

Fachhochschule der Wirtschaft
-FHDW-
Paderborn

Bachelorthesis

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

Prüfer:
Prof. Dr. Christian Ewering
Lars Kahra

Verfasser:
Ravel-Lukas Geck
Lagesche Str. 258
32758 Detmold
Matrikelnummer: 101455

Wirtschaftsinformatik
Business Process Management

Eingereicht am:
20.10.2025

Executive Summary

Diese Bachelorarbeit untersucht, wie sich Self-Service-Prinzipien und moderne Methoden der Künstlichen Intelligenz (KI) kombinieren lassen, um Prozesse der Workload-Automatisierung effizienter und zugänglicher zu gestalten. Am Beispiel der Plattform *Streamworks* von Arvato Systems wird gezeigt, dass die Verbindung von **Large Language Models (LLMs)** und **Retrieval-Augmented Generation (RAG)** die Erstellung technischer Standard-Konfigurationen vereinfachen kann.

Im Rahmen des **Design-Science-Research-Ansatzes** wurden theoretische Grundlagen, Prozessanalysen und eine prototypische Implementierung zusammengeführt. Der entwickelte Prototyp generiert aus natürlichsprachlichen Eingaben valide XML-Konfigurationen, integriert ein RAG-basiertes Hilfesystem zur kontextsensitiven Unterstützung und prüft alle Ergebnisse durch ein mehrstufiges Validierungsverfahren.

Die Evaluation zeigte, dass **82 %** der erzeugten XML-Dateien formal korrekt und **81 %** semantisch konsistent waren. Das RAG-System erreichte eine **Faktenbasiertheit von 0,85** und eine **Antwortrelevanz von 0,83**. Diese Resultate bestätigen die technische Machbarkeit KI-gestützter Self-Service-Automatisierung.

Die Arbeit liefert damit praxisrelevante Erkenntnisse für Forschung und Anwendung: Sie belegt, dass menschenzentrierte Automatisierungssysteme Fachanwender entlasten, Qualität sichern und den Weg zu einer neuen Generation intelligenter Unternehmenssoftware eröffnen.

Inhaltsverzeichnis

Inhaltsverzeichnis	III
Abbildungsverzeichnis	VI
Tabellenverzeichnis	VIII
Listingverzeichnis	IX
Abkürzungsverzeichnis	X
1 Einleitung	1
1.1 Problemstellung und Motivation	1
1.2 Zielsetzung der Arbeit	2
1.3 Methodisches Vorgehen und Aufbau der Arbeit	3
2 Theoretische Grundlagen.....	6
2.1 Workload-Automatisierung als Herausforderung	6
2.2 Self-Service durch Sprachverarbeitung	14
2.3 Kontextualisierung durch Retrieval-Augmented Generation (RAG).....	21
3 Analyse der Stream-Erstellungsprozesse bei Streamworks	28
3.1 Aktueller Prozess	28
3.2 Automatisierungspotenziale	29
3.3 Systemanforderungen.....	30
3.4 Bedenken und Risiken	32
4 Konzeption und Design	34
4.1 Zielarchitektur.....	34
4.2 Systemkomponenten	36
4.3 Prozesse	37
4.4 Abgrenzung und Prototyp-Umfang (MVP)	40
5 Implementierung	42
5.1 Auswahl des Technologiestacks	42
5.2 Backend-Implementierung	43
5.3 Frontend-Implementierung	45
5.4 Integration der Kernfunktionen	47
5.5 Validierungssystem.....	52
6 Evaluation und Ergebnisse	54
6.1 Zielsetzung der Evaluation	54
6.2 Evaluierung der XML-Generierung	54

6.3	Evaluierung des RAG-Systems.....	57
6.4	Zusammenfassung der Evaluationsergebnisse	60
7	Kritische Reflexion.....	61
7.1	Technische Limitationen und Robustheit	61
7.2	Organisatorische Rahmenbedingungen. Fehler! Textmarke nicht definiert.	
7.3	Methodische und inhaltliche Begrenzungen Fehler! Textmarke nicht definiert.	
7.4	Zwischenfazit.....	62
8	Fazit und Ausblick.....	63
8.1	Fazit und Zielerreichung.....	63
8.2	Beitrag für Forschung und Praxis.....	64
8.3	Ausblick und Weiterentwicklungen.....	66
8.4	Schlussbemerkung.....	66
9	Anhang	68
9.1	Anhangsverzeichnis	68
9.2	Screenshots des entwickelten Self-Service-Prototyps	68
9.3	Listings	69
9.4	Tabellen.....	70
9.5	Dokumentation der Nutzung von KI-Assistenz-Tools.....	78
10	Quellenverzeichnis.....	79

Abbildungsverzeichnis

Abbildung 1 Beispiel einer Stream-Konfiguration im Stream Designer von Streamworks	8
Abbildung 2 Human-Centered-Automation im Unternehmenskontext	Fehler! Textmarke nicht definiert.
Abbildung 3 Vereinfachte Architektur eines RAG-Systems mit Indexierungs- und Abfragephase	23
Abbildung 4 Typische Fehlerpunkte in RAG-Systemen	26
Abbildung 5 Ist-Prozess der Stream-Erstellung bei Streamworks.....	28
Abbildung 6 Zielarchitektur des KI-gestützten Self-Service-Systems	35
Abbildung 7 Soll-Prozess der Stream-Erstellung mit Self-Service und KI.....	38
Abbildung 8 RAG-Indexierung.....	39
Abbildung 9 RAG-Retrievalprozess.....	39
Abbildung 10 Das TRIAD-Dreieck der RAG-Bewertung	58

Tabellenverzeichnis

Tabelle 1 Fehlerpunkte (FP1-FP7) und Gegenmaßnahmen im RAG-Prozess	26
Tabelle 2 Qualitätsdimension und Prüfmethoden der XML-Validierung.....	56
Tabelle 3 Bewertungsmetriken und Formeln.....	56
Tabelle 4 Bewertungsmetriken des TRIAD-Frameworks (Retriever-Ebene).....	58
Tabelle 5 Bewertungsmetriken des RAGAS-Frameworks (Generierungs-Ebene).....	59
Tabelle 6 Stream-Parameter	70
Tabelle 7 FILE_TRANSFER-Job-Parameter.....	71
Tabelle 8 SAP-Job-Parameter	71
Tabelle 9 Standard-Job-Parameter	71
Tabelle 10 Backend-Technologien und Begründung	72
Tabelle 11 Frontend-Technologien und Begründung	72
Tabelle 12 Testdatenset zur XML-Evaluation	73
Tabelle 13 Beispiel für eine Testfall-ID.....	73
Tabelle 14 Ergebnisse der XML-Validierung	74
Tabelle 15 Ergebnisse der Retrieval-Komponente (TRIAD-Metriken)	74
Tabelle 16 Ergebnisse der Generierungskomponente (RAGAS-Metriken).....	74
Tabelle 17 Evaluierung einer RAG-Anfrage	75

Listingsverzeichnis

Listing 1 Vereinfachter XML-Export eines Streams in Streamworks	10
Listing 2 Beispiel für wohlgeformtes und nicht wohlgeformtes XML	11
Listing 3 Veranschaulichung der LLM-Funktionsweise (stark vereinfacht)	Fehler!
Textmarke nicht definiert.	
Listing 4 One-Shot-Beispiel.....	Fehler! Textmarke nicht definiert.
Listing 5 One-Shot-Beispiel.....	17
Listing 6 Beispiel für eine intrinsische Halluzination in einer XML-Konfiguration ...	20
Listing 7 Beispiel für eine extrinsische Halluzination in einer XML-Konfiguration ..	20
Listing 8 Backend-Projektstruktur.....	45
Listing 9 Frontend-Projektstruktur	46
Listing 10 Beispielhafte Parameterextraktion durch LangExtract.....	49
Listing 11 Schema-Validierung der XML-Dateien mit der Bibliothek Ixml	55
Listing 12 Beispiel – LangExtract Extraktion	69
Listing 13 Beispielhafte Berechnung der RAGAS-Metriken	70

Abkürzungsverzeichnis

API	Application Programming Interface
AI	Artificial Intelligence (Künstliche Intelligenz)
ASGI	Asynchronous Server Gateway Interface
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DSR	Design Science Research
ETL	Extract, Transform, Load
FP	Failure Point
GUI	Graphical User Interface
HCA	Human-Centered Automation
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
KI	Künstliche Intelligenz
LLM	Large Language Model
LoRA	Low-Rank Adaptation
NLP	Natural Language Processing
ORM	Object-Relational Mapping
PEFT	Parameter-Efficient Fine-Tuning
RAG	Retrieval-Augmented Generation
REST	Representational State Transfer
RPA	Robotic Process Automation
RRF	Reciprocal Rank Fusion
SAP	Systeme, Anwendungen und Produkte in der Datenverarbeitung
SST	Self-Service Technology
SSR	Server-Side Rendering
UI	User Interface

UX	User Experience
VM	Virtuelle Maschine
WLA	Workload-Automatisierung
XML	Extensible Markup Language
XSD	XML Schema Definition

1 Einleitung

Die Automatisierung von IT-Prozessen ist ein zentraler Bestandteil moderner Unternehmensarchitekturen und trägt wesentlich zur Effizienzsteigerung bei. Workload-Automatisierung (WLA) ermöglicht die zentrale Steuerung und Überwachung komplexer Abläufe über heterogene Systemlandschaften hinweg. Systeme wie *Streamworks* von Arvato Systems übernehmen hierbei eine strategische Rolle in der Orchestrierung geschäfts-kritischer Prozesse.

Parallel dazu eröffnen aktuelle Fortschritte im Bereich der *Künstlichen Intelligenz (KI)* – insbesondere durch *Large Language Models (LLMs)* – neue Möglichkeiten, den Zugang zu solchen Automatisierungssystemen zu vereinfachen.

In Verbindung mit Self-Service-Ansätzen können sie Fachanwender dabei unterstützen, natürlichsprachliche Anforderungen in strukturierte Konfigurationen zu überführen und auf dokumentiertes Wissen zurückzugreifen.

Vor diesem Hintergrund untersucht die vorliegende Arbeit, wie sich KI-Technologien im Kontext der Workload-Automatisierung einsetzen lassen und welche Potenziale sich durch ihre Integration in Streamworks ergeben. Die Machbarkeit wird anhand einer prototypischen Implementierung nachgewiesen.

1.1 Problemstellung und Motivation

Trotz ihrer strategischen Bedeutung sind Systeme der Workload-Automatisierung wie **Streamworks** für Fachanwender häufig nur schwer zugänglich. Die Erstellung und Pflege von **Streams**¹ erfordern detailliertes Expertenwissen über Parameter, Abhängigkeiten und strukturelle Richtlinien. Dadurch entsteht ein erheblicher Kommunikations- und Abstimmungsaufwand, verbunden mit einer Abhängigkeit von technischem Fachwissen und entsprechend längeren Durchlaufzeiten – selbst bei Routineanforderungen wie Dateiübertragungen oder SAP-Jobs.

An dieser Stelle setzen moderne Entwicklungen im Bereich der *Künstlichen Intelligenz* an: *Large Language Models* können natürlichsprachliche Anforderungen interpretieren und in strukturierte Konfigurationen überführen. Ergänzend ermöglicht *Retrieval-Augmented Generation* den gezielten Zugriff auf interne Dokumentationen und Wissensquel-len.

¹ Der Begriff „*Stream*“ bezeichnet in Streamworks eine Automatisierungskette

1.2 Zielsetzung der Arbeit

Ziel dieser Arbeit ist es, zu untersuchen, wie sich Self-Service-Prinzipien und moderne KI-Technologien im Bereich der Workload-Automatisierung sinnvoll kombinieren lassen, um bestehende Herausforderungen zu überwinden. Am Beispiel der Plattform Streamworks wird der Frage nachgegangen, inwieweit Large Language Models (LLMs) und Retrieval-Augmented Generation (RAG) dazu beitragen können, den Prozess der Stream-Erstellung effizienter, benutzerfreundlicher und weniger fehleranfällig zu gestalten.

Da die Arbeit im Rahmen einer Bachelorthesis durchgeführt wird, liegt der Fokus nicht auf einer produktiven Implementierung, sondern auf der Konzeption und prototypischen Realisierung eines funktionsfähigen Machbarkeitsnachweises (Proof of Concept) auf MVP-Niveau².

Im Mittelpunkt stehen zwei zentrale Forschungs- und Entwicklungsziele:

1. Automatisierte Generierung valider XML-Konfigurationen:

Natürlichsprachliche Anforderungen sollen mithilfe eines LLMs in formal korrekte, XSD-validierte XML-Strukturen überführt werden, um den manuellen Aufwand bei der Stream-Erstellung deutlich zu reduzieren.

2. Wissensbasierte Unterstützung durch RAG-Integration:

Über ein RAG-basiertes Hilfesystem soll internes Wissen aus Dokumentationen kontextsensitiv bereitgestellt werden, sodass Fachanwender gezielte Hilfestellung erhalten und Rückfragen minimiert werden.

Darüber hinaus verfolgt die Arbeit übergreifende Teilziele, die beide Ansätze miteinander verbinden:

- **Autonomie und Benutzerfreundlichkeit:** Fachanwender sollen in die Lage versetzt werden, Standardprozesse eigenständig und ohne Expertenunterstützung umzusetzen.
- **Qualitätssicherung:** Die erzeugten Konfigurationen werden durch technische Prüfmechanismen (z. B. XSD-Validierung, semantische Konsistenzchecks) automatisch überprüft.
- **Praxisbezug:** Der entwickelte Prototyp dient als Evaluationsgrundlage, um die Effizienz- und Qualitätsgewinne empirisch zu bewerten.

²Die Begriffe „Prototyp“ und „MVP (Minimum Viable Product)“ werden in dieser Arbeit synonym verwendet.

Damit leistet die Arbeit einen Beitrag zur **Machbarkeitsanalyse KI-gestützter Self-Service-Ansätze** im Bereich der Workload-Automatisierung und bewertet deren Nutzen und Relevanz für Arvato Systems kritisch.

1.3 Methodisches Vorgehen und Aufbau der Arbeit

Das Vorgehen dieser Arbeit ist darauf ausgerichtet, den wissenschaftlichen Anspruch – also die Einbettung in theoretische Grundlagen – mit einem hohen Praxisbezug durch Analyse und prototypische Umsetzung zu vereinen. Methodisch orientiert sich die Arbeit am Paradigma der **Design Science Research (DSR)**, wie es insbesondere von Hevner et al. (2004) und Peffers et al. (2007) beschrieben wird³.

DSR ist ein problemorientierter Forschungsansatz, der darauf abzielt, durch die Entwicklung und Evaluation innovativer Artefakte sowohl praktische Probleme zu lösen als auch wissenschaftliche Erkenntnisse zu generieren.

Der Forschungsprozess orientiert sich an den etablierten Phasen des Design-Science-Research-Prozessmodells⁴:

1. **Problemidentifikation und Motivation:** Analyse der manuellen und aufwändigen Stream-Erstellung in Streamworks als zentralem betrieblichen Engpass (vgl. Kapitel 3).
2. **Zieldefinition und Ableitung der Lösungsanforderungen:** Ableitung der Anforderungen an ein Self-Service- und KI-gestütztes System zur Reduktion von Komplexität und Kommunikationsaufwand (vgl. Kapitel 1.2), konkretisiert durch die Analyse bestehender Prozesse und Automatisierungspotenziale (vgl. Kapitel 3) sowie die Ausarbeitung einer Zielarchitektur und Systemkomponenten (vgl. Kapitel 4).
3. **Konzeption und Realisierung des Artefakts:** Konzeption einer Zielarchitektur und prototypische Implementierung eines Systems zur XML-Generierung und Wissensintegration (vgl. Kapitel 4 und 5).
4. **Demonstration:** Anwendung des Prototyps in exemplarischen Anwendungsfällen zur Überprüfung der Funktionsfähigkeit und Praxistauglichkeit des Artefakts (vgl. Kapitel 5)

³ Vgl. vom Brocke, J./Hevner, A./Maedche, A. (2020): *Introduction to Design Science Research* S.1f

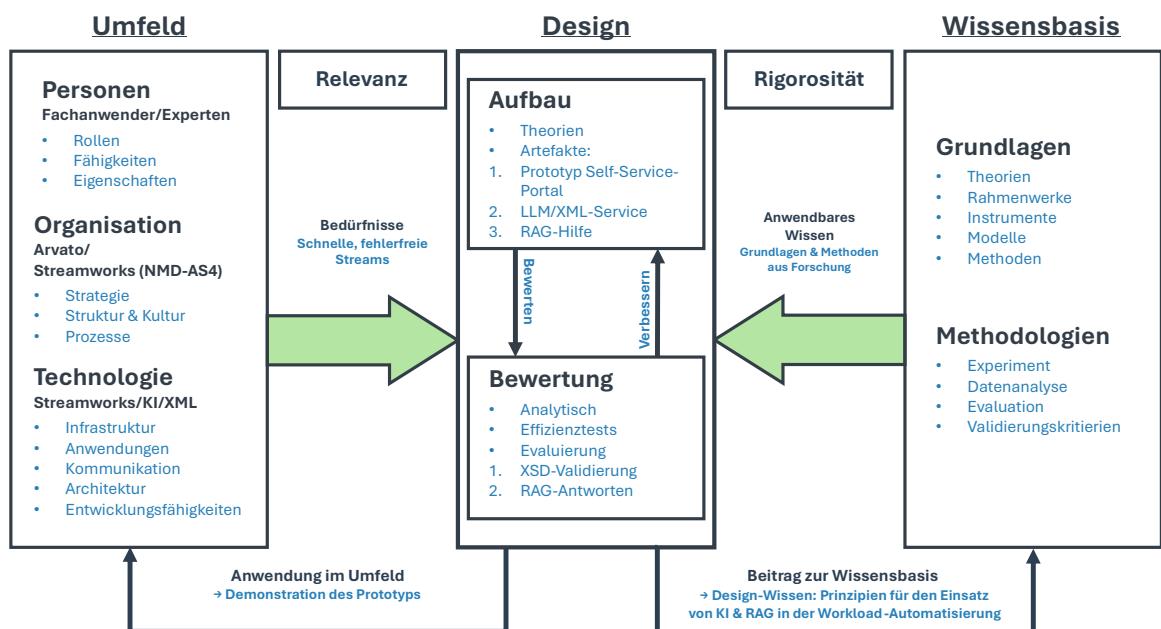
⁴ Vgl. vom Brocke, J./Hevner, A./Maedche, A. (2020), S. 3-4

5. **Evaluation:** Bewertung des entwickelten Prototyps anhand definierter Metriken (Validität, Effizienzsteigerung, Nutzerunterstützung) und Analyse der erzielten Ergebnisse (vgl. Kapitel 6).
 6. **Kommunikation:** Darstellung der Ergebnisse sowie Ableitung wissenschaftlicher und praktischer Implikationen im Rahmen der Reflexion, des Fazits und des Ausblicks (vgl. Kapitel 7 und 8).

Zur zusätzlichen Einordnung wird das methodische Vorgehen im *Design Science Research Framework* nach Hevner et al. verortet⁵.

Ein Stream bildet die logische Einheit eines automatisierten Ablaufs und fasst mehrere Komponenten zu einer ausführbaren Prozesskette zusammen. Neben den übergeordneten Stream-Eigenschaften umfasst er mindestens den StartPoint, einen oder mehrere Jobs sowie einen EndPoint. Technisch betrachtet sind *StartPoint* und *EndPoint* spezielle Ausprägungen des Job-Typs, die den Beginn bzw. das Ende der Ausführung markieren. Jede dieser Komponenten verfügt über eigene Parameter und Konfigurationseigenschaften.

Abbildung 1 zeigt die Übertragung dieses Rahmens auf den Kontext der vorliegenden Arbeit



Quelle: Eigene Darstellung in Anlehnung an Hevner et al. 2004

Arbeit: Während das **Umfeld** (Fachanwender, Organisation, Technologie bei Arvato / Streamworks) die praktischen Anforderungen und Bedürfnisse definiert, liefert die

⁵ Vgl. vom Brocke, J./Heyner, A./Maedche, A. (2020), S. 6–7

Wissensbasis (Theorien, Modelle und Methoden zu WLA, Self-Service, LLM und RAG) die wissenschaftliche Grundlage. In der **Design-Domäne** werden die Artefakte – Prototyp, LLM/XML-Service und RAG-Hilfesystem – entwickelt und evaluiert. Die Arbeit bewegt sich damit im Spannungsfeld von **Relevanz** (praktische Problemstellung), **Rigorosität** (theoretische Fundierung) und **Design** (Gestaltung und Evaluation der Lösung)

2 Theoretische Grundlagen

In diesem Kapitel werden die theoretischen und technologischen Grundlagen dargestellt, die für das Verständnis der Arbeit und die Einordnung des entwickelten Prototyps erforderlich sind. Im Sinne des *Design-Science-Research*-Ansatzes leistet dieses Kapitel den Beitrag zur **Rigorosität** der Forschung: Die systematische Aufarbeitung zentraler Konzepte – *Workload-Automatisierung*, *Self-Service*, *Large Language Models (LLMs)* und *Retrieval-Augmented Generation (RAG)* – bildet die wissenschaftliche Basis für die nachfolgende Konzeption und Implementierung des Artefakts.

2.1 Workload-Automatisierung als Herausforderung

Die Automatisierung von IT-Prozessen ist eine zentrale Komponente moderner Unternehmensarchitekturen und trägt wesentlich zur Steigerung von Effizienz und Zuverlässigkeit bei⁶. Im Fokus stehen Enterprise-WLA-Plattformen wie Streamworks, die als Schlüsseltechnologie für die übergreifende Steuerung und Koordination komplexer IT-Abläufe gilt.⁷

Trotz ihres hohen Potenzials ist die Nutzung solcher Systeme mit erheblichen Herausforderungen verbunden: Die Konfiguration von Automatisierungsketten ist aufgrund der Vielzahl an Parametern, Abhängigkeiten und technischen Rahmenbedingungen äußerst komplex. Diese Komplexität führt zu einer starken Abhängigkeit von Expertenwissen und mindert die Agilität der Fachbereiche.

Eine Reduktion dieser Hürden durch benutzerfreundliche Self-Service-Konzepte und KI-gestützte Assistenzsysteme ist daher ein entscheidender Schritt, um die Potenziale der Workload-Automatisierung voll auszuschöpfen.

2.1.1 Grundlagen der Workload-Automatisierung

Workload-Automatisierung (WLA) ermöglicht die zentrale Planung, Steuerung und Überwachung IT-gestützter Geschäftsprozesse, die sich über verschiedene Anwendungen und Plattformen erstrecken, und trägt wesentlich zur Steigerung der Prozesseffizienz bei⁸. Im Gegensatz zu nativen, anwendungsspezifischen Schedulern, die isoliert agieren, fungieren WLA-Systeme als übergreifende ‚Master Scheduler‘: Sie erlauben die Definition und Verwaltung von Abhängigkeiten zwischen Prozessen auf unterschiedlichen

⁶ Vgl. Twing, D. (2025). The Future of Workload Automation and Orchestration, S. 4

⁷ Vgl. Sabharwal, N. & Kasiviswanathan, S. (2023). Introduction to Workload Automation, S. 1

⁸ Vgl. Beta Systems. (2024). Was ist Workload Automation? - Beta Systems, S. 1f

Systemen und gewährleisten so eine ganzheitliche Sicht auf sämtliche Abläufe. Da moderne Geschäftsprozesse häufig aus verketteten Transaktionen über verschiedene Systeme bestehen, umfasst die Kernfunktionalität von WLA-Lösungen sowohl die zeit- als auch die ereignisgesteuerte Ausführung von Workflows, um flexibel auf betriebliche Anforderungen reagieren zu können⁹. Ein weiterer zentraler Aspekt ist die Einhaltung von Compliance- und Sicherheitsvorgaben, die durch detaillierte Protokollierungen und Zugriffskontrollen sichergestellt wird. Durch die Automatisierung wiederkehrender Aufgaben werden manuelle Eingriffe minimiert, wodurch das Fehlerrisiko sinkt und personelle Ressourcen für strategischere Tätigkeiten verfügbar werden¹⁰.

Mit der fortschreitenden digitalen Transformation steigen jedoch die Anforderungen an die Automatisierung, was eine zunehmende Komplexität in der Orchestrierung nach sich zieht¹¹. Die Vielzahl der zu steuernden Prozesse und die Integration heterogener Technologien erfordern tiefgehendes Expertenwissen – selbst bei der Umsetzung standardisierter Anforderungen¹². Diese wachsende Komplexität resultiert in einem Mangel an qualifiziertem Personal und etabliert zeitintensive Abstimmungsprozesse als betriebliche Praxis¹³. Besonders die manuelle Übersetzung fachlicher Anforderungen in technische Konfigurationen bildet dabei einen Engpass. Daraus entsteht eine deutliche Diskrepanz zwischen den Potenzialen von WLA-Systemen und ihrer tatsächlichen Zugänglichkeit für Fachanwender.

