Fachhochschule der Wirtschaft
-FHDW-
Paderborn

Bachelorthesis

Thema:

Effiziente Workload-Automatisierung durch Self-Service und Künstliche Intelligenz: Möglichkeiten bei Streamworks

Prüfer:
<< Erstprüfer >>
<< Zweitprüfer >>

Verfasser:
Ravel-Lukas Geck
Lagesche Str. 258
32758 Detmold

Wirtschaftsinformatik
Business Process Management

Eingereicht am:

20.10.2025

Inhaltsverzeichnis

1 Einleitung

Die Automatisierung von Enterprise-Workloads ist zu einem zentralen Erfolgsfaktor moderner IT-Landschaften geworden. Trotz erheblicher Fortschritte bei der Standardisierung von Geschäftsprozessen bleibt der Zugang zu leistungsfähigen Workload-Management-Systemen wie Streamworks weitgehend auf einen kleinen Kreis spezialisierter Experten beschränkt. Diese Exklusivität erzeugt Engpässe und verhindert eine umfassende Ausschöpfung der bestehenden Automatisierungspotenziale.

1.1 Problemstellung und Motivation

Streamworks ist derzeit primär auf eine GUI-basierte Bedienung durch Experten ausgelegt. Fachanwender aus den Geschäftsbereichen müssen ihre Anforderungen, beispielweise einen täglich automatischen File-Transfer, an das Orchestration-Team weiterleiten. Dieses übersetzt die Anforderung manuell in eine komplexe Stream-Konfiguration. Interviews mit Streamworks-Spezialisten bestätigen, dass ein erheblicher Teil dieser Anfragen standardisierbaren Mustern folgt, die keine individuelle Expertenprüfung erfordern. Der Einsatz hochqualifizierter Fachkräfte für repetitive Aufgaben stellt somit eine ineffiziente Ressourcennutzung dar.

Ein möglicher Ansatz zur Reduktion dieser Engpässe ist die Nutzung der XML-Schnittstelle von Streamworks. Prozessdefinitionen können hierüber importiert und exportiert werden, was die Grundlage für eine automatisierte Generierung von Streams bildet. Moderne Large Language Models (LLMs) eröffnen in diesem Zusammenhang neue Möglichkeiten: Sie sind in der Lage, natürlichsprachliche Anforderungen in strukturierte Datenformate wie XML zu übersetzen. So könnte ein Fachanwender künftig ein Self-Service-Formular ausfüllen oder in natürlicher Sprache eine Anforderung formulieren, die vom LLM in eine valide XML-Definition überführt und automatisch in Streamworks importiert wird.

Dieses Konzept adressiert nicht nur die Effizienzfrage, sondern auch die Benutzerautonomie. Fachanwender wären in der Lage, Standard-Jobs eigenständig zu initiieren, während Experten sich auf komplexere und strategisch relevante Aufgaben konzentrieren könnten. Die vorliegende Arbeit untersucht daher, wie ein KI-gestütztes Self-Service-System die Lücke zwischen fachlicher Anforderung und technischer Umsetzung schließen und gleichzeitig die Qualität sowie die Nachvollziehbarkeit der Konfigurationen gewährleisten kann.

1.2 Zielsetzung der Arbeit

Das zentrale Ziel dieser Arbeit besteht in der Konzeption und prototypischen Entwicklung eines hybriden Self-Service-Systems zur Erstellung von Streamworks-Konfigurationen. Im Mittelpunkt steht eine benutzerfreundliche Weboberfläche, über die Fachanwender ihre Workload-Anforderungen in natürlicher Sprache formulieren können. Diese Eingaben werden von einem Large Language Model automatisiert in valide XML-Definitionen übersetzt, die anschließend in Streamworks importiert werden können. Auf diese Weise soll die Lücke zwischen fachlicher Anforderung und technischer Umsetzung geschlossen werden, ohne dass tiefgreifende Streamworks-Kenntnisse erforderlich sind.

Neben dieser Kernfunktion wird ein Retrieval-Augmented-Generation-basiertes Supportsystem realisiert, das auf bestehende Dokumentationen, Best-Practice-Beispiele und Konfigurationsvorlagen zugreift. Es erweitert den Prototypen um eine intelligente Wissensdatenbank, die typische Fragen adressiert und den Self-Service-Prozess unterstützt.

Die Bewertung des Prototyps orientiert sich nicht an einer vollständigen Abdeckung aller denkbaren Anwendungsfälle, sondern an der Machbarkeit und dem nachweisbaren Mehrwert. Im Vordergrund stehen dabei folgende Zielgrößen:

- **Funktionale Machbarkeit:** Nachweis, dass natürlichsprachliche Anforderungen automatisiert in valide XML-Definitionen übersetzt und in Streamworks importiert werden können; Zielgröße: *Fehlerquote unter 5 % bei der XSD-Validierung.*
- **Effizienzsteigerung:** Reduktion des Arbeitsaufwands im Vergleich zum bestehenden manuellen Prozess; Zielgröße: *mindestens 50 % Zeitersparnis bei der Erstellung von Standard-Streams.*
- **Benutzerautonomie:** Fachanwender sollen in die Lage versetzt werden, Standard-Workloads eigenständig zu initiieren, ohne Expertenwissen; Zielgröße: *Reduktion des erforderlichen Experten-Supports um mindestens 70 % bei Standardfällen.*
- **Qualitativer Mehrwert:** Das RAG-basierte Supportsystem soll relevante und verständliche Antworten auf typische Anwenderfragen liefern; Zielgröße: *mindestens 80 % korrekt und nachvollziehbar beantwortete Standardfragen.*

1.3 Methodisches Vorgehen

Die Arbeit folgt einem praxisnahen und agilen Forschungsansatz, der theoretische Fundierung mit einer prototypischen Umsetzung verbindet. Methodisch wird dabei ein vierstufiges Vorgehen gewählt, das von der Grundlagenanalyse über die Anforderungsaufnahme bis hin zur Implementierung und Evaluation reicht. Jede Phase baut auf den Ergebnissen der vorherigen auf und schafft so einen strukturierten Rahmen für die Bearbeitung der Forschungsfrage.

Die Vorgehensweise lässt sich wie folgt zusammenfassen:

- **Grundlagenanalyse:** Durchführung einer systematischen Literaturrecherche zu Workload-Automatisierung, Self-Service-Konzepten, Large Language Models und Retrieval-Augmented Generation. Ziel ist die Schaffung eines fundierten theoretischen Bezugsrahmens.
- **Anforderungsanalyse:** Durchführung und Auswertung von Experteninterviews mit Streamworks-Spezialisten, um bestehende Herausforderungen, typische Anwendungsfälle und Automatisierungspotenziale zu identifizieren.
- **Prototypische Entwicklung:** Agile Umsetzung eines Self-Service-Portals mit Integration eines LLM zur automatisierten XML-Generierung sowie eines RAG-basierten Q&A-Systems. Dabei wird ein iteratives Vorgehen nach dem Prinzip des Minimum Viable Product (MVP) verfolgt.
- **Evaluation:** Bewertung des entwickelten Prototyps anhand quantitativer Metriken (z. B. Fehlerquote bei der XML-Validierung, Generierungszeit) sowie qualitativer Kriterien (z. B. Benutzerfeedback, Praxistauglichkeit des RAG-Systems).

1.4 Aufbau der Arbeit

Die Bachelorarbeit ist klar strukturiert und führt von den theoretischen Grundlagen über die Analyse der Streamworks-Umgebung und die Konzeption bis hin zur technischen Umsetzung, Evaluation und Reflexion. Auf diese Weise wird ein roter Faden geschaffen, der sowohl wissenschaftliche Fundierung als auch praktische Relevanz sicherstellt.

Der Aufbau gliedert sich in folgende Kapitel:

2 Theoretische Grundlagen

In diesem Kapitel werden die theoretischen und technologischen Grundlagen erarbeitet, die für das Verständnis der Arbeit notwendig sind. Dabei werden zunächst die Prinzipien der Workload-Automatisierung und die Rolle von Streamworks als Plattform erläutert. Anschließend wird auf XML als technisches Austauschformat eingegangen, bevor moderne KI-Technologien wie Large Language Models (LLMs) und Retrieval-Augmented Generation (RAG) vorgestellt werden. Den Abschluss bildet eine Betrachtung von Self-Service-Konzepten in der IT-Automatisierung, die den konzeptionellen Rahmen für die in dieser Arbeit entwickelte Lösung bilden.

2.1 Workload-Automatisierung mit Streamworks

Workload-Automatisierung (WLA) ist die Grundlage für effiziente, zuverlässige und skalierbare IT-Prozesse in modernen Unternehmen. Im Folgenden wird zunächst der Begriff definiert und seine Relevanz für die Praxis dargestellt. Anschließend wird mit Streamworks ein konkretes WLA-System eingeführt, dessen Architektur, Kernfunktionen und typische Anwendungsfälle erläutert werden.

2.1.1 Definition und Relevanz von Workload-Automatisierung

Workload-Automatisierung (WLA) bezeichnet den Einsatz spezialisierter Software zur zentralen Steuerung, Verwaltung und Überwachung von IT-Prozessen[1]. Diese Prozesse umfassen typischerweise zeit- oder ereignisgesteuerte Abläufe wie Batchprozesse, Dateiübertragungen und API-Calls[2]. Im Gegensatz zu einfachen, isolierten Skripten oder Task-Schedulern (wie z. B. Windows Task-Scheduler oder Cronjobs unter Unix), die oft nur auf einzelnen Systemen agieren, ermöglichen WLA-Systeme die Orchestrierung komplexer, voneinander abhängiger Workflows über heterogene IT-Landschaften hinweg. Sie integrieren Abhängigkeiten, Kalender und Geschäftslogiken, um eine durchgängige Prozesskette sicherzustellen[3]. Die Bedeutung für Unternehmen liegt vor allem in der Steigerung von Effizienz, Zuverlässigkeit und Skalierbarkeit[4]. Durch die Automatisierung wiederkehrender Aufgaben werden manuelle Eingriffe und das damit

[1] Vgl. Beta Systems. (2024). Was ist Workload Automation?, S. 1
[2] Vgl. Arvato Systems. (2020). Streamworks Produktbeschreibung.pdf, S. 1
[3] Vgl. Sabharwal, N., & Kasiviswanathan, S. (2023). 1. Introduction to Workload Automation, S. 13f
[4] Vgl. Twing, D. (2022). Global Workload Automation Market Size and Forecast 2022 to 2027.pdf, S. 4

verbundene Fehlerrisiko reduziert, Personalressourcen entlastet und die Prozessqualität nachhaltig verbessert. In der heutigen schnelllebigen digitalen Geschäftswelt ist WLA unverzichtbar, um die Agilität zu erhöhen, Compliance-Anforderungen zu erfüllen und Kosteneinsparungen zu realisieren[5].

