Versionierte und überprüfbare Labeling-Prozesse
Qualität	Mehrstufige Validierungsmechanismen
Betrieb	Nachvollziehbare Reproduzierbarkeit
Governance	Lizenz-, Datenschutz- und Nutzungsregeln
Skalierung	Steuerbarer Aufwand bei wachsendem Datenvolumen
Risiko	Bias, inkonsistente Labels, Haftungsrisiken
Typische Fehlannahmen	
»Ein einmal erhobener Datensatz reicht dauerhaft aus.«	
»Labels sind objektiv und eindeutig.«	
»Annotation kann vollständig automatisiert werden.«	
»Datenherkunft ist für den späteren Betrieb irrelevant.«	
»Rechtliche Fragen lassen sich nachträglich klären.«	

Tab. 5.1 *Architectural Impact Box: Datenakquisition und Annotation*

Diese Fehlannahmen führen häufig zu verzerrten Modellen, eingeschränkter Nachvollziehbarkeit und erhöhten Compliance-Risiken.

5.3 Gestaltung effizienter Datenpipelines und -architekturen

Im KI-Engineering spielt die Verarbeitung großer und heterogener Datenmengen eine zentrale Rolle. Damit diese Daten effizient verwaltet und für die Entwicklung und den Betrieb von KI-Modellen genutzt werden können, ist der Aufbau robuster Datenpipelines sowie geeigneter Datenarchitekturen unerlässlich. Solche Strukturen automatisieren zentrale Verarbeitungsschritte, insbesondere die Extraktion, Transformation und das Laden der Daten (sogenannte ETL-Prozesse), und gewährleisten, dass KI-Modelle konsistente, bereinigte und relevante Daten sowohl für das Training als auch für die Inferenz erhalten. Die folgenden Abschnitte geben einen systematischen Überblick über grundlegende Prinzipien sowie architektonische Gestaltungsmöglichkeiten moderner Dateninfrastrukturen für KI-Systeme.

5.3.1 Grundprinzipien effizienter Datenpipelines

Beim Aufbau von Datenpipelines und -architekturen für KI-Systeme müssen verschiedene zentrale Anforderungen berücksichtigt werden, um eine robuste, anpassungsfähige und bedarfsgerechte Datenverarbeitung sicherzustellen. Diese Anforderungen lassen sich als grundlegende Gestaltungsprinzipien verstehen, die den Aufbau und Betrieb moderner Dateninfrastrukturen prägen.

Ein wesentliches Prinzip betrifft die Verarbeitungskapazität der Pipeline. Da in KI-Anwendungen häufig sehr große Datenmengen anfallen, muss die Architektur in der Lage sein, diese Mengen effizient zu bewältigen. Entscheidend ist dabei die Parallelisierbarkeit der Verarbeitungsschritte, um die Leistungsfähigkeit bei wachsendem Datenaufkommen aufrechtzuerhalten.

Neben der Datenmenge spielt auch die Datenaktualität (im Sinne von Geschwindigkeit der Datenänderung – Data Velocity) eine wichtige Rolle. Besonders KI-Systeme, die auf sich verändernde Eingangsdaten reagieren oder Entscheidungen in Echtzeit treffen müssen, benötigen Mechanismen zur kontinuierlichen und zeitnahen Verarbeitung eingehender Datenströme. Hierfür sind Streaming-Architekturen erforderlich, die schnelle Reaktionszeiten und stabile Durchsatzraten gewährleisten.

Ein weiteres Prinzip betrifft die Datenvielfalt (im Sinne von Vielfalt der Datenquellen – Data Variety). Moderne KI-Systeme verarbeiten Informationen aus unterschiedlichsten Quellen und Formaten, von Datenbanken über Sensordatenströme bis zu Text-, Bild- oder Videodaten. Um diese Heterogenität effektiv zu handhaben, müssen Datenpipelines flexibel genug sein, um verschiedene Strukturen zu erkennen, zu vereinheitlichen und für die nachfolgenden Verarbeitungsschritte in geeigneter Form bereitzustellen.

Darüber hinaus ist die Datenqualität von zentraler Bedeutung für die Leistungsfähigkeit der KI-Modelle. Trainings- und Inferenzdaten müssen präzise und konsistent sein, um Verzerrungen und Überanpassungen in den Modellen zu vermeiden und die Generalisierungsfähigkeit sicherzustellen. Hierbei spielt auch die einheitliche Standardisierung der Datenformate, beispielsweise durch Formate wie CSV, JSON oder XML, eine wichtige Rolle für eine reibungslose, automatisierte Weiterverarbeitung innerhalb der Pipelines.

Ein weiteres Gestaltungsprinzip betrifft das Datenmanagement im Sinne eines Lebenszyklusmanagements. Analog zur klassischen Human-Engineered Softwareentwicklung, in der Artefakte wie Softwareversionen, Konfigurationen und Abhängigkeiten sorgfältig verwaltet werden, ist es im KI-Engineering erforderlich, sowohl die trainierten Modelle als auch die jeweils verwendeten Trainings- und Validierungsdatensätze systematisch zu verwalten und zu versionieren. Dies ermöglicht eine transparente Nachvollziehbarkeit der Modellentwicklung und stellt sicher, dass Modelländerungen stets mit den zugrunde liegenden Datenständen abgeglichen werden können. Somit besitzen nicht nur die Modelle, sondern auch die genutzten Datensätze eigene Lebenszyklen, die synchron verwaltet werden müssen.

Schließlich ist die Wahl des Verarbeitungsmodells, Batch oder Streaming, ein zentrales Architekturentscheidungsfeld. Klassische ETL-Prozesse setzen auf periodisch ausgeführte Batch-Jobs, während Echtzeitanwendungen eine kontinuierliche Verarbeitung erfordern. Der jeweilige Ansatz sollte sich strikt am Anwendungskontext sowie an den funktionalen und qualitativen Systemanforderungen orientieren.

5.3.2 Architekturgestaltung effizienter Datenpipelines

Für die verlässliche Verwaltung und Verarbeitung von Daten entlang des gesamten Entwicklungsprozesses eines KI-Systems, von der Datenakquise über die Vorverarbeitung bis hin zur Speicherung und Nutzung, spielt die zugrunde liegende Architektur der Datenpipeline eine zentrale Rolle. Dabei müssen nicht nur funktionale Anforderungen und Qualitätsanforderungen berücksichtigt werden, sondern ebenso projektspezifische Rahmenbedingungen, darunter Datenvolumen, Integrationsaufwand oder regulatorische Vorgaben.

Obwohl im Fachgebiet des Data Engineering bislang keine formalisierte und breit anerkannte Architekturtaxonomie etabliert wurde, lassen sich dennoch grundlegende Prinzipien und Strukturmuster identifizieren, die als konzeptionelle Leitlinien dienen können. Ziel dieses Abschnitts ist es daher, eine systematisierende, wenn auch bewusst informelle Kategorisierung vorzunehmen, die sowohl forschungsorientierte Leserinnen und Leser inspiriert als auch praxisnahen Einblick für die konkrete Umsetzung in KI-Projekten bietet.

Zur besseren Strukturierung erfolgt die Darstellung der Datenpipeline-Architektur entlang eines dreistufigen Abstraktionsmodells. Die erste Abstraktionsebene beschreibt die grundlegende Datenflussstrategie, also die operative Logik, mit der Daten innerhalb eines Systems bewegt, transformiert und weitergeleitet werden.

Auf der zweiten Ebene werden unterschiedliche Verarbeitungsmodelle betrachtet, die aufzeigen, wie Daten zeitlich-technisch innerhalb der Pipeline verarbeitet werden, etwa als Stapel (Batch), in Echtzeit oder durch hybride Verfahren. Die dritte Ebene schließlich widmet sich den Architekturparadigmen als übergeordnetes strukturelles Denkmodell, das festlegt, wie eine Pipeline als Gesamtsystem organisiert ist, welche Komponenten beteiligt sind und über welche Schnittstellen sie miteinander interagieren.

In dieser gestuften Betrachtung ergibt sich ein klarer Zusammenhang zwischen den Ebenen: Ein gewähltes Architekturparadigma definiert die grundlegende Organisation und Zusammensetzung der Systemkomponenten. Daraus leiten sich in Abhängigkeit von den Anforderungen, etwa dem Echtzeitbedarf oder der Datenherkunft, geeignete Verarbeitungsmodelle ab. Diese wiederum nutzen konkrete Strategien wie ETL oder ELT, um Daten effizient von ihrer Quelle zu einem Zielsystem zu überführen und dort nutzbar zu machen. Durch diese strukturierte Herangehensweise wird es möglich, komplexe Datenpipelines systematisch zu planen, zu analysieren und schrittweise zu realisieren.

Grundstrategien der Datenflüsse in Datenpipelines

Wie in Abschnitt 5.3.2 vorgestellt, wird die Datenflussstrategie auf der ersten Abstraktionsebene festgelegt und wie Daten bewegt, umgeformt und bereitgestellt werden. Eine etablierte Ausprägung dieser Ebene ist das sogenannte ETL-Modell mit den drei Phasen Extract, Transform und Load (vgl. Abbildung 5.1).

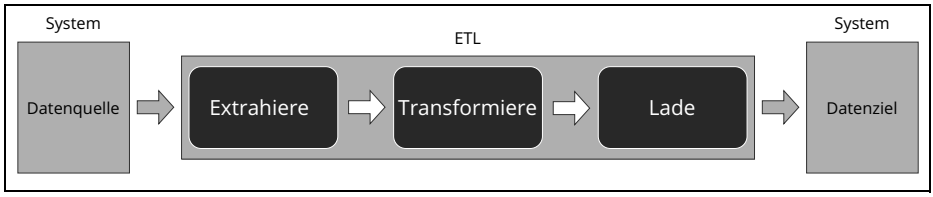

Abb. 5.1 *Schematischer Aufbau einer ETL-Pipeline*

In diesem Modell werden die relevanten Daten zunächst aus verschiedenen Quellen extrahiert, etwa aus relationalen Datenbanken, CSV-Dateien, Logdateien oder Sensordatenströmen. Da diese Datenquellen typischerweise stark heterogen sind, sowohl in Bezug auf ihre Formate als auch hinsichtlich der zugrunde liegenden Übertragungsprotokolle, ist es in der Phase der Datenextraktion besonders wichtig, technische Aspekte wie Zugriffsgeschwindigkeit, Bandbreite, Latenz und Fehlertoleranz zu berücksichtigen.

Nach der Extraktion erfolgt die Transformation der gesammelten Rohdaten. Dabei werden die Daten bereinigt, vereinheitlicht und in eine semantisch sowie technisch geeignete Struktur überführt, sodass sie für nachgelagerte Verarbeitungs- oder Analyseprozesse verwendet werden können. In klassischen ETL-Systemen findet diese Transformation außerhalb des Zielsystems statt, in separaten Verarbei-

tungsknoten, zentralen ETL-Engines oder spezialisierten Transformationsdiensten. Die transformierten und harmonisierten Daten werden anschließend in ein Zielsystem geladen, beispielsweise in ein Data Warehouse oder einen dedizierten Speicherbereich für Maschinelles Lernen. Das grundlegende Prinzip des ETL-Modells entspricht dem eines Adaptermusters: Inkompatible Quelldaten werden in ein standardisiertes Austauschformat transformiert und an die Schnittstellen des Zielsystems angepasst.

ETL-Pipelines kommen vor allem in Anwendungsfällen zum Einsatz, in denen periodische Reports erstellt oder historische Datenanalysen durchgeführt werden sollen. Sie eignen sich auch zur Vorbereitung von Trainingsdaten für maschinelle Lernmodelle, insbesondere wenn der Datenbestand stabil, strukturiert und weitgehend unverändert ist. Die vorgelagerte Transformation bietet hierbei den Vorteil, dass das Zielsystem ausschließlich bereinigte, valide und konsistente Daten erhält, wodurch sich die Betriebssicherheit und Qualität der darauf aufbauenden Anwendungen verbessern. Dennoch ist dieser Ansatz nicht in allen Fällen ideal. Gerade bei sehr großen Datenmengen kann der Transformationsprozess vor dem Laden erhebliche Rechenressourcen beanspruchen und zu langen Verarbeitungszeiten führen. Zudem ist die Flexibilität für explorative Datenanalysen eingeschränkt, da Ad-hoc-Abfragen auf den Rohdaten nicht ohne Weiteres möglich sind. Auch strukturelle Änderungen an den Datenquellen, etwa durch neue Spalten oder veränderte Schemata, erfordern in klassischen ETL-Architekturen oft eine vollständige Neudefinition und erneute Ausführung des Transformationsprozesses.

Um diesen Herausforderungen zu begegnen, hat sich im Kontext moderner, skalierbarer Datenverarbeitung das ELT-Modell (vgl. Abbildung 5.2) etabliert. Dabei werden die Daten zunächst ebenfalls extrahiert, jedoch unmittelbar in ein Zielsystem geladen, ohne zuvor transformiert zu werden. Erst im Anschluss erfolgt die Transformation innerhalb dieses Zielsystems, beispielsweise durch SQL-Abfragen, materialisierte Sichten oder modellgetriebene Workflows mit Tools wie dbt (Data Build Tool) [dbt-labs]. Das Zielsystem übernimmt somit einen Großteil der Rechenlast, wodurch die ursprünglichen ETL-Engines entlastet werden und die Verarbeitung großer Datenmengen bedarfsgerecht erfolgen kann. Durch diese Umkehr der Verarbeitungsreihenfolge können alle Rohdaten vollständig gespeichert werden, was insbesondere für Data-Science-Teams von Vorteil ist, die flexibel auf unterschiedliche Datenversionen und Analysebedarfe zugreifen möchten.

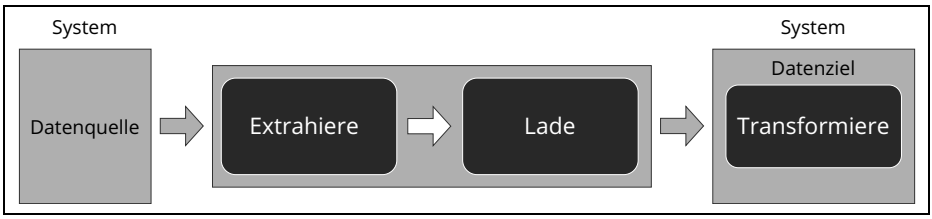

Abb. 5.2 *Schematischer Aufbau einer ELT-Pipeline*

Die ELT-Pipeline erlaubt zudem eine deutlich höhere Anpassungsfähigkeit an sich wandelnde Anforderungen. Transformationen können unabhängig von der Datenakquise neu definiert und versioniert werden. Darüber hinaus lassen sich mehrere Sichten auf dieselben Daten realisieren, ohne dass diese mehrfach physisch transformiert oder gespeichert werden müssen.

Dieser Ansatz eignet sich insbesondere für Cloud-native Architekturen, bei denen leistungsfähige Speichersysteme, wie moderne Data Warehouses oder Lakehouses (vgl. Abschnitt 5.5), als zentrales Datenrückgrat fungieren. Hier ermöglichen Plattformen wie Snowflake, Databricks oder Google BigQuery nicht nur die Speicherung großer Datenmengen, sondern auch deren unmittelbare Verarbeitung innerhalb des Systems.

Datenverarbeitungsmodelle in Datenpipelines

Neben der grundlegenden Strategie des Datenflusses, wie sie etwa durch ETL- oder ELT-Modelle vorgegeben wird, ist für die technische Ausgestaltung konkreter Lösungen innerhalb von Datenpipelines ein vertieftes Verständnis der zugrunde liegenden Datenverarbeitungsmodelle unerlässlich. Im Mittelpunkt stehen dabei Aspekte wie Latenz, Durchsatz, Zustandsverwaltung, Fehlerbehandlung und die betriebliche Steuerbarkeit. Diese Modelle definieren die zeitliche Organisation und die operativen Semantiken der Verarbeitung, also unter welchen Garantien und in welchen Zeitfenstern Daten verarbeitet und weitergegeben werden.

Ein etablierter Ansatz ist die Stapelverarbeitung, auch als Batch-Verarbeitung (vgl. Abbildung 5.3) bekannt. Dabei werden Daten über ein bestimmtes Zeitintervall hinweg gesammelt und anschließend blockweise verarbeitet. Die Verarbeitung beginnt erst, wenn ein vollständiger Datenstapel vorliegt, was eine hohe Latenz mit sich bringt. Dieses Verfahren eignet sich daher vor allem für Anwendungen, bei denen eine unmittelbare Reaktion nicht erforderlich ist, etwa bei periodischen Auswertungen, dem Training von Offline-Modellen oder dem Reporting historischer Kennzahlen.

Die Stärke der Batch-Verarbeitung liegt in ihrer Robustheit und Effizienz. Da die Daten in abgeschlossenen Einheiten verarbeitet werden, lassen sich Fehler leichter identifizieren und isolieren. Typischerweise kommen bei der technischen Umsetzung Frameworks wie Apache Hadoop oder Apache Spark zum Einsatz. Die Architektur solcher Systeme umfasst in der Regel verschiedene Komponenten: Zunächst werden die Daten aus unterschiedlichen Quellen wie Datenbanken oder Protokolldateien extrahiert. Anschließend erfolgt die Speicherung in einem Data Lake oder einem verteilten Dateisystem wie bspw. HDFS. Die eigentliche Verarbeitung geschieht durch geplante Jobs, die von spezialisierten Frameworks ausgeführt werden. Die Ergebnisse werden schließlich in Zielsysteme wie bspw. Data Warehouses (siehe Abschnitt 5.5) oder Analyseplattformen überführt.

Trotz ihrer Vorteile bei der Verarbeitung großer Datenmengen ist die Batch-Verarbeitung für Echtzeitanwendungen ungeeignet, da sie keine kontinuierliche Analyse erlaubt und aufgrund der blockweisen Verarbeitung stets mit einer gewissen Verzögerung verbunden ist.

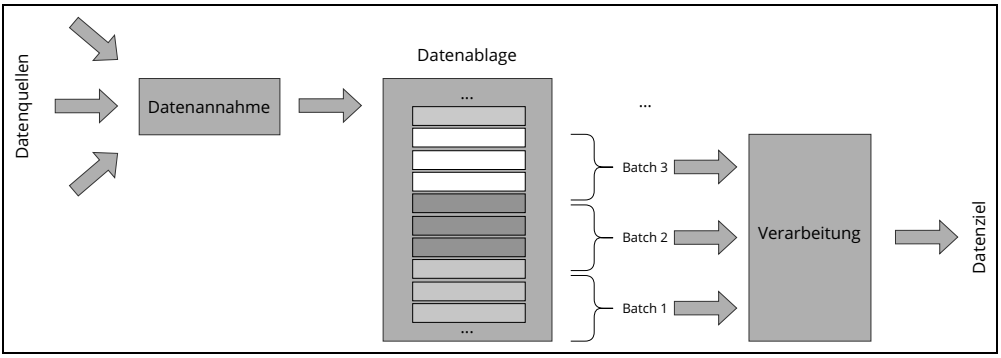

Abb. 5.3 *Schematischer Aufbau der Batch-Verarbeitung/Stapelverarbeitung*

Das Pendant zur Batch-Verarbeitung ist die Streaming-Verarbeitung. Im Gegensatz zur Batch-Verarbeitung folgt die Streaming-Verarbeitung einem ereignisgesteuerten Paradigma. Daten werden kontinuierlich beim Eintreffen verarbeitet, wodurch sehr geringe Latenzen möglich sind. Geeignet ist dies insbesondere für Szenarien mit unmittelbarem Reaktionsbedarf, etwa für Anomalie- und Ereigniserkennung in Sensornetzen, Betrugserkennung in Zahlungsströmen oder latenzkritische Online-Inferenz in KI-gestützten Diensten. Entscheidend ist dabei weniger ein »anderer Datensatz« als vielmehr die Verarbeitungsform: Modelle werden häufig offline (batchbasiert) trainiert und anschließend im Stream für die Echtzeitentscheidung bereitgestellt; optional können Aktualisierungen durch Mikro-Batch-Retraining oder kontrollierte Online-Lernverfahren erfolgen.

Technisch liegt die Herausforderung in der dynamischen Zustandsverwaltung. Ohne feste Datenblöcke muss der Anwendungszustand während der laufenden Verarbeitung konsistent geführt und fortgeschrieben werden, inklusive eventzeitbasierter Semantik (Fenster, Watermarks), Liefergarantien (z. B. exactly-once) und Rückstaukontrolle. Hierfür kommen spezialisierte Stream-Processing-Engines wie Apache Kafka Streams, Apache Flink oder Apache Beam zum Einsatz.

Eine typische Streaming-Architektur umfasst mehrere Komponenten: kontinuierliche Ereignisquellen (zum Beispiel Sensoren, Log-Streams, Messaging-Systeme wie Apache Kafka [apache-kafka]), einen Stream-Prozessor für Analyse und Entscheidungslogik sowie einen zustandsbehafteten Speicher (State Store), häufig wird eine eingebettete Key-Value-Engine wie RocksDB [rocks-db] eingesetzt. Die Ergebnisse werden primär an operative Senken ausgeliefert, etwa Aktorik-/Downstream-Services, Ereignisbusse oder Echtzeitdatenbanken für Serving und Alarmierung. Ergänzend können Visualisierungsschichten angeschlossen werden (z. B. Dashboards), sie stehen jedoch nicht im Vordergrund der Verarbeitungsarchitektur.