2.1.2 Streamworks als Enterprise WLA-Plattform

Streamworks von Arvato Systems ist eine Enterprise-Plattform für Workload-Automation und Service-Orchestration, die auf Standard-Windows-Komponenten im Backend basiert. Das Kernprinzip besteht in der zentralen Definition von Workflows, bei denen einzelne Prozessschritte – sogenannte *Jobs* – zu einer durchgängigen Prozesskette, dem *Stream*, verbunden werden¹⁴. Die Ausführung dieser Streams erfolgt plattformübergreifend auf Zielsystemen wie Windows, Unix oder Mainframes durch schlanke Agenten, die von einem zentralen Processing Server gesteuert werden¹⁵. Die Kommunikation

⁹ Vgl. Sabharwal & Kasiviswanathan (2023), S. 3

¹⁰ Vgl. Beta Systems (2024), S. 2

¹¹ Vgl. Twing (2025), S. 17

¹² Vgl. Ebd. (2025), S. 8f

¹³ Vgl. Twing (2025), S. 15

¹⁴ Vgl. Arvato Systems (o. D.), S. 2

¹⁵ Vgl. Arvato Systems (o. D.), S. 6f

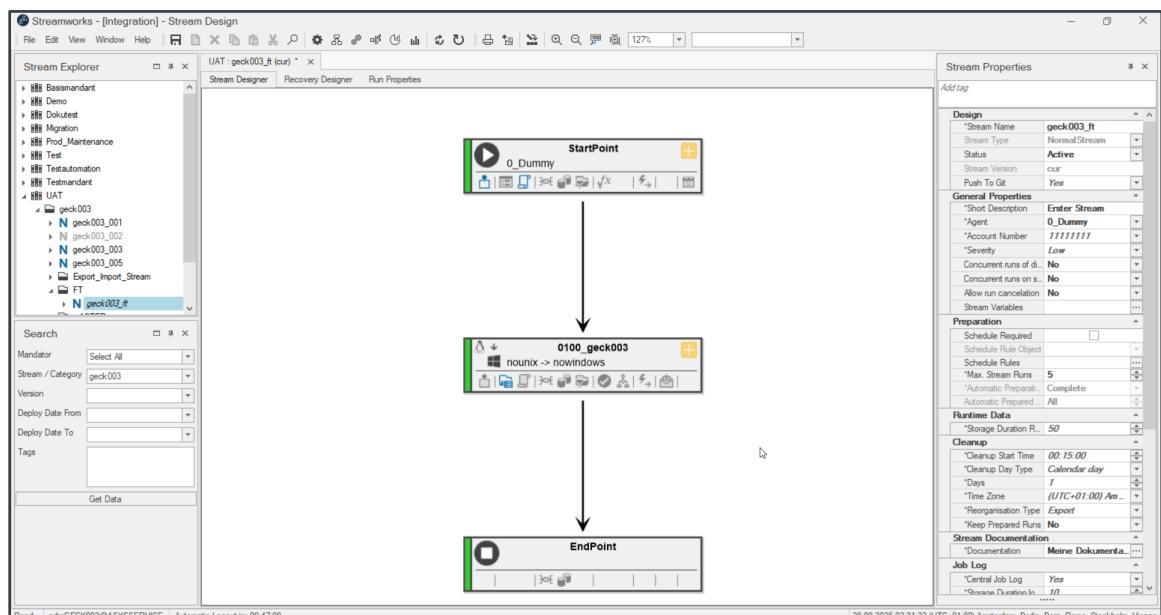
zwischen allen Komponenten ist verschlüsselt und erfüllt damit hohe Sicherheitsanforderungen¹⁶.

2.1.3 Konfiguration von Streams

Die Konfiguration von Streams und die Definition ihrer komplexen Abhängigkeiten erfolgen über die grafische Benutzeroberfläche im *Stream Designer* von Streamworks (vgl. Abbildung 2). Diese unterstützt den Erstellungsprozess visuell, erfordert jedoch aufgrund ihrer technischen Komplexität spezialisiertes Fachwissen.

Ein Stream bildet die logische Einheit eines automatisierten Ablaufs und fasst mehrere Komponenten zu einer ausführbaren Prozesskette zusammen. Neben den übergeordneten Stream-Eigenschaften umfasst er mindestens den StartPoint, einen oder mehrere Jobs sowie einen EndPoint. Technisch betrachtet sind *StartPoint* und *EndPoint* spezielle Ausprägungen des Job-Typs, die den Beginn bzw. das Ende der Ausführung markieren. Jede dieser Komponenten verfügt über eigene Parameter und Konfigurationseigenschaften.

Abbildung 1 Beispiel einer Stream-Konfiguration im Stream Designer von Streamworks



Quelle: Screenshot aus Streamworks (*Stream Design*)

Die **Stream-Eigenschaften** allein umfassen über 40 individuelle Konfigurationsparameter, hinzu kommen rund 25 Parameter pro Job sowie zusätzliche Einstellungen für Start- und Endpunkte.

Parameter lassen sich in drei Kategorien gliedern:

¹⁶ Vgl. Arvato Systems (o. D.), S. 3

1. **Pflichtparameter:** Einstellungen, die zwingend erforderlich sind, um Funktionalität und Lauffähigkeit sicherzustellen
2. **Automatisch gesetzte Parameter:** Werte, die vom System übernommen oder aus übergeordneten Einstellungen vererbt werden.
3. **Optionale Parameter:** Erweiterte Konfigurationsmöglichkeiten für eine granulare Steuerung, die für den Basisbetrieb jedoch nicht notwendig sind.

Neben einer Vielzahl optionaler Einstellungen erfordert die Konfiguration weiterhin zahlreiche manuelle Eingaben – darunter die eindeutige Benennung von Stream und Jobs, die Zuordnung des ausführenden Agenten sowie die ergänzende Dokumentation.

Ein zentrales Feature von Streamworks ist die Möglichkeit, fertig konfigurierte Streams über eine Export/Import-Funktion in ein standardisiertes XML-Format zu überführen¹⁷. Diese Funktion dient insbesondere dem Austausch zwischen verschiedenen Umgebungen, etwa beim Transfer von einer Test- in eine Produktionsumgebung. Für die Zielsetzung dieser Arbeit ist die XML-Struktur von besonderer Bedeutung, da sie die vollständige, maschinenlesbare Repräsentation eines Streams darstellt und somit den idealen Anknüpfungspunkt für eine automatisierte Generierung der Konfiguration bietet.

2.1.4 Grundlagen der XML-basierten Konfiguration

Die Extensible Markup Language (XML) ist eine universelle Metasprache zur Darstellung strukturierter Daten in einem plattform- und anwendungsunabhängigen Textformat. Sie wurde als Teilmenge der SGML (Standard Generalized Markup Language) mit dem Ziel entwickelt, sowohl für Menschen als auch für Maschinen leicht lesbar und einfach verarbeitbar zu sein¹⁸. Die grundlegende Struktur eines XML-Dokuments folgt einem Baumprinzip: Elemente sind ineinander verschachtelt, durch Start- und End-Tags begrenzt und können weitere Elemente, Attribute oder Zeichendaten enthalten¹⁹.

Das folgende Listing veranschaulicht anhand eines stark vereinfachten XML-Exports eines Streams aus Streamworks die charakteristische hierarchische Struktur, die selbst in dieser reduzierten Form deutlich erkennbar ist.

- Auf oberster Ebene befindet sich das Element <ExportableStream>, das den gesamten Exportrahmen des Streams definiert und als Wurzelement des Dokuments fungiert.

¹⁷ Vgl. Arvato Systems. (o. D.), S. 9

¹⁸ Vgl. Bray, T. et al. (2008). Extensible Markup Language (XML) 1.0 (Fifth Edition), S. 3

¹⁹ Vgl. Vonhoegen, H. (2002). Einstieg in XML, S. 52

- Innerhalb davon steht das zentrale `<Stream>`-Element, das Metadaten wie den Namen (`<StreamName>`), die Dokumentation (`<StreamDocumentation>`), den zugeordneten Agenten (`<AgentDetail>`), den Typ sowie eine Kurzbeschreibung enthält.
- Der anschließende `<Jobs>`-Block beschreibt die einzelnen Prozessschritte des Streams. Im gezeigten Beispiel ist dies das Startobjekt (`<JobName>Start-Point</JobName>`), das über seine Kategorie (`<JobCategory>Start-Point</JobCategory>`) eindeutig als Einstiegspunkt markiert ist.
- Über die verschachtelten Nachfolgeelemente (`<JobInternalSuccessors>`) wird die Abhängigkeit zum nächsten Job (`0100_geck003`) hergestellt. Dieser Job ist im Listing nicht weiter ausgeführt, da es sich um eine bewusst vereinfachte Darstellung handelt.

Listing 1 Beispielhafter XML-Export eines Streams aus Streamworks (vereinfacht)

```

<?xml version="1.0" encoding="utf-8"?>
<ExportableStream>
  <Stream>
    <StreamName>geck003_stream</StreamName>
    <StreamDocumentation><! [CDATA[Dokumentation]]></StreamDocumentation>
    <AgentDetail>TestAgent01</AgentDetail>
    <StreamType>Normal</StreamType>
    <ShortDescription><! [CDATA[Erster Stream]]></ShortDescription>
  </Stream>
  <Jobs>
    <Job>
      <JobName>StartPoint</JobName>
      <JobCategory>StartPoint</JobCategory>
      <ShortDescription><! [CDATA[Start Point for the Stream]]></ShortDescription>
      <StatusFlag>True</StatusFlag>
      <JobInternalSuccessors>
        <JobInternalSuccessor>
          <JobInternalName>0100_geck003</JobInternalName>
        </JobInternalSuccessor>
      </JobInternalSuccessors>
    </Job>
  </Jobs>
</ExportableStream>

```

Quelle: eigene Darstellung, basierend auf dem XML-Exportformat von Streamworks (Arvato Systems)

Neben den Pflichtparametern enthält ein Stream zahlreiche optionale Eigenschaften, beispielsweise für Logging, Recovery, File-Transfer-Einstellungen oder Kalenderregeln. In produktiven Umgebungen bestehen vollständige Stream-XML-Dateien meist aus mehreren hundert, teilweise über tausend Zeilen, was die hohe Detailtiefe der Konfiguration verdeutlicht.

Während Streams in der Praxis nahezu ausschließlich über die grafische Benutzeroberfläche konfiguriert werden, fungiert XML im Hintergrund als persistente und deutlich komplexere Repräsentation. Für Anwender stellt diese Detailtiefe häufig eine Barriere dar – für KI-Modelle hingegen bietet sie eine ideale Schnittstelle: XML ist strikt formalisiert, maschinenlesbar und ermöglicht eine präzise Abbildung der Automatisierungslogik. Dadurch eignet sich das Format hervorragend, um natürlichsprachliche Eingaben durch ein Large Language Model (LLM) in valide, strukturell konsistente Konfigurationen zu überführen.

Ein zentrales Prinzip von XML ist die *Wohlgeformtheit*: Alle Elemente müssen korrekt geschachtelt sein, andernfalls verweigert der XML-Prozessor die Verarbeitung. Diese strikte Syntaxprüfung gewährleistet Interoperabilität, da sie einen verlässlichen Austausch strukturierter Daten sicherstellt und durch selbstbeschreibende Tags eine klare Trennung von Inhalt und Form ermöglicht²⁰.

Listing 2 Beispiel für wohlgeformtes und nicht wohlgeformtes XML

```
<!-- wohlgeformt -->
<Stream>
  <JobName>StartPoint</JobName>
</Stream>

<!-- nicht wohlgeformt -->
<Stream>
  <JobName>StartPoint</Stream>
```

Quelle: eigene Darstellung in Anlehnung an W3C (Bray et al, 2008)

2.1.5 Qualitätssicherung durch XML Schema Definition (XSD)

Während die Wohlgeformtheit sicherstellt, dass ein XML-Dokument syntaktisch korrekt aufgebaut ist, reicht diese Eigenschaft in komplexen Systemen allein nicht aus. Für die Workload-Automatisierung in Streamworks ist darüber hinaus entscheidend, dass auch die inhaltliche Struktur den fachlichen Vorgaben entspricht – an dieser Stelle greift die Validierung über die XML Schema Definition (XSD)²¹.

Eine XSD fungiert als Bauplan: Es legt das Vokabular, die Struktur sowie die erlaubte Hierarchie und Häufigkeit von Elementen fest und unterstützt dadurch die

²⁰ Vgl. Vonhoegen, H. (2002), S. 29

²¹ Vgl. Vonhoegen, H. (2002), S. 70

Qualitätssicherung in datenintensiven Systemen²². Ein Dokument gilt als valide, wenn es nicht nur wohlgeformt ist, sondern auch vollständig den Regeln seines zugeordneten XSD-Schemas entspricht²³.

Eine besondere Stärke der XML Schema Definition (XSD) liegt in der Möglichkeit, Datentypen präzise zu definieren. Dadurch lassen sich die Inhalte von Elementen und Attributen auf vordefinierte oder benutzerdefinierte Typen beschränken. Im Kontext der Workload-Automatisierung mit Streamworks stellt ein XSD-Schema sicher, dass importierte Konfigurationen nicht nur syntaktisch korrekt, sondern auch semantisch vollständig und konsistent sind²⁴.

Das folgende XSD-Fragment-Listing (vgl. Listing 3) veranschaulicht, wie sich mit der XML Schema Definition (XSD) nicht nur die Wohlgeformtheit, sondern auch die inhaltliche Struktur und Semantik eines XML-Dokuments absichern lässt:

- Das Root-Element `<ExportableStream>` wird als eigener komplexer Typ definiert und bildet den Einstiegspunkt der Stream-Struktur.
- Mithilfe von `<xs:complexType>` und `<xs:sequence>` wird die Reihenfolge und Hierarchie der enthaltenen Elemente festgelegt. So ist beispielsweise definiert, dass ein Stream zwingend ein `<Stream>`-Element enthalten muss.
- Untergeordnete Elemente sind mit Datentypen versehen, etwa `xs:string` für Textfelder oder `xs:int` für numerische Angaben. Dadurch werden Eingaben auf zulässige Wertebereiche beschränkt.
- Das Attribut `minOccurs="0"` beim Element `<AccountNoId>` zeigt, wie optionale Felder modelliert werden können. Solche Parameter dürfen im Dokument fehlen, ohne die Validität des Dokuments zu beeinträchtigen.

²² Vgl. Gao, S. et al. (2012). W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures, S. 15

²³ Vgl. Vonhoeven, H. (2002), S. 71

²⁴ Vgl. Gao, S. et al. (2012), S. 4

Listing 3 Ausschnitt eines XSD-Schemas zur Definition der Stream-Struktur in Streamworks

```

?xml version="1.0" encoding="utf-8"?
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">

    <!-- Root Element -->
    <xs:element name="ExportableStream" type="ExportableStreamType"/>

    <!-- Definition des Stream-Containers -->
    <xs:complexType name="ExportableStreamType">
        <xs:sequence>
            <xs:element name="Stream" type="StreamType"/>
        </xs:sequence>
    </xs:complexType>

    <!-- Stream-Struktur -->
    <xs:complexType name="StreamType">
        <xs:sequence>
            <xs:element name="StreamName" type="xs:string"/>
            <xs:element name="StreamDocumentation" type="xs:CDataString"/>
            <xs:element name="AgentDetail" type="xs:string"/>
            <xs:element name="AccountNoId" type="xs:int" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Quelle: eigene Darstellung, basierend auf dem Streamworks-XSD-Schema (Arvato Systems)

Durch diese formale Beschreibung legt XSD einen verbindlichen Bauplan fest, der sicherstellt, dass alle importierten Konfigurationen nicht nur syntaktisch korrekt (wohlgeformt), sondern auch fachlich konsistent und valide sind.

In der vollständigen Stream-XSD sind darüber hinaus zahlreiche weitere Elemente definiert – etwa für Kalender, Benachrichtigungen oder Job-Logs. Ähnlich wie bei den XML-Konfigurationen umfasst auch eine produktive XSD-Datei mehrere hundert Zeilen, da sie die gesamte Struktur und Semantik der Stream-Definition formal abbildet. Diese formale Beschreibung der erwarteten Datenstruktur erhöht die Interoperabilität und stärkt die Robustheit des Automatisierungssystems, indem Konfigurationsfehler bereits vor der produktiven Ausführung erkannt werden²⁵.

Die Erzeugung XSD-valider Konfigurationen ist daher eine Kernanforderung für jedes System, das den Konfigurationsprozess der Streams automatisieren oder vereinfachen soll. Ein solches System muss die fachliche Anforderung korrekt interpretieren und in eine technisch einwandfreie XML-Struktur überführen, die den Qualitätsprüfungen des Schemas standhält.

²⁵ Vgl. Gao, S. et al. (2012), S. 124

2.2 Self-Service durch Sprachverarbeitung

Die Komplexität der XML-basierten Konfiguration führt dazu, dass Fachanwender ohne spezielles Know-how meist nicht direkt mit Streamworks interagieren können. Um diesen Engpass zu überwinden, untersucht die vorliegende Arbeit, wie sich Self-Service-Prinzipien mit moderner Sprachverarbeitung durch Large Language Models (LLMs) verbinden lassen. Dadurch entsteht eine Schnittstelle, die natürliche Sprache in formal korrekte, maschinenlesbare Konfigurationen übersetzt. So können Fachanwender Standardprozesse eigenständig initiieren und anpassen, ohne auf die manuelle Unterstützung technischer Experten angewiesen zu sein. Dieser Ansatz vereinfacht den bislang kommunikations- und abstimmungsintensiven Prozess der Stream-Erstellung grundlegend und eröffnet neue Möglichkeiten für eine effizientere, KI-gestützte Workload-Automatisierung.

2.2.1 Das Self-Service-Prinzip in der IT

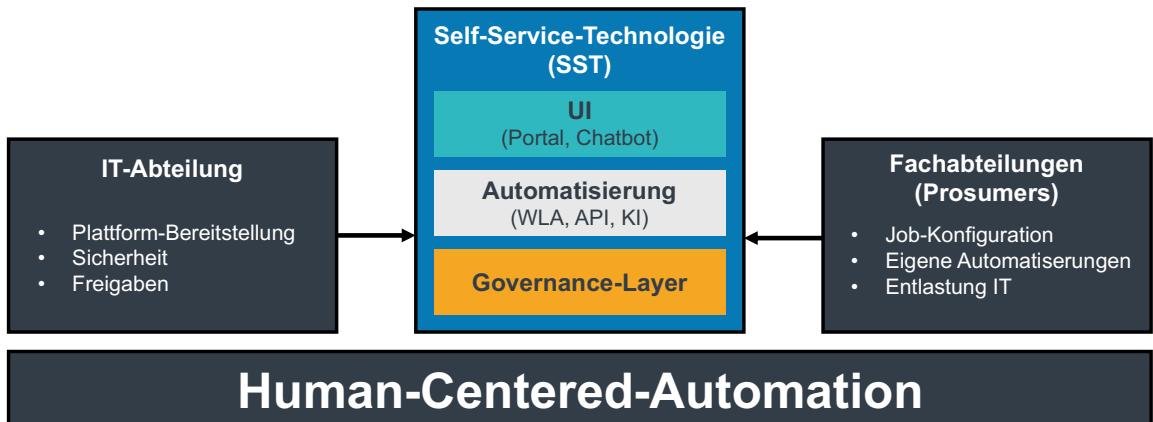
Das Self-Service-Prinzip, realisiert durch sogenannte Self-Service-Technologien (SSTs), ersetzt zunehmend die direkte Interaktion zwischen Dienstleistern und Kunden durch automatisierte Systeme. Im Unternehmenskontext werden Mitarbeitende dadurch zu „Prosumenten“ – also zugleich Produzenten und Konsumenten von Dienstleistungen, die Serviceprozesse eigenständig steuern, um individuelle Bedürfnisse zu erfüllen. Ziel solcher Technologien ist es nicht nur, die betriebliche Effizienz durch geringere Arbeitskosten und eine standardisierte Servicebereitstellung zu erhöhen, sondern auch die Nutzererfahrung zu verbessern – beispielsweise durch mehr Bequemlichkeit, Zeitersparnis und Autonomie²⁶.

Die Einführung solcher Systeme wird jedoch häufig durch eine geringe Benutzerfreundlichkeit gebremst. Komplexe Schnittstellen, mangelnde Intuitivität und die Notwendigkeit von Domänenwissen stellen erhebliche Hürden für die Akzeptanz dar – insbesondere für nicht-technische Anwender. Um diese Herausforderungen zu überwinden, ist ein menschenzentrierter Automatisierungsansatz (Human-Centred Automation, HCA) erforderlich, der die Bedürfnisse und Fähigkeiten der Nutzer konsequent in den Mittelpunkt stellt. Automatisierungslösungen dürfen nicht verlangen, dass sich Mitarbeiter an starre

²⁶ Vgl. Considine, E. & Cormican, K. (2017). The rise of the prosumer- an analysis of self-service technology adoption in a corporate context, S. 26-28

Systeme anpassen, sondern müssen so gestaltet sein, dass sie sich an bestehende Arbeitsabläufe und Kompetenzen anlehnen²⁷.

Abbildung 2 Human-Centered-Automation im Unternehmenskontext



Quelle: eigene Abbildung, in Anlehnung an Toxtli, C. (2024).

Abbildung 2 veranschaulicht das Self-Service-Ökosystem im Unternehmenskontext. Während die IT-Abteilung für die Plattformbereitstellung, Sicherheitsaspekte und Freigaben verantwortlich bleibt, übernehmen Fachabteilungen als Prosumennten zunehmend eigenständig Konfigurations- und Automatisierungsaufgaben. Im Zentrum steht eine Self-Service-Technologie, die aus drei Schichten besteht: einer benutzerfreundlichen Oberfläche (UI), einer Automatisierungsschicht (Workload-Automatisierung, APIs, KI) sowie einem Governance-Layer, das Kontrolle und Regelkonformität sicherstellt. Der Erfolg dieses Ansatzes hängt entscheidend davon ab, dass die Systeme den Prinzipien einer Human-Centered Automation folgen und damit die technische Komplexität so abstrahieren, dass sie für Fachanwender beherrschbar und nutzbar 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 gelten als eine bedeutende Innovation innerhalb der künstlichen Intelligenz²⁸. Ihre grundlegende Fähigkeit besteht darin, sprachliches und faktisches Wissen aus umfangreichen Textmengen im Rahmen des Pre-Trainings zu extrahieren und so ein universelles Modell zu schaffen, das vielseitig einsetzbar ist²⁹. Diese Entwicklung markiert einen Paradigmenwechsel: An die Stelle spezialisierter, aufgabenspezifisch trainierter Systeme traten Basismodelle (Foundation Models), die durch großflächiges, selbstüberwachtes Lernen auf

²⁷ Vgl. Toxtli, C. (2024). Human-Centered Automation, S. 1-4

²⁸ Vgl. Xiao, T. & Zhu, J. (2025). Foundations of Large Language Models, S. 36

²⁹ Vgl. Xiao, T. & Zhu, J. (2025), S. iii

unstrukturierten Daten trainiert werden und anschließend für spezifische Anwendungen durch Methoden wie Prompting, Fine-Tuning oder Retrieval-Augmented Generation (RAG) angepasst werden können³⁰.

Die Grundlage moderner Large Language Models (LLMs) bildet die **Transformer-Architektur**, die durch sogenannte *Self-Attention-Mechanismen* die parallele Verarbeitung langer Textsequenzen ermöglicht. Dadurch können Modelle Abhängigkeiten zwischen Wörtern unabhängig von ihrer Position im Satz erfassen, was die Basis für ihre hohe Leistungsfähigkeit und Kontextsensitivität bildet³¹. Während des **Pre-Trainings** werden Modelle auf umfangreichen, heterogenen Textkorpora mit Milliarden von Parametern trainiert. In diesem Prozess lernen sie durch die millionenfache Wiederholung von Vorhersageaufgaben, das jeweils wahrscheinlichste nächste Token vorherzusagen. Auf diese Weise entwickeln LLMs ein tiefes Verständnis für sprachliche Strukturen, Syntax und implizites Faktenwissen³².

Zur Veranschaulichung der Funktionsweise von Large Language Models (LLMs) zeigt das folgende Beispiel (vgl. Listing 4) in stark vereinfachter Form, wie Sprache in numerische Repräsentationen überführt und daraus Text generiert wird. Zunächst wird die Eingabe in **Tokens** zerlegt – kleinste bedeutungstragende Einheiten wie Wörter oder Satzzeichen. Diese Tokens werden anschließend durch **Vektorisierung** in hochdimensionale Zahlenräume (sogenannte *Embeddings*) abgebildet, die semantische Beziehungen zwischen Wörtern erfassen. Auf dieser Basis berechnet das Modell schließlich die **Wahrscheinlichkeit** des nächsten Tokens in der Sequenz und generiert so schrittweise den Folgetext.

Listing 4 Veranschaulichung der LLM-Funktionsweise (stark vereinfacht)

```

"Starte den Stream um 10 Uhr"

Tokenisierung
["Starte", "den", "Stream", "um", "10", "Uhr"]

Numerische Abbildung (Embeddings)
"Starte" → [0.12, -0.45, 0.87, ...]
"Stream" → [0.33, 0.05, -0.91, ...]
"10"     → [0.77, -0.12, 0.56, ...]