2.1.2 Streamworks als Enterprise WLA-Plattform

Streamworks von Arvato Systems ist eine „Workload-Automation- und Service-Orchestration-and-Automation-Platform“, die für den Einsatz in heterogenen IT-Landschaften mit einem hohen Anspruch an Automatisierung und Standardisierung konzipiert wurde [1]. Die Plattform ermöglicht die zentrale Erstellung, Steuerung und Überwachung von Workflows, indem einzelne Prozessschritte, sogenannte Jobs, miteinander verknüpft werden [2].

2.1.2.1 Architektur und Plattformreichweite

Die technische Architektur von Streamworks ist modular aufgebaut und besteht aus mehreren zentralen Komponenten, die auf einem Windows-Betriebssystem laufen und sowohl auf physischer als auch auf virtueller Hardware betrieben werden können [3]. Das Fundament bildet eine zentrale Datenbank auf Basis eines Microsoft SQL Servers, in der alle Stamm-, Laufzeitdaten und Systemnachrichten persistent gespeichert werden [4]. Die Prozesssteuerung wird vom Processing Server übernommen, der als Engine die gesamte Kommunikation mit den angebundenen Systemen sowie die Jobdurchführung verantwortet, während ein Application Server auf Basis von Microsoft IIS den Zugriff für Benutzer über verschiedene Clients wie einen Desktop-Client oder eine Web App steuert [5].

Ein wesentliches Merkmal der Plattform ist ihre Fähigkeit zur plattformübergreifenden Automatisierung (Cross-Platform-Automation), wobei schlanke Agenten auf den Zielsystemen die Ausführung der Jobs übernehmen [6]. Streamworks unterstützt dabei eine Vielzahl gängiger Betriebssysteme aus der Windows-, Unix- und Linux-Welt sowie Mainframe-Umgebungen wie z/OS, System i (AS/400) und BS2000 [7]. Zusätzlich ist ein Betrieb der Backend-Services in Containern über den Azure Kubernetes Service möglich, was die Fähigkeit zur Integration in moderne Cloud-Infrastrukturen unterstreicht [8].

[5] Vgl. Beta Systems. (2024). Was ist Workload Automation? - Beta Systems.pdf, S. 1ff

2.1.2.2 Kernfunktionen und Konzepte

Streamworks bildet IT-Prozesse durch die Verknüpfung einzelner Jobs zu sogenannten Streams ab, deren Ausführung sowohl zeit- als auch ereignisgesteuert erfolgen kann [9]. Um Konflikte bei parallel laufenden Prozessen zu vermeiden, können logische Ressourcen den Zugriff auf Systeme oder Dateien regeln [10]. Für die Integration in unternehmensweite Anwendungslandschaften bietet die Plattform spezialisierte Schnittstellen, insbesondere für die Anbindung von SAP® Netweaver Systemen über die zertifizierten Module jexa4S und jexa4BI, mit denen SAP-Batch-Jobs, -Prozessketten und -Info-Packages gesteuert und überwacht werden können [11].

Zur Förderung von Standardisierung und Effizienz implementiert Streamworks ein Master-/Realkonzept, bei dem ein Masterstream als vollständig definiertes Template für gleichartige Jobnetze dient und per Knopfdruck als neuer, anpassbarer Realstream instanziiert werden kann [12]. Die Sicherheit und Nachvollziehbarkeit von Konfigurationsänderungen wird durch ein detailliertes Rollen- und Berechtigungssystem sowie eine integrierte Versionierung gewährleistet, die das Verwalten unterschiedlicher Versionen von Prozessketten und die Anbindung an externe Versionskontrollsysteme wie GIT ermöglicht [13]. Darüber hinaus verfügt die Plattform über eine eigene Lösung zur Durchführung und zentralen Kontrolle von Dateiübertragungen, die über die Funktionalität des Standard-FTP-Protokolls hinausgeht [14].

2.1.2.3 Typische Anwendungsfälle

Die funktionalen Fähigkeiten von Streamworks ermöglichen die Abbildung vielfältiger Automatisierungsszenarien in Unternehmen. Ein zentraler Anwendungsfall ist die Automatisierung klassischer Batchprozesse, wie sie beispielsweise für nächtliche Datenverarbeitungen, Uploads oder Downloads benötigt werden [15]. Darüber hinaus wird die Plattform zur Orchestrierung komplexer, systemübergreifender Geschäftsprozesse eingesetzt, etwa bei Monatsabschlüssen, die sowohl SAP- als auch Non-SAP-Systeme involvieren [16].

Im Bereich der IT-Betriebsautomatisierung, auch als IT-Housekeeping bezeichnet, dient Streamworks der Synchronisierung von betrieblichen Notwendigkeiten mit geschäftlichen Abläufen, indem regelmäßige Wartungsaufgaben wie Server-Neustarts, Backups oder das automatisierte Einspielen von Patches gesteuert werden [17]. Ein weiteres typisches Einsatzgebiet ist die gesteuerte Daten-Synchronisation, bei der die Übertragung und Verarbeitung von Informationen zwischen unterschiedlichen Applikationen und

Datenbanken automatisiert wird, um die Konsistenz und Verfügbarkeit von Unternehmensdaten sicherzustellen [2].

2.1.2.4 Herausforderungen und Komplexität

Die dargestellte Funktionsvielfalt und die Fähigkeit zur Orchestrierung über heterogene Systemlandschaften hinweg ermöglichen die Abbildung komplexer Unternehmensprozesse, führen jedoch zwangsläufig zu einer hohen Konfigurationskomplexität [1]. Die manuelle Erstellung und Verwaltung von Streams erfordert daher tiefgreifendes Expertenwissen, weshalb die Implementierung von routinierten Experten begleitet wird, die gemeinsam mit dem Anwender passende Pilotprozesse auswählen und automatisieren [18]. Diese Notwendigkeit spezialisierter Fachkräfte stellt in der Praxis eine signifikante Hürde für Fachanwender dar und begründet den zentralen Engpass, der in dieser Arbeit durch einen KI-gestützten Self-Service-Ansatz adressiert wird.

2.2 XML als Konfigurationsstandard in Streamworks

Für die technische Umsetzung der Workload-Automatisierung spielt XML (Extensible Markup Language) eine zentrale Rolle. Dieses Kapitel beschreibt die Grundlagen der Sprache, ihre spezifische Bedeutung für den Import und Export in Streamworks sowie die Herausforderungen, die mit der manuellen Erstellung von Streams verbunden sind.

2.2.1 Grundlagen der Extensible Markup Language (XML)

Die Extensible Markup Language, kurz XML, ist eine Auszeichnungssprache, die zur Darstellung hierarchisch strukturierter Daten in einem textbasierten Format dient [1]. XML ist eine Untermenge der Standard Generalized Markup Language (SGML) und wurde mit dem Ziel entwickelt, Daten im Web auf eine Weise bereitzustellen, zu empfangen und zu verarbeiten, wie es zuvor mit HTML möglich war [1]. Ein zentrales Merkmal von XML ist die Trennung von Daten und deren Darstellung. Die Struktur eines XML-Dokuments wird durch verschiedene Schlüsselmerkmale definiert. Sogenannte *Tags* werden verwendet, um Daten zu umschließen und zu benennen, wodurch *Elemente* entstehen [1]. Diese Elemente können wiederum weitere Elemente enthalten, was die hierarchische Struktur erzeugt. Elemente können zudem *Attribute* besitzen, die zusätzliche Informationen in Form von Name-Wert-Paaren bereitstellen [1].

Ein wesentlicher Vorteil von XML ist die gleichzeitige Menschen- und Maschinenlesbarkeit [1]. Die textbasierte und selbsterklärende Syntax ermöglicht es Entwicklern und

Fachanwendern, die Struktur und die Inhalte eines XML-Dokuments leicht zu verstehen. Gleichzeitig können Computersysteme die Daten durch sogenanntes Parsen eindeutig interpretieren und verarbeiten. Dies fördert die Interoperabilität zwischen verschiedenen Systemen und erleichtert den Datenaustausch [1].

2.2.2 Die Rolle von XML für den Import/Export in Streamworks

Innerhalb der Workload-Automation-Plattform Streamworks von Arvato Systems dient XML als Standardformat für den Import und Export von Prozessdefinitionen, den sogenannten Streams [2]. Mithilfe einer Export/Import-Utility können sämtliche Prozess-Definitionen oder Stammdaten im XML-Format zwischen verschiedenen Mandanten oder unterschiedlichen Streamworks-Systemen transferiert werden [2]. Dies ermöglicht beispielsweise ein kontrolliertes Deployment von Prozessketten zwischen Test- und Produktionsumgebungen [2].

Die XML-Datei Beispiel-Stream.xml dient als Beispiel für eine solche Stream-Definition. Die Struktur der Datei zeigt eine tiefe Verschachtelung von Elementen, die die gesamte Konfiguration des Streams abbildet. Das Wurzelelement <ExportableStream> umschließt das <Stream>-Element, welches grundlegende Informationen wie den Namen des Streams (<StreamName>), eine Beschreibung (<ShortDescription>) und den Pfad (<StreamPath>) enthält [3]. Innerhalb des <Stream>-Elements befindet sich das Element <Jobs>, das eine Liste von <Job>-Elementen enthält. Jeder Job repräsentiert einen einzelnen Prozessschritt innerhalb des Streams, wobei die Verknüpfung der Jobs die AblaufloLogik definiert [3]. Diese hierarchische und verschachtelte Struktur ist typisch für die Konfiguration von komplexen Automatisierungsprozessen in Streamworks.

Zur Sicherstellung der formalen Korrektheit und Importfähigkeit der XML-Definitionen verwendet Streamworks zudem XML Schema Definitionen (XSD). Diese legen die erlaubten Elemente, Attribute und deren Struktur fest und dienen damit als ‚Bauplan' für valide Stream-Konfigurationen. Die XSD-Validierung spielt insbesondere im Kontext automatisierter Generierung – wie sie in dieser Arbeit untersucht wird – eine zentrale Rolle, um fehlerhafte oder unvollständige XML-Dateien frühzeitig auszuschließen

2.2.3 Herausforderung: Die manuelle Erstellung von Streams

Die manuelle Erstellung von Stream-Definitionen in Streamworks stellt eine erhebliche Herausforderung dar und ist die zentrale Problemstellung dieser Arbeit. Obwohl die Konfiguration über eine grafische Benutzeroberfläche (GUI) möglich ist, bildet diese die

gleiche hohe Komplexität ab, die sich auch in der XML-Struktur widerspiegelt [2]. Anwender müssen eine Vielzahl von Parametern, Abhängigkeiten und Regeln definieren, um einen funktionierenden und robusten Automatisierungsprozess zu erstellen [2].