Dank ihrer geringen Latenz eignet sich die Streaming-Verarbeitung ideal für Echtzeitanwendungen und ist eng mit eventgetriebenen Architekturen verbunden. Aus naheliegenden Gründen ist sie in der Umsetzung komplexer als die klassische Batch-Verarbeitung, insbesondere was die Konsistenz, Fehlerbehandlung und Skalierbarkeit betrifft.

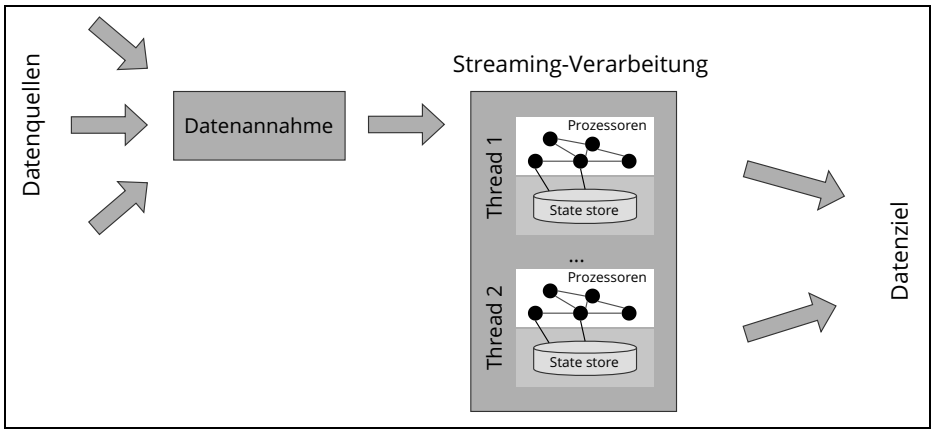

Abb. 5.4 *Schematischer Aufbau der Streaming-Verarbeitung*

Eine Zwischenform von der Batch- und Streaming-Verarbeitung stellt das sogenannte Micro-Batching (siehe Abbildung 5.5) dar, das insbesondere durch Technologien wie Apache Spark Streaming populär wurde. Bei diesem Ansatz werden eingehende Datenströme in sehr kurzen, meist sekundengenauen Intervallen gesammelt und als kleine Batches (sogenannte Minibatches) verarbeitet. Damit vereint Micro-Batching viele Vorteile der Stream-Verarbeitung mit den betrieblichen Vereinfachungen der Batch-Verarbeitung. Zwar liegt die Latenz etwas höher als bei reiner Echtzeitverarbeitung, doch lassen sich Skalierung, Fehlertoleranz und die Integration mit bestehenden Systemen oftmals deutlich effizienter realisieren.

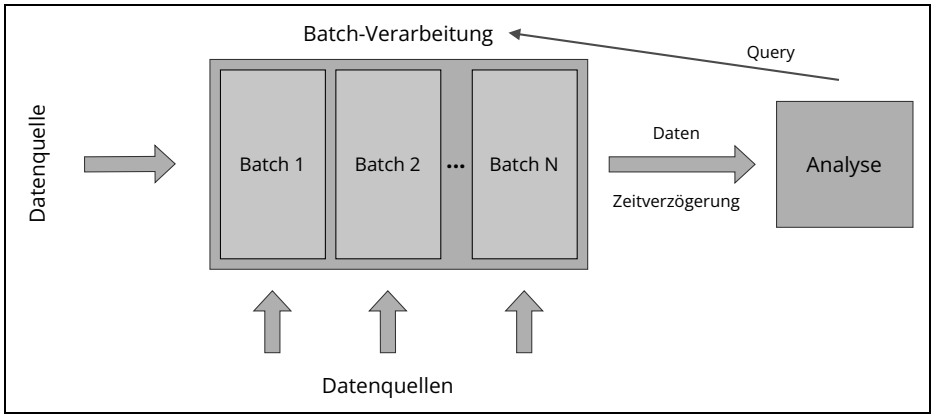

Abb. 5.5 *Schematischer Aufbau des Micro-Batchings*

Die Wahl des Verarbeitungsmodells erfolgt auf Grundlage einer fundierten Analyse funktionaler und betrieblicher Anforderungen. Systeme mit strikten Echtzeitzielen sind typischerweise mit Streaming-Ansätzen besser bedient; Batch-Verfahren spielen ihre Stärken bei periodischen oder retrospektiven Auswertungen aus. Micro-Batching bietet sich an, wenn eine Balance zwischen kurzer Latenz, technischer Umsetzbarkeit und Betriebsstabilität gesucht wird.

Architekturparadigmen der Datenpipelines

Nachdem in den vorangegangenen Abschnitten grundlegende Strategien des Datenflusses sowie verschiedene Datenverarbeitungsmodelle innerhalb von Pipelines behandelt wurden, folgt nun die Betrachtung übergeordneter architektonischer Konzepte, die den strukturellen und funktionalen Aufbau solcher Systeme prägen. Im Folgenden werden ausgewählte Paradigmen entlang zentraler Strukturmerkmale wie Steuerungslogik, Datenfluss und Integrationsmechanismus exemplarisch vorgestellt, um eine klare, anwendungsorientierte Einordnung ihrer Stärken und Grenzen darzustellen.

Ein bekanntes und häufig diskutiertes Architekturparadigma zur Kombination von Genauigkeit und Echtzeitfähigkeit in der Datenverarbeitung ist die sogenannte Lambda-Architektur, wie in Abbildung 5.6 schematisch dargestellt. Sie verfolgt einen hybriden Ansatz, indem sie Elemente der Batch- und der Streaming-Verarbeitung in einer gemeinsamen Architektur vereint. Ziel ist es, sowohl eine konsistente Sicht auf große Datenmengen zu ermöglichen als auch aktuelle Informationen mit minimaler Latenz bereitzustellen. Die Lambda-Architektur gliedert sich in drei funktionale Schichten: Batch-Layer, Speed-Layer und Serving-Layer.

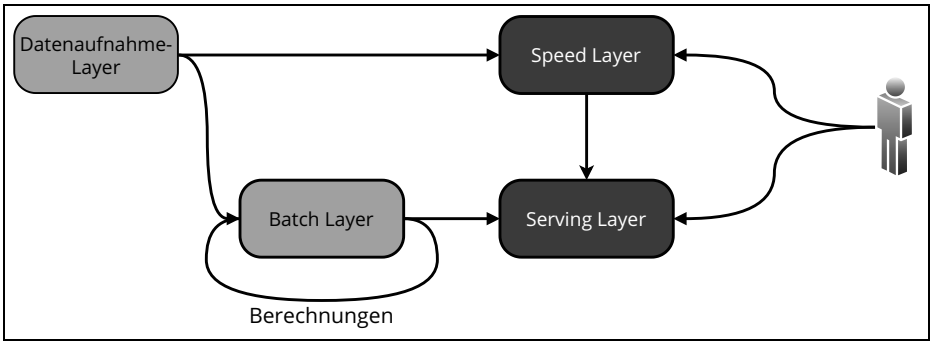

Abb. 5.6 *Schematischer Aufbau der Lambda-Architektur*

Im Batch-Layer (auch »Cold Path« genannt) werden große Datenmengen gesammelt, historisch aggregiert und in regelmäßigen Abständen verarbeitet. Hier entsteht ein sogenanntes Master Dataset, das eine vollständige und konsistente Datenbasis darstellt. Die Daten werden typischerweise in einem Data Lake gespeichert und können offline weiterverarbeitet oder angereichert werden, etwa für die Erstellung retrospektiver Berichte oder das Training von KI-Modellen.

Parallel dazu übernimmt der Speed-Layer (auch »Hot Path« genannt) die sofortige Verarbeitung neu eintreffender Daten. Sobald Daten über ereignisgesteuerte Quellen, etwa Sensorströme oder Messaging-Systeme, aufgenommen werden (»Data Ingestion«), erfolgt deren Echtzeitverarbeitung. Ziel ist es, eine möglichst zeitnahe Reaktion zu ermöglichen, etwa durch Visualisierungen, Dashboards oder die direkte Auslösung von Ereignissen, wie Warnmeldungen bei Anomalien. Da auf historische Kontextinformationen verzichtet wird, ist die Genauigkeit hier eingeschränkt, dafür aber die Reaktionsgeschwindigkeit besonders hoch.

Beide Verarbeitungswege münden im Serving-Layer, der einen einheitlichen Zugriff auf vorberechnete oder aktuell verarbeitete Daten bietet. Häufig erfolgt dies über schnelle Speicherlösungen wie In-Memory-Datenbanken, um sofortige Abfragen für Analysen oder KI-Systeme zu ermöglichen.

Die Stärke der Lambda-Architektur liegt in ihrer Robustheit gegenüber großen Datenvolumina und unterschiedlichen Latenzanforderungen. Während der Batch-Layer für die Korrektheit und Vollständigkeit der Datenbasis sorgt, stellt der Speed-Layer sicher, dass auch zwischen zwei Batch-Läufen stets aktuelle Informationen verfügbar sind. Diese Architektur ist besonders für datenintensive Anwendungen geeignet, bei denen sowohl präzise historische Analysen als auch reaktive Echtzeitelemente erforderlich sind [WM15] [AB++15].

Allerdings bringt die Lambda-Architektur auch eine erhöhte Komplexität mit sich. Die parallele Pflege zweier Verarbeitungslogiken erfordert doppelten Entwicklungsaufwand und eine aufwendige Konsolidierung im Serving-Layer. Transformationen und Aggregationen müssen in beiden Pfaden konsistent umgesetzt und synchronisiert werden. Diese Faktoren machen die Implementierung ressourcenintensiv, weshalb sie häufig nur von größeren Organisationen mit entsprechenden Infrastruktur- und Betriebskapazitäten umgesetzt wird.

Trotz dieser Herausforderungen bleibt die Lambda-Architektur ein bewährter Ansatz, insbesondere dann, wenn bestehende batchbasierte Systeme durch Echtzeitfunktionen erweitert werden sollen, ohne Kompromisse bei der Datenqualität einzugehen.

Ein alternatives Architekturparadigma zur Lambda-Architektur stellt die sogenannte Kappa-Architektur dar, wie in Abbildung 5.7 dargestellt. Sie wurde mit dem Ziel entwickelt, die konzeptionellen und operativen Komplexitäten hybrider Systeme zu reduzieren. Dies gilt insbesondere für datenintensive Echtzeitanwendungen wie das Internet der Dinge (IoT), die industrielle Prozessüberwachung oder die Streaming-Analyse in Medienplattformen. Während die Lambda-Architektur zwei getrennte Verarbeitungswege vorsieht, verfolgt die Kappa-Architektur einen radikal vereinfachten Ansatz. Es besteht ein kontinuierlicher Streaming-Pfad, der sogenannte »Hot Path«, über den sämtliche Daten verarbeitet werden, unabhängig davon, ob es sich um Echtzeitdaten oder historische Ereignisse handelt [Kre14].

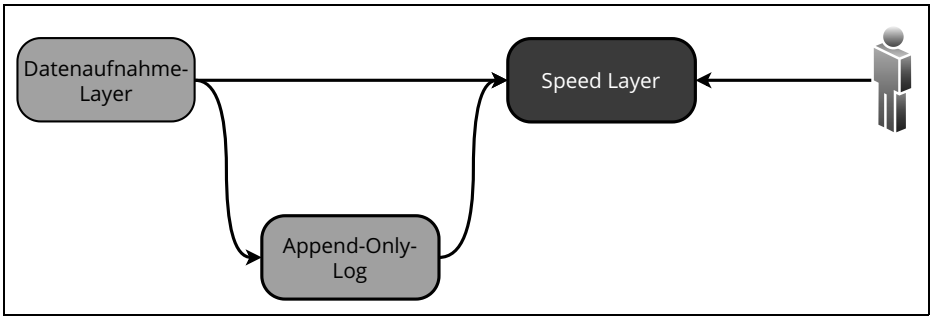

Abb. 5.7 *Schematischer Aufbau der Kappa-Architektur*

Im Zentrum der Kappa-Architektur steht ein Append-Only-Log, das alle eingehenden Daten in unveränderter Form dauerhaft speichert. Typischerweise kommen dafür Messaging-Systeme wie Apache Kafka [apache-kafka] oder Apache Pulsar [apache-pulsar] zum Einsatz, die als skalierbare, append-only Event-Logs fungieren. Die Unterstützung von Event-Replays erfolgt dabei primär über Topic-, Partition- und Offset-Mechanismen und setzt in der Praxis häufig zusätzliche Metadatenverwaltung sowie geeignete Orchestrierungsstrategien voraus, um gezielte oder teilselektive Replays zu ermöglichen [Kre14] [Kle17].

Dieses Log fungiert als zentrale Datenquelle für sämtliche nachgelagerten Verarbeitungsschritte. Historische Daten können bei Bedarf erneut eingespielt werden, etwa um Modelle neu zu trainieren oder vergangene Datenverläufe mit aktualisierten Analysekomponenten zu verarbeiten. Dadurch entfällt die Notwendigkeit einer separaten Batch-Infrastruktur.

Ein wesentlicher Vorteil dieses Ansatzes liegt in der konsistenten Umsetzung der Verarbeitungspfade. Da es keine doppelte Implementierung von Transformationslogik gibt, wie sie bei Lambda nötig ist, entstehen keine Inkonsistenzen zwischen Batch- und Streaming-Ergebnissen. Updates in der Verarbeitung oder Änderungen am Datenmodell lassen sich durch erneutes Abspielen des Logs gezielt nachvollziehen und reproduzieren. Dies ist besonders vorteilhaft in sensiblen Anwendungsfeldern wie Finanz- oder Gesundheitsdaten, wo Nachvollziehbarkeit und Konsistenz essenziell sind.

Darüber hinaus erlaubt die Kappa-Architektur eine schnelle und direkte Speicherung der verarbeiteten Daten in den Masterdatenbestand, wodurch kontinuierliche Aktualisierungen ohne Verzögerung möglich werden. Ihre geringe Komplexität im Vergleich zur Lambda-Architektur macht sie zudem leichter wartbar und reduziert den Entwicklungs- und Betriebsaufwand erheblich. Dies erklärt auch, warum Kappa insbesondere in sogenannten Cloud-nativen Architekturen eingesetzt wird, die auf zustandslose und skalierbare Dienste setzen.

Allerdings ist die Kappa-Architektur nicht uneingeschränkt vorteilhaft. Zwar entfällt die Komplexität einer hybriden Systemstruktur, doch bleiben strukturelle Herausforderungen bestehen. Anpassungen an Datenstrukturen und -schemata müssen weiterhin sorgfältig vorgenommen werden, nun allerdings im Rahmen der

Streaming-Logik. Zudem fehlt eine dedizierte Schicht für vorab berechnete, stark optimierte Analysen, wie sie im Serving-Layer der Lambda-Architektur vorgesehen ist. In Szenarien mit besonders hohen Anforderungen an Verarbeitungspersistenz, aggregierten Historienanalysen oder Performancetuning kann die Lambda-Architektur daher weiterhin die geeignetere Wahl darstellen.

Nicht zuletzt erfordert die Umsetzung der Kappa-Architektur bestimmte technologische Voraussetzungen. Die zugrunde liegenden Log-Systeme müssen in der Lage sein, auch sehr große Datenmengen über längere Zeiträume hinweg effizient vorzuhalten. Gleichzeitig bedarf es einer stabilen, fehlertoleranten Streaming-Infrastruktur, insbesondere in Bezug auf Zustandsverwaltung, Wiederanlaufmechanismen und die Resilienz bei temporären Ausfällen [PNM22].

Insgesamt eignet sich die Kappa-Architektur besonders für Systeme, die von Anfang an auf eventgetriebenem Datenfluss basieren, auf niedrige Latenzzeiten angewiesen sind und bei denen eine einfache Wartbarkeit im Vordergrund steht. Sie stellt eine moderne, skalierbare Alternative zur Lambda-Architektur dar, jedoch mit spezifischen Einschränkungen in Bezug auf analytische Tiefe und Flexibilität bei Datenrevisionen.

Ein weiteres zentrales Paradigma für moderne Datenpipelines stellt die ereignisgesteuerte Architektur (Event-Driven Architecture, kurz EDA, siehe Abbildung 5.8) dar. Im Kern basiert dieser Ansatz auf der Idee, dass sämtliche relevanten Systemereignisse, etwa das Eintreffen neuer Sensordaten, eine Nutzeraktion oder ein Statuswechsel, als eigenständige Events modelliert und verarbeitet werden. Diese Events fungieren als Auslöser für nachgelagerte Verarbeitungsschritte, die durch sogenannte Event-Handler oder spezialisierte Consumer-Komponenten übernommen werden [Fow17].

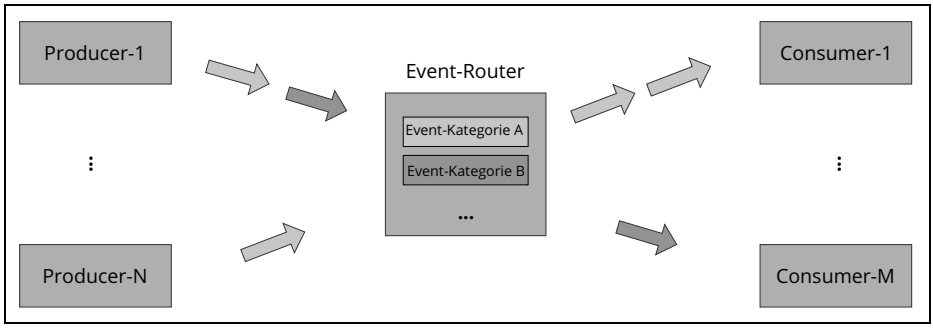

Abb. 5.8 *Schematischer Aufbau der EDA-Architektur*

Technologisch wird EDA in der Regel durch Messaging-Infrastrukturen realisiert, die eine lose gekoppelte, reaktive Softwarearchitektur ermöglichen. Zentrale Komponenten sind dabei der Event Producer, also das Gerät oder System, das das Ereignis auslöst; ein Event Router in Form eines Message-Brokers wie Apache Kafka, RabbitMQ oder Azure Event Hubs, der die Ereignisse aufnimmt und verteilt; sowie der Event Consumer, der das Event empfängt und darauf geeignet reagiert. Die zu-

grunde liegende Infrastruktur erlaubt es, Events unmittelbar nach ihrem Auftreten in einen sogenannten Event-Bus einzuspeisen. Dort werden sie asynchron an alle registrierten Konsumenten weitergeleitet, die daraufhin ihre spezifische Logik ausführen.

Diese Architekturform begünstigt niedrige Latenz, hohe Skalierbarkeit und eine modulare Entwicklung. Da alle Komponenten ausschließlich über den Event-Bus miteinander interagieren, ergibt sich eine technisch lose Kopplung auf Infrastrukturebene, die insbesondere in verteilten Systemlandschaften Vorteile bietet. Fachliche Abhängigkeiten zwischen Produzenten und Konsumenten bleiben jedoch bestehen, da nachgelagerte Komponenten inhaltlich von bestimmten Event-Typen, Datenstrukturen oder Qualitätsmerkmalen abhängig sein können.

Dennoch lassen sich Microservices, Datenverarbeitungseinheiten oder Analysekomponenten weitgehend unabhängig voneinander entwickeln, deployen und skalieren, sofern klare Event-Schemata und stabile fachliche Verträge eingehalten werden. Dies ist besonders attraktiv für Cloud-native Architekturen, domänenorientierte Datenplattformen (z. B. Data Mesh, Lakehouse, siehe Abschnitt 5.5) sowie Edge-Computing-Szenarien, bei denen nicht alle Event-Quellen zwangsläufig in der Cloud verortet sein müssen.

Auch moderne KI-Infrastrukturen wie Feature Stores, die Daten für das Training oder die Inferenz von Modellen in Echtzeit bereitstellen, greifen zunehmend auf EDA-Prinzipien zurück. Durch die eventgetriebene Verarbeitung kann eine konsistente, latenzarme und gleichzeitig skalierbare Datenversorgung gewährleistet werden, selbst unter hochdynamischen Bedingungen.

Natürlich bringt dieser Ansatz auch Herausforderungen mit sich: Das Design der Event-Schemata muss sorgfältig abgestimmt werden, um semantische Konsistenz zu gewährleisten. Zudem stellen Zustandsverwaltung, Fehlertoleranz sowie das Monitoring und Debugging verteilter Ereignisflüsse erhöhte Anforderungen an Architektur und Betrieb. Gerade in sicherheitskritischen oder regulatorisch anspruchsvollen Kontexten erfordert EDA daher ein durchdachtes Governance- und Kontrollmodell.

Neben den zuvor vorgestellten Ansätzen stellt die Data-Mesh-Architektur (siehe Abbildung 5.9) ein weiteres modernes Architekturparadigma dar, das insbesondere für datengetriebene Unternehmen mit wachsendem Skalierungsbedarf konzipiert wurde. Im Zentrum dieses Konzepts steht die Abkehr von einer zentralisierten Datenverwaltung hin zu einem dezentralen Modell, in dem Daten als domänenspezifische Produkte verstanden werden. Die Verantwortung für diese Datenprodukte liegt bei den jeweiligen Domänenteams, die nicht nur für die Bereitstellung, sondern auch für die Pflege, Qualitätssicherung und Dokumentation der Daten verantwortlich sind. Dabei werden die Produkte über klar definierte Schnittstellen und Service-Level-Agreements organisiert, wodurch ein hohes Maß an Verlässlichkeit und Nachvollziehbarkeit erreicht werden soll.

