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 4
1.1 Problemstellung und Motivation 4
1.2 Zielsetzung der Arbeit 5
1.3 Methodisches Vorgehen 5
1.4 Aufbau der Arbeit 6

2 Theoretische Grundlagen 8
2.1 Workload-Automatisierung mit Streamworks 7
2.2 XML als Konfigurationsstandard in Streamworks 10
2.3 Large Language Models als Lösungstechnologie 13
2.4 Retrieval-Augmented Generation (RAG) im Unternehmenskontext 16
2.5 Self-Service in der IT-Automatisierung 23

3 Analyse der Stream-Erstellungsprozesse 22
3.1 Der Ist-Prozess: Ein manueller, kommunikationsintensiver Workflow 22
3.2 Kernherausforderungen und operative Engpässe 23
3.3 Potenziale durch Self-Service und Automatisierung 23
3.4 Abgeleitete Anforderungen an ein KI-gestütztes System 24

4 Konzeption und Design (~6–7 Seiten / ~1.600–1.750 Wörter) 26
4.1 Zielarchitektur des Self-Service-Systems 33
4.2 User Interface und User Experience (UI/UX) des Self-Service-Portals 35
4.3 Prozess der KI-gestützten XML-Generierung 36
4.4 Konzeption des RAG-basierten Support-Systems 37
4.5 Sicherheits- und Datenschutzaspekte 37

5 Implementierung (10 Seiten / ~2500 Wörter) 27
5.1 Technologiestack und Entwicklungsumgebung 38
5.2 Umsetzung des Self-Service-Frontends 38
5.3 Integration des Large Language Models 38
5.4 Implementierung der XSD-Validierung 38
5.5 Integration des RAG-Support-Systems 38

6 Evaluierung und Testing (~6–7 Seiten / ~1.600–1.750 Wörter) 28

7 Ergebnisse und Bewertung (~3–4 Seiten / ~1.000 Wörter) 29

8 Kritische Reflexion (~3 Seiten / ~750 Wörter) 30

9 Fazit und Ausblick (~3 Seiten / ~750 Wörter) 31

10 Literaturverzeichnis 32

11 Anhang 33

11.1 Zusätzliche Abbildungen und Tabellen 33

11.2 Codebeispiele 33

1 Einleitung

Die Automatisierung von Enterprise-Workloads ist ein zentraler Erfolgsfaktor moderner IT-Landschaften. Leistungsstarke Workload-Management-Systeme wie Streamworks ermöglichen die Orchestrierung komplexer, geschäftskritischer Prozesse. Der Zugang zu diesen Systemen und die Erstellung von Automatisierungsprozessen bleiben jedoch weitgehend auf einen kleinen Kreis von Experten beschränkt. Diese Hürde führt zu Engpässen, verlangsamt die Umsetzung von Anforderungen und verhindert, dass das volle Potenzial der Automatisierung ausgeschöpft wird.

Diese Arbeit untersucht, wie moderne Ansätze der künstlichen Intelligenz (KI) diese Lücke schließen können. Im Fokus steht die Frage, ob die Verarbeitung natürlicher Sprache (Natural Language Processing, NLP) es Fachanwendern ohne technisches Spezialwissen ermöglichen kann, Automatisierungsprozesse eigenständig zu initiieren.

1.1 Problemstellung und Motivation

Die Erstellung von Automatisierungsprozessen, sogenannten Streams, in der Plattform Streamworks ist heute ein wissensintensiver und manueller Vorgang. Fachanwender aus den Geschäftsbereichen müssen ihre Anforderungen, beispielsweise einen täglichen, automatisierten Dateitransfer, an ein spezialisiertes Orchestration-Team weiterleiten. Dieses Team übersetzt die fachliche Anforderung manuell in GUI-geführte Konfiguration. Dieser Prozess ist aus mehreren Gründen ineffizient: Er ist langsam, fehleranfällig durch Medienbrüche und Missverständnisse und erzeugt einen signifikanten Kommunikationsaufwand.

Die Motivation dieser Arbeit entspringt der Beobachtung, dass ein erheblicher Teil dieser Anfragen standardisierbaren Mustern folgt. Der Einsatz hochqualifizierter Fachkräfte für solche repetitiven Aufgaben stellt eine ineffiziente Ressourcennutzung dar. Gleichzeitig eröffnen moderne Large Language Models (LLMs) die Möglichkeit, diesen Übersetzungsprozess zu automatisieren. Sie sind in der Lage, natürlichsprachliche Anforderungen zu verstehen und in strukturierte Datenformate zu überführen.

Die zentrale Motivation ist daher die Entwicklung eines Prototypen, der als Machbarkeitsstudie (Proof of Concept) dient. Es soll untersucht werden, ob ein KI-gestütztes Self-Service-System den Zugang zur Workload-Automatisierung demokratisieren kann. Ein solcher Ansatz verspricht nicht nur eine Effizienzsteigerung, sondern auch eine Erhöhung der Autonomie von Fachanwendern, während sich

Experten auf die Konzeption und Umsetzung komplexer, wertschöpfender Automatisierungslogik konzentrieren können.

1.2 Zielsetzung der Arbeit

Das übergeordnete Ziel dieser Arbeit ist die Konzeption, prototypische Entwicklung und Evaluation eines KI-gestützten Self-Service-Systems zur Erstellung von Streamworks-Konfigurationen. Um die Zielerreichung messbar zu machen, werden die folgenden drei SMART-Ziele definiert:

1. **Realisierung einer validen Konfigurationsgenerierung (Kernfunktion):** Innerhalb des Bearbeitungszeitraums der Bachelorarbeit wird eine Kernfunktion prototypisch implementiert, die natürlichsprachliche Eingaben für drei definierte Standard-Anwendungsfälle (z.B. SAP, File-Transfer) in technisch valide Streamworks-XML-Konfigurationen übersetzt. Die Qualität wird quantitativ gemessen: Ziel ist eine Erfolgsquote von über 90 % bei der Validierung der generierten XML-Dateien gegen ein vordefiniertes XSD-Schema.
2. **Entwicklung eines kontextbezogenen Hilfesystems (Unterstützungsfunktion):** Es wird ein auf Retrieval-Augmented Generation (RAG) basierendes Hilfesystem entwickelt, das auf einer kuratierten Wissensbasis aufsetzt. Das System soll in der Lage sein, einen Katalog von zehn vordefinierten, typischen Anwenderfragen zu beantworten. Die Messung des Erfolgs erfolgt qualitativ durch die Bewertung von Relevanz und Korrektheit der Antworten, wobei eine Zielquote von 80 % korrekter Antworten angestrebt wird.
3. **Integration zu einem funktionsfähigen Gesamtprototypen (Machbarkeitsstudie):** Die Kern- und Unterstützungsfunktionen werden zu einem integrierten Prototypen mit einer interaktiven Weboberfläche zusammengeführt. Dieser Prototyp dient als relevanter Machbarkeitsnachweis (Proof of Concept), der den vollständigen Self-Service-Prozess abbildet. Der Erfolg wird durch einen End-to-End-Test nachgewiesen, bei dem ein Testanwender ohne Expertenwissen in der Lage ist, einen Standard-Stream erfolgreich zu generieren.

1.3 Methodisches Vorgehen

Die Arbeit folgt einem praxisnahen und agilen Forschungsansatz, der theoretische Fundierung mit einer prototypischen Umsetzung verbindet. Das Vorgehen ist vierstufig aufgebaut, um eine systematische Bearbeitung der Forschungsfrage zu gewährleisten:

1. **Grundlagenanalyse:** Eine systematische Literaturrecherche zu den Schlüsseltechnologien – Workload-Automatisierung, Self-Service-Konzepte, Large Language Models und Retrieval-Augmented Generation – schafft den notwendigen theoretischen Bezugsrahmen.
2. **Anforderungsanalyse:** Experteninterviews mit Streamworks-Spezialisten dienen dazu, den bestehenden Prozess, dessen Herausforderungen und konkrete Automatisierungspotenziale zu identifizieren und daraus Anforderungen für den Prototypen abzuleiten.
3. **Prototypische Entwicklung:** Agile und iterative Umsetzung eines Self-Service-Portals, das die primäre Kernfunktion (Sprache-zu-Konfiguration) und die sekundäre Unterstützungsfunktion (RAG-Hilfesystem) nach dem Prinzip eines Minimum Viable Product (MVP) realisiert.
4. **Evaluation:** Systematische Bewertung des entwickelten Prototypen anhand definierter Kriterien, um die funktionale Machbarkeit, die Effizienzpotenziale und die Qualität der KI-gestützten Unterstützung zu bewerten.

1.4 Aufbau der Arbeit

Die Gliederung der Arbeit folgt der logischen Abfolge des methodischen Vorgehens und stellt einen klaren roten Faden von der theoretischen Fundierung bis zur praktischen Evaluation sicher.

- **Kapitel 2** legt die **theoretischen Grundlagen**. Es führt zunächst in die Herausforderungen der Workload-Automatisierung ein und stellt anschließend die beiden zentralen Lösungsansätze vor: die Realisierung von Self-Service durch Sprachverarbeitung (LLMs) und die Kontextualisierung von KI-Antworten durch RAG.
- **Kapitel 3 analysiert den bestehenden Prozess** der Stream-Erstellung. Basierend auf Experteninterviews werden Engpässe identifiziert und konkrete Anforderungen an den zu entwickelnden Prototypen abgeleitet.

- **Kapitel 4** widmet sich der **Konzeption des Prototypen**. Es beschreibt die Gesamtarchitektur und detailliert das Design der primären Kernfunktion (Konfigurationsgenerierung) sowie der unterstützenden RAG-Funktion.
- **Kapitel 5** dokumentiert die **technische Implementierung** des Prototypen und erläutert die Auswahl des Technologiestacks sowie die Umsetzung der konzipierten Komponenten.
- **Kapitel 6** befasst sich mit der **Evaluation des Prototypen**. Die Kern- und Unterstützungsfunktionen werden anhand definierter Metriken systematisch bewertet, um die Zielerreichung zu überprüfen.
- **Kapitel 7** enthält eine **kritische Reflexion** der Arbeit, in der die Methodik, das Vorgehen und die Limitationen des Prototypen beleuchtet werden.
- **Kapitel 8** schließt die Arbeit mit einem **Fazit und Ausblick**. Es fasst die Kernergebnisse zusammen, ordnet den Beitrag der Arbeit ein und skizziert zukünftige Entwicklungsmöglichkeiten.

2 Theoretische Grundlagen

In diesem Kapitel werden die theoretischen und technologischen Grundlagen erarbeitet, die für das Verständnis der in dieser Arbeit entwickelten Lösung notwendig sind. Zunächst wird die Domäne der Workload-Automatisierung als zentrale Herausforderung beleuchtet, indem die Funktionsweise der Plattform Streamworks und die Komplexität ihres Konfigurationsstandards XML dargestellt werden. Darauf aufbauend werden zwei komplementäre KI-basierte Lösungsansätze vorgestellt: Erstens wird das Prinzip des Self-Service durch die Verarbeitung natürlicher Sprache mittels Large Language Models (LLMs) als primärer Lösungsansatz eingeführt. Zweitens wird Retrieval-Augmented Generation (RAG) als unterstützender Ansatz zur Kontextualisierung und Absicherung der KI-Antworten erläutert.

2.1 Workload-Automatisierung als Herausforderung 4-5

Die Automatisierung von IT-Prozessen ist eine fundamentale Säule moderner Unternehmensarchitekturen und ein entscheidender Faktor für die betriebliche Effizienz [1]. Im Zentrum dieser Entwicklung steht die Workload-Automatisierung (WLA), die über einfaches Task-Scheduling hinausgeht und die komplexe Orchestrierung geschäftskritischer Abläufe über heterogene Systemlandschaften hinweg ermöglicht [2]. Enterprise-WLA-Plattformen wie Streamworks agieren in diesem Kontext, um eine flexible Konfiguration von Automatisierungsprozessen zu ermöglichen [3]. Obwohl die Konfiguration primär über eine komplexe grafische Benutzeroberfläche erfolgt, basiert die zugrundeliegende Datenstruktur zur Qualitätssicherung auf etablierten Technologiestandards wie XML und XSD [4]. Die Beherrschung dieser technologischen Komplexität stellt jedoch eine wesentliche Hürde dar, die den Zugang zur Automatisierung auf Experten beschränkt und die Agilität der Fachbereiche einschränken kann. Die Vereinfachung der Konfigurationsprozesse durch Abstraktion und intelligente Assistenzsysteme ist daher ein entscheidender Schritt, um die Potenziale der WLA vollständig auszuschöpfen.