Diese Komplexität führt im aktuellen Prozess zu einem signifikanten Engpass: Fachanwender ohne tiefgreifende technische Kenntnisse müssen ihre Anforderungen – oft über heterogene Kanäle wie E-Mails oder Word-Dokumente – an ein spezialisiertes Orchestration-Team weiterleiten. Dieses Team übersetzt die fachliche Anforderung manuell in die technische Stream-Konfiguration. Unvollständige oder inkonsistente Angaben führen dabei zu zeitaufwendigen Rückfragen und Abstimmungsschleifen, was die Durchlaufzeiten erheblich verlängert [4].

Die Einarbeitung in die manuelle Konfiguration erfordert spezialisierte Experten, die sowohl das fachliche Verständnis als auch das technische Know-how für die Umsetzung in Streamworks mitbringen. Die Notwendigkeit, hochqualifizierte Fachkräfte für repetitive Standardaufgaben einzusetzen, stellt eine ineffiziente Ressourcennutzung dar. Die Komplexität und der damit verbundene hohe manuelle Aufwand bei der Erstellung von Streams limitieren somit die Skalierbarkeit und Effizienz beim Einsatz von Workload Automation. Eine KI-gestützte Generierung der XML-Konfigurationen, wie sie in dieser Arbeit konzipiert wird, soll diese Hürde überwinden.

2.3 Large Language Models als Lösungstechnologie

Large Language Models (LLMs) sind eine Schlüsseltechnologie für die Automatisierung komplexer Aufgaben. Sie ermöglichen es, natürlichsprachliche Anforderungen in strukturierte, maschinenlesbare Formate zu übersetzen. In diesem Abschnitt werden die Grundlagen, Architektur und Funktionsweise von LLMs erläutert, bevor ihr Einsatz im Kontext von Self-Service und die damit verbundenen Chancen und Risiken diskutiert werden

2.3.1 Grundlagen und Architektur

Large Language Models sind neuronale Netzwerke, die auf der Verarbeitung natürlicher Sprache (Natural Language Processing, NLP) basieren und durch selbstüberwachte Lernverfahren auf riesigen Textdatenmengen trainiert werden [vgl. 1]. Ihr primäres Ziel ist es, sprachliche Zusammenhänge sowie implizites Weltwissen zu erfassen, um vielfältige Aufgaben wie die Generierung, Klassifikation oder Zusammenfassung von Texten

zu bewältigen [vgl. 2]. Die grundlegende Funktionsweise eines LLMs besteht darin, die Wahrscheinlichkeit von Token-Sequenzen vorherzusagen. Ein Token kann dabei ein Wort, ein Wortteil oder ein einzelnes Zeichen sein. Das Modell lernt also, welches Token mit welcher Wahrscheinlichkeit auf eine bestimmte Zeichenkette folgt, wodurch ein statistisch fundiertes Sprachverständnis aufgebaut wird [vgl. 3]. Charakteristisch für LLMs ist ihre enorme Modellgröße, die oft Milliarden von Parametern umfasst, was ihnen die Erfassung komplexer Sprachmuster und semantischer Relationen ermöglicht [vgl. 4].

Die technologische Basis für die meisten aktuellen LLMs ist die **Transformer-Architektur**, die erstmals von Vaswani et al. (2017) vorgestellt wurde. Diese Architektur verzichtet vollständig auf rekurrente neuronale Netze und setzt stattdessen auf sogenannte **Aufmerksamkeitsmechanismen (Attention Mechanisms)**, um globale Abhängigkeiten zwischen Ein- und Ausgabe zu modellieren [vgl. 5]. Ein zentrales Konzept ist dabei die **Self-Attention**. Sie erlaubt es jeder Position in einer Sequenz, alle anderen Positionen innerhalb derselben Sequenz zu berücksichtigen und deren Relevanz für das eigene Verständnis zu gewichten [vgl. 6]. Diese Fähigkeit zur parallelen Verarbeitung der gesamten Sequenz ist ein wesentlicher Vorteil gegenüber älteren, sequenziellen Architekturen. Die maximale Länge der Sequenz, die ein Modell auf einmal verarbeiten kann, wird als **Kontextfenster** bezeichnet.

Der Entwicklungsprozess von LLMs erfolgt typischerweise in zwei Phasen: **Vortraining (Pre-training)** und **Feinjustierung (Fine-tuning)**. Während des Vortrainings erwirbt das Modell auf riesigen, unstrukturierten Textmengen durch selbstüberwachte Aufgaben allgemeines Sprachverständnis und Weltwissen [vgl. 7]. In der anschließenden Feinjustierung wird das vortrainierte Modell auf kleineren, spezifischen Datensätzen für bestimmte Anwendungsfälle angepasst. Dies kann durch überwachtes Lernen mit Beispielen (**Supervised Fine-Tuning**, SFT) oder durch **bestärkendes Lernen mit menschlichem Feedback (Reinforcement Learning from Human Feedback**, RLHF) geschehen, um die Modellausgaben an menschliche Erwartungen und Präferenzen anzupassen [vgl. 8]. Eine weitere Methode zur Anpassung ist das **Prompting**, bei dem das Modell durch eine gezielte Eingabeaufforderung, die auch Beispiele enthalten kann (In-Context Learning), zur Ausführung einer Aufgabe angeleitet wird, ohne dass die Modellparameter verändert werden müssen [vgl. 9].

2.3.2 Sprachverstehen und strukturierte Ausgabe

Eine entscheidende Fähigkeit von Large Language Models ist ihre Kompetenz, natürlichsprachliche Anweisungen zu verstehen und in hochstrukturierte Datenformate wie JSON oder XML zu übersetzen. Sie fungieren als Schnittstelle zwischen menschlicher Sprache und maschinenlesbarem Code und können somit als universelle Übersetzer für technische Konfigurationen dienen. Diese Fähigkeit ist besonders relevant für die Automatisierung, wo LLMs beispielsweise Skripte oder API-Aufrufe direkt aus einer Anforderung ableiten können, was sie zu wertvollen „Coding Assistants“ macht [vgl. 10].

Ein innovativer Ansatz in diesem Bereich ist das von Minkova et al. (2024) vorgestellte Konzept **Text2Workflow**. Dieses System übersetzt natürlichsprachliche Benutzeranfragen direkt in ausführbare Workflows im JSON-Format. Der entscheidende Vorteil liegt in der Generalisierbarkeit des Ansatzes, der es ermöglicht, eine breite Palette von Geschäftsprozessen zu automatisieren, ohne auf vordefinierte, domänenspezifische Datensätze angewiesen zu sein [vgl. 12]. Für die Workload-Automatisierung mit Streamworks ist diese Fähigkeit von zentraler Bedeutung, da Prozessdefinitionen dort auf einem ähnlichen strukturierten Format, nämlich XML, basieren. Eine einfache Anforderung wie „Exportiere die SAP-Daten täglich um 6 Uhr“ kann von einem LLM in eine valide XML-Struktur überführt werden, die alle notwendigen Parameter für die Zeitsteuerung, die auszuführende Aktion und die betroffenen Systeme enthält. Dadurch wird der manuelle Übersetzungsschritt von der fachlichen Anforderung zur technischen Konfiguration erheblich verkürzt und die Grundlage für eine Self-Service-Automatisierung gelegt. Die Fähigkeit, Code zu generieren und komplexe Anweisungen zu befolgen, die LLMs während ihres Trainings erlernen, ermöglicht die Erzeugung solch strukturierter Ausgaben [vgl. 11, 12].

2.3.3 Einsatz im Self-Service-Kontext

Large Language Models eignen sich hervorragend als technologische Schnittstelle zwischen Fachanwendern ohne Programmierkenntnisse und komplexen IT-Systemen. Im Self-Service-Kontext ermöglichen sie es Nutzern, technische Prozesse durch einfache, natürliche Sprache zu initiieren und zu steuern [vgl. 13]. Praktische Anwendungen dieser Fähigkeit finden sich bereits in vielfältiger Form. LLM-basierte **Chatbots** können beispielsweise komplexe Nutzeranfragen im Kundenservice interpretieren und beantworten. **FAQ-Systeme** nutzen LLMs, um unstrukturierte Fragen mit Einträgen in Wissensdatenbanken semantisch abzugleichen. Ebenso können LLMs als intelligente

Schnittstellen für die **semantische Suche** in unternehmensweiten Dokumentenarchiven dienen, indem sie relevante Informationen kontextualisieren und zusammenfassen [vgl. 14].

Für das in dieser Arbeit konzipierte System ist dieser Aspekt zentral: Das LLM agiert als Vermittler, der es einem Fachanwender ermöglicht, eine komplexe Streamworks-Konfiguration zu erstellen, ohne die zugrundeliegende XML-Struktur oder die technischen Details der Automatisierungsplattform verstehen zu müssen. Die natürlichsprachliche Eingabe des Nutzers wird vom LLM analysiert und in eine präzise, valide XML-Datei übersetzt. Dieser Ansatz "demokratisiert" den Zugang zur Workload-Automatisierung und erlaubt es Fachexperten, Standardaufgaben eigenständig zu automatisieren, was die Abhängigkeit von spezialisierten IT-Teams reduziert und die Effizienz steigert [vgl. 12].

2.3.4 Grenzen und Risiken

Trotz ihrer fortschrittlichen Fähigkeiten bergen LLMs signifikante Risiken, die bei der Konzeption eines Self-Service-Systems berücksichtigt werden müssen. Ein zentrales und gut dokumentiertes Problem sind **Halluzinationen**, bei denen das Modell Ausgaben generiert, die plausibel klingen, jedoch sachlich falsch oder nicht durch die Quelldaten gedeckt sind [vgl. 15]. Im Kontext der XML-Generierung kann dies zu syntaktisch falschen Strukturen, der Erfindung von nicht existierenden Attributen oder der inkorrekten Verschachtelung von Elementen führen. Solche Fehler können die gesamte Workflow-Automatisierung unbrauchbar machen oder zu unvorhersehbarem Systemverhalten führen [vgl. 16].

Ein weiteres wesentliches Risiko betrifft die **Sicherheit und den Datenschutz**. Werden LLMs mit unternehmensinternen Daten trainiert oder interagieren sie mit sensiblen Informationen, besteht die Gefahr, dass diese unbeabsichtigt in den generierten Ausgaben wiedergegeben werden [vgl. 17]. Im Self-Service-Kontext, in dem Fachanwender möglicherweise Details zu Kunden, Systemen oder Geschäftsprozessen eingeben, muss sichergestellt werden, dass keine vertraulichen Daten exponiert werden.