Vorhersage des nächsten Tokens
Mögliche Fortsetzungen nach "Stream":
"um"      → 0.62
"starten" → 0.18
"läuft"   → 0.09

```

Quelle: eigene Darstellung in Anlehnung an Vaswani et al., 2017; Brown et al., 2020

³⁰ Vgl. Xiao, T. & Zhu, J. (2025), S. 1ff

³¹ Vgl. Vaswani, A. et al. (2023). Attention Is All You Need, S. 2ff

³² Vgl. Xiao, T. & Zhu, J. (2025), S. 2f

Das Listing illustriert diesen Prozess anhand des Beispiels „Starte den Stream um 10 Uhr“: Aus der Eingabe werden Tokens gebildet, numerisch codiert und über Wahrscheinlichkeitsverteilungen fortgesetzt (z. B. „um“, „starten“, „läuft“). Diese Darstellung abstrahiert den tatsächlichen Rechenprozess erheblich – reale Modelle operieren mit Milliarden Parametern, hochdimensionalen Vektorräumen und komplexen Optimierungsverfahren, um kontextabhängige Sprachmuster präzise zu erfassen.

2.2.3 Anpassung von LLMs durch Prompting

Eine der zentralen Eigenschaften von LLMs ist ihre Fähigkeit zur schnellen Adaption an neue Aufgaben durch In-Context-Learning, ohne dass eine zeit- und datenintensive Neu-konfiguration der Modellparameter (Fine-Tuning) erforderlich ist³³. Dabei werden Aufgaben und Demonstrationsbeispiele ausschließlich über die Texteingabe an das Modell übermittelt – ein Verfahren, das als **Prompting** bezeichnet wird.

Je nach Anzahl der im Prompt bereitgestellten Beispiele unterscheidet man drei Hauptkategorien³⁴.

- **Zero-Shot:** Das Modell erhält lediglich eine natürlichsprachliche Anweisung, ohne Demonstrationsbeispiele³⁵ (vgl. Listing 5).

Listing 5 Zero-Shot-Prompting-Beispiel

```
Instruktion:
Extrahiere die folgenden Parameter aus dem Text:
- Quell-Agent
- Zieldatei
- Ziel-Agent
- Zielverzeichnis

Eingabetext:
"Ich muss täglich die Datei Geck.pdf von unserem FTP-Server (agent01) in das
Zielverzeichnis :/c/Abgabe auf dem Applikationsserver (agent02) übertragen."
```

Quelle: eigene Darstellung in Anlehnung an Brown et al., 2020

³³ Vgl. Xiao, T. & Zhu, J. (2025), S. 96

³⁴ Vgl. Brown, T. B. et al. (2020). Language Models are Few-Shot Learners, S. 3-4

³⁵ Vgl. Brown, T. B. et al. (2020), S. 7

- **One-Shot:** Neben der Aufgabenbeschreibung wird ein einzelnes Beispiel gegeben, das Inhalt und Format verdeutlicht (vgl. Listing 6).

Listing 6 One-Shot-Prompting-Beispiel

```

Instruktion:
Extrahiere die folgenden Parameter aus dem Text:
- Quell-Agent
- Zieldatei
- Ziel-Agent
- Zielverzeichnis

One Shot:
Beispiel (Demonstration):
"Bitte verschiebe die Logfiles (logs.txt) vom Webserver (ws_agent)
nach /backup auf den Archiv-Server (archive_srv)."

Output:
"Quell-Agent": "ws_agent"
"Zieldatei": "logs.txt"
"Ziel-Agent": "archive_srv"
"Zielverzeichnis": "/backup"

Eingabetext:
"Ich muss täglich die Datei Geck.pdf von unserem FTP-Server (agent01) in das
Zielverzeichnis :/c/Abgabe auf dem Applikationsserver (agent02) übertragen."

```

Quelle: eigene Darstellung in Anlehnung an Brown et al., 2020

- **Few-Shot:** Das Modell wird mit mehreren Beispielen konditioniert – typischerweise zwischen 10 und 100 –, solange diese in das Kontextfenster passen³⁶.

Die Beispiele bestehen jeweils aus einer Aufgabenbeschreibung und einer erwarteten Lösung. Durch das Erkennen solcher Muster kann das Modell die gelernten Strukturen auf neue Kontexte übertragen, indem es die wahrscheinlichste Token-Sequenz vorher sagt.

Die Effektivität des In-Context-Learnings steigt mit der Modellgröße, da größere Modelle kontextuelle Informationen effizienter verarbeiten und als leistungsfähigere Meta-Lerner agieren³⁷. Diese Fähigkeit, Aufgaben unmittelbar aus dem Kontext zu erlernen, ist entscheidend für das Erstellen von Streams durch natürliche Sprachverarbeitung. Sie ermöglicht es, unterschiedlich formulierte Benutzeranforderungen flexibel in streng strukturierte Konfigurationen wie XML zu übersetzen – ohne dass für jeden Anwendungsfall ein separates Modell trainiert werden muss.

³⁶ Vgl. Brown, T. B. et al. (2020), S. 6ff

³⁷ Vgl. Brown, T. B. et al. (2020), S. 4-6

2.2.4 LLMs als Schnittstelle von Sprache zu Konfiguration

Large Language Models (LLMs) fungieren als entscheidende technologische Schnittstelle, um die Kluft zwischen der menschlichen, unstrukturierten Sprache und der streng formalisierten Welt maschinenlesbarer Konfigurationen zu überbrücken. Sie bilden damit eine Schlüsseltechnologie für einen menschenzentrierten Automatisierungsansatz (HCA), die komplexen Systeme auch für Anwender ohne tiefgehendes Spezialwissen zugänglich macht³⁸.

Im Gegensatz zu traditionellen Automatisierungsmethoden wie der Robotic Process Automation (RPA), die auf strikt regelbasierten Abläufen beruht und bei semantisch komplexen Aufgaben schnell an ihre Grenzen stößt, eröffnen LLMs neue Perspektiven. Durch ihr tiefes Sprachverständnis können sie natürlichsprachliche Anforderungen interpretieren und in ausführbare Workflows überführen, die in standardisierten Formaten wie JSON oder XML repräsentiert werden. Dieser Ansatz – auch als „*Words to Workflows*“ bezeichnet (Minkova et al., 2024) – beschreibt die Transformation freier Texteingaben in eine Sequenz ausführbarer Schritte und bildet die Grundlage für die Automatisierung komplexer Geschäftsprozesse³⁹.

Large Language Models (LLMs) können als intelligente semantische „Übersetzer“ fungieren, die natürlichsprachliche Eingaben von Fachanwendern in formal korrekte Konfigurationen überführen. Aktuelle Forschungsarbeiten belegen, dass LLMs bereits mit wenigen Beispielen (*Few-Shot Learning*) in der Lage sind, strukturierte und schema-konforme Ausgaben zu erzeugen⁴⁰.

2.2.5 Potenziale und Limitationen von LLMs: Das Halluzinationsproblem

Trotz ihrer hohen Leistungsfähigkeit zeigen Large Language Models (LLMs) weiterhin spezifische Schwächen – insbesondere das sogenannte „**Halluzinieren**“. Im Kontext der *Natural Language Generation (NLG)* bezeichnet dieser Begriff die Erzeugung von Inhalten, die entweder faktisch falsch, kontextwidrig oder semantisch unplausibel sind und somit nicht mit dem zugrunde liegenden Quellenkontext übereinstimmen⁴¹. Dieses Phänomen beeinträchtigt die Zuverlässigkeit LLM-basierter Systeme erheblich, insbesondere in Anwendungsfeldern, die eine hohe faktische Korrektheit erfordern.

³⁸ Vgl. Toxtli, C. (2024), S. 4

³⁹ Vgl. Minkova, L. et al. (2024). From Words to Workflows Automating Business Processes, S. 5-8

⁴⁰ Vgl. Minkova, L. et al. (2024), S. 20f und 27ff

⁴¹ Vgl. Ji, Z. et al. (2024). Survey of Hallucination in Natural Language Generation, S. 3-4

Die Forschung unterscheidet grundsätzlich zwei Hauptformen von Halluzinationen in generativen Sprachmodellen:

1. **Intrinsische Halluzinationen** entstehen, wenn die Ausgabe im direkten Widerspruch zu den Eingabedaten steht⁴².

Im folgenden Beispiel (vgl. Listing 7) gibt der Nutzer den Namen `geck003_filetransfer` vor, während das Modell stattdessen den Wert `geck004_filetransfer` generiert. Der erzeugte Inhalt widerspricht somit eindeutig der Benutzereingabe.

Listing 7 Beispiel für eine intrinsische Halluzination in einer XML-Konfiguration

```
<!-- Prompt: "Erstelle einen Stream mit dem Namen geck003_filetransfer." -->
<Stream>
    <StreamName>geck004_filetransfer</StreamName>
</Stream>
```

Quelle: eigene Darstellung in Anlehnung an Ji et al., 2023

2. **Extrinsische Halluzinationen** beziehen sich auf Inhalte, die sich anhand der Quelle weder bestätigen noch widerlegen lassen.⁴³

Hier ergänzt das Modell einen zusätzlichen *EndPoint*, obwohl dieser in der ursprünglichen Eingabe nicht spezifiziert war. Diese Erweiterung kann zwar technisch plausibel sein, entspricht aber nicht der Intention des Nutzers und führt potenziell zu semantischen Inkonsistenzen. (vgl. Listing 8)

Listing 8 Beispiel für eine extrinsische Halluzination in einer XML-Konfiguration