2.1.1 Grundlagen der Workload-Automatisierung

Workload-Automatisierung (WLA) ermöglicht die zentrale Planung, Steuerung und Überwachung von IT-gestützten Geschäftsprozessen, die sich über verschiedene Anwendungen und Plattformen erstrecken können, und trägt maßgeblich zur Steigerung

der Prozesseffizienz bei [5]. Im Gegensatz zu nativen, anwendungsspezifischen Schedulern, die isoliert agieren, dienen WLA-Systeme als zentrale "Master Scheduler", welche die Definition und Verwaltung von Abhängigkeiten zwischen Prozessen auf unterschiedlichen Systemen erlauben und so eine ganzheitliche Sicht auf alle Abläufe sicherstellen [6]. Moderne Geschäftsprozesse stellen oft eine Kette von Transaktionen über diverse Systeme hinweg dar, weshalb die Kernfunktionalität von WLA-Lösungen sowohl die zeitbasierte als auch die ereignisgesteuerte Ausführung von Workflows umfasst, um dynamisch auf betriebliche Anforderungen reagieren zu können [7]. Ein weiterer wesentlicher Aspekt ist die Gewährleistung von Compliance und Sicherheit, was durch detaillierte Protokollierung und Zugriffskontrollen erreicht wird [8]. Durch die Automatisierung wiederkehrender Aufgaben werden manuelle Eingriffe reduziert, was das Fehlerrisiko minimiert und wertvolle Personalressourcen für strategischere Tätigkeiten freisetzt [9].

Mit der fortschreitenden digitalen Transformation steigen die Anforderungen an die Automatisierung, was zu einer erhöhten Komplexität in der Orchestrierung führt [10]. Die Vielfalt der zu steuernden Prozesse und die Einbindung unterschiedlicher Technologien erfordern tiefgehendes Expertenwissen, selbst für die Umsetzung von Standardanforderungen [11]. Diese wachsende Komplexität führt in vielen Unternehmen zu einem Mangel an qualifiziertem Personal und etabliert zeitintensive Abstimmungsprozesse als betriebliche Normalität [12]. Insbesondere die manuelle Übersetzung fachlicher Anforderungen in technische Konfigurationen stellt einen Engpass dar. Hierdurch entsteht eine Lücke zwischen dem Potenzial der WLA-Systeme und ihrer praktischen Zugänglichkeit für Fachanwender, was den Bedarf an Self-Service-Ansätzen zur Effizienzsteigerung begründet.

2.1.2 Streamworks als Enterprise WLA-Plattform

Streamworks von Arvato Systems ist eine Enterprise-Plattform, die als Workload-Automation- und Service-Orchestration-Lösung konzipiert wurde und auf Standard-Windows-Komponenten im Backend basiert, was geringe Einarbeitungs- und Betriebsaufwände ermöglicht [13]. Das Kernprinzip der Plattform ist die zentrale Erstellung von Workflows durch die Verknüpfung einzelner Prozessschritte, sogenannter Jobs, zu einer Prozesskette, die als "Stream" bezeichnet wird [14]. Die Ausführung dieser Prozesse erfolgt plattformübergreifend auf Zielsystemen wie Windows, Unix oder Mainframes über schlanke Agenten, die von einem zentralen Processing Server

gesteuert werden [15]. Die Kommunikation zwischen allen Komponenten ist dabei verschlüsselt, um hohen Sicherheitsanforderungen zu genügen [16].

Die Konfiguration der Streams und die Definition ihrer komplexen Abhängigkeiten erfolgen in einer grafischen Oberfläche, dem "Stream Designer" [17]. Erstellte Streams können über ein Export/Import-Utility in ein XML-Format überführt werden, was den Austausch zwischen verschiedenen Systemen, beispielsweise einer Test- und einer Produktionsumgebung, ermöglicht [19]. Diese XML-Struktur stellt die technische Repräsentation eines Streams dar und bietet somit einen entscheidenden Anknüpfungspunkt für eine automatisierte Generierung.

2.1.3 XML als Konfigurationsstandard

Die Extensible Markup Language (XML) ist eine universelle Metasprache zur Darstellung strukturierter Daten in einem plattform- und anwendungsunabhängigen Textformat [20]. Als eine Untermenge der SGML wurde XML mit dem Ziel entworfen, die Lesbarkeit für Mensch und Maschine sowie eine einfache Verarbeitung zu gewährleisten [21]. Die grundlegende Struktur eines XML-Dokuments besteht aus einem Baum von ineinander verschachtelten Elementen, die durch Start- und End-Tags begrenzt werden und sowohl weitere Elemente als auch Zeichendaten sowie Attribute enthalten können [22]. Eine fundamentale Regel ist die der Wohlgeformtheit, die eine strikte Einhaltung syntaktischer Vorgaben wie die korrekte Schachtelung verlangt, wobei ein XML-Prozessor die Verarbeitung bei Verstößen verweigert [23].

Die strikte Syntaxprüfung ist entscheidend für die Interoperabilität, da sie den zuverlässigen Austausch strukturierter Daten sicherstellt und durch selbstbeschreibende Tags eine klare Trennung von Inhalt und Form erzwingt [24]. Durch die Verwendung von Namensräumen (`namespaces`) wird zudem die Eindeutigkeit von Element- und Attributnamen gewährleistet, was die Kombination verschiedener XML-Vokabulare ermöglicht [25]. In Systemen wie Streamworks wird XML aufgrund dieser Eigenschaften als Konfigurationsstandard genutzt, da es eine robuste Methode zur Serialisierung komplexer, hierarchischer Konfigurationsdaten bietet [26]. Gerade diese Komplexität und Striktheit, die aus technischer Sicht vorteilhaft ist, stellt für Fachanwender ohne XML-Kenntnisse eine erhebliche Barriere dar und begründet den Bedarf an intelligenten Schnittstellen, die natürlichsprachliche Anforderungen in dieses formalisierte Format übersetzen können.