Die **fehlende Nachvollziehbarkeit** der Modellergebnisse stellt eine weitere Herausforderung dar. Da LLMs als „Blackbox“ agieren, ist es oft schwierig zu verstehen, warum eine bestimmte Ausgabe generiert wurde. Dies erschwert die Fehleranalyse und reduziert das Vertrauen in die automatisierten Prozesse. Aus diesem Grund ist eine externe Validierung, beispielsweise durch ein XSD-Schema für die generierten XML-Dateien, unerlässlich, um die formale Korrektheit der Ergebnisse sicherzustellen.

Schließlich besteht die Gefahr, dass Nutzer den Ausgaben eines LLMs übermäßiges Vertrauen schenken, weil sie so überzeugend und flüssig formuliert sind – selbst wenn sie inhaltlich falsch sind [vgl. 15]. Im Self-Service-Umfeld kann dies dazu führen, dass Anwender fehlerhafte Konfigurationen unkritisch übernehmen, was schwerwiegende operative Folgen haben kann. Diese Risiken verdeutlichen die Notwendigkeit robuster Validierungsmechanismen und einer klaren Governance für den Einsatz von LLMs in produktiven Umgebungen.

2.4 Retrieval-Augmented Generation (RAG) im Unternehmenskontext

Retrieval-Augmented Generation (RAG) ergänzt LLMs um den Zugriff auf externe Wissensquellen und erhöht dadurch die Verlässlichkeit der Ergebnisse. Dieser Abschnitt beschreibt das Konzept, die Architektur und die Skalierbarkeit von RAG-Systemen. Zudem werden die Unterschiede zu Fine-Tuning-Ansätzen herausgearbeitet sowie typische Fehlerquellen und Maßnahmen zur Qualitätssicherung, wie etwa XSD-Validierung, dargestellt.

2.4.1 Konzept und Motivation

Die grundlegende Motivation hinter RAG ist die Überwindung der inhärenten Begrenzungen von Standard-LLMs [7]. Das Wissen dieser Modelle ist auf dem Stand ihrer Trainingsdaten eingefroren und bietet keine einfache Möglichkeit zur Erweiterung oder Revision [8]. Ein RAG-System adressiert diese Limitierung, indem es den generativen Fähigkeiten eines LLM einen Informationsabrufmechanismus (Retrieval) vorschaltet [9]. Eine Nutzeranfrage wird genutzt, um semantisch relevante Informationen aus einem Wissenspool zu suchen [10]. Diese abgerufenen Informationen werden als Kontexte bezeichnet und gemeinsam mit der ursprünglichen Anfrage an das LLM übergeben [11]. Dadurch erhält das Modell Zugriff auf spezifisches Wissen, das nicht Teil seiner ursprünglichen Trainingsdaten war [12].

Die Aufgabe des LLM verschiebt sich dadurch von der reinen Wissensreproduktion hin zur Synthese und Verarbeitung der bereitgestellten Informationen [13]. Es formuliert seine Antwort auf Basis externer, verifizierbarer Fakten [14]. Dies steigert die Zuverlässigkeit der generierten Texte erheblich und ermöglicht die sichere Nutzung von LLMs für

unternehmensinterne Daten, ohne ein kostspieliges Fine-Tuning durchführen zu müssen [15].

2.4.2 Technische Architektur

Ein RAG-System baut auf zwei zentralen Komponenten auf: einem **Retriever**, der latente Dokumente basierend auf dem Input bereitstellt, und einem **Generator**, der diese Dokumente zur Generierung der Ausgabe verwendet [16]. Die Architektur lässt sich in einen **Indexierungs-** und einen **Abfrageprozess** unterteilen [17].

Der **Indexierungsprozess** findet typischerweise in der Entwicklungsphase statt und bereitet die externe Wissensdatenbank vor [18]. Dabei werden Quelldokumente in kleinere Abschnitte ("Chunks") zerlegt [19]. Für diese Chunks werden semantische Vektor-Repräsentationen ("Embeddings") erstellt, die zusammen mit den Original-Chunks in einer Datenbank indiziert werden [20]. Zur technischen Umsetzung werden die Textpassagen mithilfe von Embedding-Modellen als Vektoren repräsentiert und in einer Vektordatenbank gespeichert, um eine effiziente semantische Ähnlichkeitssuche zu ermöglichen [21].

Im **Abfrageprozess**, der zur Laufzeit stattfindet, wird eine Nutzeranfrage zunächst verarbeitet, um sie zu generalisieren [22]. Anschließend wird die Anfrage in ein Embedding umgewandelt, um mittels Ähnlichkeitssuche die Top-K relevantesten Chunks aus der Datenbank abzurufen [23]. Diese werden nach einer Konsolidierung an das LLM übergeben [24]. Der Generator nutzt die abgerufenen Dokumente als zusätzlichen Kontext, um die finale Ausgabe zu erzeugen [25]. Die Dokumente können dabei als latente Variablen behandelt werden, über die zur Erzeugung der finalen Wahrscheinlichkeitsverteilung marginalisiert wird [26]. Das LLM erhält die Suchergebnisse somit als Ergänzung zum ursprünglichen Prompt und hat die Aufgabe, diese im Sinne der Anfrage zu verwerten, beispielsweise durch eine Zusammenfassung [27].

Dieser modulare Aufbau ermöglicht es, die Wissensbasis des Systems durch einfaches Hinzufügen oder Aktualisieren von Dokumenten im Index aktuell zu halten, ohne das Sprachmodell selbst neu trainieren zu müssen [28].

2.4.3 Retrieval-Komponenten und Skalierbarkeit

Ein entscheidender Schritt für die Qualität des Retrievals ist das **Chunking** [29]. Die Größe der Dokumentenabschnitte ist dabei eine wichtige Designentscheidung, denn sind die Chunks zu klein, geht Kontext verloren, während zu lange Chunks die Extraktion

der Antwort durch irrelevante Informationen ("Noise") erschweren [30]. Die technische Realisierung erfolgt durch spezialisierte **Embedding-Modelle**, die Text-Chunks in semantische Vektoren umwandeln [31]. Diese werden in einer **Vektordatenbank** für eine schnelle Suche indiziert, wobei sich zur Qualitätssteigerung auch hybride Ansätze aus Vektor- und Schlüsselwortsuche eignen [32].

In Bezug auf die **Skalierbarkeit** zeigen empirische Studien, dass die Leistungsfähigkeit von RAG-Systemen mit zunehmender Größe der Wissensdatenbank weiter ansteigt [33]. Im Gegensatz dazu stagniert der Leistungszuwachs beim Fine-Tuning bei wachsender Datenmenge, was auf eine überlegene Skalierbarkeit von RAG in datenreichen Szenarien hindeutet [34].

2.4.4 Integration von LLMs: RAG vs. Fine-Tuning

Um einem Large Language Model (LLM) domänenspezifisches Wissen zu vermitteln, wie es für die Erstellung von Streamworks-Konfigurationen erforderlich ist, stehen grundsätzlich zwei Ansätze zur Verfügung: Retrieval-Augmented Generation (RAG) und Fine-Tuning. Obwohl beide Ansätze oft verglichen werden, verfolgen sie unterschiedliche Ziele und sollten nicht als direkte Alternativen betrachtet werden[6].

Fine-Tuning ist ein Prozess, bei dem die internen Parameter eines vortrainierten LLMs durch zusätzliches Training auf einem spezifischen, kuratierten Datensatz angepasst werden, um das interne Wissen des Modells an domänenspezifische Muster anzugleichen[7]. Dieser Ansatz eignet sich hervorragend, um dem Modell einen bestimmten Antwortstil oder den Umgang mit speziellen Datenstrukturen, wie der Syntax von XML-Konfigurationen, beizubringen. Für das primäre Ziel, einem LLM reines Faktenwissen zu vermitteln, ist Fine-Tuning jedoch weniger geeignet und kann zu unerwünschten Nebeneffekten führen[8]. Der Prozess ist nicht nur rechen- und kostenintensiv, sondern birgt

---

[6] Vgl. Wang, C. et al. (2025): *RAG or Fine-tuning? A Comparative Study on LCMs-based Code Completion in Industry*, S. 2

[7] Vgl. Wang, C. et al. (2025): *RAG or Fine-tuning? A Comparative Study on LCMs-based Code Completion in Industry*, S. 2

[8] Vgl. Honroth, T. et al. (2024): *Retrieval Augmented Generation (RAG): Chat mit eigenen Daten*, S. 7

auch die Gefahr, dass das Modell allgemeine, zuvor erlernte Fähigkeiten vergisst – ein Phänomen, das als „Catastrophic Forgetting“ bekannt ist[9].

Im Gegensatz dazu ist RAG die Methode der Wahl, um ein LLM mit spezifischem, aktuellem und vor allem verifizierbarem Faktenwissen auszustatten, ohne die Modellparameter selbst zu verändern. Anstatt das Wissen fest in das Modell zu integrieren, wird es zur Laufzeit dynamisch aus einer externen Wissensquelle abgerufen und dem LLM als zusätzlicher Kontext für die Beantwortung einer Anfrage bereitgestellt[10]. Die Aufgabe des LLMs verschiebt sich dadurch von der reinen Wissensreproduktion hin zur Synthese und Verarbeitung der bereitgestellten Informationen, was die Wahrscheinlichkeit von Halluzinationen signifikant senkt[11]. Ein entscheidender Vorteil für den Unternehmenskontext liegt in der Wartbarkeit und Aktualität: Die Wissensbasis kann jederzeit durch das Hinzufügen oder Aktualisieren von Dokumenten erweitert werden, ohne dass ein teures und zeitaufwendiges Neutraining des LLMs erforderlich ist. Zudem erhöht RAG die Nachvollziehbarkeit und das Vertrauen in das System, da die für eine Antwort herangezogenen Quellen transparent gemacht werden können[12].

Für das in dieser Arbeit konzipierte System zur automatisierten Erstellung von Streamworks-Konfigurationen ist der RAG-Ansatz aus mehreren Gründen überlegen. Erstens ist die Wissensbasis (z. B. Dokumentationen, Best Practices) dynamisch und kann sich ändern, was RAG durch die flexible Aktualisierung des Wissensspeichers agil und kosteneffizient abbilden kann. Zweitens vermeidet RAG das Risiko des „Catastrophic Forgetting“, wodurch die robusten, allgemeinen Sprachfähigkeiten des Basis-LLMs erhalten bleiben, die für das Verstehen der natürlichsprachlichen Anforderungen der Fachanwender essenziell sind. Drittens zeigen empirische Studien, dass die Leistungsfähigkeit von RAG-Systemen mit zunehmender Größe der Wissensdatenbank weiter ansteigt, während der Leistungszuwachs beim Fine-Tuning stagniert, was auf eine überlegene Skalierbarkeit von RAG in datenreichen Szenarien hindeutet[13].