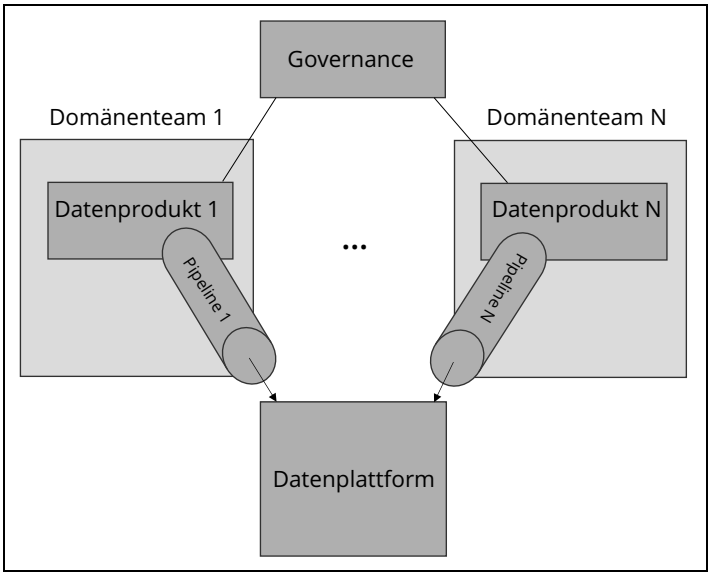

Abb. 5.9 *Schematischer Aufbau der Data-Mesh-Architektur*

Diese Architektur besteht aus domänenspezifischen Datenprodukten, die in fachlicher Verantwortung betrieben und über jeweils eigene Pipelines an eine gemeinsame Datenplattform angebunden sind. Die Plattform stellt die Basisdienste für Speicherung, Verarbeitung und Zugriffskontrolle bereit und dient als technische Drehscheibe zwischen den Produkten. Oberhalb spannt ein systematischer Governance-Rahmen den organisatorischen Überbau: Er definiert Standards, Sicherheitsrichtlinien und ein konsistentes Metadaten- bzw. Katalogmanagement und stellt damit Interoperabilität, SLAs und regulatorische Konformität sicher [Deh22].

Zu den wesentlichen Vorteilen der Data-Mesh-Architektur zählen ihre hohe Skalierbarkeit durch dezentrale Verantwortlichkeiten sowie die Förderung von Innovation und Agilität, da Entscheidungen näher an den jeweiligen Fachdomänen getroffen werden können. Gleichzeitig geht dieses Paradigma mit bestimmten Herausforderungen einher: Der Koordinationsaufwand verlagert sich von der operativen Datennutzung hin zu frühen Phasen der Datenproduktdefinition, Standardisierung und Governance. Diese zusätzliche Verantwortung in frühen Phasen stellt keinen Widerspruch zur angestrebten Agilität dar, sondern bildet deren Voraussetzung, indem sie spätere operative Abhängigkeiten reduziert.

Domänenteams müssen mehr Verantwortung in die klare Beschreibung, Dokumentation und Qualitätssicherung ihrer Datenprodukte investieren, um eine spätere Nutzung ohne laufende Kommunikations- und Abstimmungsprozesse zu ermöglichen. Darüber hinaus erfordert die Etablierung eines produktorientierten, verteilten Datenmanagements einen kulturellen Wandel innerhalb der Organisation, weg von zentraler Kontrolle hin zu klar definierter, domänenbasierter Verantwortung.

Da Data Mesh nicht nur ein organisatorisches und strukturelles Paradigma, sondern zugleich auch eine Strategie zur unternehmensweiten Datenhaltung und -speicherung darstellt, wird in Abschnitt 5.5 auch auf die speicherbezogene Perspektive dieses Architekturansatzes näher eingegangen.

Die Auswahl eines geeigneten Architekturparadigmas für die Datenpipeline sollte dabei stets im Einklang mit den fachlichen Anforderungen, den technischen Rahmenbedingungen und den betrieblichen Zielsetzungen des jeweiligen KI-Systems erfolgen. Nur so lässt sich sicherstellen, dass die Datenpipeline nicht nur funktional leistungsfähig, sondern auch langfristig wartbar, erweiterbar und zukunftsfähig ist.

5.3.3 Relevante Werkzeuge für Datenpipelines

Für den Aufbau effizienter Datenpipelines hat sich ein Ökosystem komplementärer Werkzeuge etabliert. Auf der Transformations- und Modellierungsebene adressiert dbt (Data Build Tool) [dbt-labs] SQL-basierte ELT-Workflows nach dem Ladeprozess. Apache Beam [GooBeam2024] bietet ein einheitliches Programmiermodell für Batch- und Streaming-Verarbeitung und abstrahiert von der jeweiligen Ausführungsumgebung. Für datennahe Arbeiten und Prototyping stehen Pandas/Dask [McK11] [Roc15] bereit; Dask skaliert dabei das aus Pandas bekannte API-Paradigma auf größere Datenmengen und Cluster. Im Schwerpunkt dieser Darstellung stehen Apache Spark [ZX++16] [apache-spark] und Apache Flink [CK++15] [apache-flink] als zentrale, produktionsreife Engines für verteilte Datenverarbeitung, auf deren Eigenschaften und Einsatzszenarien im Folgenden vertieft eingegangen wird.

Apache Spark

Apache Spark ist ein Open-Source-Framework für verteilte Datenverarbeitung und bildet eine generische Rechen-Engine, die große Datenmengen parallel über viele Cluster-Knoten hinweg verarbeitet. Es abstrahiert die zugrunde liegende Infrastruktur und stellt ein einheitliches Programmiermodell bereit, mit dem Daten aus lokalen Clustern, Kubernetes oder Cloud-Umgebungen geladen, transformiert, analysiert und modelliert werden können. Ursprünglich an der UC Berkeley entwickelt, zählt Spark heute zu den zentralen Werkzeugen im Big-Data- und Data-Engineering-Ökosystem.

Im Kern basiert Spark auf dem Konzept der Resilient Distributed Datasets (RDDs), also fehlertoleranten, verteilten Datensammlungen. Diese werden im Arbeitsspeicher gehalten und erlauben dadurch eine deutlich schnellere Verarbeitung als klassische Hadoop-Batch-Jobs, die nach jedem Schritt auf Festplatten schreiben. Spark ermöglicht damit In-Memory-Verarbeitung in großem Maßstab, wodurch interaktive Abfragen und iterative Berechnungen insbesondere für ML-Algorithmen effizient ausgeführt werden können. In Datenpipelines wird Apache Spark häufig als zentrale Verarbeitungs-Engine für skalierbare Datenanalysen eingesetzt. Nach der Akquise aus unterschiedlichen Quellen, beispielsweise relationalen Datenbanken, REST-APIs, IoT-Sensoren oder Messaging-Systemen wie Apache Kafka [apache-kaf-

ka], werden die Daten in Spark-Cluster geladen. Anschließend erfolgt die Bereinigung, Transformation und Aggregation der Rohdaten mit Modulen wie Spark SQL oder PySpark, die SQL-ähnliche und Python-basierte Operationen auf großen Datenbeständen ermöglichen. Für Echtzeitanalysen steht Structured Streaming bereit, das eingehende Datenströme kontinuierlich verarbeitet.

Die verarbeiteten Daten werden typischerweise in zentralen Speichersystemen wie HDFS, Amazon S3 oder in Lakehouse-Formaten wie Delta Lake, Apache Iceberg oder Hudi persistiert, die ACID-Transaktionen und die Kombination von Batch- und Streaming-Workloads unterstützen [AD++20] [AG++21] [Ice2024] [apache-hudi]. Anschließend können sie von Analyse- und Machine-Learning-Werkzeugen weiterverwendet werden, etwa für Dashboards (z. B. Tableau, Power BI) oder für Modelltraining über Spark MLlib.

Zur Steuerung der gesamten Pipeline kommen häufig Orchestrierungs-Frameworks wie Apache Airflow, Dagster oder Prefect zum Einsatz, die Aufgabenplanung, Fehlerbehandlung und Monitoring übernehmen. Dank seiner verteilten In-Memory-Architektur, der einheitlichen Unterstützung von Batch- und Streaming-Verarbeitung und der Integration mit unterschiedlichen APIs (Python, Scala, SQL) bietet Apache Spark eine skalierbare Rechenbasis, die sowohl klassische ETL-Prozesse als auch interaktive Analysen und Echtzeitanwendungen effizient unterstützt.

Apache Flink

Apache Flink ist ein Open-Source-Framework (Apache-Lizenz 2.0) zur verteilten, zustandsbehafteten Datenverarbeitung mit klarem Schwerpunkt auf Echtzeit-Streaming. Im Gegensatz zu klassischen ETL-Werkzeugen, die überwiegend auf Batch-Verarbeitung basieren, ist Flink für die kontinuierliche Verarbeitung von Datenströmen konzipiert. Das Framework berücksichtigt neben der Verarbeitungszeit auch den tatsächlichen Zeitpunkt des Ereignisses (Event Time) und kann Datenströme trotz Netzwerkverzögerungen oder verspäteter Ankunft in korrekter zeitlicher Reihenfolge auswerten. In Verbindung mit genau einmaliger Verarbeitung (Exactly-Once), bei der jede Nachricht oder jedes Ereignis genau ein einziges Mal verarbeitet und gespeichert wird, eignet sich Flink für Szenarien, in denen Aktualität, korrekte Ereignisreihenfolgen und komplexe Zustandsverwaltung entscheidend sind, etwa bei der Betrugserkennung, der Analyse von IoT-Datenströmen oder interaktiven Dashboards.

Eine typische Datenpipeline mit Apache Flink beginnt mit der Anbindung externer Quellen (z. B. Apache Kafka, Amazon Kinesis, relationale Datenbanken, Dateisysteme oder REST). Die eingehenden Daten werden als Streams verarbeitet; Aggregationen, Joins und Mustererkennung laufen kontinuierlich und zustandsbehaftet. Konsistente Zustände werden über Checkpoints/Savepoints und fehlertolerante Wiederanlaufmechanismen gewährleistet, sodass die Verarbeitung auch bei Ausfällen korrekt fortgesetzt wird.

Für das Zustandsmanagement nutzt Apache Flink eine abstrahierte Backend-Architektur, die sich je nach Anwendungsszenario anpassen lässt. Der Zustand

kann lokal im Speicher gehalten, beispielsweise über RocksDB persistiert oder in verteilten Speichersystemen abgelegt werden. Diese Mechanismen sind zudem essenziell für Upgrade-Strategien, Rolling Deployments und langfristige Wartbarkeit produktiver Systeme.

Die verarbeiteten Daten werden im Anschluss in unterschiedlichste Zielsysteme geschrieben. Häufig erfolgt dies in verteilte Speichersysteme wie Data Lakes, NoSQL-Datenbanken (z.B. Apache Cassandra, HBase) oder spezialisierte Analyse-Plattformen wie Apache Druid oder Elasticsearch. Auch eine direkte Anbindung an klassische Data Warehouses oder externe Systeme über REST-Interfaces ist möglich. Darüber hinaus lässt sich Apache Flink nahtlos in Lambda- oder Kappa-Architekturen (siehe Abschnitt 5.3.2) integrieren, in denen sowohl Batch- als auch Streaming-Datenquellen in einheitlicher Streaming-Semantik zusammengeführt werden.

Zur Anwendungsentwicklung stellt Apache Flink sowohl eine flexible Low-Level-API (Datastream-API) als auch eine deklarative, SQL-basierte API (Flink SQL) bereit. Letztere erlaubt die einfache Formulierung kontinuierlicher Abfragen und senkt die Einstiegshürden für Nutzer ohne tiefgreifende Programmierkenntnisse. Moderne Flink-Deployments erfolgen zunehmend containerisiert in Kubernetes-Umgebungen, häufig orchestriert durch Plattformen wie Ververica Platform, Apache Beam oder cloudbasierte Dienste großer Anbieter.

Ähnlich wie bei Apache Spark kommen externe Workflow-Orchestrierungstools wie Apache Airflow, Dagster oder Argo Workflows für die übergreifende Steuerung und Automatisierung der Pipeline-Schritte zum Einsatz. Sie ermöglichen eine zeit- oder ereignisbasierte Ausführung, die Integration mit weiteren Systemkomponenten sowie das zentrale Monitoring der Datenverarbeitung.

Insgesamt bietet Apache Flink eine tragfähige Grundlage für datenintensive, zustandsbehaftete Streaming-Anwendungen. Mit seiner starken Konsistenzgarantie, Echtzeitfähigkeit und hohen Skalierbarkeit ist Flink insbesondere für Systeme geeignet, die dynamisch auf kontinuierlich eingehende Daten reagieren müssen, etwa in der industriellen Prozessüberwachung, im Finanzwesen oder in komplexen IoT-Infrastrukturen.

5.3.4 Architektonische Einordnung: Datenpipelines und Datenarchitekturen

Datenpipelines und Datenarchitekturen bilden das strukturelle Rückgrat KI-basierter Softwaresysteme. Sie bestimmen, wie Daten erhoben, verarbeitet, versioniert und bereitgestellt werden, und beeinflussen damit unmittelbar Leistungsfähigkeit, Skalierbarkeit und Wartbarkeit der Gesamtlösung. Unzureichend abgestimmte Pipeline-Architekturen führen häufig zu instabilen Datenflüssen, schwer nachvollziehbaren Abhängigkeiten und erhöhtem Betriebsaufwand.

Für Softwarearchitekten ist insbesondere relevant, dass Datenpipelines nicht isoliert betrachtet werden dürfen, sondern eng mit MLOps-Prozessen, Speicherarchitekturen und Anwendungskomponenten verzahnt sind. Architekturentscheidun-

gen zu Verarbeitungsmodellen, Integrationsmechanismen und Orchestrierung wirken sich langfristig auf Betrieb, Erweiterbarkeit und Fehlertoleranz aus.

Die Architektur muss daher sicherstellen, dass Datenpipelines modular, skalierbar und transparent aufgebaut sind. Gleichzeitig sind Anforderungen an Reproduzierbarkeit, Fehlertoleranz, Monitoring und Governance systematisch in das Pipeline-Design zu integrieren. Für die Architektur ergeben sich insbesondere folgende Anforderungen:

Aspekt	**Bedeutung für die Architektur**
Datenfluss	Klare Steuerung von Ingestion und Transformation
Verarbeitung	Abstimmung von Batch- und Streaming-Modellen
Betrieb	Monitoring, Backpressure-Handling, Recovery
Skalierung	Elastische Ressourcen- und Laststeuerung
Wartbarkeit	Austauschbare Pipeline-Komponenten
Governance	Dokumentierte Verarbeitungslogik und Lineage
Risiko	Pipeline-Brüche, Inkonsistenzen, Datenverlust
Typische Fehlannahmen	
»Eine Pipeline lässt sich später leicht umbauen.«	
»Batch- und Streaming-Verarbeitung können beliebig kombiniert werden.«	
»Monitoring ist nur für Infrastruktur relevant.«	
»Pipeline-Fehler lassen sich manuell korrigieren.«	
»Datenverarbeitung ist unabhängig vom Anwendungskontext.«	

Tab. 5.2 *Architectural Impact Box: Datenpipelines und -architekturen*

Diese Fehlannahmen führen häufig zu steigender Komplexität, eingeschränkter Skalierbarkeit und hohem Betriebsaufwand.

5.4 Strategien zur Datenaufbereitung

Eine saubere, konsistente und semantisch sinnvolle Datenbasis ist die Grundlage für den Aufbau effizienter, robuster und vertrauenswürdiger KI-Systeme. Um diese Qualität der Daten sicherzustellen, ist eine zielgerichtete und methodisch fundierte Aufbereitung der erhobenen Daten notwendig. In den folgenden Abschnitten werden zentrale Strategien zur Datenaufbereitung vorgestellt, die sich in der Praxis bewährt haben.

5.4.1 Strategien für Datenbereinigung

Die Datenbereinigung ist ein zentraler Schritt der Aufbereitung mit dem Ziel, die Qualität der Eingabedaten zu sichern und damit Leistungsfähigkeit und Verlässlichkeit von KI-Modellen zu erhöhen. Fehlerhafte, fehlende oder unplausible Werte können zu Verzerrungen führen und das Training beeinträchtigen; entsprechend müssen sie systematisch erkannt, bewertet und behandelt werden, ohne die statistische Struktur der Daten unnötig zu verfälschen. Im Folgenden werden die wesentlichen Handlungsfelder in sinnvoller Bearbeitungsreihenfolge dargestellt.

Zum einen betrifft dies Redundanzen und Duplikate: Sie erhöhen das Overfitting-Risiko, weil identische Muster mehrfach gelernt werden, und können durch die Überrepräsentation einzelner Klassen oder Merkmale zu einer verzerrten Modelllogik führen. Daher empfiehlt es sich, solche Einträge frühzeitig zu identifizieren, zu entfernen und bei Bedarf zu konsolidieren. Zugleich ist zu prüfen, ob scheinbar doppelte Instanzen tatsächlich redundant sind oder legitime Wiederholungen desselben Phänomens darstellen, etwa bei hochfrequenten Zeitreihen; diese Klärung bildet die Grundlage für die nachfolgenden Schritte.

Darauf aufbauend schließt sich der Umgang mit fehlenden Werten (sog. Missing Values) an. Abhängig von Umfang, Verteilung und semantischer Bedeutung können unvollständige Zeilen oder Merkmale gelöscht werden, sofern kein relevanter Informationsverlust entsteht. Trägt das Fehlen selbst Information, ist eine explizite Kennzeichnung (z. B. »NaN«) zweckmäßig. Wo vollständige Datensätze erforderlich sind, bietet sich die Imputation fehlender Werte an. Hierfür können einfache Verfahren wie Mittelwert, Median oder Modus ebenso eingesetzt werden wie k-Nächste-Nachbarn- und Regressionsansätze sowie multiple Imputation. Bei zeitabhängigen Daten kommen zudem Interpolationsverfahren wie Vorwärts- oder Rückwärtsfüllung zum Einsatz. Ziel ist es, die Daten für nachgelagerte Schritte zu vervollständigen und gleichzeitig die Verteilungscharakteristika möglichst zu erhalten. Die zugrunde liegenden Verfahren und ihre Auswirkungen auf Datenverteilungen und Modellqualität wurden bereits in Abschnitt 4.3.2 detailliert erläutert und werden an dieser Stelle nicht erneut ausgeführt.

Schließlich ist die Behandlung von Ausreißern zu adressieren, da diese nach der Bereinigung von Redundanzen und fehlenden Werten besonders deutlich hervortreten. Ausreißer, also Werte weit außerhalb des typischen Bereichs, können auf Messfehler, seltene Ereignisse oder reale Extremzustände zurückgehen. Sie werden unter anderem über Z-Score oder Interquartilsabstand identifiziert und je nach Kontext entfernt, korrigiert oder durch Transformationen (z. B. Logarithmierung oder Wurzelbildung) in ihrer Wirkung abgeschwächt. In sicherheitskritischen Anwendungen ist zudem zu prüfen, ob vermeintliche Ausreißer sicherheitsrelevante Randfälle abbilden, die für Robustheitstests ausdrücklich erhalten bleiben sollten.

5.4.2 Strategien für Datenaggregation

Da in KI-Entwicklungsprojekten Daten häufig aus unterschiedlichen, voneinander unabhängigen Quellen stammen, weisen sie in der Regel heterogene Formate, Strukturen und Granularitätsstufen auf. Um einen konsolidierten und analysierbaren Datenbestand zu schaffen, müssen diese Datenquellen vereinheitlicht und zusammengeführt werden, ein Prozess, der als Datenaggregation bezeichnet wird.

Ein zentrales Element der Datenaggregation ist die Wahl geeigneter Join-Strategien zur Verknüpfung strukturierter oder semi-strukturierter Daten, beispielsweise aus relationalen Tabellen, Logdateien oder Sensordatenströmen. Häufig kommen dabei Inner Joins zum Einsatz, bei denen nur jene Datensätze kombiniert werden, für die in beiden Quellen ein gemeinsamer Schlüssel existiert. Diese Methode reduziert Inkonsistenzen, kann jedoch relevante, aber nur einseitig vorhandene Informationen ausschließen. Outer Joins hingegen führen alle Datensätze aus beiden Quellen zusammen und markieren fehlende Werte typischerweise mit NULL, um Vollständigkeit sicherzustellen. Für zeitlich aufgelöste Daten wird zusätzlich häufig ein zeitbasierter Join verwendet, bei dem Datensätze anhand eines Zeitfensters synchronisiert werden. Diese Technik ist insbesondere in domänenübergreifenden Anwendungen mit asynchronen Datenquellen von Bedeutung, etwa bei der Kombination von Sensor- und Kontextdaten, bei denen Sensordaten (z. B. Kamerabilder, GPS, Lidar) kontinuierlich, Kontextdaten (z. B. Wetter, Verkehrsfluss oder Karteninformationen) jedoch in anderen zeitlichen Takten erhoben werden. Ein zeitbasierter Join sorgt hier dafür, dass Messwerte aus unterschiedlichen Quellen einem konsistenten Zeitpunkt oder Ereignis zugeordnet werden und somit eine gemeinsame Datengrundlage für die Modellbildung entsteht.