2.1.4 Qualitätssicherung durch XML Schema Definition (XSD)

Während die Wohlgeformtheit die syntaktische Korrektheit eines XML-Dokuments sicherstellt, reicht sie allein nicht aus, um die inhaltliche und strukturelle Richtigkeit zu garantieren, da ein Dokument syntaktisch korrekt sein, aber dennoch gegen logische Vorgaben verstoßen kann [27]. Um diese Lücke zu schließen, wird das Konzept der Gültigkeit (`validity`) eingeführt, das durch ein sogenanntes Schema definiert wird, wobei das W3C die XML Schema Definition Language (XSD) als Standard spezifiziert hat [28]. Ein XML Schema dient als Bauplan, der das Vokabular, die Grammatik sowie die Hierarchie und Häufigkeit von Elementen festlegt und so die Qualitätssicherung in datenintensiven Systemen unterstützt [29]. Ein XML-Dokument gilt als gültig, wenn es nicht nur wohlgeformt ist, sondern auch vollständig den Regeln seines zugeordneten XSD-Schemas entspricht [30].

Eine wesentliche Stärke von XSD ist die Fähigkeit zur Definition von Datentypen, die es erlaubt, den Inhalt von Elementen und Attributen auf vordefinierte oder benutzerdefinierte Typen zu beschränken [31]. Im Kontext der Workload-Automatisierung mit Streamworks stellt ein XSD-Schema sicher, dass importierte XML-Konfigurationen nicht nur syntaktisch korrekt, sondern auch semantisch vollständig und valide sind [32]. Durch die formale Beschreibung der erwarteten Datenstruktur wird die Interoperabilität erhöht und die Robustheit des Automatisierungssystems gestärkt, indem Konfigurationsfehler vor der produktiven Ausführung verhindert werden [33]. Die Notwendigkeit, XSD-valide Konfigurationen zu erzeugen, stellt somit eine Kernanforderung für jedes System dar, das den Konfigurationsprozess automatisieren oder vereinfachen soll. Ein solches System muss nicht nur die fachliche Anforderung korrekt interpretieren, sondern diese auch in eine technisch einwandfreie XML-Struktur überführen, die den strengen Qualitätsprüfungen des Schemas standhält.

2.2 Self-Service durch Sprachverarbeitung

Die Automatisierung von IT-Prozessen stellt eine fundamentale Säule moderner Unternehmensarchitekturen dar und ist ein entscheidender Faktor für die betriebliche Effizienz. Enterprise-Workload-Automation-Plattformen ermöglichen dabei die Orchestrierung komplexer, geschäftskritischer Abläufe über heterogene Systemlandschaften hinweg, jedoch stellt die Beherrschung der technologischen Komplexität eine wesentliche Hürde dar, die den Zugang zur Automatisierung auf Experten beschränkt [1]. Diese Arbeit untersucht daher einen Lösungsansatz, der diese

Hürde durch die Kombination zweier zentraler Konzepte überwinden soll: das Self-Service-Prinzip und die Verarbeitung natürlicher Sprache mittels Large Language Models (LLMs). Ziel ist es, eine technologische Brücke zu schaffen, die es Fachexperten ohne tiefgreifendes IT-Wissen ermöglicht, Automatisierungsprozesse selbstständig zu initiieren und zu konfigurieren, wodurch der manuelle und kommunikationsintensive Prozess der Konfigurationserstellung bei Streamworks grundlegend transformiert wird.

2.2.1 Das Self-Service-Prinzip in der IT

Das Self-Service-Prinzip, realisiert durch Self-Service-Technologien (SSTs), ersetzt zunehmend die interpersonelle oder persönliche Interaktion zwischen Dienstleister und Kunde durch automatisierte Systeme [2]. Im Unternehmenskontext führt dies zur Entwicklung des Mitarbeiters zum "Prosumer" – einem Produzenten und Konsumenten von Dienstleistungen zugleich –, der in die Lage versetzt wird, Serviceprozesse eigenständig zu steuern, um seine spezifischen Bedürfnisse zu erfüllen [3]. Das primäre Ziel solcher Technologien ist nicht nur die Steigerung der betrieblichen Effizienz durch die Reduzierung von Arbeitskosten und die Standardisierung der Servicebereitstellung, sondern auch die Verbesserung der Mitarbeitererfahrung durch erhöhte Bequemlichkeit, Zeitersparnis und Autonomie [4].

Dieser Wandel hin zu einer stärkeren Automatisierung wird jedoch durch die oft geringe Benutzerfreundlichkeit bestehender Systeme behindert, denn komplexe Schnittstellen, eine mangelnde intuitive Bedienbarkeit und die Notwendigkeit von Domänenwissen stellen erhebliche Adoptionsbarrieren dar, insbesondere für nicht-technische Anwender [5]. Um diese Herausforderungen zu überwinden, ist ein menschenzentrierter Automatisierungsansatz (Human-Centered Automation, HCA) erforderlich, der die Bedürfnisse und Präferenzen der Nutzer in den Mittelpunkt des Designs stellt [6]. Anstatt von den Mitarbeitern zu verlangen, sich an starre und komplizierte Systeme anzupassen, müssen Automatisierungslösungen so gestaltet sein, dass sie sich den Arbeitsabläufen und Fähigkeiten der Nutzer anpassen [7]. Im Kontext der Workload-Automatisierung bei Streamworks bedeutet dies die Schaffung einer intuitiven Schnittstelle, die es Fachanwendern ermöglicht, ihre Anforderungen in natürlicher Sprache zu formulieren, wodurch die Komplexität der zugrundeliegenden Technologie abstrahiert und der Zugang zur Automatisierung demokratisiert wird.