Obwohl RAG für die Bereitstellung von Faktenwissen die bevorzugte Methode ist, können beide Ansätze auch komplementär eingesetzt werden, da ihre Kombination zu

---

[9] Vgl. Wang, C. et al. (2025): *RAG or Fine-tuning? A Comparative Study on LCMs-based Code Completion in Industry*, S. 8

[10] Vgl. Lewis, P. et al. (2021): *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*, S. 1-2

[11] Vgl. Honroth, T. et al. (2024): *Retrieval Augmented Generation (RAG): Chat mit eigenen Daten*, S. 2

[12] Vgl. Lewis, P. et al. (2021): *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*, S. 7

[13] Vgl. Wang, C. et al. (2025): *RAG or Fine-tuning?* S. 7

synergistischen Effekten und einer weiteren Leistungssteigerung führen kann[14]. Ein gezieltes Fine-Tuning könnte beispielsweise die Fähigkeit des LLMs verbessern, die durch den RAG-Prozess abgerufenen Kontexte besser zu verstehen und in die korrekte XML-Struktur zu übersetzen. Für die Zielsetzung dieser Arbeit – die Bereitstellung von Faktenwissen zur Unterstützung der XML-Generierung – sollte jedoch zunächst das Potenzial von RAG voll ausgeschöpft werden, da es den direkteren, flexibleren und transparenteren Weg darstellt[15].

2.4.5 Herausforderungen und Fehlerpotenzial

Obwohl RAG-Systeme viele Probleme traditioneller LLMs lösen, unterliegen sie eigenen, spezifischen Fehlerquellen, die sich über die gesamte Verarbeitungskette erstrecken [48]. Eine empirische Analyse hat sieben zentrale Fehlerpunkte ("Failure Points") identifiziert [49]. Diese umfassen **fehlenden Inhalt (FP1)**, bei dem die Antwort nicht in den Dokumenten vorhanden ist; **verfehlte Top-Dokumente (FP2)**, bei denen der richtige Inhalt nicht hoch genug eingestuft wird; **Verlust im Kontext (FP3)**, wenn relevante Informationen bei der Aufbereitung für das LLM verloren gehen; und **nicht extrahierte Antworten (FP4)**, bei denen das LLM trotz korrektem Kontext die Antwort nicht findet [50]. Weitere Fehler sind ein **falsches Format (FP5)** bei der Ausgabe, eine **falsche Spezifität (FP6)**, die nicht zur Nutzerintention passt, sowie **unvollständige Antworten (FP7)**, bei denen verfügbare Informationen im Kontext ausgelassen werden [51].

Diese Fehlerpotenziale verdeutlichen, dass die Implementierung eines robusten RAG-Systems eine sorgfältige Konfiguration und kontinuierliche Überwachung aller Prozessschritte erfordert [52].

2.4.6 Qualitätssicherung durch XSD-Validierung

Die von einem LLM generierte XML-Konfiguration ist anfällig für Fehler wie Halluzinationen oder syntaktische Ungenauigkeiten, was zu einem fehlerhaften Systemverhalten oder Importproblemen führen kann [53]. Um die Korrektheit und Zuverlässigkeit der automatisierten Stream-Erstellung zu gewährleisten, ist ein robuster Qualitätssicherungsmechanismus unerlässlich [54]. Die XML Schema Definition (XSD) bietet hierfür ein standardisiertes und leistungsfähiges Regelwerk [55].

---

[14] Vgl. Wang, C. et al. (2025): *RAG or Fine-tuning?*, S. 5

[15] Vgl. Honroth, T. et al. (2024): *Retrieval Augmented Generation (RAG): Chat mit eigenen Daten*, S. 7

XML Schema Definition (XSD) ist eine Schemasprache, die dazu dient, die Struktur eines XML-Dokuments zu beschreiben und dessen Inhalte zu beschränken [56]. Ein XSD-Schema definiert, welche Elemente und Attribute in einer XML-Datei vorkommen dürfen, in welcher Reihenfolge sie erscheinen müssen und welchen Datentyp ihre Werte haben müssen [57]. Es fungiert somit als eine Art Bauplan oder Grammatik für eine Klasse von XML-Dokumenten und stellt sicher, dass diese einem vordefinierten, formalen Standard entsprechen [57]. Für die Workload-Automation-Plattform Streamworks bildet ein solches Schema die Grundlage, um die Gültigkeit von importierten Prozessdefinitionen zu verifizieren [58].

Die XSD-Validierung ist eine Methode, mit der die syntaktische Korrektheit einer vom LLM generierten XML-Datei programmgesteuert überprüft wird [54]. Dabei wird das generierte XML-Dokument gegen das vordefinierte XSD-Schema geprüft [57]. Der Validierungsprozess stellt fest, ob das Dokument den im Schema festgelegten Regeln entspricht [57]. Findet der Validator Abweichungen, wie beispielsweise falsch benannte Tags, eine inkorrekte Elementreihenfolge oder ungültige Attributwerte, wird die Validierung als fehlgeschlagen markiert und der Import in Streamworks verhindert [54].

Die Positionierung der XSD-Validierung als finaler Schritt nach der XML-Generierung durch das LLM ist essenziell für die Risikominimierung [54]. Sie stellt eine entscheidende Sicherheitsebene dar, die verhindert, dass fehlerhafte oder inkonsistente Konfigurationen in das produktive Automatisierungssystem gelangen [54]. Indem jede generierte XML-Datei vor dem Import validiert wird, wird die Importfähigkeit in Streamworks gewährleistet und das Risiko von Laufzeitfehlern durch syntaktisch inkorrekte Prozessdefinitionen drastisch reduziert [54]. Dieser Schritt ist somit fundamental, um das Vertrauen in die KI-gestützte Self-Service-Lösung zu stärken und eine hohe Prozessqualität sicherzustellen [54].

2.5 Self-Service in der IT-Automatisierung

Self-Service-Konzepte haben sich in der IT etabliert, um Anwendern eine direkte und effiziente Nutzung von Services zu ermöglichen. Dieses Kapitel zeigt, wie sich das Prinzip auf komplexe Plattformen wie Streamworks übertragen lässt. Dazu werden Definitionen, technologische Grundlagen, Erfolgsfaktoren und Herausforderungen diskutiert. Ein besonderer Fokus liegt auf der Rolle von LLMs und RAG bei der Realisierung von Self-Service-Ansätzen für die Workload-Automatisierung.

2.5.1 Definition und Grundprinzipien

Self-Service-Technologien (SST) sind automatisierte Systeme, die es Nutzern ermöglichen, eine Dienstleistung unabhängig von der direkten Beteiligung eines menschlichen Servicemitarbeiters zu erhalten [1]. Im Unternehmenskontext manifestiert sich dieses Prinzip häufig in Form von Self-Service-Portalen, die als zentraler digitaler Zugangspunkt dienen, über den Endanwender eigenständig Serviceanfragen stellen, Informationen abrufen oder vordefinierte Prozesse anstoßen können [2]. Im Gegensatz zu klassischen Service-Prozessen wie einem Helpdesk, bei dem die Bearbeitung von Anfragen zentral durch IT-Personal erfolgt, zielt ein Self-Service-Portal darauf ab, den Nutzer zur Selbsthilfe zu befähigen, noch bevor ein Support-Ticket notwendig wird [3].

Im spezifischen Kontext der Workload-Automatisierung mit Streamworks bedeutet dies eine Abkehr vom traditionellen Vorgehen, bei dem Fachanwender ihre Anforderungen formulieren und an ein spezialisiertes Orchestration-Team weiterleiten, welches diese manuell in eine technische Konfiguration übersetzt [4]. Der Self-Service-Ansatz verlagert stattdessen die Initiative zum Anwender und verfolgt die zentrale Zielsetzung, die Verlagerung von standardisierbaren Aufgaben von IT-Spezialisten auf die Fachanwender zu ermöglichen [5]. Dadurch wird nicht nur die IT-Abteilung entlastet und Prozesse werden beschleunigt, sondern auch die Autonomie der Fachbereiche wird erhöht [6]. In der Praxis werden Fachanwender so in die Lage versetzt, Standard-Workloads wie wiederkehrende Dateiübertragungen eigenständig zu initiieren, während sich die Experten des Orchestration-Teams auf komplexere und strategisch relevante Aufgaben konzentrieren können [7].

2.5.2 Technologische Grundlagen

Die Architektur moderner Self-Service-Portale kombiniert typischerweise mehrere funktionale Elemente, darunter geführte Eingabemasken zur Ticketerstellung und einen Service-Katalog für standardisierte Leistungen [8]. Ergänzt wird dies durch eine Wissensdatenbank (Knowledge Base) mit Anleitungen und FAQs sowie die Möglichkeit zum Status-Tracking offener Anfragen [9]. Um die Qualität der Eingaben sicherzustellen und den Prozess für den Nutzer zu vereinfachen, werden Validierungs- und Prüfmechanismen eingesetzt [10]. Durch intelligente Formularlogik mit bedingten Feldern werden beispielsweise nur die für den jeweiligen Kontext relevanten Informationen abgefragt [11].

Die Notwendigkeit solcher Prüfungen wird durch die Anforderung untermauert, dass durch verpflichtende Eingabefelder und technische Validierungen nur korrekte und

vollständige Anfragen übermittelt werden dürfen, um Medienbrüche und Fehlerquellen zu vermeiden [12]. Eine nahtlose Einbettung in bestehende IT-Landschaften ist für den Erfolg eines solchen Portals entscheidend, was durch die Anbindung an IT-Service-Management-Lösungen und Automatisierungsplattformen über Schnittstellen wie APIs realisiert wird [13].

2.5.3 Erfolgsfaktoren und Herausforderungen

Die Einführung von Self-Service-Technologien bietet zahlreiche Vorteile, wie die Reduzierung des Ticketvolumens durch Selbsthilfe und eine schnellere Bearbeitung durch strukturierte Erfassung, was letztlich zu Kostensenkungen führen kann [14]. Durch die Verlagerung von Routineaufgaben auf die Anwender wird nicht nur die Effizienz gesteigert, sondern auch die Nutzerautonomie und -zufriedenheit gestärkt [15]. Ein solcher Ansatz kann zudem die allgemeine Produktivität und Innovation in einem Unternehmen fördern [16].