```
<!-- Prompt: "Erstelle einen Stream mit dem einem Startpunkt udn einem
Datenübertragungsjob." -->
<Stream>
    <Job>
        <JobName>StartPoint</JobName>
        <JobCategory>StartPoint</JobCategory>
    </Job>
    <Job>
        <JobName>FileTransferJob</JobName>
        <JobCategory>FileTransfer</JobCategory>
    </Job>
    <Job>
        <JobName>EndPoint</JobName>
        <JobCategory>EndPoint</JobCategory>
    </Job>
</Stream>
```

Quelle: eigene Darstellung in Anlehnung an Ji et al., 2023

⁴² Vgl. Ji, Z. et al. (2024), S. 4

⁴³ Vgl. Ji, Z. et al. (2024), S. 4

Die **Ursachen von Halluzinationen** in großen Sprachmodellen sind vielfältig. Sie reichen von Verzerrungen in den Trainingsdaten über fehlerhafte Dekodierungsstrategien bis hin zu einer Übergewichtung des im Modell gespeicherten Weltwissens gegenüber den konkreten Eingaben⁴⁴. Gerade im Kontext der automatisierten Erstellung von XML-Konfigurationen und für die kontextsensitive Hilfe ist dieses Risiko besonders kritisch, da bereits geringe Abweichungen zu syntaktisch invaliden oder semantisch fehlerhaften Strukturen führen können.

Dies unterstreicht die Notwendigkeit robuster **Validierungsmechanismen**, etwa durch XSD-Schema-Prüfungen, um die formale und logische Konsistenz der erzeugten Konfigurationen sicherzustellen. Für weiterführende Anwendungen ist darüber hinaus der **Zugriff auf externes, verifiziertes Wissen** erforderlich, um die faktische Korrektheit der Modellantworten zu gewährleisten

Ansätze wie Retrieval-Augmented Generation (RAG) können hier einen Beitrag leisten, indem sie die Ausgabe eines Sprachmodells durch den Rückgriff auf verlässliche Quellen absichern und so das Risiko fehlerhafter oder halluzinierter Inhalte reduzieren. Dies ist notwendig, da LLMs die Tendenz haben falsche, aber plausibel klingende Informationen mit hoher Konfidenz auszugeben.⁴⁵

2.3 Kontextualisierung durch Retrieval-Augmented Generation (RAG)

Die direkte Nutzung von Large Language Models (LLMs) im Unternehmenskontext stößt auf zentrale Einschränkungen: Ihnen fehlt der kontext- und domänensensitive Zugriff auf interne Wissensbestände, wodurch die Modelle nicht in der Lage sind, organisationsspezifische Informationen verlässlich zu berücksichtigen. Dies führt häufig zu Halluzinationen und einer mangelnden **Nachprüfbarkeit** der generierten Inhalte⁴⁶.

Der Ansatz der **Retrieval-Augmented Generation (RAG)** begegnet diesem Problem, indem er ein vortrainiertes Modell mit einem externen Wissensspeicher kombiniert. Relevante Informationen werden zur Laufzeit abgerufen, in den Prompt integriert und sichern so Aktualität und Faktentreue der generierten Ergebnisse⁴⁷.

⁴⁴ Vgl. Ji, Z. et al. (2024), S. 8

⁴⁵ Vgl. Blüthgen, C. (2025). *Technische Grundlagen großer Sprachmodelle*. Die Radiologie, 65, S. 227

⁴⁶ Vgl. Honroth, T. et al. (2024). Retrieval Augmented Generation (RAG): Chat mit eigenen Daten, S. 1f

⁴⁷ Vgl. Yu, H. et al. (2024). Evaluation of Retrieval-Augmented Generation: A Survey, S. 1ff

Für die **Self-Service-Technologie** eröffnet der Einsatz von Retrieval-Augmented Generation (RAG) insbesondere im Rahmen des **Hilfesystems** ein hohes Potenzial: Fachanwender können Anfragen in natürlicher Sprache formulieren, während das System relevante Dokumentationsinhalte abruft und durch das LLM in verständlich aufbereiteter Form bereitstellt

2.3.1 Motivation und Konzept von RAG

Large Language Models (LLMs) speichern ihr Wissen in Milliarden von Parametern, können dieses jedoch nicht immer gezielt und fehlerfrei abrufen. Dadurch entstehen häufig sogenannte Halluzinationen (vgl. Kapitel 2.2.5) – also Ausgaben, die plausibel klingen, aber nicht mit den zugrunde liegenden Fakten übereinstimmen⁴⁸.

Das Konzept der Retrieval-Augmented Generation (RAG) adressiert dieses Problem, indem es zwei Kernkomponenten kombiniert:

1. einen **Retrieval-Mechanismus**, der gezielt relevante Informationen aus vordefinierten Wissensquellen abruft, und
2. ein **generatives Sprachmodell**, dass diese Inhalte verarbeitet und zu einer konsistenten, kontextgestützten Antwort formuliert⁴⁹.

Anstatt sich ausschließlich auf das implizit im Modell gespeicherte parametrische Wissen zu verlassen, werden bei der Retrieval-Augmented Generation (RAG) zur Laufzeit relevante Dokumentenabschnitte aus einem externen Wissensspeicher abgerufen, in den Prompt integriert und anschließend durch das LLM verarbeitet⁵⁰. Das Modell fungiert dadurch nicht mehr als alleinige Wissensquelle, sondern als Synthesizer und Erklärer geprüfter Inhalte.

Auf diese Weise lässt sich das Potenzial von LLMs auch für interne Dokumentationen und Wissensdatenbanken nutzbar machen – etwa im Fall von *Streamworks* mit technischen Handbüchern, Namenskonventionen oder Best-Practice-Beispielen. Ein aufwendiges und kostenintensives Nachtrainieren (Fine-Tuning) ist dabei nicht erforderlich⁵¹. Die Kombination eines **parametrischen Speichers** (vortrainiertes Sequenz-zu-Sequenz-Modell) mit einem **nicht-parametrischen Speicher** (Vektorindex der Unternehmensdokumente) ermöglicht Antworten, die sowohl aktuell als auch kontextuell fundiert sind⁵².

⁴⁸ Vgl. Honroth, T. et al. (2024), S. 2f

⁴⁹ Vgl. Bark, M. (2025). RAG-Modell - die Zukunft von KI im Unternehmen?, S. 2f

⁵⁰ Vgl. Yu, H. et al. (2024), S. 1f

⁵¹ Vgl. Honroth, T. et al. (2024), S. 3

⁵² Vgl. Lewis, P. et al. (2021). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, S. 1ff

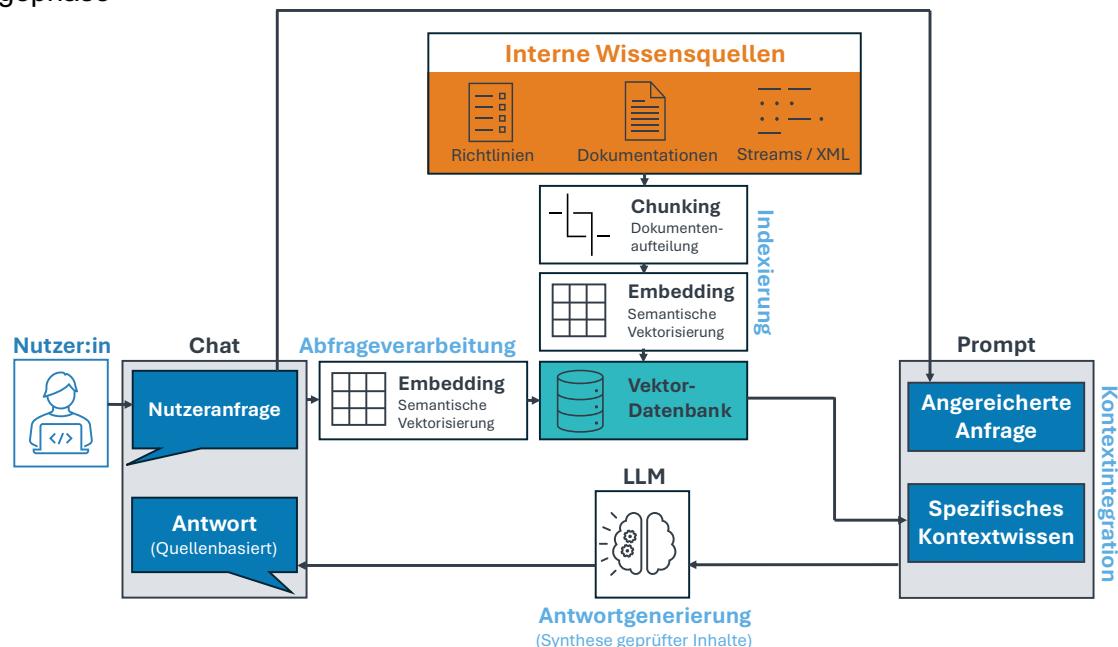
2.3.2 Technische Architektur eines RAG-Systems

Ein RAG-System besteht im Kern aus zwei Hauptkomponenten:

- dem **Retrieval-Modul**
- und dem **Generierungsmodul**

Der Gesamtprozess gliedert sich in zwei Phasen – **Indexierung** und **Abfrage**⁵³ –, die gemeinsam sicherstellen, dass relevante Informationen effizient gefunden und für die Antwortgenerierung genutzt werden können. Ziel dieser Architektur ist es, eine Nutzeranfrage vor der Verarbeitung durch das LLM mit externen Daten anzureichern und dadurch kontextualisierte sowie faktenbasierte Ergebnisse zu ermöglichen. Für den Anwendungsfall eines intelligenten Hilfesystems bei *Streamworks* ist dies besonders vorteilhaft, da die generativen Fähigkeiten eines LLMs direkt mit internen Wissensquellen gekoppelt werden können. Auf diese Weise entstehen präzise, nachvollziehbare und auf verifizierbaren Daten basierende Hilfestellungen für Anwender⁵⁴.

Abbildung 3 Vereinfachte Architektur eines RAG-Systems mit Indexierungs- und Abfragephase



Quelle: eigene Darstellung in Anlehnung an Zweitag (<https://www.zweitag.de/blog/rag-fine-tuning-lm>)

Der Gesamtprozess wird in Abbildung 3 dargestellt; im Folgenden werden die beiden Phasen näher erläutert.

⁵³ Vgl. Yu, H. et al. (2024), S. 2f

⁵⁴ Vgl. Honroth, T. et al. (2024), S. 3-4

Indexierungsprozess: Grundlage des Informationsabrufs ist die Indexierung. Hierbei werden Dokumente in kleinere, semantisch zusammenhängende **Abschnitte** (sogenannte **Chunks**) zerlegt. Diese Zerlegung erhöht die Präzision der semantischen Einbettung und reduziert das Risiko von Rauschen, das bei zu langen Textabschnitten entstehen würde.⁵⁵ Für Streamworks ist dies entscheidend, da technische Dokumentationen oder Code-Dateien oft sehr umfangreich sind, relevante Informationen jedoch meist nur in wenigen Sätzen oder Codeblöcken enthalten sind. Jeder Chunk wird anschließend mithilfe eines **Embedding-Modells** in einen hochdimensionalen Vektor überführt. Diese Vektoren werden in einer für Ähnlichkeitssuchen optimierten **Vektordatenbank** gespeichert⁵⁶.

Abfrageprozess: Der Abfrageprozess bildet das Herzstück des Systems und wird durch eine Nutzeranfrage initiiert. Optional kann ein LLM die Anfrage zunächst umformulieren, um die Suchintention zu präzisieren oder den Kontext vorheriger Dialoge zu berücksichtigen⁵⁷. Anschließend wird die Anfrage in einen Vektor transformiert und mittels semantischer Ähnlichkeitssuche die relevantesten Text-Chunks aus der Vektordatenbank abgerufen. Diese Vorgehensweise ist deutlich leistungsfähiger als eine reine Stichwortsuche, da es Anwendern erlaubt, Fragen in natürlicher Sprache zu stellen, ohne die exakten Fachbegriffe der Dokumentation kennen zu müssen. Für spezifische Suchanfragen – etwa nach Funktionsnamen oder Fehlermeldungen in Streamworks – kann eine **hybride Suche** (Kombination aus semantischer Vektorsuche und unscharfer Schlüsselwortsuche) die Retrieval-Qualität zusätzlich verbessern⁵⁸.

Die zurückgegebenen Dokumente werden anschließend neu geordnet (**Reranking**) und durch eine **Konsolidierungsstrategie** auf die für die Anfrage relevanten Informationen reduziert. Dieser Schritt ist notwendig, um die maximale Kontextlänge des Modells einzuhalten und nur relevante Inhalte in die Generierung einfließen zu lassen.

Das Generierungsmodul verarbeitet schließlich den angereicherten Prompt und erzeugt die finale, quellengestützte Antwort. Damit verändert sich die Rolle des LLMs grundlegend: Es fungiert nicht mehr als potenziell unzuverlässiger Wissensspeicher, sondern als Synthesizer verifizierter Informationen⁵⁹.

⁵⁵ Vgl. Barnett, S. et al. (2024). Seven Failure Points When Engineering a Retrieval Augmented Generation System, S. 2-3

⁵⁶ Vgl. Honroth, T. et al. (2024), S. 4

⁵⁷ Vgl. Barnett, S. et al. (2024), S. 2ff

⁵⁸ Vgl. Honroth, T. et al. (2024), S. 4ff

⁵⁹ Vgl. Barnett, S. et al. (2024), S. 2-3

2.3.3 Vergleich: Retrieval-Augmented Generation (RAG) vs. Fine-Tuning (FT)

Zur Anpassung von LLMs an spezifische Domänen – wie die internen Entwicklungsprozesse bei Streamworks – haben sich mit Retrieval-Augmented Generation (RAG) und Fine-Tuning (FT) zwei unterschiedliche Paradigmen etabliert.

Während RAG die Generierung durch externen, zur Laufzeit abgerufenen Kontext anreichert, integriert FT neues Wissen direkt in die Modellparameter, indem das Modell erneut auf domänenspezifischen Daten trainiert wird⁶⁰.

Für den Anwendungsfall bei *Streamworks* erweist sich der RAG-Ansatz als vorteilhafter, da Fine-Tuning erhebliche Nachteile in Bezug auf Flexibilität, Wartbarkeit und Kosten aufweist. Ein wesentlicher Vorteil von RAG liegt in der **dynamischen Aktualisierbarkeit des Wissens**: Durch den Austausch oder die Erweiterung des Dokumentenindex kann das System ohne erneutes Training an neue Gegebenheiten angepasst werden – ein entscheidender Faktor angesichts der sich kontinuierlich verändernden technischen Dokumentationen⁶¹. Fine-Tuning hingegen verfolgt einen **statischen Ansatz**: Jede Aktualisierung der Wissensbasis oder des zugrunde liegenden LLM erfordert einen erneuten, ressourcenintensiven Trainingsprozess. Dieser hohe Aufwand macht Fine-Tuning insbesondere für **dynamische Unternehmensumgebungen** unpraktikabel⁶².

Darüber hinaus sollte Fine-Tuning nicht zur Vermittlung von Faktenwissen eingesetzt werden, da dieses durch den Retrieval-Mechanismus von RAG wesentlich effizienter und überprüfbarer bereitgestellt werden kann⁶³.

Gleichwohl kann ein gezieltes Fine-Tuning **RAG-Systeme sinnvoll ergänzen**, beispielsweise um ein Modell auf spezifische Antwortstile, Domänenterminologien oder proprietäre Datenformate zu optimieren. Solche Optimierungen lassen sich durch **Parameter-Efficient Fine-Tuning (PEFT)**-Methoden wie **Low-Rank Adaptation (LoRA)** ressourcenschonend realisieren⁶⁴.

Da Zielsetzung und Umfang dieser Arbeit auf die **Machbarkeitsdemonstration** und den Aufbau eines prototypischen Systems ausgerichtet sind, wird bewusst auf Fine-Tuning verzichtet. Das im Prototyp implementierte Hilfesystem basiert vollständig auf dem RAG-Ansatz und untersucht dessen Potenziale im Anwendungskontext von Streamworks.

⁶⁰ Vgl. Wang, C. et al. (2025). RAG or Fine-tuning? A Comparative Study on LCMs-based Code Completion in Industry, S. 1-2

⁶¹ Vgl. Lewis, P. et al. (2021), S. 7

⁶² Vgl. Wang, C. et al. (2025), S. 4-5

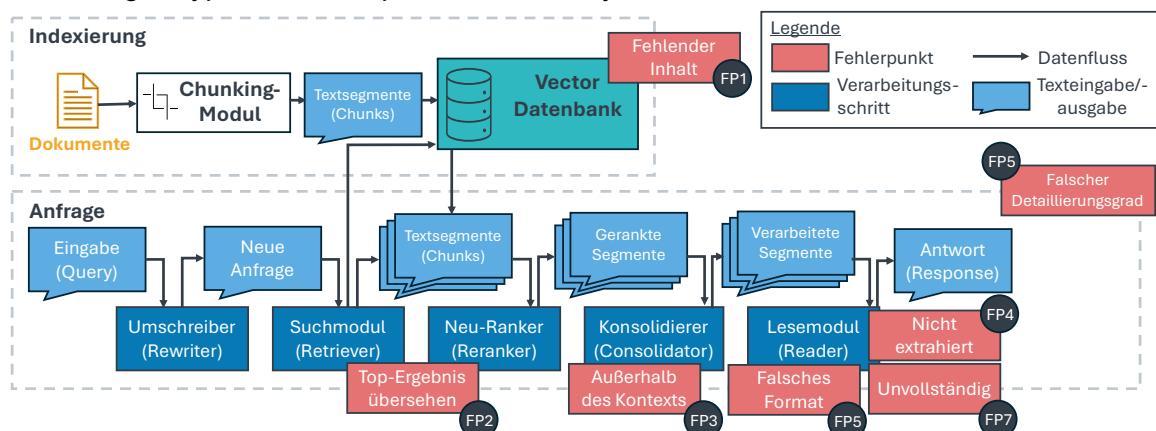
⁶³ Vgl. Honroth, T. et al. (2024, S. 7

⁶⁴ Vgl. Honroth, T. et al. (2024, S. 7ff

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. Ihre Validierung erfolgt häufig erst im laufenden Betrieb, sodass sich Robustheit und Qualität schrittweise entwickeln⁶⁵. Aufgrund der Systemkomplexität lässt sich die Leistungsfähigkeit nicht durch die isolierte Bewertung einzelner Komponenten erfassen, sondern nur durch die Analyse des Zusammenspiels von Retrieval und Generierung.

Abbildung 4 Typische Fehlerpunkte in RAG-Systemen



Quelle: eigene Darstellung in Anlehnung an Barnett et al., 2024

Die in Abbildung 4 visualisierten Prozessphasen verdeutlichen, an welchen Punkten Fehlerquellen im Zusammenspiel von Retrieval und Generierung entstehen können.

Im Folgenden werden die sieben zentralen „Failure Points“ (FP1–FP7) nach Barnett et al. (2024) detailliert beschrieben und um geeignete Gegenmaßnahmen ergänzt.⁶⁶

Tabelle 1 Fehlerpunkte (FP1-FP7) und Gegenmaßnahmen im RAG-Prozess

FP1	Fehlerpunkt (Missing Content)	Retrieval
Problem: Gesuchte Info nicht im Index; Modell könnte trotzdem antworten.		
Maßnahme: Recall ↑ (größeres k), Index-Coverage-Checks, klare „Keine ausreichenden Informationen“-Antwort, kontinuierliche Indexpflege/ETL.		
FP2	Verpasste Relevanz (Missed Top Ranked)	Retrieval
Problem: Relevantes vorhanden, aber nicht in Top-Treffern.		
Maßnahme: Hybride Suche (Vektor + BM25/Fuzzy), domänen spezifisches Reranking, Field-Boosting (Titel/Überschriften), Query-Rewrite.		

⁶⁵ Vgl. Barnett, S. et al. (2024), S. 1ff

⁶⁶ Vgl. Barnett, S. et al. (2024), S. 3-4

FP3	Nicht im Kontext (Not in Context)	Konsolidierung
	<p>Problem: Relevantes abgerufen, beim Zuschneiden entfernt.</p> <p>Maßnahme: Evidenz-Selektion mit Diversität (z. B. MMR), Kontext-Budget je Quellentyp, Anti-Redundanz-Heuristiken, Logging verworfener Evidenz.</p>	
FP4	Nicht extrahiert (Not Extracted)	Generierung
	<p>Problem: Info im Kontext, wird aber nicht sauber genutzt.</p> <p>Maßnahme: Strikte Prompt-Struktur (Zitieren → Begründen → Antwort), Quellenstellen anfordern, Temperatur niedrig, Antwort gegen Regeln prüfen.</p>	
FP5	Falsches Format (Wrong Format)	Generierung
	<p>Problem: Vorgaben (Tabelle/XML) ignoriert.</p> <p>Maßnahme: Format-Constraints (JSON/XML-Schema im Prompt), Parser/Validator (z. B. XSD für Streamworks-XML), Fail-fast & Retry.</p>	
FP6	Unpassender Detaillierungsgrad (Incorrect Specificity)	Generierung
	<p>Problem: Zu allgemein/zu granular.</p> <p>Maßnahme: Rolle/Adressat im Prompt, Ziel-Länge/Gliederung vorgeben, optional Follow-ups („Mehr Details zu X?“)</p>	
FP7	Unvollständig (Incomplete)	Generierung
	<p>Problem: Teile der Frage bleiben unbeantwortet.</p> <p>Maßnahme: Checklist-Prompts („A/B/C abdecken“), Abdeckungsprüfung am Ende, Mehrquellen-Fusion mit Lückenhinweis.</p>	

Quelle: eigene Abbildung in Anlehnung an Barnett, S. et al. (2024)

Die Analyse der Fehlerpotenziale verdeutlicht, dass RAG-Systeme trotz ihrer prinzipiellen Robustheit eine Vielzahl interdependent Schwachstellen aufweisen. Ihre Leistungsfähigkeit hängt maßgeblich von der Qualität des Indexes, der Konsistenz der Kontexte und der Kontrolle der Generierung ab. Die im Kapitel dargestellten Failure Points und Gegenmaßnahmen bilden damit eine methodische Grundlage, um die Zuverlässigkeit des im weiteren Verlauf entwickelten Prototyps gezielt zu bewerten und zu optimieren.

Nach der theoretischen Fundierung des Forschungsgegenstands widmet sich das folgende Kapitel der Analysephase des Design-Science-Research-Prozesses. Dabei werden die bestehenden Workload-Erstellungsprozesse in Streamworks systematisch untersucht, um Anforderungen und Gestaltungsziele für das zu entwickelnde Artefakt abzuleiten.

3 Analyse der Stream-Erstellungsprozesse bei Streamworks

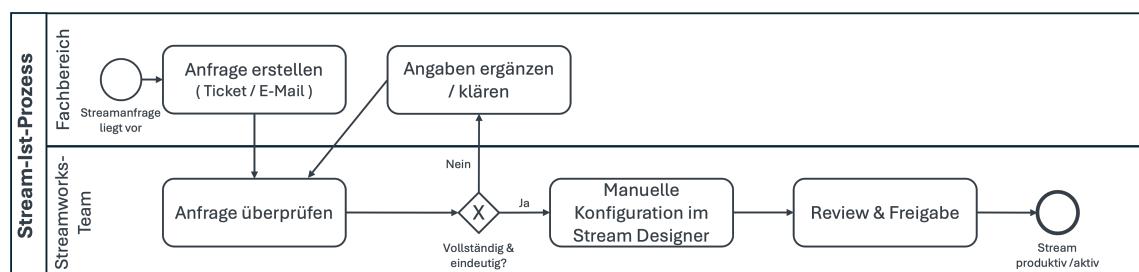
Aufbauend auf den in Kapitel 2 beschriebenen theoretischen Grundlagen untersucht dieses Kapitel die aktuelle Ausgangssituation der Workload-Automatisierung in Streamworks.

Die Analyse kombiniert interne Prozessdokumentationen mit praxisnahen Erkenntnissen aus drei Experteninterviews mit Fachverantwortlichen aus dem Streamworks-Umfeld⁶⁷. Diese beleuchten den heutigen Stream-Erstellungsprozess aus unterschiedlichen Perspektiven und liefern qualitative Einsichten zu typischen Problemfeldern, Automatisierungspotenzialen und Self-Service-Ansätzen. Die daraus gewonnenen Ergebnisse dienen als empirische Validierung der theoretisch abgeleiteten Anforderungen und bilden die Grundlage für die anschließende Systemkonzeption.

3.1 Aktueller Prozess

Die Erstellung neuer Streams in Streamworks erfolgt derzeit überwiegend manuell und ist stark von der Expertise einzelner Entwickler abhängig. Ausgangspunkt sind in der Regel Anfragen aus den Fachbereichen, die über unterschiedliche Kanäle wie Ticketsysteme, E-Mails oder Excel-Formulare eingehen⁶⁸. Diese Anfragen sind häufig uneinheitlich in Form und Detailtiefe, sodass zentrale Pflichtinformationen (z. B. Job-Typ, ausführender Agent, Abhängigkeiten) teilweise fehlen oder nur unpräzise angegeben werden.

Abbildung 5 Ist-Prozess der Stream-Erstellung bei Streamworks (BPMN-Modell)



Quelle: eigene Darstellung in Anlehnung an Experteninterviews und interne Prozessdokumente

Der anschließende Prozess, dargestellt in Abbildung 5, ist iterativ und von Rückfragen geprägt: Entwickler prüfen die Anfrage, fordern fehlende Angaben nach und beginnen

⁶⁷ Vgl. Zusammenfassung der Experteninterviews zur Streamworks-Automatisierung (separates Begleitdokument zur Bachelorarbeit)

⁶⁸ Vgl. Geck (2025): Zusammenfassung der Experteninterviews, S. 3

anschließend mit der manuellen Konfiguration des Streams im Stream Designer. Abhängig von Komplexität und Vollständigkeit der Eingangsdaten sind oft mehrere Abstimmungsschleifen erforderlich.

Neben der eigentlichen Konfiguration fallen Dokumentations- und Qualitätssicherungsschritte an, die ebenfalls manuell erfolgen. Die Durchlaufzeiten variieren entsprechend stark: Während einfache Anfragen innerhalb weniger Stunden umgesetzt werden können, benötigen komplexere Streams aufgrund wiederholter Abstimmungen und Korrekturen mehrere Tage bis hin zu Wochen⁶⁹.

Die Experteninterviews verdeutlichten dabei mehrere wiederkehrende Probleme:

- Unvollständige oder unklare Angaben in den Anfragen
→ Verzögerungen durch Rückfragen.
- Starke Abhängigkeit von individuellem Expertenwissen
→ Engpässe und Single-Point-of-Knowledge.
- Hohe Fehleranfälligkeit bei der manuellen Parametrierung (z. B. Tippfehler, Verstöße gegen Namenskonventionen).
- Erhöhter Kommunikationsaufwand zwischen Fachbereich und Entwicklung, insbesondere bei nicht standardisierten Anforderungen⁷⁰.

Diese Ausgangslage verdeutlicht, dass der bestehende Prozess erhebliche manuelle Aufwände verursacht und dadurch die Skalierbarkeit der Workload-Automatisierung begrenzt. Sie unterstreicht den Bedarf an stärkerer Standardisierung, frühzeitiger Validierung und benutzerfreundlichen Self-Service-Ansätzen.

3.2 Automatisierungspotenziale

Die Analyse des aktuellen Prozesses verdeutlicht, dass ein erheblicher Teil der Ineffizienzen aus wiederkehrenden und grundsätzlich standardisierbaren Arbeitsschritten resultiert. Daraus ergeben sich folgende Automatisierungspotenziale:

- **Standardjobs und Templates:** Viele Streams folgen wiederkehrenden Mustern (z. B. Dateiübertragungen, SAP-Jobs)⁷¹. Ein regelbasiertes Eingabesystem ermöglicht es Fachanwendern, diese Standardfälle eigenständig zu konfigurieren, während technische Details automatisiert geprüft und ergänzt werden.

⁶⁹ Vgl. Geck (2025): Zusammenfassung der Experteninterviews, S. 5

⁷⁰ Vgl. Geck (2025): Zusammenfassung der Experteninterviews, S. 5

⁷¹ Vgl. Geck (2025): Zusammenfassung der Experteninterviews, S. 6

- **Automatische Validierung:** Häufige Probleme sind unvollständige oder fehlerhafte Eingaben. Eine automatisierte Prüfung von Pflichtparametern sowie syntaktische Validierungen (z. B. mittels XSD-Schemachecks) erkennen Fehler frühzeitig und reduzieren Rückfragen.
- **Self-Service-Portal:** Eine benutzerfreundliche Oberfläche ermöglicht es Fachbereichen, Streams eigenständig zu erstellen oder vorzubereiten. Das senkt den Kommunikationsaufwand und entlastet Expertinnen und Experten.
Voraussetzung sind intuitive Bedienung, Abstraktion technischer Komplexität und klare, unmittelbare Rückmeldungen zu Vollständigkeit und Konsistenz.

Die Experteninterviews verdeutlichen, dass grundsätzlich eine hohe Akzeptanz für Self-Service-Ansätze besteht, sofern klare Regeln, Governance-Mechanismen und Validierungen sicherstellen, dass die erzeugten Konfigurationen korrekt und wartbar bleiben⁷². Ist besonders erfolgskritisch wurde eine benutzerfreundliche Eingabemaske mit transparenter Rückmeldung hervorgehoben – etwa in Form von Hinweisen, welche Angaben fehlen oder fehlerhaft sind und wie diese zu korrigieren sind..

Fazit: Durch die Kombination aus standardisierter Eingaben, frühzeitiger Validierung und intuitiven Self-Service-Funktionen kann der Prozess der Stream-Erstellung deutlich robuster und effizienter gestaltet werden – bei gleichzeitig höherer Autonomie der Fachbereiche.

3.3 Systemanforderungen

Aufbauend auf den in Kapitel 3.2 identifizierten Potenzialen werden im Folgenden die Anforderungen an ein KI-gestütztes Self-Service-System zur Stream-Erstellung formuliert.

Da diese Arbeit auf einen prototypischen Machbarkeitsnachweis abzielt (vgl. Kap. 1) und die in Kapitel 2 und 4 beschriebenen Kernprinzipien – RAG-basierte Wissensintegration und frühe technische Validierung – demonstriert, konzentriert sich der entwickelte MVP auf wenige, robuste Kernfunktionen.

Governance-Regeln und Freigabeprozesse werden konzeptionell spezifiziert, im Prototyp jedoch noch nicht technisch umgesetzt. Sie dienen als Leitplanken für spätere Ausbaustufen und gewährleisten, dass der Ansatz skalierbar und regelkonform weiterentwickelt werden kann.

⁷² Vgl. Geck (2025): Zusammenfassung der Experteninterviews, S. 7

3.3.1 Funktionale Anforderungen

Der entwickelte MVP muss die folgenden Funktionen bereitstellen, um einen nutzbaren und konsistenten End-to-End-Prozess der Stream-Erstellung zu gewährleisten:

- **Geführte Datenerfassung:** Erfassung aller Pflichtfelder je Job-Typ unter Anwendung von Plausibilitätsregeln und kontextsensitiven Hilfetexten. Dadurch wird sichergestellt, dass Eingaben vollständig, logisch konsistent und fachlich korrekt sind.
- **Sofort-Feedback:** Automatische Rückmeldungen bei fehlenden oder widersprüchlichen Eingaben in Form klarer, umsetzbarer Fehlermeldungen. Diese Funktion unterstützt Fachanwender dabei, Fehler frühzeitig zu erkennen und selbstständig zu korrigieren.
- **RAG-gestütztes Hilfesystem:** Kontextbasierte Bereitstellung quellenfundierter Informationen und Beispiele aus internen Dokumentationen. Auf diese Weise erhalten Anwender faktenbasierte Unterstützung während der Konfiguration.

3.3.2 Qualitätssicherung

Zur Vermeidung von Fehlkonfigurationen wurden im Prototyp verbindliche Maßnahmen zur Qualitätssicherung definiert

- **Technische Prüfungen:** Überprüfung der Wohlgeformtheit, XSD-Validität und definierter Regelwerke (z. B. Wertebereiche, Namenskonventionen).
- **Nachvollziehbarkeit:** Protokollierung aller Prüfungen und Entscheidungen (wer / was / wann / warum) auf MVP-Niveau zur Sicherstellung von Transparenz und Reproduzierbarkeit.

Governance und Abgrenzung manueller Schritte