2.2.2 Grundlagen von Large Language Models (LLMs)

Large Language Models (LLMs) stellen die jüngste Entwicklungsstufe im Bereich der natürlichen Sprachverarbeitung (Natural Language Processing, NLP) dar und haben sich als eine revolutionäre technologische Entwicklung in der künstlichen Intelligenz erwiesen [8]. Ihre grundlegende Fähigkeit besteht darin, Wissen über die Welt und die Sprache durch die Bearbeitung riesiger Textmengen im Rahmen des Pre-Trainings zu erwerben, um so ein universelles Modell zu schaffen, das vielfältige Probleme bearbeiten kann [9]. Dieser Paradigmenwechsel führte von spezialisierten, aufgabenspezifisch trainierten Systemen hin zu Basismodellen (Foundation Models), die durch großflächiges, selbstüberwachtes Lernen auf unstrukturierten Daten trainiert und anschließend für spezifische Anwendungen durch Methoden wie Fine-Tuning oder Prompting angepasst werden [10].

Die technologische Grundlage moderner LLMs ist die Transformer-Architektur, welche auf die sequenzielle Verarbeitung von rekurrenten Netzen verzichtet und stattdessen Self-Attention-Mechanismen nutzt, um globale Abhängigkeiten im Text zu erfassen und eine kontextualisierte Repräsentation jedes Wortes zu erstellen [11]. Durch die millionenfache Wiederholung von Vorhersageaufgaben auf einem riesigen und diversen Datensatz während des Pre-Trainings erlernt das Modell komplexe sprachliche Muster und Faktenwissen [12]. Eine zentrale Eigenschaft, die sich daraus ergibt, ist die Fähigkeit des In-Context-Learnings, bei der das Modell eine neue Aufgabe nur anhand einiger weniger Beispiele (Few-Shot) oder einer einfachen Anweisung (Zero-Shot) erlernen kann, ohne dass eine Anpassung der Modellparameter erforderlich ist [13]. Diese Fähigkeit positioniert LLMs als ideale Technologie, um als Schnittstelle zwischen menschlicher Sprache und maschinenlesbaren, strukturierten Formaten zu fungieren und somit den Self-Service-Ansatz bei Streamworks technologisch zu untermauern.

2.2.3 Anpassung von LLMs durch Prompting

Eine der leistungsfähigsten Eigenschaften von LLMs ist ihre Fähigkeit zur schnellen Adaption an neue Aufgaben durch In-Context-Learning, ohne dass eine zeit- und datenintensive Neukonfiguration der Modellparameter (Fine-Tuning) erforderlich ist [14]. Dieser Prozess, bei dem Aufgaben und Demonstrationsbeispiele rein über die Texteingabe an das Modell spezifiziert werden, wird als Prompting bezeichnet [15]. Je nach Anzahl der im Prompt bereitgestellten Beispiele wird zwischen drei Hauptkategorien unterschieden: "Zero-Shot", "One-Shot" und "Few-Shot" [16]. Im Zero-

Shot-Szenario erhält das Modell lediglich eine natürlichsprachliche Anweisung zur Beschreibung der Aufgabe, ohne jegliche Demonstrationsbeispiele [17]. Im One-Shot-Szenario wird zusätzlich zur Aufgabenbeschreibung ein einzelnes Beispiel bereitgestellt, was der Art und Weise ähnelt, wie Aufgaben an menschliche Bearbeiter kommuniziert werden, um Inhalt und Format der Aufgabe zu verdeutlichen [18].

Beim Few-Shot-Learning, der leistungsfähigsten der drei Methoden, wird das Modell mit so vielen Demonstrationsbeispielen konditioniert, wie in das Kontextfenster des Modells passen, typischerweise zwischen 10 und 100 Beispiele [19]. Durch diese Beispiele, die aus einem Kontext und einer gewünschten Vervollständigung bestehen, lernt das Modell, das Muster zu erkennen und auf neue, ähnliche Kontexte anzuwenden, indem es einfach die wahrscheinlichste Token-Sequenz vorhersagt [20]. Die Effektivität des In-Context-Learnings steigt dabei dramatisch mit der Größe des Modells an, was darauf hindeutet, dass größere Modelle fähigere Meta-Lerner sind, die kontextuelle Informationen effizienter nutzen können [21]. Diese Fähigkeit, Aufgaben "on-the-fly" zu erlernen, ist entscheidend für den Einsatz von LLMs in einem Self-Service-Portal, da es die flexible Übersetzung unterschiedlich formulierter Benutzeranforderungen in eine streng strukturierte Konfiguration wie XML ermöglicht, ohne für jeden Anwendungsfall ein eigenes Modell trainieren zu müssen.

2.2.4 LLMs als Schnittstelle von Sprache zu Konfiguration

Large Language Models (LLMs) fungieren als eine entscheidende technologische Schnittstelle, um die Kluft zwischen der menschlichen, unstrukturierten Sprache und der streng formalisierten, strukturierten Welt der Maschinenkonfigurationen zu überbrücken und sind die Schlüsseltechnologie für einen menschenzentrierten Automatisierungsansatz (HCA), der darauf abzielt, komplexe Systeme für Nutzer ohne technisches Spezialwissen zugänglich zu machen [22]. Während traditionelle Automatisierungsansätze wie Robotic Process Automation (RPA) in ihren kognitiven Fähigkeiten limitiert sind und oft an Aufgaben scheitern, die über regelbasierte Prozesse hinausgehen, erweitern LLMs diesen Horizont durch ihre Fähigkeit, natürlichsprachliche Anfragen zu verstehen und in ausführbare Workflows zu übersetzen, die beispielsweise in Formaten wie JavaScript Object Notation (JSON) oder XML repräsentiert werden können [23]. Dieser Prozess der Umwandlung von "Words to Workflows" ermöglicht es, komplexe Geschäftsprozesse zu automatisieren, indem Benutzereingaben in eine Sequenz von ausführbaren Schritten übersetzt werden [24].

Die Herausforderung bei der Integration von KI und Automatisierung liegt oft in der mangelnden Standardisierung und Interoperabilität zwischen verschiedenen Werkzeugen, was die Skalierbarkeit behindern kann [25]. Im spezifischen Kontext von Streamworks erfordert die manuelle Übersetzung fachlicher Anforderungen in eine valide XML-Struktur erheblichen Kommunikations- und Umsetzungsaufwand, wobei ein LLM als intelligenter "Übersetzer" dienen kann, der die Anforderung eines Fachanwenders entgegennimmt und diese in eine technisch valide Konfiguration überführt. Moderne Bibliotheken demonstrieren, dass LLMs mittels weniger Beispiele (Few-Shot-Examples) dazu angeleitet werden können, zuverlässig strukturierte und quelltextgenaue Ausgaben zu erzeugen, die einem vordefinierten Schema folgen [26]. Diese Fähigkeit ist von zentraler Bedeutung, um die für Streamworks erforderliche, XSD-validierbare XML-Struktur zu generieren und somit eine technologische Brücke zu schlagen, die den Zugang zur Workload-Automatisierung demokratisiert und die Effizienz steigert.