Trotz der Vorteile bestehen signifikante Herausforderungen, wie etwa eine mangelnde Akzeptanz bei den Nutzern oder eine unzureichende Integration in bestehende Systeme [17]. Weitere Hürden sind die Gewährleistung der Datenqualität sowie die Durchsetzung von Governance-Richtlinien durch Rollen- und Berechtigungskonzepte [18]. Ebenso muss die Einhaltung von Sicherheits- und Datenschutzstandards gewährleistet sein [19]. Da automatisierte Systeme nicht in der Lage sind, semantische Mehrdeutigkeiten oder implizite Anforderungen korrekt zu erfassen, bleibt bei individuellen Sonderfällen mit komplexer Logik menschliche Expertise unverzichtbar [20]. In solchen Szenarien ist ein kollaborativer Ansatz erforderlich, der menschliches Urteilsvermögen mit automatisierter Unterstützung kombiniert und als Human-in-the-Loop-Prinzip bekannt ist [21].

2.5.4 Rolle von LLMs und KI im Self-Service

Large Language Models (LLMs) sind eine Schlüsseltechnologie, um die Lücke zwischen der natürlichsprachlichen Anforderung eines Fachanwenders und der hochstrukturierten, technischen Konfiguration eines IT-Systems zu schließen [22]. Im Kontext dieser Arbeit ermöglichen sie es Anwendern, ihre Absicht in natürlicher Sprache zu beschreiben, woraufhin das Modell diese Eingabe interpretiert und in ein maschinenlesbares, strukturiertes Datenformat wie XML übersetzt [23]. Diese Fähigkeit ist entscheidend, um die komplexen, aber stark strukturierten Konfigurationsdateien für Streamworks automatisiert zu erzeugen [24]. Zur weiteren Unterstützung des Self-Service-Prozesses können

Retrieval-Augmented-Generation-Systeme (RAG) als intelligente Wissensdatenbanken fungieren, die auf Basis von Dokumentationen und Best-Practice-Beispielen typische Anwenderfragen beantworten [25]. Durch den Einsatz von LLMs und KI wird der Zugang zur Automatisierung demokratisiert, indem die Abhängigkeit von spezialisierten IT-Teams reduziert wird, sodass auch Nutzer ohne tiefgreifende technische Kenntnisse komplexe Prozesse initiieren können [26].

2.5.5 AI-in-the-Loop: Menschliche Expertise als Korrektiv

Während künstliche Intelligenz das Potenzial hat, Prozesse vollständig zu automatisieren, erfordern insbesondere komplexe und kritische Anwendungsfälle eine Brücke zwischen maschineller Effizienz und menschlicher Expertise. An dieser Schnittstelle etabliert sich das Human-in-the-Loop-Prinzip (HITL). HITL-Systeme sind darauf ausgelegt, die Risiken einer reinen KI-gesteuerten Automatisierung zu minimieren, indem sie menschliches Urteilsvermögen an entscheidenden Stellen im Prozess integrieren [1]. So wird sichergestellt, dass KI-generierte Empfehlungen stets durch kontextuelles Verständnis und kritische Bewertung eines Menschen überprüft und validiert werden können.

Eine genauere Betrachtung der existierenden Systeme offenbart jedoch eine entscheidende konzeptionelle Unterscheidung, die für die Zielsetzung dieser Arbeit von zentraler Bedeutung ist: die Differenzierung zwischen „Human-in-the-Loop“ (HIL) und „AI-in-the-Loop“ (AI²L). Obwohl beide Ansätze eine Mensch-Maschine-Interaktion beschreiben, unterscheiden sie sich fundamental in der Verteilung von Kontrolle und Verantwortung. Viele Systeme, die als HIL bezeichnet werden, sind bei genauerer Betrachtung AI²L-Systeme, bei denen „der Mensch die Kontrolle über das System hat, während die KI zur Unterstützung des Menschen da ist“ [2].

Im klassischen **HIL-Modell** liegt die primäre Steuerung beim KI-System. Die KI treibt den Inferenz- und Entscheidungsprozess voran, während der Mensch als Korrektiv eingreift, um beispielsweise Fehler zu beheben, Feedback zu geben oder Daten zu annotieren. Das Ziel ist hier oft die Optimierung und Weiterentwicklung des KI-Modells selbst. Ein typisches Beispiel ist das aktive Lernen, bei dem die KI dem Menschen gezielt unklare Fälle zur Klassifizierung vorlegt.

Im Gegensatz dazu steht das **AI²L-Modell**, das einen kollaborativen Ansatz verfolgt. Hier behält der Mensch die volle Kontrolle und trifft die endgültige Entscheidung. Die KI agiert als intelligenter Assistent, der Daten analysiert, Muster erkennt, Informationen

zusammenfasst und Handlungsempfehlungen gibt. In diesem Paradigma „ist der Mensch die Kontrollinstanz des gesamten Systems. Die Präsenz der KI in diesem System macht den Prozess lediglich effizienter und möglicherweise effektiver. Das Gesamtsystem existiert jedoch unabhängig von der Anwesenheit der KI. Dieser Unterschied ist entscheidend“ [3]. Dieser Ansatz stellt sicher, dass die Stärken beider Seiten optimal genutzt werden: die Fähigkeit der KI, riesige Datenmengen schnell zu verarbeiten, und die Fähigkeit des Menschen, kontextbezogen zu urteilen, ethische Abwägungen zu treffen und die letztendliche Verantwortung zu tragen. In einem solchen kollaborativen System greift der Mensch proaktiv ein, um die Lösung zu steuern, insbesondere wenn die KI eine geringe Konfidenz in ihre Ergebnisse hat, wodurch eine Feedbackschleife entsteht, die die Intelligenz des Gesamtsystems verbessert [4].

Für das in dieser Arbeit konzipierte Self-Service-System ist das AI²L-Modell der entscheidende konzeptionelle Rahmen. Es geht nicht darum, den menschlichen Experten durch eine autonome KI zu ersetzen, sondern ihn durch ein intelligentes Werkzeug zu befähigen. Standardprozesse können hochautomatisiert ablaufen, doch bei komplexen, neuen oder potenziell fehleranfälligen Konfigurationen sieht das System explizit einen menschlichen Review vor. Damit wird eine Balance zwischen Effizienzsteigerung und der Gewährleistung von Qualität, Sicherheit und Nachvollziehbarkeit geschaffen – eine zentrale Anforderung, die sich aus den Experteninterviews ableitet.

2.5.6 Zwischenfazit und Überleitung zur Konzeption

Die Analyse verdeutlicht, dass Self-Service ein zentraler Schlüssel ist, um in der IT-Automatisierung eine höhere Skalierbarkeit und Effizienz zu erreichen [27]. Insbesondere in Umgebungen wie Streamworks, wo Expertenwissen traditionell einen Engpass darstellt, befähigen Self-Service-Portale Fachanwender, Standardprozesse eigenständig zu verwalten [28]. Dies führt zu einer Entlastung der IT-Experten und verkürzt die Prozessdurchlaufzeiten [29]. Die wesentliche Herausforderung besteht jedoch darin, die fachliche Anforderung eines Nutzers präzise in eine komplexe technische Konfiguration zu übersetzen, ohne dass spezialisiertes Wissen über die Zielplattform erforderlich ist [30].

An dieser Stelle ermöglicht der Einsatz von künstlicher Intelligenz, insbesondere die Kombination aus Large Language Models (LLMs) und Retrieval-Augmented Generation (RAG), erstmalig eine praxistaugliche Umsetzung im Kontext hochkomplexer Systeme wie Streamworks [31]. LLMs fungieren als "Übersetzer" von natürlicher Sprache in

strukturierte Formate wie XML, während RAG-Systeme kontextbezogenes Wissen zur Unterstützung bereitstellen [32]. Diese Symbiose aus Self-Service und KI schafft die Grundlage für eine intuitive, effiziente und skalierbare Workload-Automatisierung.

Diese Erkenntnisse bilden die Basis für das in Kapitel 4 konzipierte System, das die automatisierte Erstellung von Streamworks-Konfigurationen durch ein KI-gestütztes Self-Service-Portal zum Ziel hat.

3 Analyse der Stream-Erstellungsprozesse

Dieses Kapitel bildet die Brücke zwischen der theoretischen Fundierung und der praktischen Konzeption. Es analysiert den bestehenden Prozess der Stream-Erstellung bei Arvato Systems auf Basis der durchgeführten Experteninterviews [1]. Ziel ist es, die zentralen Herausforderungen und Engpässe zu identifizieren, daraus Automatisierungspotenziale abzuleiten und einen konkreten Anforderungskatalog für die zu entwickelnde Lösung zu definieren.

3.1 Der Ist-Prozess: Ein manueller, kommunikationsintensiver Workflow

Die Erstellung neuer Streams ist heute ein Prozess, der stark von manuellen Tätigkeiten und intensiver Kommunikation geprägt ist. Wie die Experteninterviews bestätigen, gibt es keinen einheitlichen, standardisierten Kanal für die Einreichung von Anforderungen. Stattdessen erreichen Anfragen das Orchestration-Team über eine Vielzahl von Wegen. Ein Experte beschreibt die Situation treffend: „Mal sind es Excel-Listen, mal Word-Dokumente, mal nur Stichpunkte im Mailtext. Diese Heterogenität führt regelmäßig zu Rückfragen und verlangsamt die Umsetzung“ [2].

Dieser Mangel an Standardisierung setzt sich bei der Qualität der bereitgestellten Informationen fort. Für eine reibungslose Umsetzung benötigt das Team zwingend Pflichtinformationen wie Streamnamen nach Konvention, Zielsysteme und Kalender. Dennoch wird gewarnt, dass eine Automatisierung an unvollständigen oder veralteten Informationen wie alten Servernamen scheitern würde [3].

Der resultierende Prozess gliedert sich typischerweise in folgende Schritte:

1. **Eingang und Sichtung der Anforderung:** Prüfung des heterogenen Inputs.
2. **Validierung und Rückfragen:** Manuelle Prüfung auf Vollständigkeit, gefolgt von oft zeitintensiven Klärungsschleifen.
3. **Manuelle Umsetzung:** Erstellung der Stream-Konfiguration in der Testumgebung.
4. **Testing und Freigabe:** Interne Tests, gemeinsame Abnahme mit dem Kunden und Überführung in die Produktion.

Während kleine Änderungen in 15 bis 60 Minuten umgesetzt werden können, dauert die Erstellung eines neuen Streams aufgrund der Abstimmungsprozesse laut den Experten

im Idealfall eine Woche, realistisch sind jedoch drei bis fünf Wochen [4]. Der Großteil der Zeit fließt in die Behebung von Informationsdefiziten.