Nach der erfolgreichen Synchronisation der Daten stellt sich die Frage, auf welcher Granularitätsstufe die zusammengeführten Informationen weiterverarbeitet werden sollen. Diese Entscheidung ist wesentlich, da die Rohdaten oft in sehr unterschiedlicher Auflösung oder Detailtiefe vorliegen. Für KI-Modelle ist es entscheidend, eine geeignete Aggregationsstufe zu wählen. Zeitreihendaten lassen sich beispielsweise auf Intervallebene zusammenfassen, um Mittelwerte, Maxima oder Häufigkeiten zu berechnen. In räumlichen Szenarien etwa im Mobilitäts- oder Umweltbereich, kann zudem eine Aggregation nach geografischen Einheiten wie Regionen, Stadtteilen oder Postleitzahlen sinnvoll sein. Alternativ ermöglicht eine entitätsbezogene Aggregation, etwa auf Objekt-, Fahrzeug- oder Projektebene, eine domänenspezifische Verdichtung der Daten. Die gewählte Aggregationsstufe beeinflusst sowohl die Modellkomplexität als auch die Menge der Trainingsdaten: Feingranulare Daten erlauben differenziertere Modelle, erhöhen aber den Rechenaufwand und das Risiko von Datenrauschen.

Ein oft unterschätzter Aspekt der Datenaggregation ist die Konfliktauflösung bei inkonsistenten oder redundanten Informationen. Verschiedene Quellen enthalten häufig widersprüchliche Angaben zu denselben Attributen. In solchen Fällen sind transparente und nachvollziehbare Auflösungsstrategien erforderlich. Eine

Möglichkeit besteht in der Definition von Prioritätsregeln, bei denen eine Quelle als besonders vertrauenswürdig eingestuft wird. Alternativ können heuristische Verfahren wie Latest Timestamp, Majority Vote oder statistische Kennzahlen (z. B. Median, Mittelwert) zur Entscheidung herangezogen werden. Bei numerischen Abweichungen bieten sich probabilistische Ansätze an, die die Plausibilität einzelner Werte bewerten. Zusätzlich kann domänenspezifisches Wissen, etwa in Form definierter Toleranzgrenzen oder betrieblicher Regeln, dazu beitragen, fehlerhafte Werte frühzeitig zu identifizieren und auszuschließen.

Unabhängig von der gewählten Strategie ist es essenziell, dass alle Mechanismen zur Aggregation und Konfliktlösung dokumentiert, reproduzierbar und konsistent implementiert sind. Sie beeinflussen unmittelbar die Datenqualität und damit die Modellgüte, Generalisierungsfähigkeit und Validierbarkeit der resultierenden KI-Systeme.

5.4.3 Strategien für Datenstandardisierung

Die Datenstandardisierung ist ein weiterer zentraler Aspekt der Datenaufbereitung und zielt darauf ab, syntaktische wie semantische Konsistenz im gesamten Datenbestand sicherzustellen. Gerade bei großen, heterogenen Datensätzen treten häufig inkonsistente oder fehlerhafte Einträge auf, verursacht durch uneinheitliche Formatierungen, manuelle Eingabefehler oder abweichende Konventionen, etwa bei Zeitangaben, Maßeinheiten, Klassifikationen oder Codierungen [RD00] [RH22].

Zur Harmonisierung empfiehlt sich ein systematischer Ansatz aus Normalisierung und Standardisierung. Dazu gehören einheitliche Formate und Wertebereiche (z. B. ISO-konforme Datumsangaben, standardisierte Einheiten für Temperatur, Länge oder Währung) sowie die Definition und Durchsetzung verbindlicher Datenstandards mittels kontrollierter Vokabulare, Referenzkataloge oder strukturierter Klassifikationssysteme (z. B. Ländercodes, Produkttypen, medizinische Terminologien).

Für die technische Umsetzung stehen u. a. reguläre Ausdrücke zur Formatvalidierung, Lookup-Tabellen zur Umcodierung bekannter Varianten und Schemata bzw. Constraints zur Prüfung zulässiger Wertebereiche, Datenformate und Pflichtfelder zur Verfügung. Ergänzend sind Plausibilitätsprüfungen über mehrere Felder hinweg erforderlich, um widersprüchliche Einträge zu erkennen (z. B. »digitales« Produkt mit physischer Versandadresse). In sensiblen Bereichen wie Adressdaten, medizinischen Attributen oder Produktkategorien ist zudem der Abgleich mit externen Referenzdatenbanken (ISO-Normen, Postleitzahlverzeichnisse, medizinische Klassifikationen) sinnvoll, um die Datenqualität langfristig zu sichern.

5.4.4 Strategien für Datentransformation

Die Datentransformation bildet die Brücke zwischen aufbereiteten Rohdaten und den für das Training erforderlichen Eingabeformaten. Während Abschnitt 4.6 die grundlegenden Prinzipien und gängigen Verfahren skizziert hat, richtet sich der Fo-

kus hier auf die strategische Planung, die reproduzierbare Umsetzung und die Bewertung der Transformationsschritte innerhalb konsistenter Datenpipelines.

Ausgangspunkt ist ein einheitliches Transformationsschema, das festlegt, welche Merkmale wie und in welcher Reihenfolge verarbeitet werden. Dazu zählen u. a. Formatkonvertierungen (z. B. ISO-konforme Zeitstempel), die Skalierung numerischer Variablen oder die Codierung kategorialer Merkmale. Diese Schritte sollten modular aufgebaut, automatisiert und versionsgeführt sein (ETL/ELT-Jobs, parametrische Preprocessing-Pipelines), um Wiederholbarkeit, Austauschbarkeit und Tests zu ermöglichen.

Die Auswahl der konkreten Methoden erfolgt modell- und datenabhängig und wird empirisch abgesichert. Lineare Modelle profitieren in der Regel stärker von normierten Skalen als baumbasierte Verfahren; neuronale Netze benötigen konsistente Wertebereiche und profitieren oft von kontinuierlichen Einbettungen. Anstelle pauschaler Standardrezepte (z. B. »immer One-Hot, immer StandardScaler«) werden Verfahren anhand von Validierungsergebnissen, Ressourcenbedarf (Speicher/Runtime) und Erhalt der semantischen Aussagekraft gewählt und dokumentiert.

Feature Engineering erweitert den Datenraum gezielt und hypothesenbasiert. Abgeleitete zeitliche, räumliche oder semantische Variablen werden nicht ad hoc eingeführt, sondern entlang klarer Annahmen entworfen und über definierte Metriken (z. B. Δ-AUC, Δ-RMSE, Kalibrierung) bewertet. So wird sichergestellt, dass jede Transformation einen messbaren Beitrag zur Modellgüte leistet und nicht unnötig Komplexität oder Rauschen einführt.

Schließlich ist Datentransformation kein einmaliger Vorbereitungsschritt, sondern ein iterativer Bestandteil des MLOps-Lebenszyklus. Änderungen an Datenquellen, Merkmalsdefinitionen oder Modellarchitekturen fließen kontinuierlich zurück in die Pipeline. Versionierung (Code, Daten, Artefakte), Data-Lineage, Tests (Unit/Property-based Tests) und Drift-Monitoring sichern Nachvollziehbarkeit, Validierbarkeit und langfristige Betriebssicherheit der Modelle.

5.4.5 Strategien für Datenanreicherung und -augmentierung

Auch wenn in vielen KI-Projekten bereits Datensätze vorliegen, sind sie für ein leistungsfähiges, robustes und gut generalisierbares Modell häufig zu klein oder zu wenig vielfältig, insbesondere bei datenintensiven Verfahren (z. B. tiefen neuronalen Netzen) oder hoher Szenariovariabilität. In solchen Fällen werden gezielte Maßnahmen zur Datenanreicherung und -augmentierung eingesetzt. Als abschließender Schritt der Datenaufbereitung erhöhen sie Informationsgehalt und Vielfalt der Datenbasis und erweitern den Kontext der vorhandenen Daten, im Unterschied zu den vorangehenden Schritten, die primär Konsistenz, Korrektheit und Standardisierung sicherstellen.

Ein strategischer Ansatz zur Datenanreicherung besteht in der Integration zusätzlicher Informationsquellen, die den Kontext bestehender Daten präzisieren. Dazu zählen etwa Umwelt-, Sensordaten, sozioökonomische Indikatoren oder geo-

grafische Attribute. Entscheidend ist hierbei, dass neue Quellen nicht beliebig hinzugefügt, sondern hinsichtlich ihrer Relevanz, Konsistenz und Kompatibilität geprüft werden. Eine zu starke Heterogenität kann sonst zu Rauschen und Modellverzerrungen führen. Daher sollte jede Erweiterung auf einer Hypothese beruhen, welcher zusätzliche Kontext tatsächlich zu einer messbaren Verbesserung der Modellleistung beiträgt.

Neben der Integration externer Quellen können bestehende Daten durch gezielte Ableitung neuer Merkmale systematisch erweitert werden. Derartige Transformationen sind Bestandteil eines geplanten Feature-Engineering-Prozesses, der in Abschnitt 4.4.4 bereits beschrieben wurde. Im Kontext der Anreicherung steht dabei weniger die technische Umsetzung einzelner Features im Vordergrund, sondern deren funktionaler Beitrag zu einem reichhaltigeren, semantisch vollständigen Datenraum.

Die Datenaugmentierung verfolgt ergänzend das Ziel, künstliche Vielfalt zu erzeugen, um die Robustheit eines Modells gegenüber Variationen in der Eingabe zu erhöhen. Dabei werden neue Dateninstanzen synthetisch generiert oder bestehende modifiziert. Diese Verfahren sind insbesondere in Bereichen mit hohen Erfassungs- oder Beschaffungskosten von Daten relevant. Entscheidend ist in der Praxis weniger die Aufzählung einzelner Transformationen als vielmehr die Abstimmung der Augmentierungslogik auf den intendierten Modellzweck, beispielsweise darauf, ob natürliche Störungen simuliert oder gezielt Randfälle erzeugt werden sollen. Zur Qualitätssicherung sollte jede Augmentierungsstrategie validiert werden, etwa über Kreuzvalidierung oder gezielte Robustheitstests.

In datenarmen Szenarien kann zudem Domänenadaption eine zentrale Rolle spielen. Sie ermöglicht es, bereits vortrainierte Modelle aus verwandten Domänen zu übernehmen und durch kontrollierte Anpassung (Finetuning) an die Zielumgebung zu übertragen. Im Rahmen der Datenaufbereitung betrifft dies vor allem die systematische Auswahl und Transformation der Trainingsdaten, um den Übergang zwischen Quell- und Zieldomäne zu unterstützen.

Insgesamt dienen Datenanreicherung und -augmentierung nicht primär der quantitativen Vergrößerung des Datenbestands, sondern der qualitativen Erweiterung des Merkmalsraums. Strategisch eingesetzt verbessern sie die Ausdrucksstärke, Robustheit und Generalisierungsfähigkeit von KI-Modellen und bilden damit eine zentrale Grundlage für belastbare und praxisnahe Anwendungen.

5.4.6 Architektonische Einordnung: Datenaufbereitung

Die Datenaufbereitung ist ein zentraler architekturrelevanter Bestandteil von KI-Systemen, da sie die semantische Struktur der Eingabedaten und damit unmittelbar das Verhalten der trainierten Modelle bestimmt. Unsystematisch definierte Bereinigungs-, Aggregations- und Transformationsprozesse führen häufig zu schwer nachvollziehbaren Modellentscheidungen, eingeschränkter Reproduzierbarkeit und erhöhtem Wartungsaufwand.

Für Softwarearchitekten ist insbesondere relevant, dass Aufbereitungslogik nicht als Ad-hoc-Vorverarbeitung implementiert werden darf, sondern als versionierbarer, testbarer und dokumentierter Bestandteil der Softwarearchitektur zu gestalten ist. Änderungen an Aufbereitungsregeln wirken sich oft stärker auf das Systemverhalten aus als Änderungen an Modellparametern.

Die Architektur muss daher sicherstellen, dass Datenaufbereitung deterministisch, reproduzierbar und transparent umgesetzt wird. Gleichzeitig sind Qualitätsanforderungen, Validierungsmechanismen und Governance-Vorgaben systematisch in die Aufbereitungsprozesse zu integrieren.

Für die Architektur ergeben sich insbesondere folgende Anforderungen:

- Versionierung und Dokumentation aller Aufbereitungs- und Transformationsregeln
- Modularisierung von Bereinigungs-, Aggregations- und Feature-Prozessen
- Integration automatisierter Qualitäts- und Plausibilitätsprüfungen
- Sicherstellung konsistenter Transformationen für Training und Inferenz
- Unterstützung reproduzierbarer Preprocessing-Pipelines
- Nachvollziehbare Bewertung von Feature-Engineering-Entscheidungen

Aspekt	**Bedeutung für die Architektur**
Datenqualität	Systematische Bereinigung und Validierung
Transformation	Konsistente und versionierte Preprocessing-Logik
Reproduzierbarkeit	Deterministische Verarbeitungspipelines
Betrieb	Überwachung von Verteilungs- und Feature-Drift
Wartbarkeit	Austauschbare Feature- und Transformationsmodule
Governance	Dokumentation und Prüfpfade
Risiko	Informationsverlust, Bias, Overfitting
Typische Fehlannahmen	
»Datenaufbereitung ist reine Vorarbeit für das Training.«	
»Transformationen müssen nicht dokumentiert werden.«	
»Feature Engineering ist ein kreativer Einzelschritt.«	
»Kleine Änderungen haben keine systemischen Auswirkungen.«	
»Preprocessing unterscheidet sich nicht zwischen Training und Betrieb.«	

Tab. 5.3 *Architectural Impact Box: Datenaufbereitung in KI-Systemen*

Diese Annahmen führen häufig zu inkonsistenten Modellergebnissen, erschwerter Wartung und eingeschränkter Nachvollziehbarkeit.

5.5 Konzeption der Datenarchitekturen und Datenspeicherung

In den vorherigen Abschnitten wurden zentrale Aspekte der Datenbereitstellung und -verarbeitung behandelt. Ein weiterer essenzieller Bestandteil des Datenmanagements bei der Entwicklung von KI-Systemen ist die Datenspeicherung bzw. -persistenz. Dabei können Daten in unterschiedlichen Formaten und über verschiedene Speichertechnologien hinweg verteilt abgelegt werden.

5.5.1 Datenarchitekturen und Technologien zur Datenspeicherung

Ein häufig verwendetes, aber einfaches Speicherformat ist die Ablage in CSV-Dateien. Diese zeilen- oder spaltenorientierten Textformate sind leicht lesbar und universell kompatibel, was sie für den Austausch und die schnelle Inspektion kleiner bis mittelgroßer tabellarischer Datensätze geeignet macht. Für komplexe, verschachtelte oder hierarchische Strukturen stoßen sie jedoch rasch an Grenzen. Insbesondere die fehlende Typisierung und eingeschränkte Metadatenunterstützung erschweren eine skalierbare und konsistente Weiterverarbeitung in datenintensiven KI-Projekten.

Für strukturierte und persistente Speicherung größerer Datenmengen kommen relationale Datenbanksysteme (RDBMS) wie MySQL oder PostgreSQL zum Einsatz. Diese basieren auf festen Schemata und ermöglichen durch Transaktionen, Schlüsselbeziehungen und Integritätsbedingungen eine zuverlässige Verwaltung strukturierter Daten mit klar definierten Entitäten und Attributen. In KI-Systemen finden relationale Datenbanken häufig Anwendung bei der Speicherung von Labeldaten, Metadaten aus Experimenten oder annotierten Datensätzen, da sie durch Joins, Indizes und Abfragen eine effiziente und konsistente Datenverarbeitung gewährleisten.

Im Gegensatz dazu bieten NoSQL-Datenbanken mehr Flexibilität bei sich verändernden oder heterogenen Datenstrukturen. Dokumentenbasierte Systeme wie MongoDB, Cassandra und Dynamo speichern Daten in JSON-ähnlichen Formaten mit dynamischem Schema. Dadurch eignen sie sich besonders für semi-strukturierte oder unstrukturierte Daten, etwa zur Verarbeitung von Text-, Sensor- oder Multimediadaten. Graphdatenbanken, wie bspw. Neo4j sind wiederum speziell für netzwerkartige Beziehungen zwischen Entitäten konzipiert und ermöglichen eine effiziente Abfrage und Analyse komplexer Beziehungsstrukturen, beispielsweise im Bereich der Wissensgraphen oder Empfehlungssystemen.

Nicht-relationale Datenbanksysteme sind besonders relevant in modernen Big-Data-Architekturen und Streaming-Plattformen, da sie hohe Skalierbarkeit, Verfügbarkeit und Performance bei gleichzeitiger Flexibilität in der Datenmodellierung bieten. Die Wahl der geeigneten Speicherlösung sollte sich daher stets am Datenformat, der Änderungsdynamik, den Integritätsanforderungen sowie der geplanten Nutzung innerhalb des KI-Systems orientieren.

Außer den vorgestellten Technologien spielen Ansätze der übergeordneten Datenarchitekturen ebenfalls eine zentrale Rolle im Datenmanagement für KI-Systeme. Zu den gängigen Architekturen zählen das Data Warehouse, der Data Lake, das Data Lakehouse sowie das Data Mesh. Diese Ansätze unterscheiden sich hinsichtlich ihrer Datenorganisation, ihrer Einsatzszenarien und technologischen Anforderungen und werden im Folgenden vorgestellt.

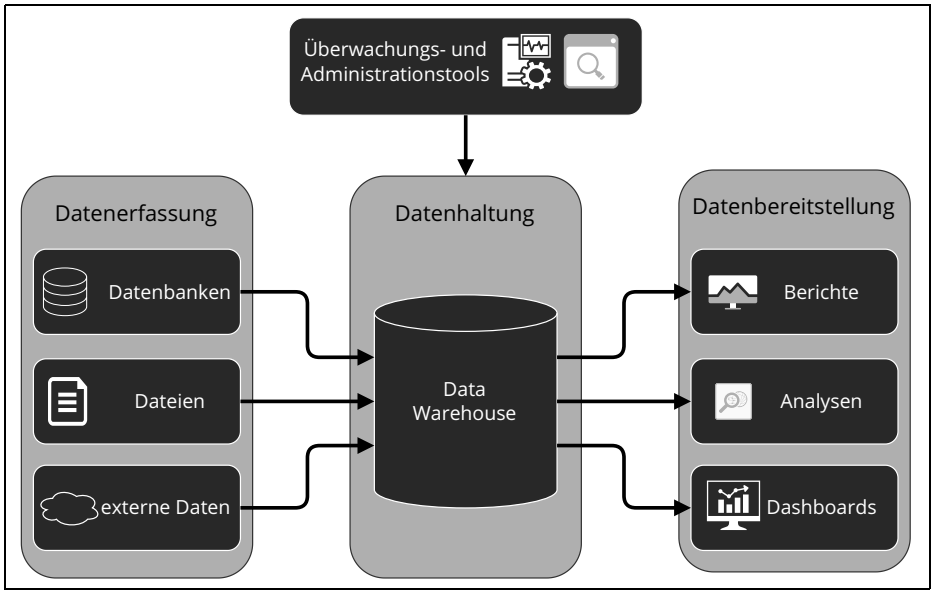

Abb. 5.10 *Schematischer Aufbau der Architektur für Data Warehouse*

Ein Data Warehouse ist eine zentralisierte Datenplattform zur Speicherung strukturierter, bereinigter und historisierter Daten aus unterschiedlichen Quellsystemen (siehe Abbildung 5.10) [Inm05]. Die Daten werden im Vorfeld durch ETL-Prozesse in ein standardisiertes Schema überführt, wodurch eine hohe Konsistenz und Qualität gewährleistet wird. Aufgrund dieses sogenannten Schema-on-Write-Prinzips eignet sich das Data Warehouse besonders für analytische Abfragen, etwa in der Business Intelligence oder für Reportings. Allerdings bringt diese Strukturierung auch Einschränkungen mit sich. Die Flexibilität ist begrenzt, Schemaänderungen sind aufwendig und die Systemkosten tendenziell hoch, insbesondere bei komplexen Optimierungen oder bei wachsendem Datenvolumen. Auch kann es durch die vorgelagerten Verarbeitungsprozesse zu einer erhöhten Latenz bei der Verfügbarkeit neuer Daten kommen. Typische Implementierungen sind Amazon Redshift, Google BigQuery, Snowflake oder Microsoft SQL Server [amazon-redshift] [bigquery] [snowflake] [ms-synapse].