2.2.5 Potenziale und Limitationen von LLMs: Das Halluzinationsproblem

Trotz ihrer beeindruckenden Fähigkeiten sind LLMs nicht frei von Schwächen, von denen das sogenannte Halluzinieren eine der bedeutendsten darstellt [27]. Im Kontext der Natural Language Generation (NLG) wird Halluzination als die Generierung von Inhalten definiert, die unsinnig sind oder dem bereitgestellten Quellenkontext untreu sind [28]. Dieses Phänomen untergräbt die Zuverlässigkeit von LLM-basierten Systemen und stellt insbesondere in Anwendungsfällen, die eine hohe faktische Korrektheit erfordern, ein erhebliches Risiko dar. Die Forschung unterscheidet primär zwischen zwei Arten von Halluzinationen: intrinsische und extrinsische Halluzinationen [29].

Intrinsische Halluzinationen treten auf, wenn die generierte Ausgabe direkt im Widerspruch zu den Informationen aus der Quelle steht [30]. Im Anwendungsfall der Streamworks-Konfigurationserstellung könnte dies bedeuten, dass ein LLM einen falschen Parameterwert generiert, obwohl der korrekte Wert in der Benutzereingabe explizit genannt wurde. Extrinsische Halluzinationen hingegen umfassen generierte Informationen, die sich aus der Quelle weder belegen noch widerlegen lassen [31]. Ein Beispiel hierfür wäre die Ergänzung eines Automatisierungsprozesses um einen zusätzlichen, vom Nutzer nicht spezifizierten Prozessschritt. Solche extrinsischen Ergänzungen sind nicht zwangsläufig fehlerhaft, da sie auf faktisch korrektem, aus dem Pre-Training erlerntem Wissen basieren können; sie erhöhen jedoch das Risiko von sachlich falschen oder vom Nutzer nicht intendierten Ergebnissen, da ihre Herkunft

unklar ist [32]. Die Ursachen für Halluzinationen sind vielfältig und reichen von Artefakten in den Trainingsdaten über fehlerhafte Dekodierungsstrategien bis hin zu einer Verzerrung durch das im Modell parametrisch gespeicherte Wissen, welches das Modell gegenüber den Informationen aus der konkreten Nutzereingabe priorisiert [33]. Für die automatisierte Erstellung von XML-Konfigurationen ist dieses Risiko kritisch, da jede Form von Halluzination zu einer technisch invaliden oder semantisch falschen Konfiguration führen kann. Dies unterstreicht die Notwendigkeit von robusten Validierungsmechanismen wie der XSD-Schema-Prüfung und Ansätzen wie Retrieval-Augmented Generation (RAG), die darauf abzielen, die Generierung durch verlässliche, externe Wissensquellen abzusichern und so das Halluzinationsrisiko zu minimieren

2.3 Kontextualisierung durch Retrieval-Augmented Generation (RAG)

Die direkte Nutzung von großen Sprachmodellen (Large Language Models, LLMs) im Unternehmenskontext, wie es für die Prozessautomatisierung bei Streamworks geplant ist, stößt auf grundlegende Limitationen. Insbesondere die Verarbeitung interner, domänenspezifischer Wissensbestände stellt eine zentrale Hürde dar [1]. Um diesen Herausforderungen zu begegnen und das Potenzial von LLMs für unternehmensinterne Daten nutzbar zu machen, etabliert sich der Ansatz der Retrieval-Augmented Generation (RAG) [2]. Dieser Lösungsansatz zielt darauf ab, die Verlässlichkeit der generierten Antworten signifikant zu steigern, indem er die implizite Wissensbasis eines vortrainierten, parametrischen Modells mit einem expliziten, nicht-parametrischen Wissensspeicher kombiniert [3].

2.3.1 Motivation und Konzept von RAG

Große Sprachmodelle speichern zwar Wissen in ihren internen Parametern, können dieses aber nicht gezielt und fehlerfrei abrufen und erfinden daher mitunter Fakten, weshalb das Kernkonzept von RAG darin besteht, dem LLM externe Wissensquellen zur Seite zu stellen, sodass das Wissen aus diesen angebundenen Quellen entnommen wird [4]. Im Kontext der Automatisierung von Support- und Entwicklungsprozessen bei Streamworks stellt die Tendenz zur Halluzination ein inakzeptables Risiko dar [5]. Durch das Einbetten des Generierungsprozesses in einen externen Informationsabruf-Mechanismus werden die Verlässlichkeit und der Detailreichtum der generierten Inhalte signifikant verbessert [6].

Das Sprachmodell erhält somit die Aufgabe, die extern abgerufenen Suchergebnisse im Sinne der ursprünglichen Anfrage zu verarbeiten und zusammenzufassen, anstatt auf potenziell ungenaues, internes Wissen zurückgreifen zu müssen [7]. Dieser Ansatz ermöglicht es, das volle Potenzial von LLMs auch für interne Dokumentenbestände und Datenbanken zu nutzen, wie sie bei Streamworks in Form von technischen Dokumentationen vorliegen, ohne dass dafür ein aufwendiges Nachtrainieren des Modells erforderlich ist [8]. Die Kombination aus einem vortrainierten Sequenz-zu-Sequenz-Modell als parametrischem Speicher und einem dichten Vektorindex von Unternehmensdokumenten als nicht-parametrischem Speicher ermöglicht die Erstellung von Antworten, die durch aktuelle und genaue Informationen fundiert sind [9].