Eine nachhaltige Automatisierung setzt klare Zuständigkeiten, Freigabeprozesse und Kontrollmechanismen voraus. Im Rahmen dieser Arbeit wurden die entsprechenden Governance-Anforderungen konzeptionell analysiert und definiert, jedoch im Prototyp nicht technisch implementiert. Sie bilden die Grundlage für eine spätere Erweiterung des Systems zu einem vollwertigen Rollen- und Workflow-basierten Freigabeprozess.

Rollen und Verantwortlichkeiten:

- **Fachanwender:** Erfassung und Vorbereitung standardisierter Anwendungsfälle im Self-Service.
- **Entwickler / Reviewer:** Durchführung technischer Prüfungen, Bearbeitung von Sonderfällen sowie finale Freigabe.

Freigabeprozesse:

Es wurden klare Prozesspfade spezifiziert, die eine abgestufte Prüfung und Freigabe sicherstellen

(*autovalidiert → optionaler Review → Freigabe → Import*).

Diese konzeptionelle Modellierung gewährleistet, dass Governance-Aspekte bereits in der Architektur des MVP berücksichtigt sind und in künftigen Ausbaustufen technisch umgesetzt werden können.

3.4 Bedenken und Risiken

Trotz der in Kapitel 3.2 und 3.3 aufgezeigten Potenziale ist die Einführung eines KI-gestützten Self-Service-Systems mit spezifischen Risiken verbunden. Die Experteninterviews verdeutlichen dabei insbesondere drei Risikodimensionen: **technische Zuverlässigkeit, organisatorische Akzeptanz sowie Governance und Verantwortung**.

3.4.1 Technische Zuverlässigkeit

Die Generierung technischer Artefakte durch Large Language Models (LLMs) kann fehlerhafte, unvollständige oder strukturell inkonsistente Ausgaben erzeugen⁷³. Im Kontext von Streamworks führt dies unmittelbar zu nicht validen oder fachlich widersprüchlichen XML-Konfigurationen. Für den MVP sind daher verbindliche Kontrollmechanismen vorgesehen:

- **Strikte Formatvorgaben:** Einsatz fester Output-Constraints im Prompt (z. B. XML/JSON), geringe Temperatur und „Fail-fast“-Strategie mit automatisiertem Wiederholungsversuch.
- **Validierung:** Prüfung jeder generierten Datei durch XSD-Validierung und Parser-Checks, um fehlerhafte Strukturen vor der Weitergabe abzufangen.

3.4.2 Organisatorische Risiken

Die Akzeptanz eines Self-Service-Systems hängt maßgeblich von seiner Benutzerfreundlichkeit und Zuverlässigkeit ab. Fehlende Führung, unklare Pflichtfelder oder unplausible Standardwerte führen dazu, dass Anwender das System umgehen und wieder auf manuelle Anfragen zurückgreifen.

Für den MVP wurden daher zentrale Gegenmaßnahmen definiert:

⁷³ Vgl. Geck (2025): Zusammenfassung der Experteninterviews, S. 9

-
- Einsatz vordefinierter **Templates** für Standardfälle,
 - konsequente **Pflichtfeld- und Plausibilitätsprüfungen**,
 - sowie ein unmittelbares, klar verständliches **Sofort-Feedback** bei Eingabefehlern.

Diese Mechanismen fördern Vertrauen, verringern Rückfragen und bilden die Grundlage für eine nachhaltige Akzeptanz in der Praxis.

3.4.3 Governance und Verantwortung

Während Standardfälle im Self-Service eigenständig abgedeckt werden können, erfordern komplexe Abhängigkeiten, sicherheitsrelevante Parameter oder Sonderfreigaben weiterhin eine manuelle Überprüfung.

Für den MVP wurden daher Rollen (Fachanwender, Reviewer, Betrieb) sowie ein schlanker Freigabepfad konzeptionell definiert. Die technische Umsetzung eines vollständigen Rollen- und Workflow-Systems ist für spätere Ausbaustufen vorgesehen, um Governance-Anforderungen schrittweise zu operationalisieren.

Der Erfolg des Self-Service-Ansatzes hängt nicht allein von der Leistungsfähigkeit der eingesetzten Modelle ab, sondern vom Zusammenspiel aus strikter Validierung, benutzerzentriertem Design und klar definierten Governance-Regeln. Im MVP werden die technischen Sicherungsmechanismen vollständig umgesetzt (Format-Constraints, XSD-Validierung, Checklisten), während Governance-Strukturen zunächst konzeptionell festgelegt und als Grundlage für spätere Ausbaustufen verankert werden.

Insgesamt zeigt die Analyse, dass der aktuelle Prozess der Stream-Erstellung noch stark von manuellen Eingriffen, uneinheitlichen Eingabeformaten und unzureichenden Validierungen geprägt ist. Die daraus abgeleiteten Anforderungen bilden die Basis für die im folgenden Kapitel entwickelte Self-Service-Architektur, die diese Schwachstellen gezielt adressiert und den Automatisierungsgrad nachhaltig erhöht.

4 Konzeption und Design

Aufbauend auf den in Kapitel 3 identifizierten Problemfeldern wird in diesem Kapitel ein technologie-agnostisches Design für ein Self-Service- und KI-gestütztes System zur Stream-Erstellung in Streamworks entwickelt. Kapitel 4 beschreibt die konzeptionelle Architektur sowie die logischen Komponenten und Prozesse, die als Grundlage für die prototypische Implementierung in Kapitel 5 dienen. Im Rahmen des Design-Science-Research-Ansatzes entspricht dieses Kapitel der Gestaltungs- und Entwicklungsphase, in der eine Lösung systematisch entworfen wird, bevor sie in Kapitel 5 technisch realisiert wird.

4.1 Zielarchitektur

Die Zielarchitektur definiert die logische Struktur des geplanten Systems und orientiert sich an übergreifenden Gestaltungsprinzipien. Sie bildet das konzeptionelle Fundament, auf dem spätere technische Entscheidungen und Implementierungen aufbauen.

4.1.1 Leitprinzipien der Zielarchitektur

Die Zielarchitektur orientiert sich an grundlegenden Gestaltungsprinzipien, die die technische Stabilität, Nutzerzentrierung und Zukunftsfähigkeit des Systems sicherstellen.

- **Modularität und lose Kopplung:** Komponenten werden unabhängig voneinander entwickelt und über klar definierte Schnittstellen integriert. Eine geringe Kopplung minimiert unerwünschte Wechselwirkungen zwischen Modulen und gilt als zentrales Qualitätsmerkmal wartbarer und erweiterbarer Architekturen⁷⁴.
- **Nutzerzentrierte Automatisierung:** Das System unterstützt Fachanwender bei der Umsetzung, ohne deren fachliche Verantwortung zu ersetzen. Der Mensch bleibt Entscheidungsträger – die KI agiert als Assistenzsystem.
- **Validierung als Sicherheitsanker:** Alle generierten Konfigurationen werden automatisch geprüft, um logische Fehler frühzeitig abzufangen.
- **Erweiterbarkeit:** Neue Job-Typen und Eigenschaften können schrittweise integriert werden, ohne die Gesamtarchitektur zu beeinträchtigen.

Auf Basis dieser Leitprinzipien wird im Folgenden das Architekturmodell dargestellt, das den strukturellen Aufbau und die funktionale Gliederung des geplanten Systems beschreibt.

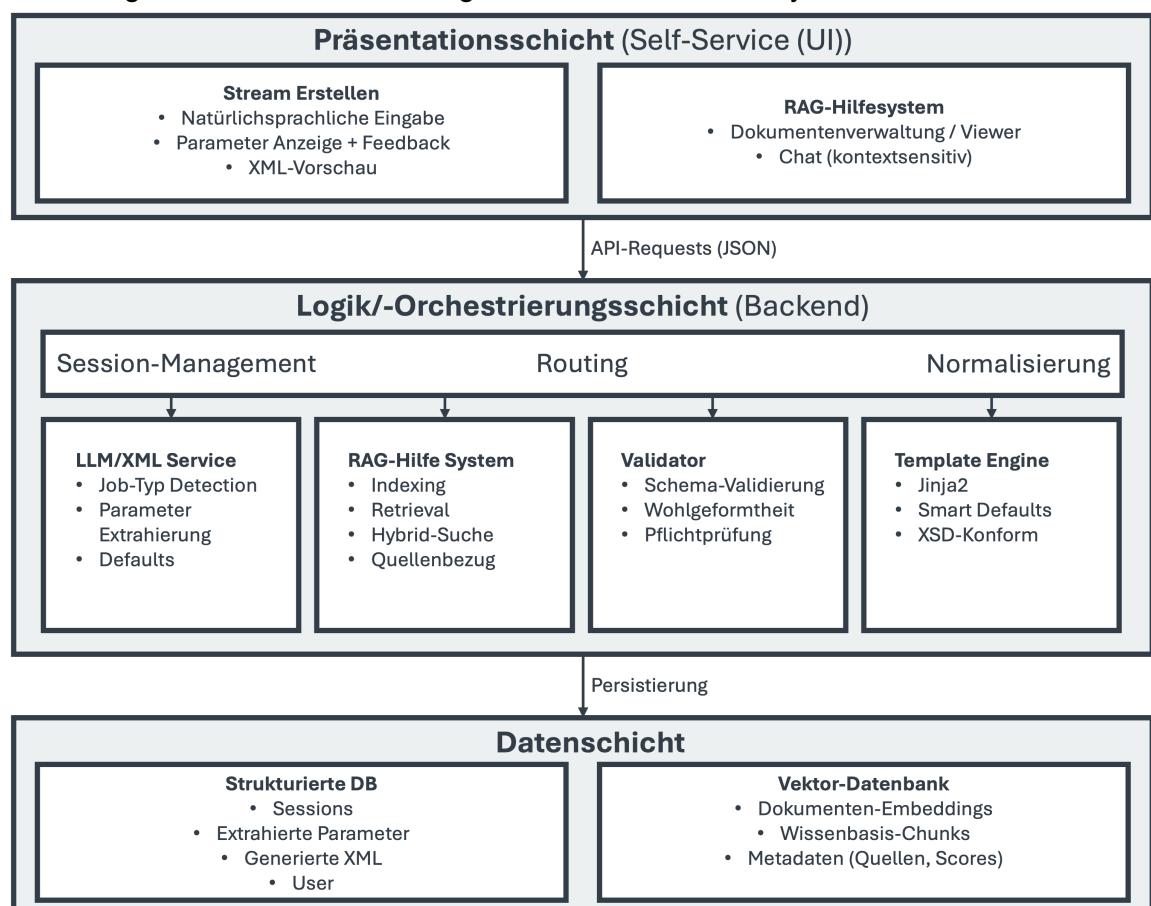
⁷⁴ Vgl. Meyer, A. (2018). *Softwareentwicklung: Ein Kompass für die Praxis*, S. 91

4.1.2 Architekturmodell

Die Systemarchitektur folgt einem dreischichtigen Modell (vgl. Abbildung 6), das eine klare funktionale Trennung der Verantwortlichkeiten sicherstellt und damit Modularität, Wartbarkeit und Erweiterbarkeit fördert.

- **Präsentationsschicht:** Sie ermöglicht die Eingabe von Anforderungen in natürlicher Sprache und bietet Funktionen zur Parameteranzeige, XML-Vorschau sowie kontextbezogene Unterstützung durch das RAG-Hilfesystem.
- **Logik bzw. Orchestrierungsschicht:** Diese Schicht steuert den Datenfluss zwischen Benutzeroberfläche, LLM/XML-Service, Validator, Template-Engine und RAG-Hilfesystem. Sie übernimmt das Session-Management und die Normalisierung der Eingaben.
- **Datenschicht:** Hier werden Eingaben, Parameter und generierte XML-Dateien in einer relationalen Datenbank gespeichert. Dokument-Embeddings und Wissens-Chunks werden zusätzlich in einer Vektordatenbank abgelegt, um semantische Suchvorgänge zu ermöglichen.

Abbildung 6 Zielarchitektur des KI-gestützten Self-Service-Systems



Quelle: eigene Abbildung

Das Schichtenmodell bildet die Grundlage einer robusten, erweiterbaren und benutzerfreundlichen Architektur und dient zugleich als konzeptioneller Rahmen für die nachfolgenden Systemkomponenten und Prozessmodelle.

4.2 Systemkomponenten

Die Zielarchitektur wird durch fünf funktionale Kernkomponenten realisiert, die jeweils spezifische Aufgaben innerhalb der Architekturnschichten übernehmen und gemeinsam den End-to-End-Prozess der Stream-Erstellung abbilden. Sie interagieren über klar definierte Schnittstellen und bilden zusammen die architektonische Kernstruktur des Prototyps.

4.2.1 Self-Service-Frontend (Benutzeroberfläche)

Das Self-Service-Frontend bildet die zentrale Interaktionsschicht zwischen Fachanwendern und System. Es abstrahiert technische Komplexität und ermöglicht sowohl natürliche Spracheingaben als auch strukturierte Formulareingaben.

Die Hauptfunktionen sind:

- Eingabe von Automatisierungsanforderungen in Freitext
- Sofort-Feedback bei unvollständigen oder unplausiblen Eingaben
- Vorschau und Bearbeitung generierter Konfigurationen
- Einbindung kontextsensitiver Hilfestellungen aus dem integrierten Wissenssystem

4.2.2 Orchestrierungsschicht (Backend-Logik)

Die Orchestrierungsschicht bildet das logische Rückgrat der Architektur und koordiniert den Informationsfluss zwischen allen Systemkomponenten. Sie normalisiert Eingaben, ruft spezialisierte Dienste auf und bereitet Ergebnisse für die Darstellung im Frontend auf.

Zentrale Aufgabenbereiche sind:

- Konsolidierung heterogener Eingaben
- Session-Management für Dialoge über mehrere Interaktionsrunden
- Steuerung der Aufrufe an LLM/XML-Service, RAG-System und Validator
- Zentrale Fehlerbehandlung und definierte Fallback-Mechanismen

4.2.3 LLM/XML-Service (Übersetzung Sprache → Struktur)

Der LLM/XML-Service stellt die Schnittstelle zwischen der natürlichsprachlichen Eingabe und der strukturierten Konfiguration dar. Er erkennt Job-Typen, extrahiert Parameter, berücksichtigt Namenskonventionen und Abhängigkeiten und generiert daraus eine valide XML-Struktur. Damit übernimmt der Service die zentrale Transformationslogik des Systems und bildet die Grundlage für die automatisierte Stream-Erstellung.

4.2.4 RAG-Hilfesystem (Wissensintegration)

Das RAG-Hilfesystem erweitert die Architektur um eine wissensbasierte Komponente, die Fachanwender bei der Eingabe und Konfiguration unterstützt. Es indexiert relevante Dokumentationen, ruft bei Anfragen passende Inhalte ab und stellt diese in verständlicher, quellengestützter Form bereit. Auf diese Weise erhalten Anwender faktenbasierte und nachvollziehbare Antworten mit direktem Quellenbezug, was Transparenz und Vertrauen in die Systemausgaben erhöht.

4.2.5 Validator (Qualitätssicherung)

Der Validator stellt die Qualitätssicherung des Systems sicher und überprüft alle generierten Konfigurationen vor ihrer Freigabe. Die Prüfung erfolgt in mehreren aufeinander aufbauenden Stufen:

1. Strukturelle Validierung (Wohlgeformtheit)
2. Schema-Validierung (Übereinstimmung mit XSD)
3. Semantische Validierung (logische Konsistenz)
4. Vollständigkeitsprüfung (Überprüfung aller Pflichtparameter)

Durch die klare Trennung der Zuständigkeiten entsteht eine lose gekoppelte Architektur, die eine hohe Wartbarkeit, Testbarkeit und zukünftige Erweiterbarkeit sicherstellt.

4.3 Prozesse

Die Architektur bildet zwei zentrale Kernprozesse ab, die durch klar definierte Schnittstellen miteinander verbunden sind. Sie beschreiben den vollständigen Ablauf der Systeminteraktionen – von der automatisierten XML-Generierung bis zur wissensbasierten Unterstützung durch das RAG-System.

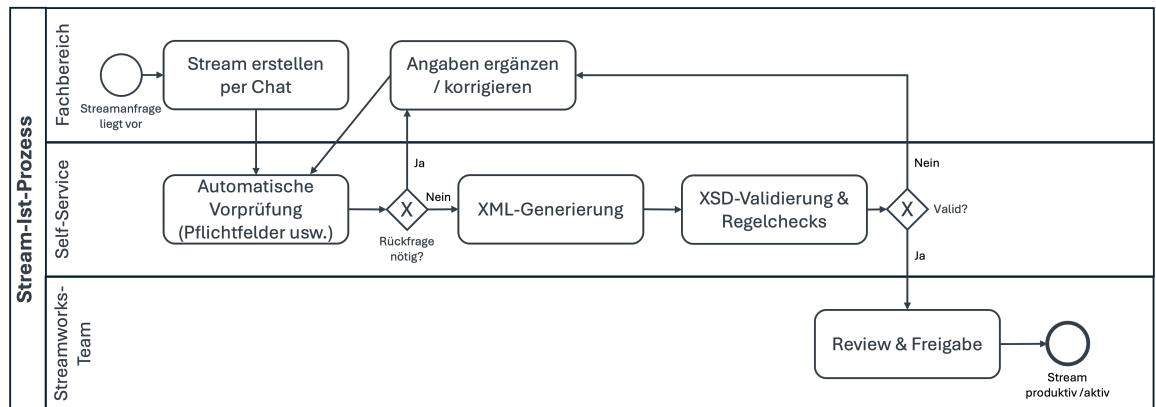
4.3.1 Prozess der XML-Generierung

Der Prozess der XML-Generierung beschreibt den zentralen Ablauf, über den natürliche Spracheingaben schrittweise in valide Streamworks-Konfigurationen überführt werden.

Ablaufschritte:

- **Eingabe:** Fachanwender geben Anforderungen in natürlicher Sprache ein.
- **Vorverarbeitung:** Automatische Vorprüfung, Normalisierung und Ergänzung von Metadaten; bei Unklarheiten erfolgt eine Rückfrage an den Nutzer.
- **Übersetzung:** Der LLM/XML-Service generiert auf Basis der Eingaben eine XML-Konfiguration.
- **Validierung:** Prüfung der generierten Struktur durch XSD- und Regelchecks; bei Fehlern erfolgt eine Rückführung in die vorherigen Schritte des Fachanwenders.
- **Freigabe & Bereitstellung:** Manuelle Review- und Freigabephase durch das Streamworks-Team, anschließend Bereitstellung und Rückmeldung im Frontend, optional mit Exportfunktion.

Abbildung 7 Soll-Prozess der Stream-Erstellung mit Self-Service und KI (BPMN-Modell)



Quelle: eigene Darstellung

Der dargestellte Soll-Prozess zeigt den End-to-End-Ablauf von der Anforderung bis zur Bereitstellung einer validierten XML-Konfiguration.

4.3.2 Prozess der RAG-gestützten Unterstützung

Das RAG-System ergänzt die automatisierte Stream-Erstellung um eine wissensbasierte Komponente. Es umfasst zwei Teilprozesse:

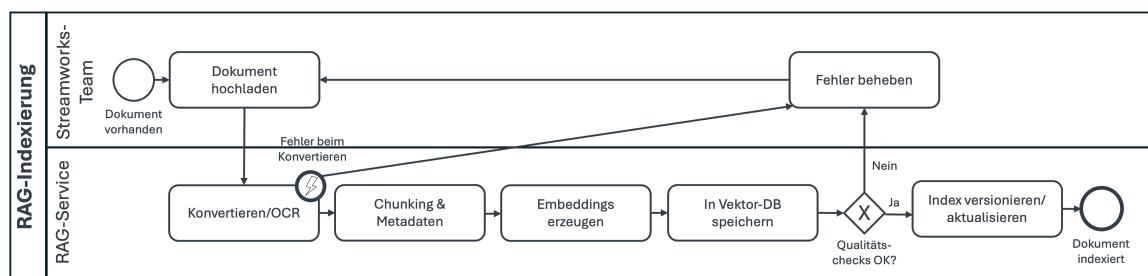
1. die **Indexierung**, in der Dokumente aufbereitet und in den Vektorindex überführt werden, sowie
2. das **Retrieval**, bei dem diese Wissensbasis zur Beantwortung von Fachanfragen genutzt wird.

4.3.2.1 RAG-Indexierung – Verarbeitung und Qualitätsprüfung

Die Indexierung bildet die Grundlage der Wissensbasis. Nach dem Upload durch das Streamworks-Team konvertiert der RAG-Service die Datei in ein Textformat. Fehler beim Import werden über ein Boundary-Event abgefangen und führen zur manuellen Korrektur. Anschließend erfolgt das **Chunking** mit Metadatenanreicherung, die Erzeugung von **Embeddings** sowie das Speichern der Vektoren in **einer Vektordatenbank**.

Abbildung 8 veranschaulicht den Ablauf der RAG-Indexierung im BPMN-Modell. Ein Qualitäts-Gateway prüft Vollständigkeit und Konsistenz. Bei erfolgreicher Validierung wird der Index versioniert und veröffentlicht.

Abbildung 8 RAG-Indexierung (BPMN-Modell)



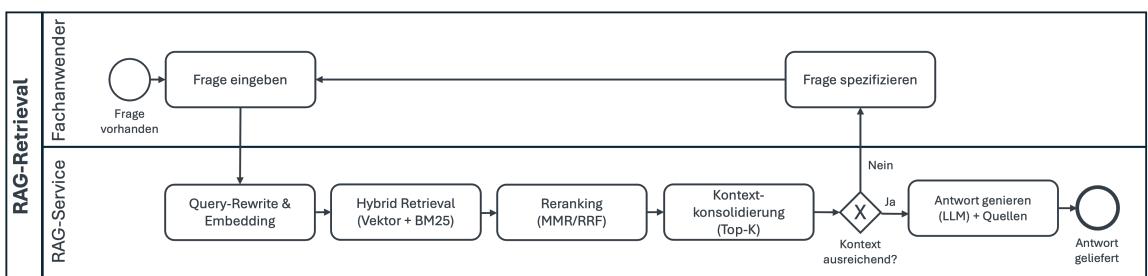
Quelle: eigene Abbildung

4.3.2.2 RAG-Retrieval – Kontextbasierte Antwortgenerierung

Das Retrieval nutzt die indexierten Daten, um natürlichsprachliche Fragen semantisch auszuwerten. Nach der Frageeingabe erfolgt ein Query-Rewrite und Embedding, gefolgt von Hybrid-Retrieval (Vektor + BM25), Reranking und Kontextkonsolidierung.

Abbildung 9 zeigt den Prozessablauf des RAG-Retrievals: Ein Gateway prüft, ob der Kontext ausreichend ist – andernfalls wird der Anwender zur Präzisierung aufgefordert. Bei ausreichendem Kontext generiert das LLM eine faktenbasierte Antwort mit Quellenangabe, die direkt im Self-Service-Portal angezeigt wird.

Abbildung 9 RAG-Retrievalprozess (BPMN-Modell)



Quelle: eigene Abbildung

4.4 Abgrenzung und Prototyp-Umfang (MVP)

Zur Überprüfung der in dieser Arbeit formulierten These, wie sich Workload-Automatisierung durch KI-gestützte Self-Service-Ansätze effizienter gestalten lässt, wird ein **Minimum Viable Product (MVP)** entwickelt.

Es dient als technischer Machbarkeitsnachweis und fokussiert sich auf die wesentlichen Funktionen des Konzepts, während komplexe Enterprise-Aspekte zunächst ausgeklammert bleiben.

4.4.1 Im MVP enthalten

Das MVP umfasst die folgenden Kernfunktionen:

- **LLM-gestützte Generierung von Konfigurationen:** Transformation natürlichsprachlicher Eingaben in strukturierte XML-Dateien unter Anwendung definierter Namenskonventionen und Standardwerte.
- **Automatische Validierung:** Prüfung der generierten Konfigurationen auf Wohlgeformtheit, Schema-Konformität (XSD) und logische Konsistenz, um fehlerhafte Streams frühzeitig zu erkennen.
- **RAG-basiertes Hilfesystem:** Kontextsensitiver Zugriff auf relevante Dokumentationen und Best-Practice-Beispiele zur Unterstützung der Fachanwender.
- **Self-Service-Frontend:** Dialogorientierte Benutzeroberfläche für Freitexteingaben und Formularerfassung mit unmittelbarem Feedback zu Pflichtfeldern, Plausibilitäten und Validierungsfehlern.

4.4.2 Nicht im MVP enthalten

Das MVP beschränkt sich bewusst auf jene Funktionen, die zum Nachweis der Machbarkeit und zur Validierung der Arbeitsthese erforderlich sind. Erweiterte, für einen produktiven Betrieb relevante Komponenten werden konzeptionell berücksichtigt, aber nicht implementiert. Dazu zählen:

- **Governance-Mechanismen:** Rollen- und Rechtemanagement, Freigabeprozesse und Audit-Trails.
- **Enterprise-Skalierung:** Hochverfügbarkeit, Lastverteilung und Mandantenfähigkeit.
- **Erweiterte Sicherheit:** Verschlüsselung ruhender Daten, MFA/SSO-Integration, Intrusion Detection.

- **Monitoring & Observability:** Umfassendes Logging, Tracing und Alerting auf Produktionsniveau.
- **Produktiv-Integrationen:** Direkte Kopplung an produktive Streamworks-Umgebungen.

4.4.2.1 Begründung der Abgrenzung

Die Konzentration auf diese Kernfunktionen ermöglicht eine **fokussierte Validierung der zentralen Hypothese** mit vertretbarem technischem und zeitlichem Aufwand. Erweiterte Enterprise-Funktionen werden zwar im Systemdesign berücksichtigt, aber aufgrund ihres Implementierungsaufwands bewusst in spätere Ausbaustufen verschoben.

Das MVP stellt damit eine praxisnahe und fokussierte Grundlage dar, um die Wirksamkeit KI-gestützter Self-Service-Mechanismen im Kontext der Workload-Automatisierung prototypisch nachzuweisen.

5 Implementierung

Dieses Kapitel beschreibt die technische Realisierung des in Kapitel 4 entworfenen Prototyps. Im Rahmen des Design-Science-Research-Ansatzes (DSR) stellt es die *Build-Phase* dar, in der die zuvor konzipierten Artefakte in eine funktionsfähige Anwendung überführt werden. Während Kapitel 4 die Architektur, Systemkomponenten und Prozesse konzeptionell definierte, konzentriert sich das vorliegende Kapitel auf die konkrete Umsetzung dieser Elemente in Form von Technologien, Code-Strukturen und Abläufen. Ziel ist es, die Machbarkeit der entworfenen Lösung nachvollziehbar zu demonstrieren und die getroffenen technischen Entscheidungen im Hinblick auf Anforderungen, Architekturprinzipien und wissenschaftliche Fundierung zu begründen.

5.1 Auswahl des Technologiestacks

Die Technologieauswahl orientiert sich an den in Kapitel 3 beschriebenen funktionalen und nicht-funktionalen Anforderungen sowie am architektonischen Entwurf aus Kapitel 4. Wesentliche Auswahlkriterien waren:

- **Self-Service-Eignung und Benutzerfreundlichkeit:** Unterstützung dialogbasierter Interaktionen mit direktem Feedback.
- **Validierung und Qualitätssicherung:** Schema-basierte Prüfung generierter Datenstrukturen.
- **Erweiterbarkeit und Wartbarkeit:** Modularer Aufbau mit getrennten Komponenten und standardisierten Schnittstellen.
- **Performance und Skalierbarkeit:** Asynchrone Verarbeitung und lokaler Betrieb ohne Cloud-Abhängigkeit.
- **Reproduzierbarkeit und Nachvollziehbarkeit:** Typisierte Modelle, deterministische Ergebnisse und transparente Fehlerbehandlung.

Die folgenden Abschnitte erläutern die gewählten Backend- und Frontend-Technologien und ihre Begründung.

5.1.1 Backend-Technologien

Das Backend bildet die zentrale Orchestrierungsschicht des Prototyps. Es koordiniert die Datenflüsse zwischen Frontend, Datenbank, KI-Diensten und Validierungslogik. Grundlage ist **Python** mit dem asynchronen Webframework **FastAPI**, das hohe Performance, Typensicherheit und automatische API-Dokumentation bietet.

Die **Datenvalidierung** erfolgt über **Pydantic**, eine Bibliothek zur Datenvalidierung über Python Type Hints. Dadurch werden alle Request- und Response-Objekte typisiert und konsistent geprüft.

Als **persistente Datenspeicherlösung** dient **PostgreSQL**, abstrahiert über **SQLAlchemy** mit asynchronem ORM-Zugriff und strukturiertem Schema-Management.

Für semantische Abfragen und die **RAG-Funktionalität** wird die **Vektordatenbank Qdrant** mit lokaler Ausführung und REST/gRPC-Schnittstellen eingesetzt. Die **Sprachverarbeitung** basiert auf **Mistral 7B Instruct** als LLM, während **LangExtract** zur schema-basierten Parameterextraktion verwendet wird.

Die deterministische **XML-Erzeugung** erfolgt über die Template-Engine **Jinja2**.

Eine Übersicht der Backend-Technologien findet sich im Anhang (vgl. Tabelle 11).

5.1.2 Frontend-Technologien

Das Frontend basiert auf modernen, benutzerfreundlichen Webtechnologien. Durch die komponentenbasierte Struktur von **React** folgt das System den von Martin (2017) beschriebenen Prinzipien der klaren Verantwortlichkeit (*Single Responsibility Principle*). Jede Komponente ist unabhängig testbar und erweiterbar, was die Wartbarkeit langfristig sichert⁷⁵.

Kern des Systems ist **Next.js**, das serverseitiges Rendering, dateibasiertes Routing und API-Routen in einer Codebasis ermöglicht. **React** dient als Komponentenbibliothek, ergänzt durch **TypeScript** für strikte Typisierung und hohe Codequalität.

Styling, Zustandsverwaltung und Animationen basieren auf **TailwindCSS**, **React Query/Zustand** und **Framer Motion**. **Radix UI** und **Headless UI** stellen barrierefreie Grundkomponenten bereit, während der **Monaco Editor** die Bearbeitung und Vorschau von XML-Konfigurationen ermöglicht.

Eine detaillierte Übersicht der Frontend-Technologien befindet sich im Anhang (vgl. Tabelle 12).

5.2 Backend-Implementierung

Das Backend bildet das technische Rückgrat des Prototyps und stellt die zentrale Logik für Automatisierung, Validierung und Wissensabruft bereit. Aufbau und Struktur

⁷⁵ Robert C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design (Prentice Hall, 2018), S. 61-69

orientieren sich an den in Kapitel 4 definierten Architekturprinzipien und setzen diese nach dem Konzept der **Clean Architecture** nach *Martin* (2017) um⁷⁶. Ziel ist eine skalierbare, wartbare Systembasis, die fachliche Logik von technischen Details entkoppelt und so Erweiterbarkeit und Testbarkeit sicherstellt.

5.2.1 Architekturprinzipien

Die Architektur des Backends ist schichtenbasiert aufgebaut und trennt Geschäftslogik, Infrastruktur und Schnittstellen, um eine modulare, wartbare und testbare Struktur sicherzustellen.

- **Service-Orientierung:** Fachlogik ist in eigenständige Services gekapselt (z. B. XML-Service, Validation-Service, RAG-Service). Dieser Ansatz folgt dem Paradiagramm der Service Oriented Architecture (SOA), bei dem Funktionalitäten als eigenständige, zur Laufzeit nutzbare Dienste bereitgestellt werden⁷⁷. Das reduziert Seiteneffekte und erhöht die Wartbarkeit.
- **Dependency Injection:** Ein zentraler Container verwaltet Abhängigkeiten und ermöglicht Mocking für Unit-Tests. Dadurch wird die Testbarkeit erhöht – ein wesentliches Kriterium im Design-Science-Research-Ansatz, bei dem Artefakte evaluiert und iterativ verbessert werden (vgl. Hevner et al., 2004).
- **Asynchrones Design:** Alle I/O-Operationen (PostgreSQL, Qdrant, LLM-Aufrufe) nutzen `async/await`, um eine nicht-blockierende Verarbeitung zu ermöglichen und gleichzeitige Anfragen effizient zu handhaben.
- **Schema-basierte Validierung:** Pydantic-Modelle erzwingen typsichere Datenflüsse zwischen Frontend und Backend und reduzieren Integrationsfehler.

Diese Prinzipien sorgen dafür, dass die technische Umsetzung nicht nur funktional, sondern auch wissenschaftlich fundiert ist: Sie stützt die Evaluierbarkeit und Reproduzierbarkeit der Ergebnisse (Kriterium der „Rigorosität“ im DSR-Framework).

5.2.2 Projektstruktur

Die Implementierung folgt einer klaren Modultrennung, die den Schichten der **Clean Architecture** entspricht. Die Aufteilung nach Verantwortlichkeiten orientiert sich am von *Martin* (2017) beschriebenen **Open-Closed-Principle**, wonach Software offen für Erweiterung, aber geschlossen für Veränderung sein soll.

⁷⁶ Vgl. Robert C. Martin (2018), S. 303f

⁷⁷ Vgl. Meyer (2018), S. 96

Neue Services – etwa für zusätzliche Jobtypen – können integriert werden, ohne bestehende Kernmodule anzupassen.⁷⁸.

Listing 9 Backend-Projektstruktur