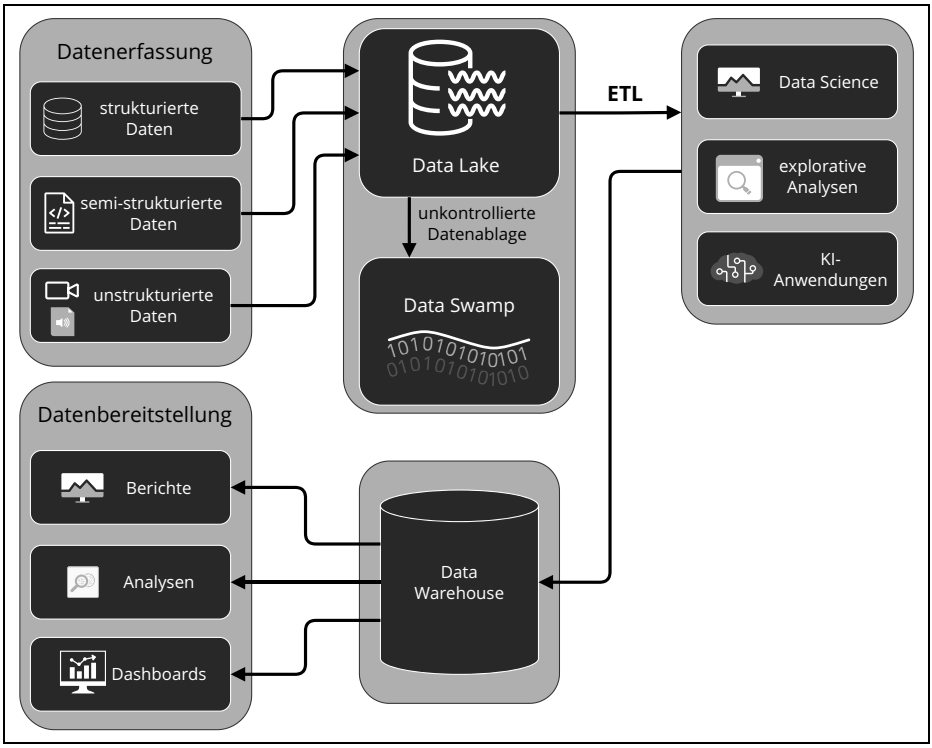

Abb. 5.11 *Schematischer Aufbau der Architektur für Data Lake mit Swamp*

Im Gegensatz dazu verfolgt ein Data Lake (siehe Abbildung 5.11) einen deutlich flexibleren Ansatz. Er dient als zentrales Repository für strukturierte, semi-strukturierte und unstrukturierte Rohdaten in unterschiedlichen nativen Formaten wie JSON, CSV oder Parquet. Durch das Schema-on-Read-Prinzip wird das Datenmodell erst zur Analysezeit angewendet, wodurch eine hohe Agilität und Anpassungsfähigkeit sichergestellt werden. Die ELT-Verarbeitung erlaubt es, Daten zunächst kosteneffizient zu speichern und erst später bedarfsgerecht zu transformieren. Speicher und Rechenleistung sind dabei häufig entkoppelt, beispielsweise durch Kombinationen wie AWS S3 (Storage) und Amazon EMR (Verarbeitung). Data Lakes sind besonders geeignet für explorative Analysen, Data Science Workflows sowie für KI-Anwendungen, bei denen große und vielfältige Datenmengen verarbeitet werden müssen. Allerdings besteht aufgrund der fehlenden einheitlichen Struktur und Kontrolle die Gefahr einer unkontrollierten Datenablage. Dieser sogenannte Data Swamp bezeichnet den Zustand eines Data Lakes, in dem Datenqualität, Übersichtlichkeit und Wiederverwendbarkeit verloren gehen, ein Risiko, das durch mangelhafte Governance oder fehlende Metadatenmanagementsysteme verstärkt wird.

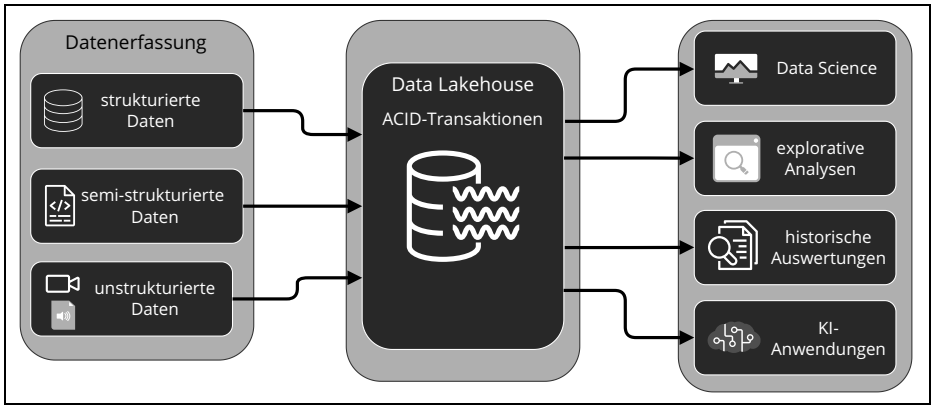

Abb. 5.12 *Schematischer Aufbau der Architektur für Data Lakehouse*

Basierend auf den Stärken von Data Warehouse und Data Lake wurde die Architektur des Data Lakehouse (siehe Abbildung 5.12) entwickelt, die versucht, die Flexibilität und Skalierbarkeit eines Data Lakes mit den transaktionalen und analytischen Fähigkeiten eines Data Warehouses zu vereinen. Ziel ist es, eine einheitliche Plattform für strukturierte, semi-strukturierte und unstrukturierte Daten bereitzustellen, die sowohl für klassische BI-Anwendungen als auch für moderne Machine-Learning-Workflows geeignet ist.

Ein zentrales Merkmal des Data Lakehouse ist die Unterstützung von ACID-Transaktionen (A: Atomarität, C: Konsistenz, I: Isolation, D: Dauerhaftigkeit). Dadurch wird sichergestellt, dass gleichzeitige Lese- und Schreibvorgänge auf denselben Daten konsistent und fehlertolerant ablaufen können. Dies ist insbesondere für datenintensive Trainingsprozesse von ML-Modellen essenziell, da Trainings-, Feature-Engineering- und Dateninkrementierungsprozesse häufig parallel stattfinden und konsistente, reproduzierbare Datenstände voraussetzen.

Technisch wird diese ACID-Unterstützung im Data-Lakehouse-Kontext nicht durch klassische, datenbankinterne Sperrmechanismen realisiert, sondern durch die Kombination aus unveränderlichen Datendateien, expliziten Transaktionsprotokollen und versionsbasierter Metadatenverwaltung. Schreiboperationen erzeugen dabei neue Dateiversionen, während Leseoperationen stets auf konsistente Schnappschüsse (Snapshots) zugreifen. Auf diese Weise lassen sich parallele Zugriffe isolieren, fehlgeschlagene Schreibvorgänge sauber zurückrollen und frühere Datenzustände reproduzierbar rekonstruieren.

Um die Effizienz bei großen Datenmengen zu verbessern, unterstützt das Data Lakehouse zudem inkrementelle Updates, bei denen nur neue oder veränderte Daten verarbeitet werden. Dies wird typischerweise durch Änderungsverfolgung auf Dateiebene oder durch Transaktionslogs ermöglicht und stellt eine wichtige Grundlage für skalierbare ML-Workflows sowie kontinuierliche Datenverarbeitung dar.

Ein weiterer Vorteil liegt in der nahtlosen Integration von Batch- und Streaming-Daten. Im Gegensatz zu klassischen Architekturen, bei denen diese Daten-

ströme getrennt behandelt werden, ermöglicht das Data Lakehouse eine einheitliche Speicherung und Abfrage beider Typen innerhalb derselben Tabellenstruktur. Das bedeutet, dass sowohl Echtzeitanalysen als auch historische Auswertungen auf dieselbe konsistente Datenbasis zugreifen können. Dies ist insbesondere für Anwendungen mit kontinuierlichem Feedback oder für zeitkritische Entscheidungen von großer Bedeutung.

Typische Technologien, die das Lakehouse-Paradigma umsetzen, sind Delta Lake, Apache Iceberg und Databricks Lakehouse Plattform [apache-iceberg] [databricks]. Sie stellen die zugrunde liegende Infrastruktur bereit, um große Datenmengen effizient, konsistent und flexibel zu verwalten. Damit schaffen sie eine skalierbare Datenbasis für moderne KI-Anwendungen.

Im Gegensatz zu zentralisierten Datenarchitekturansätzen verfolgt das Data Mesh (siehe Abbildung 5.9 oben) einen dezentralen, domänenspezifischen Ansatz, wie im »Architekturparadigmen der Datenpipelines« als organisatorisch-methodisches Paradigma eingeführt und hier mit Blick auf seine architektonisch-technische Umsetzung (Self-Serve-Plattform, Datenprodukte, Governance-Automatisierung) vertieft. Daten werden hierbei nicht als zentral verwaltete Ressourcen betrachtet, sondern als eigenständige, wertschöpfende Produkte, für die jeweils die fachlich verantwortlichen Domänenteams, etwa in den Bereichen Vertrieb, Logistik oder Produktion, zuständig sind. Diese Teams besitzen das notwendige Kontextwissen zu Struktur, Semantik und Qualität der von ihnen erzeugten und genutzten Daten und sind daher in der Lage, diese effizienter zu verwalten und zielgerichteter bereitzustellen. Dies führt zu schnelleren Entwicklungszyklen, besserer fachlicher Passung und klar definierten Verantwortlichkeiten.

Die Datenprodukte in einem Data-Mesh-System sollen zuverlässig, auffindbar, verständlich und wiederverwendbar sein. Hierzu gehören definierte Schnittstellen (z. B. APIs), standardisierte Formate und explizite SLAs, etwa zur Aktualisierungsfrequenz, Verfügbarkeit oder Latenz. Darüber hinaus ermöglichen klar definierte Qualitätsmetriken eine kontinuierliche Bewertung und Verbesserung der Datenqualität, beispielsweise hinsichtlich Konsistenz, Vollständigkeit oder Aktualität.

Trotz der dezentralen Verantwortungsstruktur ist eine zentrale technische Plattform erforderlich, um Interoperabilität, Wiederverwendbarkeit und Sicherheit zu gewährleisten. Diese Plattform stellt gemeinsame Infrastrukturen bereit, etwa Speicherlösungen, Datenpipelines, Kataloge, Monitoring-Werkzeuge und Zugriffskontrollen. Durch integrierte Governance-Funktionalitäten können unternehmensweite Standards, Richtlinien und Metadatenmanagement automatisiert durchgesetzt werden, ohne die Autonomie der Fachdomänen zu beeinträchtigen. So lassen sich etwa Zugriffspolicies, Sicherheitsanforderungen oder standardisierte Datenformate systemseitig sicherstellen.

Die dezentrale Organisation ermöglicht eine skalierbare und flexible Weiterentwicklung datengetriebener Systeme. Neue Datenprodukte können parallel, unabhängig und in hoher Qualität entstehen, ohne auf zentrale Koordinierungsprozesse angewiesen zu sein. Die Verantwortung liegt dabei bei den Domänenteams, was

nicht nur zu einer besseren Wartbarkeit führt, sondern auch zur Stärkung der Datenverantwortung auf Fachebene.

Auf organisatorischer Ebene geht dieser Paradigmenwechsel mit einer veränderten Form der Koordination und einer grundlegenden kulturellen Anpassung einher. Verantwortlichkeiten für Datenprodukte werden explizit bei den Domänenteams verankert, die Aspekte wie Dokumentation, Wartung und Qualitätskontrolle direkt übernehmen. Dadurch verlagert sich Koordination von zentralen Vermittlungsstellen hin zu den fachlich verantwortlichen Teams. Der damit verbundene initiale Formalisierungsaufwand ermöglicht es langfristig, Kommunikationswege zu verkürzen, Abhängigkeiten zu reduzieren und eine skalierbare Nutzung der Datenprodukte sicherzustellen. Zentrale Dateninstanzen geben dabei ihre kontrollierende Rolle zugunsten eines unterstützenden, plattformorientierten Rollenverständnisses auf.

Ziel des Data-Mesh-Ansatzes ist es, datengetriebene Organisationen in technischer, organisatorischer und kultureller Hinsicht nachhaltig skalierbar zu machen, durch Selbstverantwortung, Standardisierung und dezentrale Datenkompetenz.

5.5.2 Architektonische Einordnung: Datenarchitekturen und Datenspeicherung

Datenarchitekturen und Speicherlösungen bilden die langfristige Grundlage für den Betrieb, die Weiterentwicklung und die Skalierung von KI-Systemen. Entscheidungen für bestimmte Speicherparadigmen, Plattformen und Organisationsmodelle beeinflussen Kostenstrukturen, Governance-Fähigkeit und Integrationsmöglichkeiten über den gesamten Lebenszyklus hinweg. Fehlende strategische Abstimmung führt häufig zu fragmentierten Datenlandschaften, eingeschränkter Wiederverwendbarkeit und erhöhtem Betriebsaufwand.

Für Softwarearchitekten ist insbesondere relevant, dass Speicherarchitekturen nicht isoliert betrachtet werden dürfen, sondern eng mit Datenpipelines, MLOps-Prozessen und Sicherheitskonzepten verzahnt sind. Die gewählte Architektur bestimmt, wie konsistent Datenstände reproduziert, wie Zugriffe kontrolliert und wie Datenprodukte bereitgestellt werden können.

Die Architektur muss daher sicherstellen, dass Speicherlösungen skalierbar, Governance-fähig und langfristig wartbar gestaltet sind. Gleichzeitig sind Anforderungen an Datenschutz, Zugriffskontrolle, Auditierbarkeit und Kostensteuerung systematisch zu berücksichtigen.

Für die Architektur ergeben sich insbesondere folgende Anforderungen:

- Auswahl geeigneter Speicherparadigmen (Warehouse, Lake, Lakehouse, Mesh)
- Unterstützung konsistenter Snapshots und Versionierung
- Integration von Metadatenmanagement und Datenkatalogen
- Durchsetzung von Zugriffskontrollen und Sicherheitsrichtlinien
- Trennung von operativen, analytischen und Trainingsdaten
- Steuerung von Speicher- und Rechenkosten

Aspekt	Bedeutung für die Architektur
Speicherstruktur	Zentrale oder domänenspezifische Organisation
Zugriff	Rollenbasierte Zugriffskontrolle und Audit-Logs
Konsistenz	Snapshot- und Transaktionsmechanismen
Skalierung	Entkopplung von Storage und Compute
Governance	Metadaten, Kataloge, Richtlinien
Betrieb	Backup-, Recovery- und Archivstrategien
Risiko	Datenfragmentierung, Kostenexplosion, Sicherheitslücken
Typische Fehlannahmen	
»Die Speicherlösung lässt sich später leicht austauschen.«	
»Ein zentraler Data Lake reicht für alle Anforderungen.«	
»Zugriffskontrollen sind ein reines Infrastrukturthema.«	
»Kosten skalieren linear mit dem Datenvolumen.«	
»Metadatenpflege ist optional.«	

Tab. 5.4 *Architectural Impact Box: Datenarchitekturen und Speicherung*

Diese Fehlannahmen führen häufig zu langfristigen Strukturproblemen, steigenden Betriebskosten und eingeschränkter Governance-Fähigkeit.

5.6 Takeaways

Datenmanagement und Datenverarbeitung sind zentrale architekturbestimmende Faktoren für KI-Systeme und beeinflussen maßgeblich deren Qualität, Stabilität und Wartbarkeit. Eine leistungsfähige Modellarchitektur kann strukturelle Defizite in der Datenpipeline nicht kompensieren.

Reproduzierbarkeit entsteht nur durch konsequent versionierte Datenstände, dokumentierte Transformationsregeln und nachvollziehbare Labeling-Prozesse. Ohne diese Grundlagen lassen sich Modellverhalten und Qualitätsänderungen im Betrieb nicht zuverlässig erklären oder auditieren.

Die Gestaltung von Datenpipelines erfordert bewusste Architekturentscheidungen im Spannungsfeld von Latenz, Konsistenz, Skalierbarkeit und Betriebskomplexität. Batch-, Streaming- und hybride Ansätze müssen an den fachlichen und organisatorischen Kontext angepasst werden.

Datenaufbereitung, Feature Engineering und Datenanreicherung sind integrale Bestandteile der Softwarearchitektur, da sie die semantische Struktur der Modellinputs festlegen und langfristig das Systemverhalten prägen. Die Wahl von Speicherarchitekturen und Datenplattformen bestimmt nicht nur technische Leistungs-

merkmale, sondern auch Governance-Fähigkeit, Kostenstruktur und organisatorische Verantwortlichkeiten.

Ein nachhaltiges Datenmanagement erfordert die enge Verzahnung von Data Engineering, MLOps, Governance und Softwarearchitektur. Erst durch diese Integration entstehen belastbare, wartbare und vertrauenswürdige KI-Systeme.

5.7 Architectural Definition Box

Begriff	**ETL / ELT**
Definition	Datenflussstrategien zur Extraktion, Transformation und Bereitstellung von Daten. Während ETL Daten vor der Speicherung transformiert, erfolgt die Transformation bei ELT erst im Zielsystem.
Relevanz	Die Wahl zwischen ETL und ELT beeinflusst Speicherarchitektur, Skalierbarkeit, Performance und Flexibilität der Datenverarbeitung.

Aspekt	**Einordnung**
Mess-/Beobachtbarkeit	Verarbeitungsdauer, Ressourcenauslastung, Transformationsaufwand, Datenlatenz
Typische Fehlinterpretation	»ETL und ELT unterscheiden sich lediglich in der Reihenfolge einzelner Schritte.«
Architekturkonsequenz	Erfordert bewusste Entscheidungen hinsichtlich Speicherkonzepten, Rechenressourcen und Datenverarbeitungsmodellen

Begriff	**Data Mesh**
Definition	Dezentraler Architekturansatz, bei dem Daten als eigenständige Produkte organisiert und von fachlichen Domänen verantwortet werden
Relevanz	Klassische zentrale Datenplattformen stoßen bei wachsender organisatorischer und technischer Komplexität häufig an Grenzen. Data Mesh adressiert diese Einschränkungen durch domänenspezifische Verantwortlichkeiten und skalierbare Datenstrukturen.

Aspekt	**Einordnung**
Mess-/Beobachtbarkeit	Anzahl domänenspezifischer Datenprodukte, SLA-Einhaltung, Wiederverwendungsgrad, Datenqualitätsmetriken
Typische Fehlinterpretation	»Data Mesh beschreibt primär eine neue Speichertechnologie.«
Architekturkonsequenz	Erfordert klare Domänenverantwortung, standardisierte Schnittstellen, Governance-Mechanismen und Datenprodukte mit definierten Qualitätsmerkmalen

Begriff	**Lambda-Architektur**
Definition	Hybrides Architekturmodell zur Kombination von Batch- und Streaming-Verarbeitung, um sowohl vollständige historische Datenanalysen als auch Echtzeitverarbeitung zu unterstützen
Relevanz	Viele KI-Systeme benötigen gleichzeitig hochaktuelle Informationen und konsistente historische Datenbestände. Die Lambda-Architektur verbindet beide Anforderungen innerhalb eines gemeinsamen Modells.

Aspekt	**Einordnung**
Mess-/Beobachtbarkeit	Verarbeitungslatenz, Konsistenz zwischen Verarbeitungspfaden, Synchronisationsaufwand, Fehlerraten
Typische Fehlinterpretation	»Die Kombination mehrerer Verarbeitungspfade verbessert Systeme grundsätzlich.«
Architekturkonsequenz	Erfordert parallele Verarbeitungslogiken, konsistente Synchronisation sowie erhöhte Anforderungen an Betrieb und Wartbarkeit

Begriff	**Kappa-Architektur**
Definition	Architekturparadigma, bei dem sämtliche Daten ausschließlich über einen kontinuierlichen Streaming-Pfad verarbeitet und dauerhaft als Ereignisstrom gespeichert werden
Relevanz	Kappa reduziert die Komplexität hybrider Architekturen und eignet sich besonders für Systeme mit hohen Echtzeitanforderungen und kontinuierlichen Datenströmen.