2.3.2 Technische Architektur eines RAG-Systems

Ein RAG-System besteht im Kern aus zwei Hauptkomponenten: einem Retrieval- und einem Generierungsmodul [10]. Der gesamte Prozess lässt sich in eine Indexierungs- und eine Abfragephase unterteilen, die sicherstellen, dass relevante Informationen effizient gefunden und zur Antwortgenerierung genutzt werden können [11]. Die Architektur ist darauf ausgelegt, eine Nutzeranfrage durch die Anreicherung mit externen Daten zu kontextualisieren, bevor sie von einem LLM verarbeitet wird [12]. Für den Anwendungsfall eines intelligenten Hilfesystems bei Streamworks ist diese Architektur von besonderem Vorteil, da sie es ermöglicht, die generativen Fähigkeiten eines LLMs direkt an die internen Wissensquellen des Unternehmens zu koppeln und somit präzise, kontextbezogene und auf verifizierbaren Daten basierende Unterstützung für Entwickler und Support-Mitarbeiter zu leisten [13].

Indexierungsprozess: Die Grundlage für den Informationsabruf bildet der Indexierungsprozess, bei dem die Wissensbasis aufbereitet wird. Zunächst werden die Dokumente in kleinere, semantisch zusammenhängende Abschnitte, sogenannte Chunks, zerlegt, da die semantische Einbettung für kurze Abschnitte präziser ist und eine zu große Chunk-Länge das Risiko von "Rauschen" in den an das LLM übergebenen Kontexten erhöht [14]. Dieser Schritt ist für Streamworks entscheidend, da technische Dokumentationen und Code-Dateien oft sehr lang sind, die für eine spezifische Nutzeranfrage relevante Information jedoch meist in wenigen Sätzen oder einem Code-Block enthalten ist. Anschließend wird für jeden dieser Chunks mithilfe eines spezialisierten Embedding-Modells eine semantische Repräsentation in Form eines hochdimensionalen Vektors erzeugt [15]. Diese Vektoren werden zusammen mit den ursprünglichen Text-Chunks in einer für Ähnlichkeitssuchen optimierten

Vektordatenbank gespeichert, wobei Open-Source-Lösungen wie Milvus, Weaviate oder Qdrant etablierte Technologien für den produktiven Einsatz darstellen [16].

Abfrageprozess: Der Abfrageprozess wird durch eine Nutzeranfrage initiiert und stellt das Herzstück des interaktiven Hilfesystems dar. Die Anfrage wird optional von einem LLM umformuliert (Rewrite), um die Suchintention zu präzisieren und den Kontext aus vorherigen Dialogen für mehrdeutige Anfragen wie "Erkläre dieses Konzept genauer" aufzulösen [17]. Daraufhin wird die Anfrage ebenfalls in einen Vektor umgewandelt, um mittels einer Ähnlichkeitssuche, typischerweise basierend auf der Kosinus-Ähnlichkeit, die semantisch passendsten Text-Chunks aus der Vektordatenbank abzurufen (Retrieve) [18]. Diese semantische Suche ist für ein Hilfesystem leistungsfähiger als eine reine Stichwortsuche, da sie es Nutzern ermöglicht, Probleme in natürlicher Sprache zu beschreiben, ohne die exakten Fachbegriffe der Dokumentation kennen zu müssen. Für Anfragen nach spezifischen Funktionsnamen oder Fehlermeldungen bei Streamworks könnte eine hybride Suche, die semantische Vektorsuche mit einer unscharfen Schlüsselwortsuche kombiniert, die Retrieval-Qualität weiter steigern [19]. Die zurückgegebenen Dokumente werden anschließend neu geordnet (Rerank), um die relevantesten Chunks zu priorisieren, bevor eine Konsolidierungsstrategie (Consolidate) die für die Übergabe an das LLM notwendigen Informationen auswählt, was insbesondere zur Einhaltung von Token-Limits der LLM-Dienste notwendig ist [20]. Abschließend verarbeitet das Generierungsmodul (Reader), das LLM, den angereicherten Prompt und generiert daraus die finale, quellengestützte Antwort, die dem Nutzer präsentiert wird, wodurch die Rolle des LLMs von einem potenziell unzuverlässigen Wissensspeicher zu einem zuverlässigen Synthesizer von verifizierten Informationen transformiert wird [21].

2.3.3 Vergleich: RAG vs. Fine-Tuning

Zur Anpassung von LLMs an spezifische Domänen, wie die internen Entwicklungsprozesse bei Streamworks, haben sich mit RAG und Fine-Tuning (FT) zwei unterschiedliche Paradigmen etabliert [22]. Während RAG die Generierung durch externen, zur Laufzeit abgerufenen Kontext verbessert, zielt FT darauf ab, Wissen direkt in die internen Parameter des Modells durch ein erneutes Training auf domänenspezifischen Daten zu integrieren [23]. Für den Anwendungsfall bei Streamworks erweist sich der RAG-Ansatz als überlegen, da Fine-Tuning signifikante Nachteile in Bezug auf Flexibilität, Wartbarkeit und Kosten aufweist.

Der entscheidende Vorteil von RAG liegt in der dynamischen Aktualisierbarkeit des Wissens, denn durch den Austausch des zugrundeliegenden Dokumentenindex kann das Systemwissen ohne erneutes Training an neue Gegebenheiten angepasst werden, was für die sich ständig weiterentwickelnden Projektdokumentationen bei Streamworks unerlässlich ist [24]. Fine-Tuning hingegen stellt einen statischen Ansatz dar: Jede Aktualisierung der Wissensbasis oder Weiterentwicklung des Basis-LLMs erfordert einen erneuten, rechen- und zeitintensiven Trainingsprozess [25]. Dieser hohe Aufwand in der Vorbereitungs- und Wartungsphase macht FT für dynamische Unternehmensumgebungen unpraktikabel [26]. Ferner sollte Fine-Tuning grundsätzlich nicht dafür eingesetzt werden, einem LLM Faktenwissen anzutrainieren; diese Aufgabe wird durch den Retrieval-Mechanismus von RAG effizienter und zuverlässiger gelöst [27].