```
backend/
├── main.py          # API-Gateway und Lifespan-Management
├── config.py        # Zentrale Konfigurationsdateien (Umgebungsvariablen)
└── database.py      # Datenbank-Engine und Session Management

└── routers/          # API-Endpunkte
    ├── xml_streams.py
    ├── langextract_chat.py
    ├── sessions.py
    └── health.py

└── services/         # Geschäftslogik in eigenständigen Modulen
    ├── ai/             # KI-Integration und LLM-Services
    ├── xml_generation/ # Template Engine und Mapper
    └── rag_service.py

└── models/           # Datenbank- und Pydantic-Modelle
└── schemas/          # Validierungsregeln und XSD-Schemas
└── templates/         # XML-Templates (Jinja2)
└── Dockerfile         # Containerdefinition
```

Quelle: eigene Abbildung

Listing 9 zeigt die Backend-Projektstruktur mit getrennten Bereichen für API-Routen, Services, Modelle und Templates.

5.2.3 API-Gateway und Dependency Injection

Das **API-Gateway** (*main.py*) initialisiert die Anwendung, bindet Middleware (CORS, Error-Handler) und registriert alle Routen nach Funktionsdomänen (z. B. *xml_router*, *rag_router*, *validation_router*).

Über den zentralen **Dependency-Injection-Container** (*core/container.py*) werden Services wie *LangExtractService*, *QdrantRAGService* und *ValidationOrchestrator* instanziert und an die Router übergeben.

Dies ermöglicht lose Kopplung und den Einsatz simulierter Testinstanzen während der Evaluation.

5.3 Frontend-Implementierung

Das Frontend bildet die Präsentationsschicht des Prototyps und stellt die Schnittstelle zwischen Fachanwender und Backend-Services dar. Es überführt die in Kapitel 4

⁷⁸ Vgl. Robert C. Martin (2018), S. 69ff

beschriebenen Architekturprinzipien in eine nutzerzentrierte, interaktive Oberfläche, die die komplexen Funktionen der Workload-Automatisierung in einfache Bedienabläufe übersetzt. Ziel ist es, die in Kapitel 3 identifizierten Einstiegshürden zu reduzieren und Self-Service-Funktionen bereitzustellen, die technische Komplexität abstrahieren und unmittelbares Feedback bieten.

5.3.1 Framework und Architektur

Das Frontend basiert auf **Next.js** und **React**, ergänzt durch **TypeScript** zur strikten Typisierung und verbesserten Codequalität. **Next.js** übernimmt Routing und serverseitiges Rendering (SSR) für schnelle Darstellung und klare Trennung zwischen Seiten und Inhaltenkomponenten, was die Wartbarkeit fördert.

Die Komponentenarchitektur folgt den in Kapitel 5.2 beschriebenen Prinzipien der Modularität und losen Kopplung: Jede Komponente erfüllt eine klar abgegrenzte Aufgabe und kommuniziert über definierte Schnittstellen mit anderen Modulen. So bleibt das System leicht erweiterbar, etwa durch neue Dialoge oder Funktionen, ohne bestehende Teile anzupassen.

Das Styling erfolgt mit **Tailwind CSS**, das einen Utility-First-Ansatz nutzt und eine konsistente Gestaltung ohne separate CSS-Dateien ermöglicht. **Radix UI** und **Headless UI** liefern barrierefreie Grundkomponenten und trennen Design von Interaktionslogik, was eine klare Abgrenzung zwischen Präsentation und Funktionalität unterstützt.

5.3.2 Architekturprinzipien und Struktur

Das Frontend folgt einer klaren Struktur, die sich an den in Kapitel 4 definierten Schichten orientiert und eine Trennung von Darstellung, Daten und Logik ermöglicht.

Listing 10 Frontend-Projektstruktur

```
frontend/src/
├── app/
│   └── layout.tsx      # Next.js Pages (dokumentenverwaltung, xml, chat)
├── components/
│   └── ui/              # UI-Komponenten (ParameterOverview, XMLPreview, Chat, Status)
├── services/           # API-Clients (Fetch-Wrapper, XML-Streams)
├── types/               # TypeScript Typdefinitionen (API-Types)
└── lib/                 # Hilfsfunktionen (Utilities, className merge)
```

Quelle: eigene Abbildung

Diese Gliederung stellt sicher, dass Geschäftslogik (*Hooks, API-Clients*) und Präsentationslogik (*Components, Pages*) strikt getrennt bleiben.

Funktionen wie Stream-Erstellung oder RAG-Abfragen können dadurch unabhängig erweitert werden – entsprechend dem in Kapitel 4 beschriebenen Prinzip der Erweiterbarkeit bei stabiler Kernlogik.

5.3.3 State Management und API-Kommunikation

Die Zustandsverwaltung trennt **Server- und Client-State**:

- **React Query** verwaltet Server-Daten mit optimiertem Caching, Synchronisierung und automatischem Refetching.
- **Zustand** steuert UI-bezogene Zustände wie Sidebar-Sichtbarkeit oder Editor-Einstellungen.

Diese Aufteilung ermöglicht reaktive Oberflächen und ein robustes Verhalten bei asynchronen Operationen.

Die Kommunikation mit dem Backend erfolgt über einen modularen **Fetch-Wrapper**, der Fehlerbehandlung, Authentifizierung und Header-Verwaltung zentralisiert. Spezifische **API-Clients** (z. B. für XML-Streams oder RAG-Anfragen) bieten typisierte Methoden, um Integrationsfehler zu vermeiden und die Entwicklung zu vereinfachen.

5.4 Integration der Kernfunktionen

Dieser Abschnitt führt die zentralen Funktionskomponenten des Prototyps zusammen: die automatisierte Stream-Erstellung, die RAG-gestützte Wissensintegration und die Validierung durch Schema- und Pflichtparameterprüfungen. Diese Elemente greifen ineinander, um Fachanwender bei der Erstellung valider Stream-Konfigurationen zu unterstützen, Wissenslücken zu schließen und die technische Qualität der Ergebnisse sicherzustellen.

5.4.1 Automatisierte Stream-Erstellung mit LangExtract

5.4.1.1 Einführung und Zielsetzung

LangExtract ist die zentrale Komponente der automatisierten Stream-Erstellung. Das System ermöglicht es Fachanwendern, Anforderungen in natürlicher Sprache zu formulieren und diese in strukturierte, valide Streamworks-Parameter zu überführen. Dadurch wird der bisher manuelle Prozess zu einer teil- oder vollautomatisierten Pipeline weiterentwickelt. Ziel ist es, eine Brücke zwischen intuitiver Bedienbarkeit und der Erzeugung formal korrekter XML-Konfigurationen zu schlagen.

5.4.1.2 Komponenten

- **LangExtractInterface (Frontend):** Chat-basierte Eingabeoberfläche mit Parameterliste.
- **LangExtractService (Backend):** Orchestrert Kommunikation mit der LangExtract-API, verwaltet Sessions und führt schema-basierte Extraktionen aus.
- **JobTypeDetector:** Regelbasierte Klassifikation des Jobtyps (z. B. FILE_TRANSFER, SAP) mit Confidence-Werten.
- **LangExtract API (Google):** Führt KI-gestützte Schemaextraktion durch und mappt Freitext auf vordefinierte Pydantic-Schemas.
- **SessionService:** Persistiert Sessions und Dialoghistorien in PostgreSQL.
- **ParameterClassifier:** Trennt globale von job-spezifischen Parametern, prüft Pflichtfelder und übergibt die validierten Werte an die nachfolgenden Verarbeitungsschritte.

5.4.1.3 Funktionsweise von LangExtract

LangExtract kombiniert regelbasierte Logik mit KI-gestützter Schemaextraktion nach dem Prinzip des *Few-Shot-Learnings*. In den Prompts werden Beispieldialoge eingebettet, um Benutzereingaben stabil zuzuordnen und konsistente Ergebnisse zu erzeugen.

Die von Google entwickelte Bibliothek nutzt Large Language Models, um strukturierte Informationen aus Texten gemäß vordefinierter Instruktionen zu extrahieren und so „reliable structured outputs“ zu liefern⁷⁹.

Im Prototyp erfolgt die Umsetzung in drei Schritten:

1. **Klassifikation:** Der *JobTypeDetector* erkennt den Automatisierungstyp (z. B. FILE_TRANSFER, SAP) anhand typischer Schlüsselbegriffe.
2. **Extraktion:** Die *LangExtract API* parst den Eingabetext gegen ein definiertes Schema und verwendet dabei Few-Shot-Beispiele als Referenz.
3. **Validierung:** Der *ParameterClassifier* teilt die Ergebnisse in Stream- und Job-Parameter und prüft sie auf Vollständigkeit.

Ein konkretes Beispiel verdeutlicht den Ablauf:

⁷⁹ Vgl. Google (2025). *google/langextract: A Python library for extracting structured information from unstructured text using LLMs with precise source grounding and interactive visualization*

Listing 11 Beispielhafte Parameterextraktion durch LangExtract

```
Beispiel:
Eine Eingabe wie: „Ich brauche einen täglichen Datentransfer von Server QuellServer
nach Server ZielServer, alle CSV-Dateien aus /data/export sollen nach /data/import
kopiert werden.“
{
    "job_type": "FILE_TRANSFER",
    "source_agent": "QuellServer",
    "target_agent": "ZielServer",
    "source_path": "/data/export/*.csv",
    "target_path": "/data/import/",
    "schedule": "täglich"
}
```

Quelle: eigene Abbildung

Ein Beispiel der schema-basierten Extraktion ist in Listing 11 dargestellt. Das aktuelle Schema erkennt rund 20 Parameter aus natürlichsprachlichen Eingaben – darunter allgemeine Stream-Eigenschaften (*StreamName*, *Agent*, *StartTime*, *SchedulingRequiredFlag*) sowie job-spezifische Merkmale (z. B. *FILE_TRANSFER*, *SAP*). Durch die Kombination aus regelbasierter Erkennung und LLM-gestützter Schemaextraktion können selbst komplexe Angaben, etwa Mandanten, Programme oder Pfade, konsistent zugeordnet werden. Eine vollständige Übersicht der Parameter befindet sich im Anhang 10.3.

5.4.1.4 Prozessablauf

Der technische Ablauf der automatisierten Stream-Erstellung folgt einer klar definierten Prozesskette, in der alle beteiligten Komponenten synchron zusammenarbeiten:

1. **Session-Initialisierung:** Beim Öffnen des LangExtract-Interfaces wird eine neue Session mit eindeutiger UUID angelegt und in der Datenbank gespeichert.
2. **Eingabe & Job-Typ-Erkennung:** Der Benutzer formuliert seine Anforderung in natürlicher Sprache; der *JobTypeDetector* klassifiziert den Typ mit Confidence-Werten.
3. **Schemaextraktion:** Die *LangExtract API* parst den Text gegen ein Pydantic-Schema und generiert strukturierte Parameter
4. **Parameterklassifikation & Validierung:** Der *ParameterClassifier* trennt globale und job-spezifische Felder, prüft Pflichtparameter und gibt Feedback an das Frontend.
5. **Iterative Vervollständigung:** Fehlende Angaben werden ergänzt, bis alle Pflichtfelder vollständig validiert sind.
6. **Manuelle Nachbearbeitung:** Optional können Parameter im Frontend bearbeitet werden; Änderungen werden serverseitig geprüft.

5.4.1.5 Design-Entscheidungen

- **Session-basiert:** Ermöglicht Unterbrechungen und Wiederaufnahmen.
- **Hybridansatz:** Kombination aus regelbasierter Typ-Erkennung und KI-gestützter Extraktion.
- **Transparenz:** Confidence-Werte werden im Frontend angezeigt, um Nachvollziehbarkeit zu gewährleisten.

5.4.2 RAG-gestütztes Hilfesystem

5.4.2.1 Einführung und Zielsetzung

Das RAG-System (Retrieval-Augmented Generation) erweitert die automatisierte Stream-Erstellung um ein kontextsensitives Hilfesystem. Es adressiert das Problem fehlenden Domänenwissens bei Fachanwendern, indem es bestehende Dokumentationen in natürlicher Sprache zugänglich macht. Ziel ist es, faktenbasierte Antworten mit Quellenangaben bereitzustellen und so sowohl die Benutzerfreundlichkeit als auch die Qualität der Eingaben zu verbessern.

5.4.2.2 Komponenten

- **RAGChatInterface (Frontend):** Chatbasierte Oberfläche mit Anzeige von Antworten, Quellen und Confidence-Werten.
- **QdrantRAGService (Backend-Orchestrator):** Orchstriert Indexierung, semantisches Retrieval und Antwortgenerierung.
- **Qdrant Vector DB:** Speichert hochdimensionale Embeddings (768 D) und ermöglicht semantische Suche per Cosine Similarity.
- **EmbeddingService:** Erzeugt Embeddings für Dokumente und Benutzeranfragen (lokal mit *EmbeddingGemma-300m*).
- **DocumentProcessor:** Zerlegt Dokumente in 512-Token-Chunks, analysiert Layouts und extrahiert Metadaten.
- **HybridRetriever:** Kombiniert Vektor- und Keyword-Suche, fusioniert Ergebnisse mittels *Reciprocal Rank Fusion (RRF)*.
- **AnswerGenerator:** Generiert kontextbasierte Antworten und ergänzt sie um nachvollziehbare Quellenangaben.

5.4.2.3 Funktionsweise und Prozessablauf

1. **Dokumenten-Indexierung:** Beim Upload werden die Dokumente eingelesen, in semantische Chunks zerlegt und vektorisiert. Die Embeddings und

zugehörigen Metadaten (Dokumentname, Abschnitt, Seite) werden in **Qdrant** gespeichert.

2. **Frage-Eingabe:** Fachanwender formulieren ihre Anfrage in natürlicher Sprache (z. B. „Wie konfiguriere ich einen SAP-Export?“).
3. **Semantisches Retrieval:** Die Anfrage wird vektorisiert und in einem hybriden Verfahren (Vektor- + Keyword-Suche) mit den Dokument-Embeddings verglichen. Die Ergebnisse werden per *Reciprocal Rank Fusion (RRF)* zu einem Gesamtranking zusammengeführt.
4. **Kontext-Konsolidierung:** Die relevantesten Textabschnitte (Top 5) werden zu einem Kontextblock zusammengeführt und mit Quellenangaben versehen.
5. **Antwort-Generierung und Anzeige:** Das LLM erzeugt auf Basis des Kontexts eine faktenbasierte Antwort mit Quellenzitierung. Diese wird im Chat-Interface angezeigt; Confidence-Werte visualisieren die Ergebnisqualität.

5.4.2.4 Dokumentenverwaltung

Die Qualität der Antworten hängt unmittelbar von den indexierten Quellen ab. Daher wurde eine Verwaltungsschicht implementiert, die Upload, Konvertierung und Aktualisierung der Dokumente automatisiert.

- **Upload:** Neue Dateien (PDF, DOCX, Markdown) können über das Admin-Interface oder direkt im *documentation*-Verzeichnis hinzugefügt werden.
- **Konvertierung:** Mithilfe von **Docling** (Google Research) werden Dokumente vor der Indexierung in strukturierte Textrepräsentationen überführt. Dabei werden Layouts, Tabellen und Überschriften erkannt, um konsistente semantische Chunks zu erzeugen.
- **Versionierung & Aktualisierung:** Dateien erhalten Hash- und Zeitstempel; Änderungen werden beim Systemstart erkannt und automatisch neu indexiert.
- **Metadatenmanagement:** Neben Dateinamen werden Seiten, Abschnitte und Änderungsdatum erfasst.
- **Strukturierung:** Dokumente werden thematisch (z. B. SAP, Fehlermeldungen, Agentenverwaltung) kategorisiert, um das Retrieval zu optimieren.

5.4.2.5 Design-Entscheidungen

- **Automatisierung:** Dokumentänderungen werden erkannt und neu indexiert – ohne manuellen Eingriff.

- **Nachvollziehbarkeit:** Hashes und Metadaten gewährleisten eindeutige Identifikation jeder Quelle.
- **Skalierbarkeit:** Die Architektur erlaubt die Verarbeitung mehrerer hundert Dokumente ohne spürbaren Performanceverlust.

5.4.2.6 Kontextintegration und Zugriff

Das RAG-System ist nicht nur über das eigenständige **RAGChatInterface** nutzbar(vgl. abbildung x), sondern auch systemweit kontextsensitiv integriert. Über ein dauerhaft sichtbares Icon (vgl. abbildung x) kann der Assistent aus jeder Ansicht des Prototyps heraus aufgerufen werden.

Beim Öffnen erscheint ein kompaktes Chatfenster, das automatisch das aktuelle Kontextfenster erkennt – etwa ob sich der Benutzer in der Stream-Erstellung oder im Parameter-Editor. Dadurch können Fragen gezielt im jeweiligen Arbeitsbereich beantwortet werden, ohne den Arbeitsfluss zu unterbrechen (vgl. abbildung x).

5.5 Validerungssystem

5.5.1.1 Einführung und Zielsetzung

Die Validierung dient als durchgängige, mehrstufige Qualitätssicherung entlang der gesamten Pipeline. Frühe Prüfungen erfolgen vor der XML-Generierung (Pflichtfelder, Typ- und Formatvalidierung), spätere nach dem Rendering (XSD- und Semantikchecks). Ein Export ist erst zulässig, wenn alle Stufen erfolgreich sind. Ziel ist schnelles, verständliches Feedback für Fachanwender ohne technisches Spezialwissen – inklusive klarer Blocker, Warnungen und Korrekturhinweise.

5.5.1.2 Komponenten

- **RequiredParameterValidator:** Prüft jobtypspezifische Pflichtfelder auf Präsenz und Werte
- **PydanticValidator:** Führt Typ-, Format- und Constraint-Prüfungen gemäß Schema durch.
- **XMLSchemaValidator:** Kontrolliert Wohlgeformtheit und XSD-Konformität (lxml) nach der Generierung.
- **SemanticValidator:** Überprüft logische Konsistenz (z. B. Pfad-Ungleichheit, Zeitlogik, Agenten-Existenz).

- **ValidationOrchestrator:** Steuert Reihenfolge, Aggregation und Priorisierung der Ergebnisse.
- **ErrorFormatter:** Übersetzt technische Fehler in klare, deutschsprachige Meldungen mit Beispielen.

5.5.1.3 Validierungsstufen

1. **Pflichtparameterprüfung:** Fehlende Felder (z. B. *stream_name*, *start_time*) verhindern die Generierung.
2. **Typ- und Formatprüfung:** erkennt ungültige Werte wie falsche Zeitformate oder nicht-numerische SAP-Clients.
3. **XML-Schema-Prüfung:** Nach der Generierung wird das XML gegen das XSD geprüft (Elemente, Attribute, Datentypen, Kardinalität).
4. **Semantische Prüfung:** Konsistenzregeln prüfen Zeitlogik, Pfad-Ungleichheit und Agenten-Existenz.
5. **Fehleraggregation:** Ergebnisse werden nach Schweregrad (Blocker / Warnung) gewichtet und an das Frontend übergeben.

5.5.1.4 Besonderheiten und Design-Entscheidungen

- **Multi-Stage-Ansatz:** Frühzeitige Abbrüche sparen Rechenzeit und liefern direktes Feedback.
- **Blocker vs. Warnungen:** Kritische Fehler verhindern den Export, während Warnungen nur Hinweise geben.
- **Deterministische Fehlermeldungen:** Statische Mapping-Tabellen gewährleisten reproduzierbare Rückmeldungen und konsistente User Experience.

Die Implementierung zeigt, dass sich die in Kapitel 4 konzipierte Architektur mit modernen Open-Source-Technologien effizient umsetzen lässt. Durch die Kombination aus LLM-gestützter XML-Parametrisierung, RAG-basierter Wissensintegration und mehrstufiger Validierung werden die Kernanforderungen der DSR-Build-Phase erfüllt. Damit entsteht eine fundierte Basis, um in Kapitel 6 die Funktionalität, Effizienz und Benutzerfreundlichkeit des Prototyps systematisch zu evaluieren.

6 Evaluation und Ergebnisse

Die Evaluation dient dazu, die Wirksamkeit des entwickelten Artefakts zu bestätigen und die Forschungsfrage zu beantworten, ob KI-gestützte Self-Service-Mechanismen in Streamworks eine effizientere und konsistenter Workload-Automatisierung ermöglichen. Gemäß dem Design-Science-Research-Ansatz erfolgt die Bewertung sowohl analytisch als auch empirisch-technisch, um die Leistungsfähigkeit und Zuverlässigkeit des Prototyps zu belegen.

6.1 Zielsetzung der Evaluation

Auf formale Usability- oder Akzeptanztests mit Endanwendern wurde bewusst verzichtet, da der entwickelte Prototyp primär als technischer Machbarkeitsnachweis konzipiert ist. Ziel der Evaluation ist die Überprüfung seiner Leistungsfähigkeit in Bezug auf **funktionale Korrektheit** und **Antwortqualität**. Gemäß dem Design-Science-Research-Ansatz erfolgt die Bewertung auf zwei Ebenen:

1. Analyse der syntaktischen, strukturellen und semantischen Validität der generierten XML-Dateien und
2. Messung der Faktentreue und Relevanz der RAG-Antworten anhand objektiver Qualitätsmetriken.

Die Ergebnisse bilden die empirische Grundlage für die kritische Reflexion in Kapitel 7.

6.2 Evaluierung der XML-Generierung

Die Evaluierung der XML-Generierung untersucht, inwieweit der entwickelte LLM/XML-Service in der Lage ist, natürlichsprachliche Eingaben in valide Streamworks-Konfigurationen zu überführen. Im Fokus stehen Wohlgeformtheit, Schema-Konformität und semantische Konsistenz der erzeugten Dateien.

6.2.1 Methodisches Vorgehen

Die Prüfung erfolgte mittels einer automatisierten Test-Suite in Python mit 50 repräsentativen Testfällen. Für jede Eingabe wurde eine XML-Datei generiert und anschließend automatisch gegen das offizielle Streamworks-XSD-Schema validiert. Fehler wurden protokolliert und nach Fehlertypen (syntaktisch, strukturell, semantisch) klassifiziert.

Die Testfälle wurden in drei Komplexitätsstufen unterteilt (einfach, mittel, komplex; vgl. Tabelle 13 im Anhang). Zur Ergänzung wurde eine qualitative Fehleranalyse

durchgeführt, um Abweichungen nach Fehlerarten (z. B. fehlende End-Tags, semantische Fehlinterpretationen, Namensverletzungen) zu klassifizieren.

Die Ergebnisse bilden die empirische Basis für die Bewertung der technischen Korrektheit und Zuverlässigkeit des Prototyps.

6.2.2 Bewertung der XML-Struktur

Die Qualität eines automatisch erzeugten XML-Dokuments bemisst sich in erster Linie an seiner Wohlgeformtheit, Schema-Konformität und Validität. Zur Bewertung wurde die Python-Bibliothek `lxml` genutzt. Die Wahl fiel auf dieses Werkzeug, da es als Python-Binding für die hochoptimierten C-Bibliotheken `libxml2` und `libxslt` eine hohe Verarbeitungsgeschwindigkeit und Speichereffizienz gewährleistet⁸⁰. Entscheidend für die Qualitätssicherung im Rahmen dieser Arbeit ist zudem die umfassende Unterstützung für standardkonforme Validierungsmechanismen, insbesondere die Prüfung gegen eine XML Schema Definition (XSD). Dies ermöglicht eine robuste und performante Überprüfung der syntaktischen und strukturellen Korrektheit der generierten Konfigurationen mit minimalem Overhead.

6.2.2.1 Wohlgeformtheit

Ein Dokument gilt als wohlgeformt, wenn alle Elemente korrekt geschachtelt und geschlossen sind und keine inkonsistenten Tags vorhanden sind.

Die Validierung erfolgte durch den Parser:

```
from lxml import etree
xml_doc = etree.parse(file)
schema = etree.XMLSchema(file='streamworks_stream.xsd')
schema.assertValid(xml_doc)
```

Listing 12 Schema-Validierung der XML-Dateien mit der Bibliothek `lxml`

Fehler werden im Attribut `error_log` erfasst und automatisch kategorisiert.

6.2.2.2 Schema-Konformität (XSD-Validation)

Neben der reinen Syntax prüft `lxml` die inhaltliche Struktur gegen die definierte XML-Schema-Definition (XSD). Diese Validierung stellt sicher, dass alle Pflichtelemente (z. B. `<streamName>`, `<AgentDetail>`) vorhanden sind und Datentypen (z. B. `xs:string`,

⁸⁰ Vgl. `lxml` project (2025). *lxml: The lxml XML toolkit for Python*.

`xs:int`) eingehalten sind. Die Schema-Konformität ist entscheidend für den Produktiveinsatz in Streamworks, da das System nur XSD valide XMLs akzeptiert.

6.2.2.3 Validität und Semantik

Ein XML muss nicht nur formal korrekt sein, sondern inhaltlich auch konsistent – beispielsweise dürfen Quell- und Zielagent nicht identisch sein und Zeitangaben müssen logisch plausibel bleiben. Diese Regeln wurden in einer zusätzlichen Validierungsschicht (Pydantic-Validator) überprüft, um semantische Fehler zu erfassen.

Damit wurden drei Qualitätsdimensionen unterschieden:

Tabelle 2 Qualitätsdimension und Prüfmethoden der XML-Validierung

Dimension	Beschreibung	Werkzeug / Methode
Syntaktische Validität	Wohlgeformtheit der XML-Tags	lxml Parser
Strukturelle Validität	XSD-Schema-Prüfung	lxml XMLSchema()
Semantische Konsistenz	Logische Regeln und Plausibilitäten	Pydantic Validator

6.2.2.4 Metriken und Ergebnisse

Die quantitative Analyse umfasste 50 Testfälle (25 einfache, 15 mittlere, 10 komplexe, vgl. Tabelle 13). Die Bewertung erfolgte anhand der folgenden Kennzahlen:

Tabelle 3 Bewertungsmetriken und Formeln

Metrik	Beschreibung	Formel
Validitätsrate (VR)	Misst den Anteil der XSD-konformen Dokumente.	$VR = \frac{n_{valide}}{n_{gesamt}} \times 100$
Vollständigkeitsrate (VR_f)	Bewertet, wie viele Pflichtparameter erfolgreich extrahiert und gesetzt wurden.	$VR_f = \frac{p_{korrektgesetzt}}{p_{erforderlich}} \times 100$
Konsistenzrate (KR)	Anteil der XMLs ohne semantische Widersprüche (z. B. doppelte Job-Namen, gleiche Quell-/Zielagenten).	$KR = \frac{n_{fehlerfreikonsistent}}{n_{gesamt}} \times 100$
Fehlerquote (FQ)	Anteil der fehlerhaften XMLs, unabhängig vom Fehlertyp.	$FQ = \frac{n_{fehlerhaft}}{n_{gesamt}} \times 100$

Ein exemplarischer Testfall ist in Tabelle 14 aufgeführt und veranschaulicht die konkrete Anwendung des Evaluationsverfahrens anhand eines typischen SAP-Jobs.

Die Ergebnisse der Evaluierung (vgl. Tabelle 15 im Anhang) belegen, dass der entwickelte LLM/XML-Service in der Lage ist, natürlichsprachliche Eingaben mit hoher

formaler Präzision in valide Streamworks-Konfigurationen zu überführen. Mit einer durchschnittlichen **Validitätsrate von 82 %** und einer **Konsistenzrate von 81 %** erreicht das System ein für einen Prototyp stabil hohes Qualitätsniveau. Die Ixml-basierte XSD-Validierung erwies sich als wirksamer Mechanismus zur Erkennung syntaktischer Abweichungen, während die zusätzliche Pydantic-Schicht semantische Inkonsistenzen frühzeitig identifizierte.

6.3 Evaluierung des RAG-Systems

Die Evaluierung des entwickelten RAG-Hilfesystems zielt darauf ab, die Leistungsfähigkeit des kombinierten Retrieval- und Generierungsansatzes hinsichtlich der Antwortqualität, der Kontextrelevanz und der Faktentreue zu beurteilen. Grundlage bilden die in Kapitel 5.4 beschriebenen Komponenten aus semantischem Retriever, Hybrid-Search-Mechanismus und LLM-basiertem Antwort-Generator. Ziel ist es, die technische Qualität des Hilfesystems anhand objektiver, reproduzierbarer Kennzahlen zu bewerten.

6.3.1 Methodisches Vorgehen

Zur Evaluierung wurden 50 Streamworks-Dokumente aus zehn Themenbereichen (z. B. Jobkonfiguration, Agentenkommunikation, Fehlerbehandlung) indexiert. Für jede Kategorie wurden fünf praxisnahe Fachfragen definiert, die typische Nutzeranfragen abbilden. Das System musste jeweils die relevanten Dokumente abrufen und eine quellen-gestützte Antwort erzeugen.

Zur quantitativen Bewertung wurde ein kombiniertes Messmodell aus dem **TRIAD-Framework⁸¹** und den **RAGAS-Metriken⁸²** verwendet. Beide Ansätze ergänzen sich und ermöglichen eine differenzierte Betrachtung der Leistungsfähigkeit auf Retrieval- und Generierungsebene.

6.3.1.1 TRIAD-Framework

Das **TRIAD-Framework** gliedert die Leistungsbewertung eines RAG-Systems in drei Hauptdimensionen:

- **T – Treue (Faithfulness):** faktische Korrektheit der Antwort im Kontext der abgerufenen Quellen.

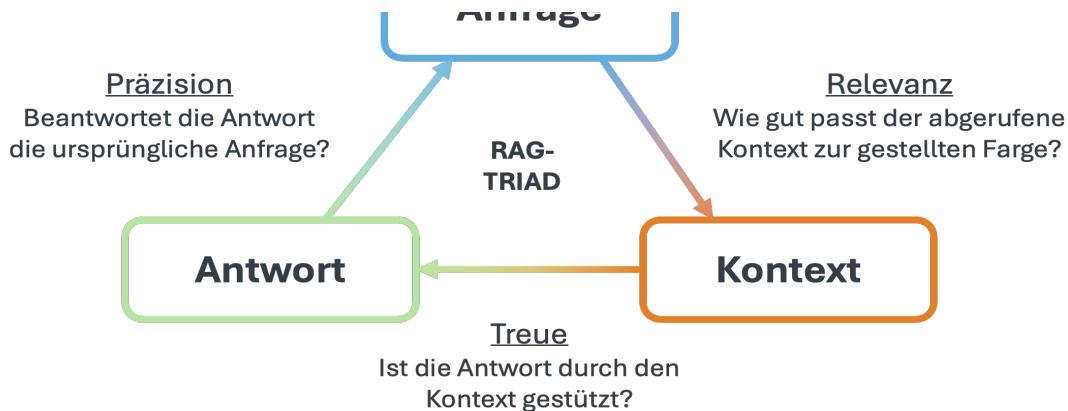
⁸¹ Vgl. MyScale (2025). *Der ultimative Leitfaden zur Bewertung von RAG-Systemkomponenten*

⁸² Vgl. Es et al., (2025): Ragas: Automated Evaluation of Retrieval Augmented Generation

- **R – Relevanz (Relevance):** thematische und inhaltliche Passgenauigkeit zur Benutzerfrage.
- **IAD – Information Accuracy & Diversity:** Abdeckung der relevanten Informationsvielfalt bei minimierter Redundanz.

Abbildung 10 stellt die drei Dimensionen des TRIAD-Modells – Treue, Relevanz und Information Accuracy & Diversity – schematisch dar und verdeutlicht das Zusammenspiel von Retrieval, Kontext und Antwort.

Abbildung 10 Das TRIAD-Dreieck der RAG-Bewertung



Zur Bewertung der Retrieval-Leistung werden die in Tabelle 4 dargestellten etablierten Information-Retrieval-Metriken herangezogen. Metriken wie Precision@k, Recall@k,

Quelle: eigene Abbildung in Anlehnung an https://www.trulens.org/getting_started/core_concepts/rag_triad/

F1-Score, Mean Reciprocal Rank (MRR) und Mean Average Precision (MAP) quantifizieren die Genauigkeit und Vollständigkeit des Systems auf der Retriever-Ebene. Im Kontext des TRIAD-Ansatzes helfen diese Metriken, die Dimension der **Relevanz** objektiv zu messen.

Tabelle 4 Bewertungsmetriken für die Retrieval-Komponente

Metrik	Beschreibung	Formel
Precision@k	Anteil relevanter Dokumente unter den Top-k-Ergebnissen.	$Precision@k = \frac{ Rel_k }{k}$
Recall@k	Anteil aller relevanten Dokumente, die erfolgreich abgerufen wurden.	$Recall@k = \frac{ Rel_k }{ Rel_{gesamt} }$
F1-Score	Harmonisches Mittel aus Precision und Recall.	$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$
MRR (Mean Reciprocal Rank)	Durchschnittliche Position des ersten relevanten Dokuments in der Rangliste.	$MRR = \frac{1}{Q} \sum_{i=1}^Q \frac{1}{rank_i}$

Mean Average Precision (MAP)	Mittlere Präzision über alle Abfragen hinweg.	$MAP = \frac{1}{Q} \sum_{i=1}^Q AvgPrecision_i$
-------------------------------------	---	---

6.3.1.2 RAGAS-Framework

Ergänzend zur Retrieval-Analyse wurde die Generierungsqualität anhand des **RAGAS-Frameworks** bewertet. Ragas ist ein Toolkit zur Evaluation von LLM-Anwendungen, das objektive, datengestützte Metriken bereitstellt⁸³. Es ermöglicht eine automatische, referenzfreie Qualitätsprüfung und analysiert die Beziehung zwischen Frage, Antwort und Kontext mittels eines LLM-gestützten Evaluators.

Zur Bewertung wurden die folgenden, vom Ragas-Framework bereitgestellten Metriken herangezogen⁸⁴:

Tabelle 5 Bewertungsmetriken des RAGAS-Frameworks (Generierungs-Ebene)

Metrik	Beschreibung	Formel
Faithfulness	Anteil der durch den Kontext gestützten Aussagen.	$Faithfulness = \frac{ S_{supported} }{ S_{total} }$
Answer Relevance (AR)	Semantische Übereinstimmung zwischen Frage und Antwort (Embedding-Ähnlichkeit).	$AR = \frac{1}{n} \sum_{i=1}^n \text{cosine}(q, q_i)$
Context Relevance (CR)	Anteil der relevanten Sätze im abgerufenen Kontext.	$CR = \frac{ S_{relevant} }{ S_{gesamt} }$

6.3.1.3 Automatisierte Berechnung der RAGAS-Metriken

Ein repräsentativer Testfall ist in Tabelle 18 dargestellt. Er zeigt, wie das System bei der Frage „Wie kann in Streamworks ein automatischer Filetransfer zwischen zwei Agents konfiguriert werden?“ vier relevante Dokumente richtig identifizierte und eine vollständig kontextgestützte Antwort generierte (Faithfulness = 0,88).

Die Berechnung der Qualitätsmetriken erfolgte automatisiert über das **RAGAS-Framework**, das für jedes Frage-Antwort-Kontext-Triple eine interne LLM-basierte Bewertung durchführt. Dabei werden die Faktentreue, semantische Relevanz und Kontextpräzision der generierten Antwort überprüft, ohne dass manuelle Referenzbewertungen erforderlich sind.

⁸³ Vgl. Ragas project (2025): *Ragas: Supercharge Your LLM Application Evaluations*.

⁸⁴ Vgl. Ragas project (2025)

Der zugrunde liegende Evaluationscode ist im Listing 14 im Anhang dokumentiert und veranschaulicht, wie die Berechnung der Metriken im Prototyp implementiert wurde. Dieses Beispiel verdeutlicht die Funktionsweise des automatisierten Bewertungsprozesses und unterstreicht die Nachvollziehbarkeit sowie Plausibilität der gewählten Metriken.

6.3.1.4 Ergebnisse

Die aggregierten Resultate der 50 Testanfragen sind in Tabelle 16 (TRIAD) und Tabelle 17 (RAGAS) zusammengefasst.

Der Retriever erreichte eine durchschnittliche **Precision@5 von 0,78** und einen **Recall@10 von 0,85**, was auf eine hohe Trefferqualität bei gleichzeitiger Breite der Abdeckung hindeutet. Der **MAP-Wert von 0,81** verdeutlicht eine stabile Ranggenauigkeit über alle Fragen hinweg.

Die Generierungskomponente erzielte eine **Faithfulness von 0,85**, **Answer Relevance von 0,83** und **Context Relevance von 0,82**. Diese Werte belegen eine hohe faktische Korrektheit und semantische Passgenauigkeit der Antworten. Vereinzelt traten leichte Abweichungen bei mehrdeutigen oder zu breit formulierten Anfragen auf.

6.4 Zusammenfassung der Evaluationsergebnisse

Die Evaluation belegt, dass die gewählte Testmethodik geeignet war, die Leistungsfähigkeit des entwickelten Prototyps objektiv zu bewerten. Sowohl für die XML-Generierung als auch für das RAG-Hilfesystem konnten quantitative Metriken definiert und automatisiert gemessen werden, wodurch Reproduzierbarkeit und Nachvollziehbarkeit gewährleistet sind.

Insgesamt bestätigt die durchgeführte Testmethodik die Eignung des Ansatzes, die Forschungsfrage dieser Arbeit empirisch zu stützen.

Die Ergebnisse der Evaluation liefern damit die empirische Grundlage für die in Kapitel 7 folgende kritische Reflexion der technischen, organisatorischen und methodischen Aspekte.“

7 Kritische Reflexion

Das entwickelte Self-Service-System zur automatisierten Stream-Erstellung zeigt, dass Large Language Models (LLMs) und Retrieval-Augmented Generation (RAG) technisch in den Kontext der Workload-Automatisierung integrierbar sind. Die Ergebnisse aus Kapitel 6 belegen eine hohe formale Validität der erzeugten XML-Konfigurationen (82 % XSD-konform, 81 % semantisch konsistent) sowie eine solide Faktenbasiertheit der RAG-Antworten (Faithfulness = 0,85). Gleichwohl ist die Praxistauglichkeit solcher Systeme an technische, organisatorische und methodische Bedingungen geknüpft. Im Folgenden werden die zentralen Erkenntnisse kritisch eingeordnet.

7.1 Technische Limitationen und Robustheit

Die Ergebnisse belegen die technische Machbarkeit der LLM-gestützten XML-Generierung, zeigen jedoch Grenzen bei komplexen oder mehrstufigen Konfigurationen. Je stärker Eingaben von den trainierten Mustern abweichen, desto höher ist das Risiko semantischer Fehlinterpretationen. Für klar strukturierte Eingaben könnten regel- oder templatebasierte Verfahren oder Low-Code-Plattformen ähnlich leistungsfähig, aber deterministischer und ressourcenschonender sein. Der Einsatz eines LLMs bietet zwar Flexibilität, birgt jedoch ein Overengineering-Risiko und erhöht den Rechen- und Energiebedarf. Auch die technische Skalierbarkeit bleibt offen – weder Parallelverarbeitung noch Langzeitbetrieb wurden getestet, sodass die Robustheit bislang nur konzeptionell belegt ist.

7.2 Qualität der Evaluierung und Testdesign

Die Evaluation erfolgte unter kontrollierten Laborbedingungen mit 50 synthetischen Testfällen. Dies gewährleistet Reproduzierbarkeit, schränkt jedoch die Generalisierbarkeit auf reale Umgebungen ein. Zufallsfaktoren wie Prompt-Variationen oder Modelltemperatur wurden minimiert, bleiben aber inhärente Unsicherheiten generativer Modelle. Zudem fehlt eine Vergleichsgruppe zu manueller oder Low-Code-basierten Prozessen, wodurch der tatsächliche Effizienzgewinn nur relativ eingeschätzt werden kann. Da die Tests vom Entwickler selbst durchgeführt wurden, ist ein gewisser Confirmation Bias nicht auszuschließen. Eine unabhängige Validierung durch Dritte oder eine Cross-Evaluation in produktionsnahen Szenarien würde die Aussagekraft deutlich erhöhen.

7.3 Fehlende empirische und nutzerbezogene Evaluation

Usability-, Akzeptanz- und Effektivitätstests mit Fachanwendern konnten im Rahmen dieser Arbeit nicht durchgeführt werden. Damit bleibt offen, wie das System in realen Workflows angenommen wird und ob es tatsächlich zu messbaren Zeit- und Qualitätsgewinnen führt. Auch eine Human-in-the-Loop-Bewertung – etwa durch schrittweise Interaktion und Fehlerkorrektur – hätte wertvolle Erkenntnisse über die tatsächliche Zusammenarbeit zwischen Mensch und KI geliefert. Die beobachteten Effizienzgewinne sind somit als potenzielle, nicht empirisch bestätigte Effekte zu interpretieren.

7.4 Systemische und wissenschaftliche Grenzen

Die Einführung eines KI-gestützten Self-Service-Systems erfordert klare Rollen, Freigabeprozesse und Verantwortlichkeiten. Diese Governance-Mechanismen wurden konzeptionell berücksichtigt, aber nicht technisch umgesetzt. Ohne Audit-Trails, Rollentrennung oder Versionierung entstehen Risiken für Nachvollziehbarkeit, Compliance und Qualitätssicherung.

Gleichzeitig bleibt die wissenschaftliche Übertragbarkeit begrenzt: Der entwickelte Prototyp ist stark auf Streamworks zugeschnitten, sodass seine Generalisierbarkeit auf andere Plattformen noch zu prüfen ist. Auch der DSR-Beitrag konzentriert sich auf den technischen Nachweis der Machbarkeit; soziale, ökonomische und Governance-Aspekte konnten nur theoretisch betrachtet werden. Die verwendeten Evaluationsmetriken (Validitätsrate, Faithfulness etc.) sind zudem modellabhängig und unterliegen stochastischen Schwankungen. Reproduzierbarkeit wurde durch deterministische Parameter verbessert, kann aber nicht vollständig garantiert werden

7.5 Zwischenfazit

Die Reflexion zeigt, dass der Prototyp ein funktionsfähiges und innovatives Artefakt darstellt, dessen wissenschaftlicher Beitrag in der Integration von LLM-Generierung, RAG-Wissensabruf und Validierungsmechanismen liegt. Gleichzeitig bestehen Limitationen hinsichtlich Robustheit, Datengrundlage, Governance und empirischer Überprüfung. Die Arbeit liefert damit ein solides Fundament für weiterführende Forschung, deren Ziel die Überführung des Machbarkeitsnachweises in eine produktionsnahe, nutzerzentrierte und empirisch validierte Lösung ist.

8 Fazit und Ausblick

Die vorangegangenen Kapitel haben gezeigt, dass der entwickelte Prototyp wesentliche Potenziale und zugleich Grenzen KI-gestützter Self-Service-Automatisierung aufzeigt. Nach der kritischen Analyse technischer, organisatorischer und methodischer Aspekte folgt nun die abschließende Einordnung der Ergebnisse im Gesamtzusammenhang der Arbeit. Kapitel 8 fasst die erzielten Erkenntnisse zusammen, bewertet die Zielerreichung im Hinblick auf die zentrale Forschungsfrage und leitet daraus Beiträge für Wissenschaft und Praxis ab. Anschließend werden Perspektiven für zukünftige Entwicklungen und Forschungsvorhaben aufgezeigt, bevor eine abschließende Reflexion den inhaltlichen Bogen der Arbeit schließt. Ziel ist es, die gewonnenen Resultate nicht nur zu bilanzieren, sondern auch ihre Bedeutung im Kontext der Workload-Automatisierung und des Design-Science-Research-Ansatzes kritisch zu würdigen.

8.1 Fazit und Zielerreichung

Ziel dieser Arbeit war es, zu untersuchen, wie sich Self-Service-Prinzipien und moderne KI-Technologien – insbesondere Large Language Models (LLMs) und Retrieval-Augmented Generation (RAG) – in den Kontext der Workload-Automatisierung mit Streamworks integrieren lassen, um die Erstellung technischer Konfigurationen zu vereinfachen und zu beschleunigen. Der entwickelte Prototyp zeigt, dass die Kombination aus sprachbasierter Eingabe, automatisierter XML-Generierung, wissensgestützter RAG-Integration und mehrstufiger Validierung einen konsistenten, funktionsfähigen Ansatz bildet.

Die Evaluation belegte eine durchschnittliche **Validitätsrate von 82 %** bei der XML-Erzeugung sowie eine **Faktenbasiertheit von 0,85 (= 85 %)** bei der RAG-gestützten Beantwortung technischer Fragen. Diese Ergebnisse zeigen, dass sich sowohl strukturelle als auch semantische Qualitätsziele weitgehend erfüllen lassen. Zugleich belegt der Prototyp, dass KI-gestützte Self-Service-Ansätze den Kommunikations- und Abstimmungsaufwand in der Stream-Konfiguration deutlich reduzieren können.

Damit wurde die Forschungsfrage dieser Arbeit positiv beantwortet:

KI-gestützte Self-Service-Systeme können die Effizienz und Zugänglichkeit der Workload-Automatisierung in Streamworks signifikant steigern, ohne die technische Qualität der Ergebnisse zu gefährden.

Die Arbeit bestätigt zugleich, dass der produktive Einsatz an Rahmenbedingungen geknüpft bleibt – insbesondere an die Qualität der Wissensbasis, die Robustheit der Modelle und die Integration klarer Governance-Mechanismen. Der entwickelte Prototyp ist

daher als **Machbarkeitsnachweis** zu verstehen, nicht als vollständig produktionsreife Lösung.

8.2 Beitrag für Forschung und Praxis

8.2.1 Wissenschaftlicher Beitrag

Im Sinne des **Design-Science-Research-Ansatzes** leistet die Arbeit einen Beitrag zum *Design Knowledge* für KI-gestützte Automatisierungssysteme. Sie demonstriert, wie LLM-basierte Generierung und RAG-basierte Wissensintegration durch technische Validierung kombiniert werden können, um komplexe, domänenspezifische Prozesse in eine nutzerzentrierte Form zu überführen.

Damit wird die Brücke zwischen theoretischen Konzepten der *Human-Centered Automation* und der praktischen Realisierung in Enterprise-Umgebungen geschlagen. Darüber hinaus liefert die Arbeit ein Evaluationskonzept, das sich auf andere KI-Artefakte übertragen lässt – bestehend aus syntaktischen, strukturellen und semantischen Qualitätsmetriken (z. B. Validitätsrate, Faithfulness, Relevance).

8.2.2 Praktischer Beitrag

Für **Arvato Systems / Streamworks** bietet die Arbeit konkrete Implikationen:

- Der entwickelte Prototyp zeigt die technische Machbarkeit eines Self-Service-Portals zur automatisierten Stream-Erstellung.
- Das RAG-Hilfesystem kann als Grundlage für eine intelligente, quellenbasierte Support-Komponente weiterentwickelt werden.
- Die Validierungslogik ermöglicht eine sichere Governance- und Freigabestruktur für künftige Produktivversionen.

Aus betrieblicher Sicht belegt die Arbeit damit, dass KI-gestützte Self-Service-Mechanismen die interne Prozesslandschaft effizienter, transparenter und weniger fehleranfällig gestalten können – ein klarer Wettbewerbsvorteil in zunehmend datengetriebenen IT-Umgebungen.

8.3 Ökonomische Bewertung und ROI-Analyse

Neben der technischen Machbarkeit ist die wirtschaftliche Tragfähigkeit einer produktiven Weiterentwicklung des Prototyps zu bewerten. Die folgende ROI-Betrachtung fasst die zentralen Kosten- und Nutzenparameter zusammen. Als Basisannahmen dienen

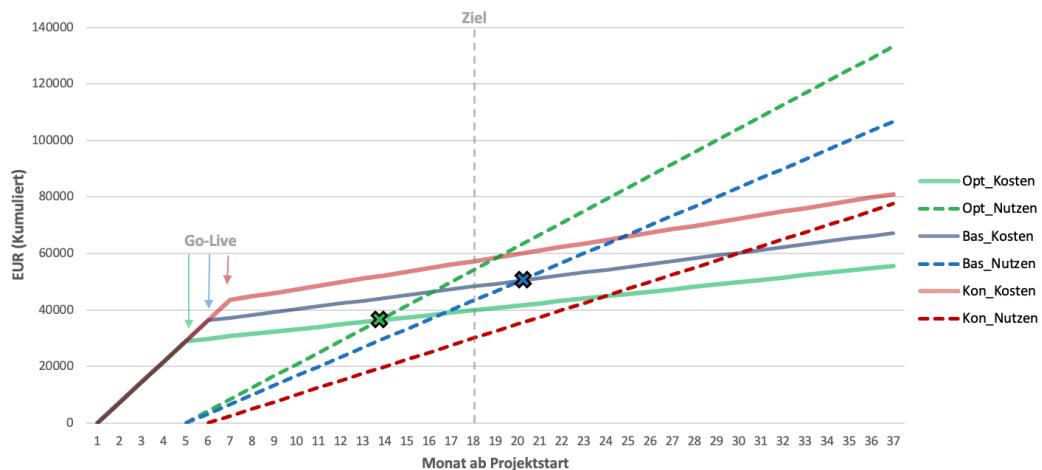
interne Personalkosten, eine konservative Entwicklungsdauer bis zum MVP sowie realistische Betriebs- und Wartungsaufwände. Tabelle 6 zeigt die kompakten Steuergrößen; **sämtliche Nebenrechnungen** sind im Anhang 9.4.5 (Tabelle 19 bis Tabelle 23) dokumentiert.

Tabelle 6 Übersicht der ökonomischen Parameter der ROI-Analyse

Parameter	Beschreibung		Kosten / Wert
$C_{Entwickler}$	1 FTE inkl. Nebenkost. (70 k € + ca. 25 %)		$\approx 87.000 \text{ € p.a.}$
$T_{Entwicklung}$	Entwicklungsdauer		5 Monate
$K_{Entwicklung}$	Initialkosten		$\approx 36.250 \text{ €}$
$K_{Betrieb,jährlich}$	Hosting/DB/Monitoring + Wartung		$\approx 12.000 \text{ € p.a.}$
$N_{Nutzen,jährlich}$	Zeitersparnis + weniger Rückfragen		$\approx 40.000 \text{ € p.a.}$
ROI	ab Go-Live	$\frac{N_{jährlich} - K_{betrieb}}{K_{Entwicklung}}$	$\frac{40.000 - 12000}{36.250} \approx 77\%$
$t_{Amortisation}$	ab Go-Live	$\frac{K_{Entwicklung}}{N_{jährlich} - K_{betrieb}}$	$\frac{36.250}{(40.000 - 12000)} = 1,295 \quad (\approx 15,5 \text{ Monate})$
$t_{Amortisation}$	ab Projektstart	5 Monate + 15,5 Monate	= 20,5 Monate

Abbildung 11 zeigt die Sensitivitätsanalyse des Amortisationsverlaufs: Dargestellt sind kumulierte Kosten (Voll) und kumulierte Einsparungen (gestrichelt) für drei Szenarien. Die Break-even-Zeitpunkte liegen bei $\approx 12,7$ Monaten (optimistisch), $\approx 20,5$ Monaten (Basis) und $\approx 40,8$ Monaten (konservativ). Die zugrunde liegenden Annahmen und Rechenschritte sind im Anhang 9.4.5 (Tabelle 21 bis Tabelle 23) dokumentiert.

Abbildung 11 Sensitivitätsanalyse der Amortisation



Quelle: eigene Berechnungen auf Grundlage interner Annahmen und Erfahrungswerte aus dem Streamworks-Umfeld

8.4 Ausblick und Weiterentwicklungen

Die Ergebnisse eröffnen mehrere Perspektiven für Forschung und Praxis:

1. Technische Weiterentwicklung:

Zukünftige Arbeiten sollten die semantische Validierung vertiefen, z. B. durch graphbasierte Regelwerke oder Ontologien, die logische Abhängigkeiten zwischen Jobs erfassen. Auch die Integration fortschrittlicherer Embedding- und Re-Ranking-Strategien (z. B. ColBERT, Hybrid RRF) kann die Präzision des RAG-Retrievals erhöhen.

2. Integration in bestehende Systeme:

Mit Blick auf die laufende Entwicklung der *Streamworks-Web-App* bietet sich eine direkte Einbettung des Prototyps in die produktive Oberfläche an. Automatisierte Stream-Erstellung, Validierung und RAG-Unterstützung könnten so zu nativen Funktionen der Plattform werden.

3. Empirische Validierung:

Aufbauend auf dem technischen Machbarkeitsnachweis sollten in einem nächsten Schritt **Usability-Studien mit Fachanwendern** durchgeführt werden, um Akzeptanz, Vertrauensbildung und tatsächliche Effizienzgewinne empirisch zu belegen.

4. Generalisierung:

Eine Übertragung des Konzepts auf andere Workload-Automation-Plattformen (z. B. Control-M, Automic) wäre machbar, sofern XML- oder JSON-Schnittstellen vorhanden sind.

8.5 Schlussbemerkung

Die Arbeit zeigt, dass KI-gestützte Self-Service-Konzepte nicht nur ein theoretisches Ideal, sondern eine realistische Zukunftsoption für die Workload-Automatisierung darstellen.

Durch die prototypische Umsetzung wurde nachgewiesen, dass Sprachschnittstellen, Wissensintegration und automatisierte Validierung technische Komplexität erheblich reduzieren können, ohne Kontrolle oder Qualität zu verlieren.

Damit liefert diese Bachelorarbeit einen Beitrag zur Entwicklung **menschenzentrierter, erkläbarer und effizienter Automatisierungssysteme** – ein Schritt hin zu einer neuen Generation intelligenter Unternehmenssoftware, die Fachanwender befähigt, selbstständig, sicher und regelkonform zu agieren.

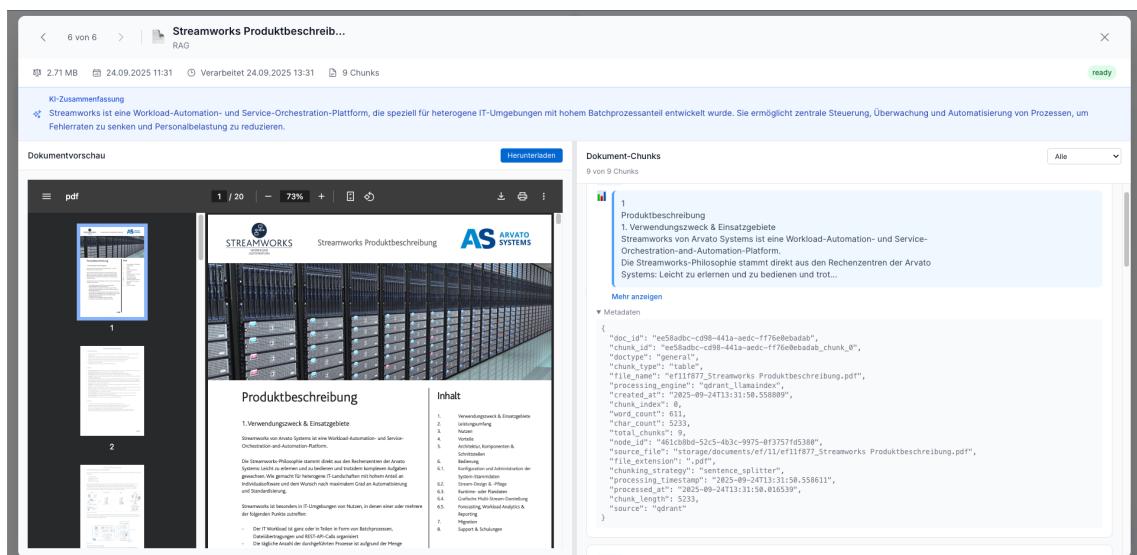
KI-gestützter Self-Service ist keine Vision mehr, sondern ein praktikabler Weg zu höherer Effizienz – vorausgesetzt, Technologie, Governance und Mensch werden als gleichwertige Elemente eines integrierten Systems verstanden.

9 Anhang

9.1 Anhangsverzeichnis

10.1 Anhangsverzeichnis	68
10.2 Screenshots.....	Fehler! Textmarke nicht definiert.
10.3 Listings.....	69
10.3.1 Beispiel – LangExtract Extraktion	69
10.4 Tabellen.....	70
10.4.1 Extrahierbare Parameter des Prototyps	70
10.4.2 Backend-Technologien und Begründung	72
10.4.3 Frontend-Technologien und Begründung	72
10.4.4 Testfälle: XML-Generierung	73
10.4.5 Testfälle: RAG-Hilfesystem	74
10.5 Dokumentation der Nutzung von KI-Assistenz-Tools	78

9.2 Screenshots des entwickelten Self-Service-Prototyps



9.3 Listings

9.3.1 Beispiel – LangExtract Extraktion

Listing 13 Beispiel – LangExtract Extraktion