Aspekt	**Einordnung**
Mess-/Beobachtbarkeit	Ereignisdurchsatz, Verarbeitungslatenz, Event-Replay-Zeiten, Recovery-Dauer
Typische Fehlinterpretation	»Streaming ersetzt sämtliche Anforderungen klassischer Datenverarbeitung.«
Architekturkonsequenz	Benötigt dauerhaft verfügbare Event-Logs, zustandsbehaftete Stream-Verarbeitung und fehlertolerante Infrastrukturmechanismen

Inhaltsverzeichnis

Vorwort zur 1. Auflage 15

1 Einleitung 17

1.1 KI-Systeme zwischen Innovation und Risiko ... 17
1.2 Warum dieses Buch entstanden ist ... 17
1.3 Was dieses Buch nicht ist ... 18
1.4 Voraussetzungen ... 18
1.5 Leitfaden für den Leser ... 18
1.6 Zielpublikum ... 19
1.7 Hinweis zu Aktualität und fachlicher Weiterentwicklung ... 19
1.8 Die Reviewer ... 20
1.9 Danksagungen ... 21

2 Einführung in softwarearchitekturrelevante Konzepte für Künstliche Intelligenz 23

2.1 Architektonische Einordnung dieses Kapitels ... 24
2.2 Künstliche Intelligenz und ihre Familienmitglieder ... 24
2.2.1 Was ist Künstliche Intelligenz? ... 26
2.2.2 Familienporträt der KI ... 28
2.2.3 Maschinelles Lernen ... 30
2.2.4 Neuronales Netz und Deep Learning ... 40
2.2.5 Generative KI ... 53
2.2.6 Data Science ... 59
2.3 Anwendungen der KI ... 62
2.3.1 Übersichtsmodell der KI-Anwendungen ... 62
2.3.2 Konzeptionelle Anwendungen der KI ... 64
2.3.3 Endnutzeranwendungen der KI und deren Domänen ... 66
2.3.4 Stärken und Grenzen der KI ... 68
2.3.5 Risiken bei der Anwendung der KI ... 70

2.4 KI-Engineering im Vergleich zur traditionellen Human-Engineered Softwareentwicklung ... 71
2.4.1 Adressierte Problemstellungen des KI-Engineerings ... 71
2.4.2 Herausforderungen der Arbeitsergebnisse von KI-Engineering ... 75
2.4.3 Rollen im Softwareentwicklungsprozess für KI-Engineering ... 80
2.4.4 Auswirkung der Integration der KI auf Produktivität ... 85
2.5 Takeaways ... 87
2.6 Architectural Definition Box ... 87
2.7 Referenzen ... 91

3 KI-Governance: Compliance, Sicherheit und Alignment in der KI-Softwarearchitektur 97

3.1 Architektonische Einordnung dieses Kapitels ... 98
3.2 Einleitung und Grundbegriffe ... 98
3.2.1 Die Schutzschilde der KI: Compliance, Sicherheit und Alignment ... 99
3.2.2 Datenschutz, Ethik und Risiko: Die Fragen der KI-Zukunft ... 101
3.3 Rechtliche Grenzen von KI-Systemen ... 103
3.3.1 Recht als architektonischer Gestaltungsrahmen ... 104
3.3.2 Verantwortung, Haftung und Zurechenbarkeit ... 104
3.3.3 Nicht-Delegierbarkeit und Entscheidungshoheit ... 105
3.3.4 Nachvollziehbarkeit, Nachweisbarkeit und transparente Datenketten ... 106
3.3.5 Organisatorisch-technische Kopplung ... 107
3.3.6 Rechtliche Spannungsfelder ... 108
3.3.7 Dokumentation als architektonisches Fundament für KI-Systeme ... 111
3.4 Die Bedeutung des Datenschutzes – architektonische Perspektiven ... 112
3.4.1 Dokumentation als Fundament für Transparenz, Erklärbarkeit und Sicherheit in ML-Modellen ... 113
3.4.2 Verarbeitung der Daten ... 114
3.4.3 Speicherung von Daten ... 115
3.4.4 Balance zwischen Innovation und Regulierung ... 116
3.5 Sicherheit in KI-Systemen – architektonische Perspektive ... 117
3.5.1 Sicherheitsbedrohungen in KI-Systemen ... 118
3.5.2 Sicherheit von Anfang an ... 124
3.5.3 Dokumentationsfähigkeit als Sicherheitsmerkmal ... 126

3.6 Ethik in der KI ... 127
3.6.1 Ethik-Leitlinien als architektonischer Bezugsrahmen ... 128
3.6.2 Ethische Grenzen der KI ... 128
3.7 Der EU AI Act ... 134
3.7.1 Der europäische Regulierungsansatz und seine architektonische Bedeutung ... 134
3.7.2 Risikoklassifizierung im EU AI Act und ihre architektonischen Konsequenzen ... 135
3.7.3 Risikoklassifizierung als architektonisch verankerter Prozess ... 136
3.7.4 Transparenzanforderungen und architektonische Umsetzung ... 137
3.7.5 Aufsicht, Sanktionen und organisatorische Anschlussfähigkeit ... 139
3.7.6 Best Practices zur architektonischen Umsetzung des EU AI Acts ... 141
3.8 ISO/IEC 42001: Managementsysteme als architektonischer Ordnungsrahmen für KI ... 142
3.9 Verantwortungsvolle KI: Ein tabellarischer Vergleich internationaler Leitlinien ... 143
3.10 Regulatory Sandboxes als architektonische Experimentierräume ... 144
3.10.1 Ziel und Funktion von Regulatory Sandboxes ... 145
3.10.2 Regulatory Sandboxes als soziotechnische Systemräume ... 145
3.10.3 Architekturen für Regulatory Sandboxes ... 145
3.10.4 Übergang von der Sandbox in den Regelbetrieb ... 146
3.10.5 Rolle von Sandboxes im KI-Lebenszyklus ... 146
3.11 Takeaways ... 146
3.12 Architectural Definition Box ... 147
3.13 Referenzen ... 150

4 Entwurf und Entwicklung von KI-Systemen 153

4.1 Architektonische Einordnung dieses Kapitels ... 153
4.2 Lebenszyklus eines KI-Projekts ... 153
4.2.1 Übersicht über den Lebenszyklus eines KI-Projekts ... 154
4.2.2 Explorative Datenanalyse ... 154
4.2.3 Datenbereinigung und -aufbereitung ... 155
4.2.4 Feature Engineering ... 156
4.2.5 Modellauswahl und -training ... 157
4.2.6 Proof of Concept (PoC) ... 158
4.2.7 Deployment ... 159
4.2.8 Wartung ... 160
4.2.9 Architektonische Einordnung: Lebenszyklusmanagement ... 160

4.3 Vorgehensmodelle für die Softwareentwicklung von KI-Systemen ... 161
4.3.1 CRISP-DM ... 161
4.3.2 GenAI Life Cycle ... 162
4.3.3 CRISP-ML(Q) ... 163
4.3.4 Team Data Science Process (TDSP) ... 164
4.3.5 Architektonische Einordnung: Vorgehensmodelle ... 165
4.4 Daten und ihre Anforderungen ... 166
4.4.1 Überblick über Datenarten und -klassen ... 166
4.4.2 Anforderungen an Daten ... 168
4.4.3 Datentrennung ... 169
4.4.4 Underfitting vs. Overfitting ... 169
4.4.5 Architektonische Einordnung: Datenmanagement ... 170
4.5 ML-Problemstellungen und deren Anforderungen ... 171
4.5.1 Überwachtes-Lernen-basierte ML-Probleme und ihre spezifischen Anforderungen ... 172
4.5.2 Unüberwachtes-Lernen-basierte ML-Probleme und ihre spezifischen Anforderungen ... 173
4.5.3 Bestärkendes-Lernen-basierte ML-Probleme und ihre spezifischen Anforderungen ... 174
4.6 Eingabedaten für verschiedene KI-Algorithmen ... 175
4.6.1 Vorbereitung der Eingabedaten durch Datentrennung ... 176
4.6.2 Numerische Vektoren, Matrizen und Tensoren für neuronale Netze ... 176
4.6.3 One-Hot-Encodings ... 177
4.6.4 Embeddings ... 178
4.6.5 Architektonische Einordnung: Problemklassifikation ... 180
4.7 Behandlung der Herausforderungen bei der Entwicklung von KI-Systemen ... 181
4.7.1 Nicht-Determinismus ... 181
4.7.2 Mangelnde Datenqualität ... 182
4.7.3 Modell-Drift ... 183
4.8 Transfer-Learning und Finetuning ... 186
4.8.1 Transfer-Learning ... 187
4.8.2 Finetuning ... 188
4.9 Entwurfsmuster für KI-Systeme ... 189
4.9.1 Topologie-Muster für ML-Systeme ... 190
4.9.2 Pipeline-Architektur-Muster ... 202
4.9.3 Model-Training-Muster ... 213
4.9.4 Model-Serving-Muster ... 220
4.9.5 Model-Deployment-Muster ... 225

4.10 Architektur und Aufgaben von Machine-Learning-Systemen 232
4.10.1 Komponenten eines ML-Systems 233
4.10.2 Aufgaben und Anwendungsfälle 240
4.10.3 Integration von ML-Modellen in bestehende Systeme ... 245
4.11 Metriken und Leistungsmessung von ML-Modellen 249
4.11.1 Grundlegende Metriken 250
4.11.2 Systemische Metriken 254
4.11.3 Kombination von Metriken 255
4.11.4 Optimierung technischer Performance 256
4.12 Skalierbarkeit und Robustheit in ML-Systemen 260
4.12.1 Skalierbarkeit 260
4.12.2 Robustheit 262
4.13 Zuverlässigkeit und Verfügbarkeit in ML-Systemen 270
4.13.1 Stabilität 271
4.13.2 Verfügbarkeit 272
4.13.3 Konzepte der Fehlertoleranz 274
4.14 Bias, Fairness und Qualitätssicherung 275
4.14.1 Erkennung und Reduktion von Bias 276
4.14.2 Ergebnisqualität und Reproduzierbarkeit 279
4.15 Sicherheit und Datenschutz in ML-Systemen 282
4.15.1 Technische Maßnahmen zur Stärkung des Datenschutzes 283
4.15.2 Sicherheitsanforderungen in Compound-AI-Systemen .. 284
4.15.3 Sicherheitsmaßnahmen und Monitoring-Ansätze 285
4.15.4 Compliance und Nachvollziehbarkeit in KI-Systemen ... 287
4.16 Gestaltung effektiver Benutzeroberflächen 293
4.16.1 Nutzerzentrierung 294
4.16.2 Transparenz bei KI-Systemen 294
4.16.3 Informationsarchitektur und Layout 295
4.16.4 Interaktive Elemente und Feedback 296
4.16.5 Responsives und barrierefreies Design 297
4.16.6 KI-spezifische UI-Elemente 297
4.16.7 Visualisierung von Daten und Ergebnissen 298
4.17 Zukunftsperspektiven 299
4.17.1 Generative KI 299
4.17.2 Compound AI 300
4.17.3 KI-Agenten 301
4.18 Takeaways 302
4.19 Architectural Definition Box 303
4.20 Referenzen 306

5 Datenmanagement und Datenverarbeitung für KI-Systeme 311

5.1 Architektonische Einordnung dieses Kapitels ... 311
5.2 Methoden der Datenakquisition und Datenannotation ... 312
5.2.1 Datenakquisition ... 312
5.2.2 Datenannotation (Labeling) ... 313
5.2.3 Plattformen für öffentlich zugängliche Daten ... 314
5.2.4 Architektonische Einordnung: Datenakquisition und Datenannotation ... 315
5.3 Gestaltung effizienter Datenpipelines und -architekturen ... 317
5.3.1 Grundprinzipien effizienter Datenpipelines ... 317
5.3.2 Architekturgestaltung effizienter Datenpipelines ... 318
5.3.3 Relevante Werkzeuge für Datenpipelines ... 330
5.3.4 Architektonische Einordnung: Datenpipelines und Datenarchitekturen ... 332
5.4 Strategien zur Datenaufbereitung ... 333
5.4.1 Strategien für Datenbereinigung ... 334
5.4.2 Strategien für Datenaggregation ... 335
5.4.3 Strategien für Datenstandardisierung ... 336
5.4.4 Strategien für Datentransformation ... 336
5.4.5 Strategien für Datenanreicherung und -augmentierung ... 337
5.4.6 Architektonische Einordnung: Datenaufbereitung ... 338
5.5 Konzeption der Datenarchitekturen und Datenspeicherung ... 340
5.5.1 Datenarchitekturen und Technologien zur Datenspeicherung ... 340
5.5.2 Architektonische Einordnung: Datenarchitekturen und Datenspeicherung ... 345
5.6 Takeaways ... 346
5.7 Architectural Definition Box ... 347
5.8 Referenzen ... 349

6 Wichtige Qualitätsmerkmale für den Betrieb von KI-Systemen 353

6.1 Einordnung und fachlicher Rahmen ... 353
6.1.1 Referenzrahmen der vorhergehenden Kapitel ... 353
6.1.2 Bezug zu Qualitätsmodellen und Abgrenzung ... 354
6.1.3 Abgrenzung zu MLOps, Lebenszyklusmodellen und Werkzeugdarstellungen ... 354
6.1.4 Architektonische Einordnung dieses Kapitels ... 354
6.2 Betriebliche Qualitätsdimensionen von KI-Systemen ... 355
6.2.1 Zeitabhängigkeit als architektonische Dimension ... 355
6.2.2 Nachvollziehbarkeit und Rekonstruierbarkeit im Betrieb ... 356

6.2.3 Kontrollierbarkeit und Eingreifbarkeit im Betrieb 357
6.2.4 Betriebliche Steuerbarkeit als Zusammenführung der Dimensionen 358
6.3 Architekturdomänen des KI-Betriebs 359
6.3.1 Betriebsinfrastruktur als architektonische Domäne 359
6.3.2 Infrastruktur- und Hardwaredomäne 360
6.3.3 Modell- und Laufzeitdomäne 376
6.3.4 Betriebs- und Integrationsdomäne 393
6.3.5 Organisations- und Governancedomäne 403
6.4 Exkurs: MLOps-Systeme und die Build-vs.-Buy-Entscheidung ... 405
6.5 Exkurs: MLOps-Werkzeuge und End-to-End-Plattformen 406
6.5.1 Neue Akteure: MLOps-Werkzeuge und Plattformen im Überblick 406
6.6 Takeaways 409
6.7 Architectural Definition Box 410
6.8 Referenzen 412

7 Systemarchitekturen und Plattformen für generative KI-Systeme 413

7.1 Architektonische Einordnung des Kapitels 413
7.2 Grundlagen und Qualitätsmerkmale Generativer KI 414
7.3 Arten Generativer KI 416
7.3.1 Large Language Models 416
7.3.2 Codegenerierende Sprachmodelle 417
7.3.3 Bildgenerierende Modelle 418
7.3.4 Musikgenerierende Modelle 420
7.3.5 Multimodale Modelle 420
7.3.6 Foundation Models und universelle Basismodelle 421
7.4 Auswahlkriterien für Generative KI 422
7.4.1 Modellgröße und Architektur 422
7.4.2 Leistung und Benchmarks 423
7.4.3 Kontextlänge 423
7.4.4 Qualität und Vielfalt der Trainingsdaten 424
7.4.5 Kosten und Lizenz 424
7.4.6 Latenz und Skalierbarkeit 425
7.4.7 Anpassbarkeit und Multimodalität 425
7.4.8 Datenschutz und Sicherheit 426
7.5 Integration von KI in die Gesamtarchitektur einer IT-Landschaft 426
7.5.1 Integrationsebenen von KI 427
7.5.2 Strategische KI-Integration 429
7.5.3 Architekturmuster zur Integration von KI 431

7.6 Patterns und Techniken zur Verbesserung der Antwortqualität generativer KI-Modelle ... 434
7.6.1 Prompt-Engineering ... 434
7.6.2 Retrieval-Augmented Generation ... 436
7.7 Multi-Task-Finetuning generischer Sprachmodelle ... 439
7.8 Layered-Caching-Strategie ... 440
7.9 Memory Cognition für LLMs ... 441
7.10 Mischen von regelbasierten und generativen Ansätzen ... 441
7.11 AI Query Router ... 442
7.12 Grundlagen agentischer Systeme ... 442
7.13 Agentic Workflows ... 448
7.13.1 Reflexion ... 448
7.13.2 Werkzeugunterstützung ... 448
7.13.3 Planung ... 449
7.13.4 Multi-Agenten-Kollaboration ... 449
7.14 Evaluierung von LLM-Anwendungen ... 450
7.14.1 Scoring ... 450
7.14.2 Manuelle Evaluierung ... 451
7.14.3 Modellbasierte Evaluierung ... 451
7.14.4 Vergleichende Evaluierung ... 452
7.15 Takeaways ... 452
7.16 Architectural Definition Box ... 453
7.17 Referenzen ... 456

8 Fallstudie und Praxisprojekt 459

8.1 Lernziele ... 459
8.2 KI-gestütztes Störungsmanagement im ÖPNV ... 459
8.2.1 Ausgangssituation & Problemstellung ... 459
8.2.2 Integration in die bestehende Systemlandschaft ... 460
8.3 Architekturanforderungen ... 461
8.3.1 Funktionale Anforderungen ... 461
8.3.2 Qualitätsanforderungen ... 464
8.3.3 Randbedingungen ... 467
8.4 Lösungsansatz ... 469
8.5 Einsatz von KI ... 470
8.6 Technische Umsetzung ... 470
8.7 Risiken und Gegenmaßnahmen ... 472
8.8 Ergebnisse und Nutzen ... 473
8.9 Reflexion ... 474
8.10 Referenzen ... 474

9 Zukunft der Softwarearchitektur von KI-Systemen 475

9.1 Einleitung 475
9.2 Trends in der KI und ihre architektonischen Implikationen 476
9.2.1 Federated Learning und dezentrale Trainingsansätze 476
9.2.2 Edge-KI und die Verschiebung von Intelligenz 476
9.3 Von Human-in-the-Loop zu Responsibility-by-Design 477
9.3.1 Grenzen prozessualer Kontrolle 477
9.3.2 Verantwortung als architektonische Eigenschaft 477
9.3.3 Architekturmuster zur Umsetzung von Responsibility-by-Design 477
9.4 KI in hybriden Architekturen und in der Entwicklung 478
9.5 Quantum Computing als architektonische Perspektive 479
9.6 Perspektiven für zukünftige KI-Architekturen 479
9.7 Architectural Definition Box 480

Glossar 483

Index 503

Index

A

A/B-Testing 452
Abhängigkeiten, datengetriebene 24
Activeloop 406
Adversarial Attacks 119
Agentenframeworks
  Kernfunktionen 450
  Laufzeitumgebungen 446
Agentic Workflows 446, 448
  Function Calling 448
  Multi-Agenten-Kollaboration 449
  Planung 449
  Reflexion 448
  Werkzeugunterstützung 448
Agentische Systeme
  Agent 443
  Agentenframework 446
  Agentenframeworks 450
  Agentic Workflows 446, 448
  Architekturkomponenten 445
  Entscheidungskriterien 447
  Grundlagen 442
  handlungsfähige Softwaresysteme 447
  Model Context Protocol 443
  Nichtdeterministik 444
  Orchestrierungsschicht 445
  Speichermechanismen 446
  Think-Act-Observe-Zyklus 446
  zielorientierte Systeme 444
AI Query Router, Routing-Strategien 442
AI-Engineering-Ebene, Prompt-Engineering 427
AIMS 142
Aktivierungsfunktion 42
Alignment 100, 148
Alignment, normative Steuerung 101
AlphaCode 2 244
Analytische KI 29
Anforderungserfassung, KI-gestützt 86
Anomalieerkennung 65
Anti-Corruption Layer 430, 454
Anwendungsebene
  horizontale Integration 427
  Plug-in-basierte Integration 427
Apache Flink 331
  APIs 332
  Checkpoints und Savepoints 331
  Event Time 331
  Exactly-Once 331
  zustandsbehaftete Datenverarbeitung 331
  Zustandsmanagement 331
Apache Spark 330
  In-Memory-Verarbeitung 330
  Resilient Distributed Datasets 330
  Spark MLlib 331
  Spark SQL und PySpark 331
  Structured Streaming 331
Aporia 406
Architekturanforderungen
  funktionale Anforderungen 461
  Qualitätsanforderungen 464
  Randbedingungen 467
Architekturen, hierarchische 198
Architekturentscheidungen, KI-Kontext 24
Architekturparadigmen
  Data-Mesh-Architektur 328
  Event-Driven Architecture 327
  Kappa-Architektur 325
  Lambda-Architektur 324
Argo 406

Artificial Intelligence Management Systems 142
Asilomar AI Principles 143
ATLAS 123
AUC-ROC 252
Ausführungsumgebungen
  Cloud 366
  Edge 365
  hybride Architekturen 366
  On-Premises 365
  Verortung 365
Ausreißerbehandlung 155
  Z-Score und Interquartilsabstand 334
Autoencoder 49

B

Backpropagation 43
  Forward Propagation 43
Batch-Layer 324
Batch-Pipelines 396
BentoML 407
Benutzeroberflächen
  Barrierefreiheit 297
  interaktive Elemente und Feedback 296
  Lokalisierung 297
  Nutzerzentrierung 294
  responsives Design 297
  responsives und barrierefreies Design 297
Bestärkendes Lernen
  Belohnungsfunktion 175
  Bewertung 175
  Zustands- und Aktionsraum 175
Betriebsinfrastruktur
  Echtzeitanforderungen und Live-Training 364
  Embedded- und Edge-Architekturen 360
  Inferenzarchitekturen 363
  Kosten-, Energie- und Skalierungsprofile 360
  Skalierung 369
  Trainings- und Inferenzlasten 359
  Trainings-, Inferenz- und Echtzeitanforderungen 363
  Trainingsarchitekturen 363
  Wirtschaftlichkeit 373
  zentrale Bestandteile 360
Betriebsmodelle, SaaS und Self-Hosting 365
Betriebsstabilität 410
  kontrollierte Übergänge 369
  Rückführbarkeit 369
  Übergangs- und Migrationsarchitekturen 369
Betroffenenrechte 113
Bias 70
  Annotation Bias 276
  Arten und Ursprungspunkte 276
  Erkennung und Reduktion 276
  Sampling Bias 276
Bildgenerierende Modelle
  asynchrone Integration 420
  Diffusionsprinzip 418
  Inpainting 419
  Latent Diffusion 419
  latenter Repräsentationsraum 419
  Outpainting 419
Blackbox 47, 90, 108
Blackbox-Problematik, offene KI-Ökosysteme 108
BLEU 255
Bounded Contexts 454
  Decision Support 469
  Event Ingestion 469
  Information Delivery 469
  KI-Analyse 469
Build-vs.-Buy-Entscheidung 405

C

Capability Maps 430
Cloud-Abhängigkeit, Hybridarchitektur 473
Clustering 35, 65
Clustering-Metriken 253
  Davies-Bouldin Index 253
  Silhouette Coefficient 253
Codegenerierende Sprachmodelle
  Codegenerierung 417
  Integration in Entwicklungsprozesse 418
  Qualitätssicherung 418
  Testfallgenerierung 417
Comet ML 407
Compliance 99
  architektonische Übersetzungsaufgabe 100
  Erklärbarkeit 287
  KI-unterstützte Auditing-Tools 293
  Transparenzanforderungen 287