3.2 Kernherausforderungen und operative Engpässe

Aus dem beschriebenen Prozess ergeben sich drei zentrale Herausforderungen:

1. **Prozessuale Ineffizienz durch fehlende Standardisierung:** Die heterogenen Eingangsformate und die schwankende Qualität der Anforderungen verhindern eine durchgängig standardisierte Bearbeitung und erschweren eine Skalierung.
2. **Hoher Kommunikationsaufwand durch Informationsdefizite:** Die häufigsten Probleme sind laut den Interviews „Verständnisprobleme“ [5]. Fachliche Anforderungen müssen in technische Logik übersetzt werden, wobei semantische Lücken und Sprachbarrieren einen erheblichen Kommunikations-Overhead erzeugen.
3. **Ineffiziente Ressourcennutzung:** Hochqualifizierte Experten wenden einen signifikanten Teil ihrer Arbeitszeit für repetitive Standardaufgaben auf. Diese Tätigkeiten binden wertvolle Ressourcen, die für komplexe, strategisch wichtige Automatisierungsaufgaben fehlen.

3.3 Potenziale durch Self-Service und Automatisierung

Trotz der Herausforderungen sehen die befragten Experten ein klares Potenzial zur Optimierung durch Self-Service und Automatisierung. Der aussichtsreichste Ansatzpunkt liegt in der **Standardisierung wiederkehrender Anwendungsfälle**. Die Experten stimmen überein, dass sich einfache Jobs hervorragend für eine Automatisierung eignen und schlagen ein Self-Service-Portal vor, in dem Kunden klar definierte Parameter direkt pflegen könnten [6].

Ein solches Portal würde die identifizierten Probleme direkt adressieren:

- Es **erzwingt eine standardisierte Eingabe** und eliminiert die Heterogenität der Kanäle.
- Es **reduziert den Kommunikationsaufwand** durch Echtzeit-Validierung.
- Es **steigert die Effizienz**, indem es Fachanwender befähigt, Standardaufgaben eigenständig zu erstellen.

Die Experten betonen jedoch auch die Grenzen eines solchen Systems. Die Akzeptanz hängt von einer klaren **Governance** ab. Komplexe, individuelle Streams sollten weiterhin manuell geprüft werden. Ein Experte bringt es auf den Punkt: „Alles, was echtes Verständnis der Geschäftslogik erfordert, bleibt beim Menschen“ [7]. Automatisierung soll das Fachurteil unterstützen, aber nicht ersetzen. Dieses kollaborative Prinzip, das im Rahmen dieser Arbeit als AI-in-the-Loop (AI²L) umgesetzt wird, ist entscheidend für die Qualitätssicherung.

3.4 Abgeleitete Anforderungen an ein KI-gestütztes System

Aus der Analyse des Ist-Prozesses, der Herausforderungen und der Potenziale lassen sich folgende Kernanforderungen für die Konzeption eines KI-gestützten Self-Service-Systems ableiten.

- **Eine flexible und zentrale Eingabeschnittstelle bereitstellen:** Das System muss die heterogenen Kanäle ablösen und eine zentrale Schnittstelle bieten, die **sowohl eine geführte Eingabe für Standardfälle (z.B. über einen Wizard) als auch eine flexible, natürlichsprachliche Eingabe (z.B. über einen Chat) für erfahrenere Nutzer ermöglicht.** Ziel ist es, die Nutzer zur vollständigen und korrekten Eingabe aller Pflichtinformationen anzuleiten, unabhängig vom gewählten Interaktionsmodus.
- **Eingaben automatisiert validieren:** Das System muss die formalen und technischen Vorgaben von Streamworks durchsetzen. Dies umfasst die Prüfung von Namenskonventionen, die Validierung von Parametern und die Sicherstellung der syntaktischen Korrektheit der generierten Konfiguration durch ein XSD-Schema.
- **Die Übersetzung von Anforderungen in Code automatisieren:** Eine KI-Komponente (LLM) soll die Nutzereingaben (egal ob formular-basiert oder natürlichsprachlich) in eine valide Streamworks-XML-Konfiguration übersetzen. Dies ist der Kern der Automatisierung, der den manuellen Umsetzungsaufwand eliminiert.
- **Einen AI-in-the-Loop-Workflow ermöglichen:** Für alle generierten Streams muss ein Freigabeprozess etabliert werden, der eine **manuelle Prüfung und explizite Freigabe durch einen Experten** vor der Produktivsetzung vorsieht. Dies stellt sicher, dass die menschliche Expertise die finale Kontrollinstanz bleibt.

- **Nachvollziehbarkeit und Transparenz gewährleisten:** Alle Aktionen im System, insbesondere die automatische Generierung von Streams, müssen lückenlos protokolliert und versioniert werden, um die Revisionssicherheit und das Vertrauen in den Prozess zu sichern.
- **Kontextbezogene Unterstützung anbieten:** Ein integriertes Hilfesystem (z. B. auf RAG-Basis) soll Nutzer bei der Eingabe unterstützen, indem es auf Basis der bestehenden Dokumentation und Best Practices typische Fragen beantwortet und so die Nutzerautonomie weiter stärkt.

Diese Anforderungen bilden das Fundament für die in Kapitel 4 folgende Konzeption der Zielarchitektur und des Systemdesigns.

4 Konzeption und Design (~6–7 Seiten / ~1.600–1.750 Wörter)

Aufbauend auf den in Kapitel 2 erarbeiteten theoretischen Grundlagen und den aus den Experteninterviews in Kapitel 3 abgeleiteten Anforderungen, wird in diesem Kapitel das detaillierte Konzept für ein KI-gestütztes Self-Service-System zur Automatisierung der Streamworks-Stream-Erstellung entwickelt. Das Ziel ist es, eine technische Blaupause zu entwerfen, die den Engpass der manuellen Konfiguration auflöst, Fachanwender befähigt und gleichzeitig die Prozessqualität durch einen kollaborativen AI-in-the-Loop-Ansatz (AI²L) sicherstellt. Zunächst wird die Gesamtarchitektur des Systems vorgestellt. Anschließend werden die zentralen Komponenten – das Self-Service-Portal, der Prozess der KI-gestützten XML-Generierung und das RAG-basierte Support-System – detailliert konzipiert.

4.1 Zielarchitektur des Self-Service-Systems

Die Zielarchitektur ist als moderne, entkoppelte Webanwendung konzipiert, die aus spezialisierten, miteinander kommunizierenden Komponenten besteht. Dieser modulare Aufbau nach dem Microservices-Prinzip gewährleistet Flexibilität, Skalierbarkeit und Wartbarkeit. Die Architektur gliedert sich in ein Frontend, das als Benutzeroberfläche dient, und ein Backend, das die Kernlogik kapselt und externe KI-Dienste integriert.

HiereineAbbildungderZielarchitektureinfu¨gen,diediefolgendenKomponentenundderen-Interaktionzeigt

Abbildung X: Zielarchitektur des KI-gestützten Self-Service-Systems

Die Hauptkomponenten der Architektur sind:

- **Frontend (Self-Service-Portal):** Eine webbasierte Benutzeroberfläche, entwickelt als Single-Page-Application (SPA) auf Basis eines modernen TypeScript-Frameworks. Sie dient als zentraler, interaktiver Einstiegspunkt für Fachanwender und erfüllt die in Kapitel 3.4 abgeleitete Anforderung nach einer zentralen, geführten Eingabeschnittstelle.
- **Backend (API-Schicht):** Eine zentrale serverseitige Anwendung, implementiert mit einem leistungsfähigen Python-Framework. Sie stellt eine RESTful-API bereit und fungiert als Orchestrator des Gesamtsystems. Zu ihren Aufgaben gehören

die Entgegennahme von Anfragen, die Steuerung der KI-Dienste, die Durchführung der Validierungen und die Verwaltung der Datenpersistenz.

- **Large Language Model (LLM) Service:** Ein externer, über API angebundener KI-Dienst, der auf die Übersetzung von strukturierten Nutzereingaben in valides Streamworks-XML spezialisiert ist.
- **Retrieval-Augmented Generation (RAG) Modul:** Eine im Backend integrierte Komponente, bestehend aus einer Vektordatenbank und einer Abfragelogik. Dieses Modul stellt dem LLM kontextrelevante Informationen aus internen Wissensquellen zur Verfügung.
- **XSD-Validator:** Eine serverseitige Komponente im Backend, die als entscheidende Instanz der Qualitätssicherung dient. Sie validiert jedes generierte XML-Dokument programmgesteuert gegen das offizielle Streamworks XSD-Schema und stellt so die **syntaktische Korrektheit** (Anforderung 2) sicher.

4.1.1 Funktionale und Nicht-funktionale Anforderungen

Funktionale Anforderungen:

- **Multimodale Eingabe:** Nutzer müssen Streams sowohl über einen geführten Wizard als auch über eine konversationelle Chat-Schnittstelle erstellen können.
- **Automatisierte XML-Generierung:** Das System muss aus den Nutzereingaben (egal ob aus Wizard oder Chat) eine syntaktisch korrekte XML-Datei erzeugen.
- **Validierung:** Jede generierte Datei muss gegen formale Regeln und das XSD-Schema geprüft werden.
- **Rollenbasiertes Freigabesystem:** Ein zweistufiger Prozess für Fachanwender und Experten muss implementiert sein.
- **Wissensabruf:** Nutzer müssen über eine RAG-basierte Komponente kontextbezogene Hilfe erhalten.
- **Auditierbarkeit:** Alle Generierungsvorgänge müssen nachvollziehbar protokolliert werden.

Nicht-funktionale Anforderungen (NFRs):

- **Performance:** Die Benutzeroberfläche muss responsiv sein (< 200ms Ladezeit). Der XML-Generierungsprozess inkl. LLM-Anfrage und Validierung sollte eine durchschnittliche Antwortzeit von unter 15 Sekunden anstreben.

- **Zuverlässigkeit:** Das System muss robust gegenüber fehlerhaften oder mehrdeutigen Eingaben sein. Bei Ausfall externer Dienste (z. B. LLM-API) muss ein geregeltes Fehler-Handling greifen.
- **Skalierbarkeit:** Die Architektur muss es ermöglichen, bei steigender Nutzerzahl oder komplexeren Anwendungsfällen einzelne Komponenten unabhängig voneinander zu skalieren.

4.1.2 Datenpersistenz

Zur Gewährleistung der Nachvollziehbarkeit und zur Verwaltung des Systemzustands werden alle relevanten Daten in einer relationalen Datenbank gespeichert. Das Datenmodell sieht primär Entitäten für die Speicherung von Nutzer- und Rolleninformationen sowie für den revisionssicheren Audit-Log vor. Für jeden Generierungslauf werden das Eingabe-JSON, ein Hash des verwendeten Prompts, die genutzte Modell-Version, das generierte XML-Artefakt, der Validierungsreport sowie der finale Freigabe-Status inklusive Zeitstempel und verantwortlichem Nutzer persistiert.

4.2 User Interface und User Experience (UI/UX) des Self-Service-Portals

Die Gestaltung der Benutzeroberfläche ist entscheidend für die Akzeptanz und den Erfolg des Systems. Sie muss die Komplexität der Streamworks-Konfiguration für den Fachanwender abstrahieren und gleichzeitig Flexibilität bieten.