Obwohl ein gezieltes Fine-Tuning theoretisch die Leistung eines RAG-Systems ergänzen kann, indem es das Modell auf einen bestimmten Antwortstil oder den Umgang mit speziellen Datenformaten trainiert, handelt es sich dabei um eine komplexe Optimierung, die mit moderatem Aufwand durch Parameter-effiziente Methoden (PEFT) wie LoRA technisch machbar ist [28]. Angesichts des Ziels, zunächst ein robustes und wartbares Basissystem zu entwickeln, wird im Rahmen dieser Arbeit auf ein Fine-Tuning verzichtet. Die Priorität liegt auf der Ausschöpfung der Potenziale des RAG-Ansatzes, der eine bessere Skalierbarkeit bei wachsenden Wissensdatenbanken verspricht und die Grundlage für ein flexibles Hilfesystem bei Streamworks bildet [29].

2.3.4 Herausforderungen und Fehlerpotenziale

Obwohl RAG-Systeme darauf ausgelegt sind, die Zuverlässigkeit von LLMs zu erhöhen, sind sie selbst nicht frei von potenziellen Fehlerquellen, weshalb ihre Validierung oft nur im laufenden Betrieb realisierbar ist und ihre Robustheit sich evolutionär entwickelt [30]. Die Komplexität des Gesamtsystems bedeutet zudem, dass seine Leistungsfähigkeit nicht durch die isolierte Bewertung der einzelnen Komponenten erfasst werden kann, sondern das Zusammenspiel von Retrieval und Generierung bewertet werden muss [31]. Bei der Konzeption und Implementierung eines RAG-Systems für den Einsatz bei Streamworks müssen sieben zentrale Fehlerpunkte berücksichtigt werden, die in den verschiedenen Phasen des Abfrageprozesses auftreten können [32].

Fehlerpunkt	Phase	Beschreibung

FP1: Missing Content	Retrieval	Die Antwort auf eine Frage ist in den indexierten Dokumenten nicht vorhanden. Das System könnte dennoch versuchen, eine Antwort zu halluzinieren, anstatt zu signalisieren, dass keine Information verfügbar ist.
FP2: Missed Top Ranked	Retrieval	Die korrekte Information ist zwar im Wissensspeicher vorhanden, wird aber vom Retrieval-Algorithmus nicht unter den Top-K-Ergebnissen platziert und erreicht somit nie die Generierungsphase.
FP3: Not in Context	Consolidation	Das relevante Dokument wird zwar abgerufen, aber im Konsolidierungsschritt, der die an das LLM übergebenen Kontexte auswählt und zusammenfügt, fälschlicherweise herausgefiltert.
FP4: Not Extracted	Generation	Die Antwort ist im an das LLM übergebenen Kontext enthalten, wird vom Modell jedoch nicht korrekt extrahiert. Dies kann durch zu viel "Rauschen" oder widersprüchliche Informationen im Kontext verursacht werden.
FP5: Wrong Format	Generation	Das LLM ignoriert spezifische Formatierungsanweisungen in der Anfrage und gibt die Antwort in einem falschen Format aus (z.B. als Fließtext statt als Tabelle).
FP6: Incorrect Specificity	Generation	Die generierte Antwort ist zwar korrekt, aber entweder zu allgemein oder zu spezifisch, um die eigentliche Intention der Nutzeranfrage zufriedenstellend zu beantworten.
FP7: Incomplete	Generation	Die Antwort ist unvollständig, obwohl die fehlenden Informationen im Kontext vorhanden waren und hätten extrahiert werden können.

Tabelle 1: Sieben zentrale Fehlerpunkte im RAG-Prozess (basierend auf Barnett et al., 2024) [33]

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.

---

[1] Geck, R.-L. (2025). *Zusammenfassung der Experteninterviews zur Streamworks-Automatisierung*

[2] Geck, R.-L. (2025). *Zusammenfassung der Experteninterviews zur Streamworks-Automatisierung, S. 3*

[3] Geck, R.-L. (2025). *Zusammenfassung der Experteninterviews zur Streamworks-Automatisierung, S. 4*

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

---

[4] Geck, R.-L. (2025). *Zusammenfassung der Experteninterviews zur Streamworks-Automatisierung, S. 4*

[5] Geck, R.-L. (2025). *Zusammenfassung der Experteninterviews zur Streamworks-Automatisierung, S. 5*

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

---

[6] Geck, R.-L. (2025). *Zusammenfassung der Experteninterviews zur Streamworks-Automatisierung, S. 6*

[7] Geck, R.-L. (2025). *Zusammenfassung der Experteninterviews zur Streamworks-Automatisierung, S. 7*

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 8-10

- **Ziel:** Eine detaillierte technische Blaupause für die Lösung entwerfen.

- **Stichpunkte:**

  - **4.1 Gesamtarchitektur:** Ausführliche Vorstellung der Systemkomponenten und deren Interaktion (detailliertes Architekturdiagramm).
  - **4.2 UI-Konzept und User-Flows:** Detaillierte Beschreibung der dualen UI (Wizard vs. Chat) mit Mockups oder Wireframes. Beschreibung der User-Flows für Fachanwender und Experten (AI²L-Workflow).
  - **4.3 Kernfunktion (Konfigurationsgenerierung):** Detaillierte Beschreibung des mehrstufigen Prozesses, inkl. Prompt-Design und Schema-Definition für `Langextract`.
  - **4.4 Unterstützende Funktion (RAG-Hilfesystem):** Detaillierte Konzeption der Wissensbasis und der RAG-Abfragekette (Prompt-Templates für RAG).

5 Implementierung 10-12

- **Ziel:** Die technische Umsetzung transparent dokumentieren.

- **Stichpunkte:**

    - **5.1 Technologiestack:** Begründete Auswahl der konkreten Bibliotheken.
    - **5.2 Umsetzung der Kernfunktion:** Relevante Code-Ausschnitte für die LLM-Ansteuerung, Schema-Definition und XSD-Validierung.
    - **5.3 Umsetzung des Hilfesystems:** Code-Beispiele für Indexierung und RAG-Abfragekette.
    - **5.4 Integration zum Gesamtprototypen:** Beschreibung der API-Endpunkte und des Zusammenspiels von Frontend und Backend.

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

Google (2025). google/langextract: A Python library for extracting structured information from unstructured text using LLMs with precise source grounding and interactive visualization. Online unter: https://github.com/google/langextract, letzter Abruf am 22.09.2025

11 Anhang

11.1 Zusätzliche Abbildungen und Tabellen

11.2 Codebeispiele