Compound AI
  Austauschbarkeit von Modulen 301
  modulare Architekturen 300
  Multi Agent Systems 301
  orchestrierte Ökosysteme 301
  Orchestrierung und Evaluation 301
Compound-AI-Systeme 245
  Modulinteraktionen 284
  Sicherheitsanforderungen 284
Computer Vision Annotation Tool 314
Concept-Drift 185
Context Maps 430
Convolutional Neural Network 51
  Feature Map 51
  Parameter-Sharing 51
  Pooling-Layer 52
Corner Cases 77
CRISP-DM 161
  iterativer Ansatz 162
CRISP-ML 163

D

DagsHub 407
Dartmouth Conference 25
Data Analyst 83
Data Annotator 84
Data Engineer 83
Data Extraction 120
Data Lake
  Data Swamp 342
  Schema-on-Read 342
Data Lakehouse
  ACID-Transaktionen 343
  inkrementelle Updates 343
  Snapshots 343
Data Leakage 169, 305
Data Mesh 347
  Datenprodukte 328
  Domänenteams 328
  Governance-Rahmen 329
  produktorientiertes, verteiltes Datenmanagement 329
  Service-Level-Agreements 328
Data Poisoning 119
Data Science 59
  Abgrenzung zu KI 60
  architektonische Einordnung 61
Data Scientist 83
Data Warehouse, Schema-on-Write 341
Databricks 407
Data-Drift 185
Daten
  Anforderungen 166, 168
  Datenmenge und Datenvielfalt 168
  metrische 167
  nominale 166
  ordinale 167
  semistrukturierte 167
  strukturierte 167
  unstrukturierte 167
Daten- und Modellversionierung 76
Datenabdeckung 31
Datenaggregation 335
  entitätsbezogene Aggregation 335
  Granularitätsstufe 335
  Join-Strategien 335
  Konfliktauflösung 335
  räumliche Aggregation 335
  zeitliche Aggregation 335
Datenakquisition 312
  automatisierte Verfahren 312
  bestehende Datenquellen 312
  Datenkuratierung 313
  manuelle Datenerhebung 312
  sensorgestützte Datenerfassung 312
  synthetische und simulierte Daten 313
  Web Scraping und Web Crawling 312
Datenakquisition und Annotation
  architektonische Einordnung 315
  Datenherkunft 315
Datenannotation 313
  halbautomatische Datenannotation 314
  Labeling 313
  Labeling-Regeln 316
  manuelle Annotation 313
  Mehrfachannotation und Expertenvalidierung 314
  Qualitätssicherung 314
  vollautomatisierte Annotation 314
Datenanreicherung und Datenaugmentierung
  Domänenadaption 338
  qualitative Erweiterung des Merkmalsraums 338
Datenarchitekturen 317
  architektonische Einordnung 345

Data Lake 342
Data Lakehouse 343
Data Mesh 344
Data Warehouse 341
Datenflüsse, Speicherstrukturen und Transformationsprozesse 311
Datenspeicherung 340
Speicherlösungen 345
Datenarten, statistische 166
Datenaufbereitung 46
architektonische Einordnung 338
Aufbereitungslogik 339
Datenaggregation 335
Datenanreicherung 337
Datenanreicherung und Datenaugmentierung 337
Datenaugmentierung 338
Datenbereinigung 334
Datenstandardisierung 336
Datentransformation 336
Strategien 333
Datenbereinigung 334
Ausreißerbehandlung 334
Imputation 334
Missing Values 334
Redundanzen und Duplikate 334
Datenbereinigung und -aufbereitung 155
Datenbeschriftung 33
Datengetriebene Abhängigkeiten 24
Datenintegration 156
Datenkette, transparente 106, 149
Datenmanagement 170, 311
Datenspeicherung und Datenpersistenz 340
Lebenszyklusmanagement 318
Datenpipeline-Architekturen
Architekturparadigmen 319
Datenflussstrategie 318
dreistufiges Abstraktionsmodell 318
Verarbeitungsmodelle 319
Datenpipelines 75, 317
architektonische Einordnung 332
Architekturgestaltung 318
Batch und Streaming 318
Datenaktualität und Data Velocity 317
Datenqualität 318
Datenverarbeitungsmodelle 321
Datenvielfalt und Data Variety 317
ELT-Modell 320
ETL-Modell 319
ETL-Pipelines 320
Grundprinzipien 317
Orchestrierungs-Frameworks 331
Parallelisierbarkeit 317
Pipeline-Architekturen 332
Pipeline-Design 333
Standardisierung der Datenformate 318
Streaming-Architekturen 317
Verarbeitungskapazität 317
Werkzeuge 330
Workflow-Orchestrierung 332
Datenpipeline-Werkzeuge
Apache Beam 330
Apache Flink 331
Apache Spark 330
Data Build Tool 330
Pandas und Dask 330
Datenqualität 168
automatisierte Qualitätsprüfungen 183
mangelnde 182
mangelnde Datenqualität im Betrieb 183
Datenquellen
Community- und Crowdsourcing-Plattformen 315
öffentlich zugängliche Daten 314
staatliche Datenplattformen 315
wissenschaftliche Datenplattformen 315
Datenschutz 101
Anonymisierung 283
Anonymisierung und Pseudonymisierung 114
Architekturtreiber 117
automatisierte Entscheidungen 114
Betroffenenrechte 113
DataOps-Prozesse 283
Datenlokalisierung 115
Datensammlung 113
Datenspeicherung 115
Datenverarbeitung 114
Datenverfügbarkeit und Datenminimierung 113, 116
Innovation und Regulierung 116
internationale Regelwerke 112
internationale Regulierungsansätze 116
KI-Softwarearchitektur 112
Löschbarkeit 115

personenbezogene Daten 112
Privacy-by-Default 291
Pseudonymisierung 283
Reaktionsmechanismen für Nutzeranfragen 289
regionale Datenräume 113
Systemanforderungen 112
technische Maßnahmen in ML-Systemen 283
Verantwortlichkeit und Rechenschaftspflicht 114
Verantwortungszuordnung 116
Verarbeitungskontexte 114
Zweckbindung 113
Datenschutz und Sicherheit, Prompt Injection und System Prompt Leakage 426
Datenspeicherung
CSV-Dateien 340
Datenschutz 115
Graphdatenbanken 340
NoSQL-Datenbanken 340
relationale Datenbanksysteme 340
Speicherarchitekturen 345
Datenstandardisierung
Normalisierung und Standardisierung 336
Plausibilitätsprüfungen 336
syntaktische und semantische Konsistenz 336
Datentransformation
Preprocessing-Pipelines 337
Transformationsschema 337
Datentrennung 169
Trainings-, Validierungs- und Testdaten 169
Datenverarbeitung 311
Datenschutz 114
Datenverarbeitungsmodelle
Batch-Verarbeitung 321
Micro-Batching 323
Streaming-Verarbeitung 322
Datenversionierung 305
DDD 429
DDoS-Angriffe 122
Deep Learning 25, 45–46
Architektonische Einordnung 52
Determinismus/Nichtdeterminismus 89
Deterministische Regeln 23
DIKW-Pyramide 60
Dimensionsfluch 179
Dimensionsreduktion 35
Distribution Shift 188
Dokumentationsartefakte, Model Cards und Datasheets for Datasets 281
Dokumentationsfähigkeit 150
Domain-Driven Design 429
Anti-Corruption Layer 430
Bounded Context 430
Context Maps 430
Domänenwissen 28, 88
Domino Data Lab 407
DSGVO 313
Durchsatzoptimierung, Datenparallelismus und Modellparallelismus 258
DVC 407

E

Echtzeitanforderungen, deterministische Latenzen 364
EDA-Architektur 327
Edge-Architektur 412
Edge-KI, Modellverteilung und Modellkontrolle 476
Einsatz von KI
NLP-Komponente 470
Reinforcement Learning 470
Trend- und Prognosemodelle 470
Embeddings 178
kontextuelle Embeddings 180
Word-Embeddings 179
End-to-End-Plattformen 406
kommerzielle 409
End-to-End-Systeme 76
Energieverbrauch
Betriebsmodelle und Skalierung 372
energetische Grundprofile 371
energetische Wirkungskette 373
Energieeffizienz 372
Lastprofile 371
Entscheidungshoheit 148
Erklärbarkeit, Merkmalswichtigkeitsanalyse 288
Ethically Aligned Design Frameworks 143
Ethik 101

architektonische Gestaltungsdimension 127
Autonomie, Abhängigkeit und Manipulation 134
Erklärbarkeit als Systemeigenschaft 129
Ethik-Leitlinien für KI 128
Fairness und Bias 129
gesellschaftliche Werte und normative Erwartungen 128
Grenzen 128
KI-Systeme 127
menschliche Kontrolle und Eingriffsmöglichkeiten 132
Querschnittsvorgaben für Architekturentscheidungen 128
Transparenz und Erklärbarkeit 130
Verantwortung und Rechenschaft 131
ETL/ELT 347
ETL-Modell
Datenextraktion 319
Laden in ein Zielsystem 320
Transformation 319
EU AI Act 134
Aufsichtsfähigkeit 139
begrenztes Risiko 136
Best Practices 141
Eingriffsfähigkeit 139
europäischer Regulierungsansatz 134
hohes Risiko 135
Kennzeichnung und Kontextoffenlegung 138
minimales Risiko 136
rechtliche Konsequenzen 140
Regulatorik 141
regulatorische Konformität 135
Risikoklassifizierung 135, 137
Risikotreiber 137
Sanktions- und Durchsetzungsregime 140
Transparenzanforderungen 137
unzulässiges Risiko 135
Evaluierung von LLM-Anwendungen
manuelle Evaluierung 451
modellbasierte Evaluierung 451
Scoring 450
vergleichende Evaluierung 452
Eventbasierte Systeme
Echtzeit-Streaming-ML 196
reaktive Muster 197
Event-Driven Architecture 432
Claim Check 433
Event-Bus 328
Event-Schemata 328
lose Kopplung 432
Producer, Router und Consumer 327

F

F1-Score 251
Fairness in ML-Systemen 278
Fallstudie, KI-gestütztes Störungsmanagement im ÖPNV 459
Feast 408
Feature Engineering 46, 90, 156
Feature Space 35
Feature Store 304
Federated Learning 480
synchronisierte verteilte ML-Systeme 201
verteilte Lernarchitekturen 476
verteiltes Modelltraining 200
Feedforward Neural Network 48
Fehlertoleranz 274, 401
generative KI-Systeme 275
Logging 274
Mean Time to Detection 274
Modellmanagement 274
Monitoring und Alerting 274
Orchestrierungsmechanismen 275
Ressourcenmanagement 274
Finetuning 188
aufgabenspezifischer Kopf 189
domänenspezifische Daten 188
Einfrieren von Modellschichten 189
Lernrate 189
Modellparameter 188
Multi-Task- Finetuning 439
Foundation Models, One Model, Many Tasks 421
FrugalGPT 273
Fully-Connected Netze 50

G

GenAI 162
Life Cycle 162
Generalisierungsfähigkeit 91
Generative Adversarial Networks 54
Generator und Discriminator 55

Generative KI 53
  anpassbare Modelle 300
  Architektonische Einordnung 58
  dynamische Modellportfolios 300
  Kontextpersonalisierung 300
Generative KI-Systeme 413
  agentische Systeme 442
  AI Query Router 442
  Antwortqualität 434
  architektonische Einordnung 413
  architektonische Schichtung 414
  Architektur-Patterns 413
  Arten 416
  bildgenerierende Modelle 418
  codegenerierende Sprachmodelle 417
  Definition 414
  Foundation Models 421
  Grundlagen und Qualitätsmerkmale 414
  Guardrails 414
  Integration in IT-Landschaften 426
  Interpretierbarkeit 415
  Layered Caching 440
  Low-Rank Adaptation 415
  Memory Cognition 441
  Modellauswahl 422
  multimodale Modelle 420
  Multi-Task-Finetuning 439
  musikgenerierende Modelle 420
  Plattform- und Integrationsarchitekturen 413
  Prompt-Engineering 434
  Prompt-Sanitization 415
  Prompt-Tests 414
  regelbasierte und generative Ansätze 441
  Retrieval-Augmented Generation 436
  Softwarearchitekturen 413
  Systemarchitekturen und Plattformen 413
  Wartbarkeit 415
Gold Standard 33
GPU 45
Ground Truth 33, 89
Grundlegende Metriken 250
Guardrails 456

H

H2O.ai 407
Halluzination 70
Hardwarearchitekturen
  HPC-Systeme 362
  Prozessorarchitekturen 361
Heatmaps 298
Hebb'sches Lernprinzip 41
Hierarchische Architekturen 198
  Ensemble-Architekturen 198
  geschichtete Modellstrukturen 199
HITL 105
HOTL 105
Human-Engineered Softwareentwicklung 71
  Qualitätsmerkmale 79
Human-Engineered Systems 23, 89
Human-in-the-Loop 105
  Grenzen prozessualer Kontrolle 477
Human-on-the-Loop 105
Hybride Architekturen
  deterministische und probabilistische Systemteile 478
  inhärente Unsicherheit 478
  Unsicherheitsmodellierung 478
Hybride KI 30

I

IEEE 143
Imputation 155
Inferenzarchitekturen
  Edge-nahe Inferenz 364
  serverseitige Inferenz 364
Informationsarchitektur, progressive Offenlegung 295
Infrastruktur
  Inferenz-Server 429
  Vektorindizes und Vektordatenbanken 429
Integrationsebenen
  AI-Engineering-Ebene 427
  Anwendungsebene 427
  Infrastruktur 428
  Modellentwicklung 428
Integrationsmuster 397
Integrationspunkte, Serviceklassen 398
Integrationsschichten 398
Interaktive Elemente
  Tooltips und Hilfetexte 296
  Undo/Redo 296
ISO/IEC 22989, KI-Systeme 27

ISO/IEC 42001
Artefakt- und Dokumentationsführung 142
Managementstrukturen 142
risikobasierter Ansatz 142
Überwachung und kontinuierliche Verbesserung 143
ISO/IEC 42001:2023 142

J

J-Kurve 85
Join-Strategien
Inner Join 335
Outer Joins 335
zeitbasierter Join 335

K

Kappa-Architektur 348
Append-Only-Log 326
Event-Replays 326
Kedro 408
KI
analytische 29
Generative 53
hybride 30
schwache 29
starke 29
symbolische 29
verantwortungsvolle 143
KI-Agenten 301
Agentenarchitekturen 301
agentenbasierte Systeme 302
Continual Learning 302
ReAct, Reflexion und CoT-basiertes Planen 301
KI-Algorithmen 175
Eingabedaten 175
KI-Anwendungen 62
Automobilbereich 67
Bildungssektor 67
E-Commerce 66
Einsatzfelder 63
Endnutzeranwendungen 66
Finanz- und Versicherungssektor 66
Gesundheitswesen 66
konzeptionelle Domänen 63
KI-Architekturen
architektonische Abgrenzung 104
Architekturtreiber 476
Datenflüsse, Datenqualität und Anpassung 476
Dokumentationsfähigkeit 111
hybride Architekturen 478
hybride Architekturen und Entwicklung 478
Inferenzarchitekturen 480
kontrollierbare Entwicklungspfade 105
Lizenzbedingungen 109
Verantwortung 479
KI-Betrieb
Abgrenzung zu MLOps und Werkzeugdarstellungen 354
architektonische Einordnung 354
architektonische Funktionsklassen 353
Architekturdomänen 354, 359
Ausführungsumgebungen 365
betriebliche Qualitätsdimensionen 355
betriebliche Steuerbarkeit 359
Betriebsinfrastruktur 359
Betriebsstabilität 368
dauerhafte Betriebsfähigkeit 353
Energie als Betriebsfaktor 371
Energie, Kosten und Governance 373
Energieverbrauch und ökologische Wirkungen 371
Feedback-Architekturen und ökologische Langzeitwirkungen 372
Infrastruktur- und Hardwaredomäne 360
Kontrollierbarkeit und Eingreifbarkeit 357
Kostenstrukturen 373
Kostenwirkungen von Infrastrukturentscheidungen 373
MLOps als Zusammenspiel von Architekturdomänen 359
Nachvollziehbarkeit und Rekonstruierbarkeit 356
ökologische Wirkung 373
Resilienz 368
Skalierungseigenschaften 369
Verfügbarkeit und Steuerbarkeit 367
Verfügbarkeit, Resilienz und Betriebsstabilität 367
versteckte Kosten 375
wirtschaftliche und ökologische Steuerbarkeit 353

Zeitabhängigkeit 355
Zuständigkeiten und Übergänge 359
KI-Compliance-Manager 84
KI-Dokumentation 111
Metadaten 111
KI-Engineer 81
KI-Engineering 24, 71
Artefakte 75
Compliance 78
Datenanforderungen 75
datengetriebene Ansätze 72
Datenwertschöpfungskette 84
datenzentrierte Ansätze 76
Domänenexperten 82
dynamische Umgebungen 73
erweiterte Verantwortungen 80
heterogene Eingabedaten 74
Modellarchitektur 76
Problemstellungen 71
Qualitätsmerkmale 79
Rollen 80
Softwarearchitekt 81
spezielle Rollen 83
KI-Entscheidungen, Fairness 70
KI-Gateway 455
KI-gestützte Anforderungserfassung 86
KI-gestütztes Störungsmanagement im ÖPNV
Architekturanforderungen 461
Ausgangssituation und Problemstellung 459
Einsatz von KI 470
Ergebnisse und Nutzen 473
Lösungsansatz 469
proaktives Betriebsmodell 460
Reflexion der Fallstudie 474
Risiken und Gegenmaßnahmen 472
Systemintegration 460
technische Umsetzung 470
KI-Governance 97, 147
Alignment 99
Compliance 99
Datenschutz 102
energetische Steuerung 373
Energieeffizienz 373
Ethik 102
Grundbegriffe 98
Human-in-the-Loop 475, 477
Infrastrukturkosten 375
IT-Governance und Unternehmensarchitektur 97
menschliche Aufsicht 475
Responsibility-by-Design 477
Risiko 102
Sicherheit 99
Steuerungsebenen 99
Zuordenbarkeit von Kostenströmen 375
KI-Integration
Architektur- und Integrationsstrategie 429
Architekturmuster 431
Event-Driven Architecture 432
fachliche Verortung 429
Integrationsebenen 427
Microservices 431
orchestrierungsbasierte Integration 433
Plug-in-basierte Integration 434
strategische KI-Integration 429
KI-Interfaces
Informationsarchitektur und Layout 295
Nutzerfeedback 295
Qualitätsindikatoren 296
Transparenz 294
UI-Elemente 297
Visualisierung von Daten und Ergebnissen 298
KI-Modelle
datenschutzrechtliche Speicherobjekte 116
Deployment 159
Modellauswahl und -training 157
Modellverwaltung und Retraining 470
offene 108
Transfer-Learning und Finetuning 186
Versionierung 159
Wartung 160
KI-Ökosysteme 108
offene 108
KI-Projekte 153
explorative Datenanalyse 154
Fallstudie und Praxisprojekt 459
Lebenszyklus 154
KI-Sicherheit 100
Algorithmensicherheit 100
architektonische Perspektive 117
Bedrohungen 118

Daten- und Modelllebenszyklen 125
datenbasierte Angriffe 119
Datensicherheit 100
Dokumentationsfähigkeit 126
Infrastrukturangriffe 122
Kontextbindung von Modellen und Schnittstellen 126
Kontrollierbarkeit, Nachvollziehbarkeit und Beobachtbarkeit 125
modellbasierte Angriffe 120
organisatorische und regulatorische Einbettung 126
Safety Cages 285
Schwachstellenanalysen und Penetrationstests 286
Sicherheit von Anfang an 124
Sicherheitsmaßnahmen und Monitoring 285
Sicherheitsmonitoring 285
Systemsicherheit 100
Threat Landscape 123
Trennung von Modellfunktion und Steuerung 124
Wirkungs- und Verantwortungsräume 125
Zugriffsrechtekontrolle 285
KI-Systeme 17
architektonische Einordnung 311
Architektur 23
Benutzeroberflächen 293
Betrieb 353
Betrieb und Qualitätsmerkmale 353
Compliance und Nachvollziehbarkeit 287
Datenakquisition und Datenannotation 312
Datenarchitektur 311
Datenmanagement und Datenverarbeitung 311
Datenpipelines und Datenarchitekturen 317
Definition 27
Dokumentation 111
Entscheidungshoheit 105
Entstehungs- und Nutzungskontexte 106
Entwurf und Entwicklung 153
Entwurfsmuster 190
Herausforderungen in der Entwicklung 181
Lebenszyklus 107
Mensch-KI-Interaktion 294
Nachvollziehbarkeit und Nachweisbarkeit 106
Nicht-Delegierbarkeit 105
Quantum Computing 479
rechtliche Anschlussfähigkeit 104
rechtliche Grenzen 103
rechtliche Zurechenbarkeit 103
Risiken 17
strukturelles Rückgrat 332
Urheberrecht 103–104
Verantwortung, Haftung und Zurechenbarkeit 104
Vorgehensmodelle 161
Zukunft der Softwarearchitektur 475
zukünftige KI-Architekturen 479
Zukunftsperspektiven 299
KI-Trends
Edge-KI 476
Federated Learning und dezentrale Trainingsansätze 476
KI-Visualisierung
interaktive Visualisierungen 298
Modellkonfidenzen und Merkmalsbedeutung 298
KI-VO 134
KI-Winter 25
neuronale Netze 45
Klassifikation 64
Klassifikationsmetriken 250
AUC-ROC 252
F1-Score 251
Genauigkeit 250
Präzision 251
Sensitivität 251
Klassifikationsproblem 33
Kommerzielle End-to-End-Plattformen 409
Konfliktauflösung, heuristische Verfahren 336
Kontextlänge, Kontextreduktion und Segmentierung 423
Kontrollierbarkeit
Eingriffspunkte 357
kontrollierte Übergänge 358
Rollouts und Rollback-Strategien 358