4.2.1 Duale Interaktionsmodi: Wizard und Chat-Assistent

Um unterschiedlichen Nutzerpräferenzen gerecht zu werden, bietet das Portal zwei alternative Wege zur Stream-Erstellung an. Beide Modi sind darauf ausgelegt, denselben robusten Generierungsprozess im Backend anzustoßen.

- **Geführter Wizard (ein schrittweiser Dialog-Assistent):** Dieser Modus ist der primäre Weg für die Erstellung der definierten Standardfälle. Er bietet maximale Sicherheit und Vollständigkeit durch Pflichtfelder und Echtzeit-Validierung und eignet sich besonders für Nutzer, die eine klare Struktur bevorzugen.
- **Konversationeller Chat-Assistent:** Für erfahrenere Nutzer bietet das System eine Chat-Schnittstelle. Hier kann der Anwender seine Anforderung in natürlicher Sprache formulieren (z.B. „Lege einen SAP-Job für P01 an, der den Report

‚Z_PROG_01‘ täglich um 3 Uhr morgens startet.“). Im Backend analysiert das LLM die Eingabe mittels **Entity Extraction**, um die relevanten Parameter zu identifizieren und in die gleiche JSON-Struktur zu überführen, die auch der Wizard erzeugt. Bei Unklarheiten kann der Assistent aktiv Rückfragen stellen.

4.2.2 Definition der Self-Service-fähigen Standardfälle (Scope V1)

Der Scope der ersten Version (V1) wird auf drei häufige, gut standardisierbare Anwendungsfälle begrenzt, die über beide Interaktionsmodi erstellbar sind.

4.2.3 UI-Konzept und Rollenmodell

Das UI-Konzept setzt auf eine klare, intuitive Bedienung und ein striktes, rollenbasiertes Berechtigungskonzept, um den AI²L-Workflow abzubilden:

- **Fachanwender:** Kann Standard-Streams über Wizard oder Chat erstellen und zur Freigabe einreichen.
- **Experte (Orchestration-Team):** Prüft eingereichte Streams, gibt diese frei und ist für den manuellen Export/Import in Streamworks verantwortlich.

4.3 Prozess der KI-gestützten XML-Generierung

Um die Risiken von LLM-Halluzinationen zu minimieren, verfolgt das Konzept einen **Schema-First-Ansatz**.

1. **Input-Aufbereitung und Vorvalidierung:** Die im Frontend gesammelten Daten (aus Wizard oder extrahiert aus Chat) werden in einem strukturierten JSON-Format an das Backend übermittelt und dort einer regelbasierten Vorvalidierung (Pre-Rules) unterzogen.
2. **Deterministisches Template-Mapping:** Im Backend wird eine serverseitige Template-Engine genutzt, um eine XML-Vorlage mit den deterministischen Werten aus dem JSON zu befüllen.
3. **Gezielter LLM-Einsatz für komplexe Lücken:** Das LLM wird nur für semantische Übersetzungen eingesetzt (z.B. Zeitangaben in Cron-Ausdrücke).

4. **Zweistufige Validierung als Qualitätstor:**
   - **Stufe 1 (Pre-Rules):** Regelbasierte Prüfung auf Namenskonventionen, Längen etc.
   - **Stufe 2 (XSD-Validation):** Das finale XML wird gegen das offizielle Schema validiert.
5. **AI²L-Freigabeworkflow und Audit-Log:**
   - Ein generierter Stream wird mit dem Status "Zur Freigabe" gespeichert.
   - Ein Experte prüft das XML im Portal und setzt den Status auf "Freigegeben".
   - Erst danach ist der manuelle Export des XML möglich.

HiereinFlussdiagrammdesAI2L−Freigabeworkflowseinfu¨gen

4.4 Konzeption des RAG-basierten Support-Systems

Das RAG-System unterstützt beide Interaktionsmodi.

- **Architektur und Scope (V1):** Die Wissensbasis besteht aus zwei Vektor-Collections:

1. **"How-to/Handbuch":** Offizielle Streamworks-Dokumentation.
2. **"Beispiel-XML-Snippets":** Anonymisierte, valide XML-Beispiele.

- **Integration in das Portal:** Im Wizard dient das RAG als Inline-Hilfe. Im Chat-Assistenten ist es die primäre Wissensquelle, um Nutzerfragen zu beantworten und die Entity Extraction zu unterstützen.

4.5 Sicherheits- und Datenschutzaspekte

Das Konzept berücksichtigt Sicherheits- und Datenschutzaspekte durch:

- **Rollenbasiertes Zugriffskonzept (Least Privilege)**
- **Secrets Management**
- **Vermeidung sensitiver Daten im Prompt**
- **Ausblick auf Enterprise-LLMs** zur Wahrung der Datenhoheit.

5 Implementierung (~8–9 Seiten / ~2.000–2.250 Wörter)

Ziel: Die technische Umsetzung des Konzepts transparent und nachvollziehbar beschreiben. Der Leser muss verstehen, was du praktisch getan hast.

5.1 Technologiestack und Entwicklungsumgebung

- Begründete Auswahl der Technologien (z.B. React/Vue für Frontend, Python/FastAPI für Backend, OpenAI-API für LLM, eine Vektor-DB wie ChromaDB für RAG).
- Vorstellung der Entwicklungsumgebung und relevanter Bibliotheken.

5.2 Umsetzung des Self-Service-Frontends

- Beschreibung der Implementierung der UI-Komponenten und Formulare.
- Einbindung der API-Schnittstelle.
- (Code-Snippets im Text, vollständiger Code im Anhang).

5.3 Integration des Large Language Models

- Technische Anbindung an die LLM-API.
- Implementierung des Prompt-Engineerings für die XML-Generierung (mit finalem Beispiel-Prompt).
- Verarbeitung und Darstellung der LLM-Antwort im Frontend.

5.4 Implementierung der XSD-Validierung

- Aufbau des finalen XSD-Schemas für einen ausgewählten Standardfall.
- Technische Umsetzung der automatisierten Prüfung und des Feedbacks an den Nutzer bei Fehlern.

5.5 Integration des RAG-Support-Systems

- Prozess der Indexierung der Dokumente (Chunking, Embedding).
- Implementierung der Vektor-Suche und der Kopplung mit dem LLM.
- Darstellung der Q&A-Funktion in der Benutzeroberfläche.

6 Evaluierung und Testing (~6–7 Seiten / ~1.600–1.750 Wörter)

Ziel: Systematisch nachweisen, dass dein Prototyp die in der Einleitung definierten Ziele erreicht. Jede Behauptung muss durch Testergebnisse belegt werden.

6.1.1 Testmethodik und Testumgebung

- Beschreibung des Test-Setups und der verwendeten Testdaten (z.B. 10 typische Standardanforderungen).
- Definition der Testarten: Unit-Tests, Integrationstests, Akzeptanztests mit (simulierten) Fachanwendern.

6.1.2 Bewertung der generierten Streamworks-Konfigurationen (Ziel 1)

- **Metrik 1 (Korrektheit):** Erfolgsquote der XSD-Validierung (z.B. 9 von 10 Tests erfolgreich).
- **Metrik 2 (Vollständigkeit):** Manueller Vergleich der generierten XMLs mit einer manuell erstellten Musterlösung.

6.1.3 Messung der Effizienzsteigerung (Ziel 2 & 3)

- **Metrik 3 (Zeitersparnis):** Zeitmessung im Vergleich: Manueller Prozess vs. Self-Service-Prozess.
- **Metrik 4 (Reduktion Experten-Support):** Analyse, wie viele der Testfälle ohne Experten-Eingriff (Review) auskommen.

6.1.4 Bewertung des RAG-basierten Support-Systems (Ziel 4)

- **Metrik 5 (Qualität der Antworten):** Bewertung der Antworten auf 5 typische Support-Fragen anhand von Relevanz und Korrektheit.

- Qualitatives Feedback (simuliert) zur Nützlichkeit der Funktion.

7 Ergebnisse und Bewertung (~3–4 Seiten / ~1.000 Wörter)

Ziel: Die Ergebnisse aus der Evaluierung zusammenfassen und im Hinblick auf die Forschungsfrage bewerten.

7.1.1 Zusammenfassung der Evaluationsergebnisse

- Präsentation der wichtigsten Kennzahlen in einer übersichtlichen Tabelle.

7.1.2 Bewertung der Zielerreichung

- Diskussion, inwieweit die in der Einleitung formulierten quantitativen und qualitativen Ziele erreicht wurden.

7.1.3 Praktischer Nutzen für Arvato Systems

- Ableitung des konkreten Mehrwerts der Lösung für das Unternehmen.

8 Kritische Reflexion (~3 Seiten / ~750 Wörter)

Ziel: Eine ehrliche und selbstkritische Auseinandersetzung mit den Grenzen und Schwächen deiner Arbeit. Das zeigt wissenschaftliche Reife.

8.1.1 Limitationen der LLM-basierten Generierung

- Diskussion von Zuverlässigkeit, Halluzinationen und der Abhängigkeit vom Prompt-Engineering.

8.1.2 Herausforderungen bei der Qualitätssicherung

- Grenzen der XSD-Validierung (prüft nur Syntax, keine Semantik/Logik).
- Notwendigkeit des menschlichen Reviews (AI²L) als unverzichtbares Sicherheitsnetz.

8.1.3 Skalierbarkeit und Integrationsaufwand

- Diskussion, was nötig wäre, um den Prototyp zu einem produktiven System auszubauen.

9 Fazit und Ausblick (~3 Seiten / ~750 Wörter)

Ziel: Die Arbeit abrunden, die Kernaussage auf den Punkt bringen und zeigen, wie es weitergehen könnte.

9.1.1 Zusammenfassung der Kernergebnisse

- Kurze, prägnante Wiederholung der wichtigsten Erkenntnisse.

9.1.2 Beitrag der Arbeit zur Workload-Automatisierung

- Herausstellung des innovativen Charakters (Kombination von Self-Service, LLM, RAG für WLA).

9.1.3 Zukünftige Entwicklungsmöglichkeiten

- Ideen für den weiteren Ausbau: Unterstützung weiterer Anwendungsfälle, Fine-Tuning des LLMs, proaktive Vorschläge der KI etc.

10 Literaturverzeichnis

Geck, R.-L. (2025): **Zusammenfassung der Experteninterviews zur Streamworks-Automatisierung**. Unveröffentlichtes Dokument im Rahmen der Bachelorarbeit an der FHDW, Paderborn

11 Anhang

11.1 Zusätzliche Abbildungen und Tabellen

11.2 Codebeispiele