Kontrollierte Übergänge 411
Kopplung, organisatorisch-technische 107, 149
Kostenarchitekturen
  Cloud und On-Premises 374
  hybride Kostenlogiken 375
  Kostenlogik 374
  SaaS und Self-Hosting 374
Kreuzvalidierung 158
Künstliche Intelligenz 24
  Anpassungsfähigkeit 68
  Bausteine 27
  Datenabhängigkeit 69
  Definition 26
  ethische und gesellschaftliche Herausforderungen 69
  gesellschaftliche Risiken 70
  Grundlagen 23
  kognitive Grenzen 69
  Kontextabhängigkeit 69
  Mustererkennung 68
  Privatsphäre und Überwachung 71
  Risiken 70
  Skalierbarkeit 68
  Stärken 68
  technische und ökologische Grenzen 69
Künstliche Neuronale Netze 25
  Architekturen 47
  Grundlagen 40

L

Label-Encoding 156
Lambda-Architektur 348
  Batch-, Speed- und Serving-Layer 324
  Master Dataset 324
Large Language Models 26
  Chunking 417
  Finetuning und domänenspezifische Anpassung 86
  Positional Encoding 416
  Self-Attention 416
  Token- und Sequenzgrenzen 417
  Tokenisierung 416
  Transformer-Architektur 56
  Zustandslosigkeit 417
  Zustandsmanagement 417
Latent Space 49
Latenz und Skalierbarkeit, First-Token-Latenz 425
Latenzoptimierung
  Datenbank- und Datenverarbeitungsstrategien 257
  Hardware-Optimierungen 257
  Modellkomprimierung 258
  Monitoring und kontinuierliche Optimierung 258
  Netzwerktechnologien 257
  Software-Optimierung 257
  Verkehrsmanagement und Lastverteilung 257
Layered Caching
  mehrstufige Caching-Ansätze 440
  Prompt-Caching 441
Lebenszyklusmanagement 161
Lernen
  überwachtes 32
  unüberwachtes 34
  verstärkendes 37
Lernrate 44
LIME 255
Lizenzklassen, KI-Modelle, Daten und Artefakte 109
LLM-Anwendungen, Evaluierung 450
LLM-Jailbreaks 118
Log-Loss 252
Lösungsansatz, Bounded Contexts 469

M

Machine Learning Operations 77, 165
Mangelnde Datenqualität 182
Man-in-the-Middle 122
Manuelle Evaluierung, Reinforcement Learning from Human Feedback 451
Maschinelles Lernen
  Definition 30
  Lernansätze 31
  Mustererkennung und Generalisierung 31
  Problemstellungen und Anforderungen 171
  Problemstruktur, Datenbasis und Modellkomplexität 171
  sicherheitskritische Ereignisse 171
  siehe auch ML
  Trainings- und Produktivdaten 171

Maturity Model 86
MCP 443
Memory Cognition, Langzeitkontext 441
Merkmalsraum 35
Metaflow 408
Metriken
  grundlegende 250
  Klassifikationsmetriken 250
  systemische 254
Metrische Daten 167
Microservices
  KI-Gateways 432
  Resilienzmechanismen 471
  Service Mesh 432
Microservices-Architektur
  Integration für ML-Systeme 194
  modularisierte ML-Komponenten 193
MitM 122
ML 171
ML-Anwendungsfälle 240
  automatisiertes Programmieren 244
  Betrugserkennung 241
  Bildklassifikation 240
  Empfehlungssysteme 242
  Sprachverarbeitung 243
ML-Integration 246
  API-basierte Zugriffspunkte 247
  containerisierte Bereitstellung 247
  ereignisbasierte Integration 247
  Herausforderung 246
  Modellbereitstellung 247
  orchestrierte Laufzeiten 247
  Rückkopplung und Retraining 248
  Serverless-Varianten 247
  Serving und Monitoring 248
  Versions- und Freigabekontrolle 248
  Workflow-Integration 246
ML-Metriken 250
  Clustering und Ähnlichkeitsanalyse 253
  Klassifikation 250
  Kombination von Metriken 255
  kontinuierliche Beobachtung 255
  Rankingmetriken 255
  Regression 253
  systemische Metriken 254
ML-Modelle
  Bewertung 172
  Integration 245
  Metriken und Leistungsmessung 249
MLOps 77, 165, 303
  Automatisierung von Datenverarbeitung und Training 165
  Build-vs.-Buy-Entscheidung 405
MLOps-Engineer 82
MLOps-Werkzeuge, Überblick 406
ML-Problemstellungen
  bestärkendes Lernen 174
  Problemklassifikation 180
  überwachtes Lernen 172
  unüberwachtes Lernen 173
MLRun 408
ML-Systeme
  Architektur und Aufgaben 232
  Aufgaben und Anwendungsfälle 240
  Ausgaben 234
  Bias, Fairness und Qualitätssicherung 275
  Datenfluss und Pipelines 236
  Eingaben 233
  Ergebnisqualität und Reproduzierbarkeit 279
  eventbasierte Systeme 196
  Experimentmanagement und Metadaten-Tracking 239
  Federated-Learning-Topologien 200
  Fehlertoleranz 270
  hierarchische Architekturen 198
  Integration von ML-Modellen 245
  Integrationstechnologien 240
  Komponenten 233
  Microservices-Architektur 193
  Model-Deployment-Muster 225
  Modelle und Lernalgorithmen 238
  Model-Serving-Muster 220
  Model-Training-Muster 213
  monolithische Systeme 192
  Multi-Modal-Muster 200
  Pipeline-Architektur 195
  Pipeline-Architektur-Muster 202
  Prüfbarkeit 279
  Reproduzierbarkeit 279
  Robustheit 262
  Rückkopplung zwischen Eingaben und Ausgaben 235
  Sicherheit und Datenschutz 282
  Skalierbarkeit 260
  Speicher- und Zugriffslösungen 239

Stabilität 271
technische Performance 256
Topologie-Muster 189
Überwachung und Feedbackschleifen 239
Verfügbarkeit 272
Zuverlässigkeit 270
Zuverlässigkeit und Verfügbarkeit 270
Model Decay 77
Model Inversion 120
Model Stealing 121
Model Validator 82
Model-Deployment-Muster 225
A/B-Testing 230
Automated Deployment 228
Blue-Green Deployment 229
Canary Deployment 230
Edge Deployment 231
Manual Deployment 227
Pre-trained Model Utilization 228
Modellarchitektur
Dense-Modelle 422
Mixture of Experts 422
Modellauswahl
Anpassbarkeit und Multimodalität 425
Benchmarks 423
Datenschutz und Sicherheit 426
Kontextlänge 423
Kosten und Lizenz 424
Latenz und Skalierbarkeit 425
Modellgröße und Architektur 422
Trainingsdaten 424
Modellbasierte Evaluierung
LLM-as-a-Judge 451
RAGAS 452
Red-Blue-Team-Pattern 452
Modell-Drift 77, 183, 304
Aktualisierungsstrategien 185
Drift-Detektion 185
Ensemble-Ansätze 186
inkrementeller Drift 184
kontextabhängige Driftformen 185
plötzlicher Drift 183
potenzieller Drift 184
schrittweiser Drift 184
Ursachen 184
wiederkehrender Drift 184
Modelle, probabilistische 23
Modellentwicklung, Evaluation 428
Modellgröße
Parameteranzahl 422
Quantisierung 422
Modellversionierung 306
Model-Serving-Muster 220
A/B-Testing-Serving 223
Batch-Serving 221
Multi-Model-Serving 222
Online-Serving 221
Prompt-Engineering 222
Shadow-Model-Deployment 224
Transfer-Serving 224
Model-Training-Muster 213
Batch-Training 215
Federated Training 218
Finetuning Deployment 218
Incremental Training 217
Online-Training 216
Personalization 219
Transfer-Learning 217
Multi-Agenten-Kollaboration
Multiplexing 449
Swarm of Generative AI Agents 450
Multimodale Modelle
Early Fusion 421
Late Fusion 421
Multi-Task-Finetuning, Aufgabengewichtung 440
Musikgenerierende Modelle
Konditionierung 420
Urheberrechte und Persönlichkeitsrechte 420
zeitliche und rhythmische Strukturen 420

N

Nachvollziehbarkeit
Architekturartefakte 356
Prozess- und Entscheidungsdokumentation 289
Rekonstruktionsfähigkeit 356
strukturelle Kopplung 356
Nearline-Pipelines 396
Neuronale Netze
Architektonische Einordnung 52
Batches 177
Normalisierung und Standardisierung 177
numerische Eingabestrukturen 177

Nicht-Delegierbarkeit 149
Nicht-Determinismus 181
  Reproduzierbarkeit und Validierbarkeit 182
Nominale Daten 166
Notbetriebsfähigkeit 401
Nutzerzentrierung
  Personas und Use Cases 294
  Usability-Tests 294

O

OECD AI Principles 143
Offene KI-Modelle 108
  Missbrauchsrisiken 109
  Offenheitsgrade 108
Offene KI-Ökosysteme 108
One-Hot-Encoding 156, 177
  Dummy-Variable-Trap 178
Orchestrierungsbasierte Integration, Orchestrator 433
Ordinale Daten 167
Organisatorisch-technische Kopplung 107, 149
Overfitting 158, 169

P

Perzeptron 41
Pipeline-Architektur 195
  Active-Feedback-Loop-Muster 210
  automatisierte ML-Workflows 196
  Dynamic-Pipeline-Orchestration-Muster 210
  End-to-End-Pipelines 207
  ETL-orientierte Pipelines 205
  Federated Pipelines 211
  Hybrid-Pipeline-Muster 212
  Iteration-Feedback-Loop-Muster 209
  modellzentrierte Pipelines 206
  Modular-Pipeline-Muster 207
  Parallel-Pipeline-Muster 208
  Reinforcement Feedback Pipeline 213
  sequenzielle Datenpipelines 195
  Streaming-Pipelines 208
Pipelines, Arten 396
Planung, graphenbasierte Planung 449
Plattformen
  kommerzielle End-to-End-Plattformen 409
  Überblick 406
PoC 158
Prefect 408
PrimeHub 408
Probabilistische Modelle 23
Prompt-Engineering 222, 434
  Arten von Prompts 435
  Chain-of-Thought 435
  Few-Shot-Prompting 435
  geschlossene Prompts 435
  offene Prompts 435
  Role-Playing 436
  strukturierte Prompts 435
  Zero-Shot-Prompting 435
Prompt-Injection 118
Proof of Concept 158
Provenienz, externe KI-Artefakte 107
Prozessorarchitekturen
  CPUs 361
  DPUs 362
  GPUs 361
  LPUs 362
  NPUs 362
  QPUs 362
  TPUs und KI-Beschleuniger 362

Q

Quality of Service 258
Quantenalgorithmen 479
Quantum Computing 479
  hybride Architekturen 479
  Quantenalgorithmen 479

R

RAG *siehe* Retrieval-Augmented Generation
RAG-Assessment 452
Rankingmetriken
  Mean Reciprocal Rank 255
  Precision@k 255
Recurrent Neuronales Netz 48
Regeln, deterministische 23
Regression 64, 253
Regressionsmetriken 253
  Bestimmtheitsmaß 253
  Mean Absolute Error 253
  MSE und RMSE 253

Regulatory Sandboxes 144
  architektonische Experimentierräume 144
  Architekturen 145
  KI-Lebenszyklus 146
  soziotechnische Systemräume 145
  temporäre Zielarchitekturen 144
  Übergang in den Regelbetrieb 146
  Umgang mit Unsicherheit 145
Regulierungsansätze, Datenschutz 116
Reinforcement Learning, Architektonische Einordnung 39
Reproduzierbarkeit 78, 304
  Prozessdokumentation 280
  Versionierung und Änderungsprotokoll 280
Resilienz 411
  architektonische Entkopplung 367
  kaskadierende Effekte 367
  Redundanz, Replikation und Lastverteilung 368
  Rückfallstrukturen 368
Responsibility-by-Design 481
  Entscheidungs-Gates 477
  Regulierung als architektonischer Treiber 478
  Verantwortung als technische Systemeigenschaft 477
Retrieval-Augmented Generation 244, 436, 453
  Adaptive Retrieval 438
  Context Enrichment 438
  Embedding 437
  Ensemble Retrieval 438
  Intelligent Re-Ranking 438
  Iterative Retrieval 438
  Metriken 255
  Query Transformation 438
  RAG-Architekturen 437
  Retrieval-Prozess 437
  Simple RAG 437
  Talk to your documents 437
  Wissensgraphen 439
Risiken und Gegenmaßnahmen
  Bias in Trainingsdaten 472
  Blackbox-Entscheidungen 472
  Cloud-Abhängigkeit 473
  Concept Drift 473
Robustheit
  Distributional Shift 264
  Fallback-Mechanismen 263
  Fehlerpropagation 267
  Fehlertoleranz 262
ROUGE 255
Rückführbarkeit 411
Rückverfolgbarkeit 72

S

Sample-Effizienz 39
Sampling Bias 184
Sandboxes, Regulatory 144
Schwache KI 29
Scoring, Task Success Rate 451
Semistrukturierte Daten 167
Service Mesh 455
Serving-Layer 325
SHAP 255
Sicherheitsmonitoring
  Ausgabedaten 285
  Eingabedaten 285
  Modulinteraktionen 286
  Systemverhaltensüberwachung 285
  Zusammenspiel mit Logging und Zugriffskontrolle 286
Sicherheitsrisiken
  Fehlkonfiguration 284
  Prompt Injection 284
Skalierbarkeit
  asynchrones Training 261
  Datenparallelismus 260
  Modellparallelismus 261
  Parameter-Server-Modell 260
  Ring-AllReduce 260
  Straggler-Problem 261
  synchrones Training 261
Skalierung
  datenseitige 369
  Dimensionen 369
  horizontale 370
  infrastrukturelle 369
  langfristige Infrastrukturstrategie 371
  modellseitige 369
  Nutzungsskalierung 369
  Training, Inferenz und Datenpipelines 370
  vertikale 370

Softwarearchitekten
KI-Governance 97
Rolle und Aufgabenprofil 476
Softwarearchitektur
normative Disziplin 480
Problemraum 479
Recht als Gestaltungsrahmen 104
Softwareentwicklung, Spec-driven Development 479
Spec-driven Development 481
Speed-Layer 325
Stabilität
End-to-End-Optimierung 272
Ensembling und Redundanz 271
Multi-Step Chains 271
Retrieval-Augmented Generation 272
Starke KI 29
Strategische KI-Integration
Capability Maps 430
Domain-Driven Design 429
Wardley Mapping 430
Streaming, Apache Kafka 470
Streaming-Pipelines 396
Streaming-Verarbeitung
dynamische Zustandsverwaltung 322
Semantik, Garantien und Rückstaukontrolle 322
State Store 322
Strukturierte Daten 167
Symbolische KI 29
Systemintegration
APIs und Ereignisströme 461
entkoppelte Schicht 460
Lagebild 460
Systemische Metriken 254
Fairness und Robustheit 254
Latenz und Durchsatz 254
Speichereffizienz 254
Verfügbarkeit und Ausfallsicherheit 254

T

Target Encoding 157
TDSP 164
Team Data Science Process 164
Operationalisierung von KI-Projekten 165
Technische Performance
Durchsatz 258
Latenz 256
Latenzoptimierung 256
Technische Umsetzung
KI-Modelle 470
Microservices 470
Observability und Security 471
Streaming 470
Tensoren 177
Testen von KI-Systemen, Metriken 77
Textgenerierungsmetriken
BLEU 255
ROUGE 255
Tiefe Neuronale Netze 46
Traceability 72
Trainings- und Testdatensätze 158
Trainingsarchitekturen, Cluster- und Pool-Architekturen 363
Transfer-Learning 187
Feature-Extractor 187
Modellkapazität 188
Quell- und Zielaufgabe 187
vortrainierte Modelle 187
Transformer-Architektur 56
Attention 56
Embedding und Positional Encoding 56
Masked Multi-Head Attention 58
Tokens 56
Transparente Datenkette 106, 149
Transparenz
Darstellung erklärbarer Ausgaben 295
Unsicherheitsdarstellung 295
Turing-Test 25

U

Übergänge, kontrollierte 411
Überwachtes Lernen 32
architektonische Einordnung 33
Klassenungleichgewicht 173
Labelqualität und Annotation 173
UI-Elemente
Nachvollziehbarkeit von Interaktionen 298
Prompt-Felder und Konfiguration 297
Underfitting 158, 169
Unstrukturierte Daten 167
Unüberwachtes Lernen 34
architektonische Einordnung 36
Bewertung 174
Distanz- und Ähnlichkeitsmetriken 174

V

Validierung 35
Variational Autoencoder 54
  Latent Space 54
Verantwortungsvolle KI 143
Verfügbarkeit
  dynamische Komponenten 273
  Fehlerzonen 272
  LLMOps und DataOps 273
  Mean Time Between Failures 272
  Mean Time to Repair/Recover 272
  Modularität und Skalierbarkeit 272
  ressourcenschonende KI 273
Vergleichende Evaluierung, A/B-Testing 452
Verstärkendes Lernen 37
  Agent und Umgebung 38
  modellbasiertes Lernen 38
  modellfreies Lernen 38

W

Wardley Mapping 430
WhyLabs 408

Z

Zeitabhängigkeit
  Data Drift und Concept Drift 356
  Qualitätsverschiebungen 356
ZenML 408
Zukunft der Softwarearchitektur von KI-Systemen, Trends und architektonische Implikationen 476
Zukunftsperspektiven
  Compound AI 300
  Generative KI 299
  KI-Agenten 301

Praxiswissen Softwarearchitektur für KI-Systeme

- Grundlagen moderner Softwarearchitekturen für KI-Systeme
- Entwicklung skalierbarer, wartbarer und erweiterbarer KI-Lösungen
- Praktische Ansätze zur Integration von KI in bestehende IT-Landschaften

In diesem Buch lernen Sie die zentralen Architekturkonzepte, die für den Entwurf und die Umsetzung von KI-Systemen notwendig sind, kennen. Es richtet sich an Softwareentwickler, Softwarearchitekten und Data Scientists, die Machine-Learning- und Generative-KI-Lösungen entwickeln und in bestehende Systeme einbinden wollen.

Die Autoren waren maßgeblich an der Entstehung des SWARC4AI-Lehrplans beteiligt. Sie zeigen Ihnen, wie Sie grundlegende Prinzipien der Softwarearchitektur auf KI-Anwendungen anwenden können – von der Analyse spezifischer Qualitätsanforderungen über den Umgang mit Daten- und Modelllebenszyklen bis zur strukturierten Integration von KI-Komponenten in bestehende Softwarelandschaften.

Neben den theoretischen Grundlagen lernen Sie Architektur-Patterns kennen, die im KI-Kontext besonders relevant sind. Dazu gehören Muster für Datenpipelines, Trainings- und Bereitstellungsarchitekturen, Modellverwaltung sowie das Zusammenspiel klassischer Komponenten mit KI-basierten Bausteinen. Anhand praxisnaher Beispiele lernen Sie, KI-Systeme so zu gestalten, dass sie robust, erweiterbar und langfristig wartbar bleiben – zentrale Anforderungen in einer dynamischen technologischen Umgebung.

Mahbouba Gharbi ist Geschäftsführerin und Chef-Architektin bei ITech Progress GmbH und aktives Mitglied im iSAQB® e. V. Sie ist Autorin zahlreicher Fachartikel und Bücher, regelmäßige Sprecherin auf internationalen Konferenzen sowie Kuratorin des SWARC4AI-Lehrplans.

Prof. Dr. Holger Klus lehrt und forscht in den Bereichen Softwareentwicklung und IT-Architekturen. Zudem berät er Unternehmen bei der langfristigen Planung und Weiterentwicklung ihrer IT-Landschaften.

Holger Tiemeyer ist erfahrener Senior Softwarearchitekt und aktives Mitglied des iSAQB® e. V. Er gibt Schulungen zur Softwarearchitektur, hält Vorträge an Fachhochschulen, ist Autor von Fachartikeln und Referent auf internationalen Konferenzen.

Dr.-Ing. Meng Zhang ist aktives Mitglied im iSAQB® e. V. und erforscht seit 2014 am Institut für Software and Systems Engineering der TU Clausthal die Grundlagen sicherer Softwarearchitekturen für KI-gestützte und autonome cyberphysische Systeme. Seine Expertise umfasst u. a. KI-Engineering und hybride Softwarearchitekturen.

Gedruckt in Deutschland
Mineralölfreie Druckfarben
Zertifiziertes Papier

€ 49,90 (D)

ISBN 978-3-98889-059-7