```

Zweck: Demonstriert die Extraktion aus natürlicher Sprache.

import os
import langextract as lx

# 1) USER INPUT
user_message = "Ich möchte einen Datentransfer von Server geckquelle nach
geckzielserver einrichten"

# 2) FEW-SHOT BEISPIELE (aus langextract_schemas.json)
langextract_examples [
    lx.data.ExampleData(
        text "Datentransfer von Quell_Server nach Ziel_Agent für alle CSV Dateien
täglich um 08:00",
        extractions [
            lx.data.Extraction(extraction_class "StreamName",
extraction_text "Quell_Server_CSV_Transfer",
            lx.data.Extraction(extraction_class "source_agent",
extraction_text "Quell_Server"),
            lx.data.Extraction(extraction_class "target_agent",
extraction_text "Ziel_Agent"),
            lx.data.Extraction(extraction_class "StartTime",
extraction_text "08:00"),
            ],
        ),
    lx.data.ExampleData(
        text "File Sync zwischen TestAgent1 und BackupServer für alle wichtigen
Dateien",
        extractions [
            lx.data.Extraction(extraction_class "StreamName",
extraction_text "TestAgent1_Backup_Sync"),
            lx.data.Extraction(extraction_class "source_agent",
extraction_text "TestAgent1"),
            lx.data.Extraction(extraction_class "target_agent",
extraction_text "BackupServer"),
            ],
        ),
    lx.data.ExampleData(
        text "streamname ist teststream567",
        extractions [
            lx.data.Extraction(extraction_class "StreamName",
extraction_text "teststream567"),
            ],
        ),
    lx.data.ExampleData(
        text "maxx 99 läufe",
        extractions [
            lx.data.Extraction(extraction_class "MaxStreamRuns",
extraction_text "99"),
            ],
        ),
    )
]

# 3) EXTRACTION PROMPT (vereinheitlicht & präzisiert)
extraction_prompt
Extrahiere File-Transfer-Parameter aus einer deutschen User-Nachricht.

FOKUS auf 9 essentielle Parameter:
1) StreamName (string, REQUIRED) – Eindeutiger Stream-Name
2) Agent (string, optional) – Ausführender Agent/Server (global)
3) MaxStreamRuns (integer, optional) – Parallelle Ausführungen (1-99)
4) SchedulingRequiredFlag (boolean, optional) – Zeitgesteuert (true) oder manuell
(false)
5) StartTime (string, optional) – Startzeit HH:MM (nur bei Scheduling)
6) source_agent (string, REQUIRED) – Quell-Agent/Server
7) target_agent (string, REQUIRED) – Ziel-Agent/Server
8) source_path (string, optional) – Quell-Dateipfad oder Pattern
9) target_path (string, optional) – Ziel-Dateipfad

Erkenne relevante Keywords: datentransfer, übertragung, server, agent, von, nach,
dateien, kopieren, sync.
Zähle/Patterns: "max X läufe", "maximal Y parallel", "bis zu Z ausführungen".

Gib als Ausgabe ein JSON mit:
- "extractions": Liste von Objekten mit Feldern
    - "extraction_class": <string>, "extraction_text": <string>, "source_span": <string>
    - "start": <int>, "end": <int>
    - "metadata": { "model": <string>, "confidence": <0..1> }

# 4) LANGETRACT API CALL
result = lx.extract(
    user_message=user_message,
    examples=langextract_examples,
    prompt_description=extraction_prompt,
    temperature=0.1,
    return_char_offsets=True, # Source Grounding (Start/End Offsets)
)

# 5) POST-PROCESSING: StreamName robust bestimmen (inkl. Fallback)
def _pick(extractions, cls):
    for e in extractions:
        if e.get("extraction_class") == cls:
            return e.get("extraction_text")
    return None

stream_name = _pick(result.get("extractions", []), "StreamName")
if not stream_name:
    src _pick(result.get("extractions", []), "source_agent") or "SOURCE"
    tgt _pick(result.get("extractions", []), "target_agent") or "TARGET"
    stream_name = f"({src})_{tgt}_Transfer"

```

9.3.2 Berechnung der RAGAS-Metriken

Listing 14 Beispielhafte Berechnung der RAGAS-Metriken

```

from ragas import evaluate
from datasets import Dataset

# Beispielhafte Evaluationsdaten
data = {
    "question": [
        "Wie kann in Streamworks ein Filetransfer zwischen zwei Agents eingerichtet werden?"
    ],
    "contexts": [
        "Der FILE_TRANSFER-Jobtyp erlaubt den automatischen Austausch zwischen Agents...",
        "Parameter source_agent und target_agent müssen gesetzt sein."
    ],
    "answer": [
        "Ein automatischer Filetransfer wird über den FILE_TRANSFER-Jobtyp realisiert..."
    ]
}

dataset = Dataset.from_dict(data)

# RAGAS-Metriken auswählen
from ragas.metrics import faithfulness, answer_relevance, context_precision

results = evaluate(dataset=dataset, metrics=[faithfulness, answer_relevance, context_precision])
print(results)

faithfulness: 0.88
answer_relevance: 0.85
context_precision: 0.82

```

9.4 Tabellen

9.4.1 Extrahierbare Parameter des Prototyps

Die nachfolgenden Tabellen zeigen, welche Parameter das entwickelte KI-System (LangExtract Schema) für verschiedene Stream- und Job-Typen automatisch aus natürlichsprachlichen Eingaben extrahieren kann. Grundlage sind vordefinierte Muster und Validierungsregeln im Backend des Prototyps.

Tabelle 7 Stream-Parameter

Parameter	Daten-typ	Beschreibung	Beispiele
StreamName	String	Eindeutiger Name des Streams	SAP_Daily_Export

Agent	String	Agent-Name, auf dem der Stream ausgeführt wird	TestAgent1
MaxStreamRuns	Integer	Maximale gleichzeitige Ausführungen des Streams	5, 10, 20
SchedulingRequired-Flag	Boolean	Zeitgesteuerte Ausführung	true, false
StartTime	String	Startzeit im Format HH:MM	08:00, 14:30

Tabelle 8 FILE_TRANSFER-Job-Parameter

Parameter	Daten-typ	Beschreibung	Beispiele
source_agent	String	Quell-Agent / Server, von dem Dateien gesendet werden	GT123_Server, TestAgent1
target_agent	String	Ziel-Agent / Server für den Empfang	TestAgent2, BackupServer
source_path	String	Pfad oder Pattern der Quelldateien	/data/export/*.csv, C:\Transfer\
target_path	String	Zielpfad / Verzeichnis für die Ablage	/backup/import/, D:\Incoming\

Tabelle 9 SAP-Job-Parameter

Parameter	Daten-typ	Beschreibung	Beispiele
system	Enum	SAP-System (Produktiv, Test o. Dev)	PA1, PT1, PD1, GT123, ZTV
client	String	SAP-Mandant / Clientnummer	300, 514, 100
program	String	SAP-Programm / Report	EXE_CAL_EXPORT, BTCSPPOOL
user	String	SAP-Benutzer zur Anmeldung	SAPCOMM, BATCHUSER
variant	String	Ausführungs-Variante	STANDARD, ZTEST_VAR01
parameters	String	Zusätzliche Parameter (Key:Value)	OUT_FILE:calendar_01
output_settings	String	Spool-/ Ausgabeeinstellungen	OUT_DIR:c:\temp, -SPOOL

Tabelle 10 Standard-Job-Parameter

Parameter	Daten-typ	Beschreibung	Beispiele
MainScript	String	Haupt-Befehl / Skript	python analyze_data.py --input=/data, dir C:\temp
JobType	Enum	Betriebssystem-Typ des Jobs (automatisch erkannt)	Windows, Unix

9.4.2 Backend-Technologien und Begründung

Tabelle 11 Backend-Technologien und Begründung

Kategorie	Technologie	Begründung
Web Framework	FastAPI	Asynchrone Verarbeitung (ASGI), automatische API-Dokumentation (OpenAPI), einfache Integration
Typ-Validierung	Pydantic	Erzwingt Schema-Konformität, reduziert Laufzeitfehler, ermöglicht reproduzierbare Schnittstellen
Relationale DB	PostgreSQL	ACID-konform, stabil und weit verbreitet in Enterprise-Umgebungen
ORM	SQLAlchemy 2.0	Abstraktion der Datenbanklogik, Unterstützung asynchroner Zugriffe
Vektor-Datenbank	Qdrant	Ermöglicht semantische Suche, lokal ausführbar, REST/gRPC-Schnittstellen
KI-gestützte Extraktion	LangExtract	Schema-basierte Extraktion, konsistente und typisierte Outputs
Template Engine	Jinja2	Deterministische XML-Generierung mit Smart Defaults
Embedding-Modell	Embedding-Gemma-300m	Lokale Embedding-Erzeugung, Datenschutz durch Offline-Betrieb
LLM-Modell	Ollama Qwen2.5:7b	Sprachmodell für Extraktion, Kontextverstehen und RAG-Antwortgenerierung

9.4.3 Frontend-Technologien und Begründung

Tabelle 12 Frontend-Technologien und Begründung

Kategorie	Technologie	Begründung
Framework	Next.js	Serverseitiges Rendering (SSR), Routing, hohe Skalierbarkeit und Integration mit API-Routen
UI-Bibliothek	React	Komponentenbasiert, großes Ökosystem, Wiederverwendbarkeit
Typisierung	TypeScript	Strenge Typprüfung, Compile-Time-Fehlererkennung, bessere Wartbarkeit
Styling	TailwindCSS	Utility-First-Ansatz, konsistente Gestaltung ohne manuelles CSS
UI-Primitives	Radix UI / Headless UI	Barrierefreiheit, modulare und anpassbare Komponenten
Code Editor	Monaco Editor	Professionelle XML-Bearbeitung mit Syntax-Highlighting und Fehlerfeedback
Server-State	React Query	Caching, Refetching und Optimistic Updates für performante API-Kommunikation
Client-State	Zustand	

		Leichtgewichtiger Store für UI-Zustände, einfache Integration
Animatio-nen	Framer Motion	Flüssige Übergänge, verbesserte Nutzererfahrung und visuelles Feedback

9.4.4 Evaluation XML-Generierung

Tabelle 13 Testdatenset zur XML-Evaluation

Komplexi-tätsstufe	Anzahl Testfälle	Beschreibung	ID
Einfach	25	Standard-Dateitransfer, einfache Job-Ausführung ohne Abhängigkeiten	T001–T025
Mittel	15	SAP-Jobs, Jobs mit einfachen Zeitabhängigkeiten	T026–T040
Komplex	10	Mehrere Job-Abhängigkeiten, spezifische Fehlerbehandlung	T041–T050

Tabelle 14 Beispiel für eine Testfall-ID

Attribut	Wert	Beschreibung
ID	T037	Fortlaufende Testfallnummer (37. Fall im Datensatz)
Job-Typ	SAP	Vom System erkannter Job-Typ
Eingabe (Kurzfassung)	„Erstelle einen SAP-Export-Job, mit dem Programm ZTV_CALENDAR PA1_100 mit Client 300, jeden Tag um 6 Uhr.“	Natürlichsprachliche Anforderung (Testeingabe)
Erwartete Pflichtparameter	System=PA1_100, Program=ZTV_CALENDAR, Client=300	Falscher Wert für „System“
Validierungsergebnis	Fehlgeschlagen (Schema-Fehler)	XML enthielt den fehlerhaften Wert System="export" statt System="PA1_100"
Fehlerursache	Parameter „System“ wurde semantisch fehlinterpretiert (Verwechslung von Tätigkeitsverb „exportieren“ mit Systemname)	Automatisch erkannt durch lxml.error_log.last_error
Komplexitätsstufe	Mittel	Einzelgeschritt mit Zeitbedingung

Die Ergebnisse der automatisierten Tests sind in der folgenden Tabelle zusammengefasst:

Tabelle 15 Ergebnisse der XML-Validierung

Metrik	Ein-fach	Mit-tel	Schwer	Ge-samt	Hauptursache für Abweichungen
Validitätsrate (VR)	92%	80%	60%	82%	Strukturelle Verschachtelung (fehlende End-Tags)
Vollständigkeitsrate (VR_n)	98%	93%	80%	93%	Fehlende Pflichtparameter in optionalen Blöcken
Konsistenzrate (KR)	89%	78%	62%	81%	Logische Widersprüche zwischen Quell-/Zielagent oder doppelten Jobnamen
Fehlerquote (FQ)	8%	20%	40%	18%	Parsing-Fehler, fehlerhafte Tag-Verschachtelung, falsche Namensräume

9.4.5 RAG-Evaluierung

Tabelle 16 Ergebnisse der Retrieval-Komponente (TRIAD-Metriken)

Metrik	Durch-schnitt	Min-i-mal-wert	Maxi-mal-wert	Interpretation
Preci-sion@5	0,78	0,62	0,85	Anteil relevanter Dokumente unter den Top-5-Ergebnissen.
Re-call@10	0,85	0,70	0,91	Anteil aller relevanten Dokumente, die erfolgreich abgerufen wurden. Leichte Schwankungen durch begrenzte Kontextfenster.
F1-Score	0,81	0,68	0,86	Harmonisches Mittel aus Precision und Recall. Zeigt gute Balance zwischen Präzision und Abdeckung.
MRR	0,79	0,63	0,84	Durchschnittliche Position des ersten relevanten Dokuments.
MAP	0,81	0,66	0,85	Präzision über alle Abfragen hinweg.

Tabelle 17 Ergebnisse der Generierungskomponente (RAGAS-Metriken)

Metrik	Durch-schnitt	Min-i-mal-wert	Maxi-mal-wert	Interpretation
Faithfulness	0,85	0,74	0,89	Anteil der Aussagen, die durch den Kontext gestützt werden. Vereinzelt leichte Halluzinationen bei generischen Antworten.
Answer Relevance	0,83	0,72	0,88	Semantische Nähe zwischen Anfrage und Antwort. Gute thematische Abdeckung, aber gelegentliche Unschärfen.

Context Relevance	0,82	0,70	0,86	Maß für die inhaltliche Fokussierung der genutzten Kontextdokumente. Geringe Redundanz im Abruf.
-------------------	------	------	------	--

Tabelle 18 Evaluierung einer RAG-Anfrage (Beispiel)

Benutzeranfrage		
Wie kann in Streamworks ein automatischer Filetransfer zwischen zwei Agents konfiguriert werden?		
Abgerufene Kontexte (Retriever)		
<ol style="list-style-type: none"> 1. <i>AgentConfiguration.pdf</i> – Abschnitt: <i>FileTransferSettings</i> 2. <i>JobTypeReference.docx</i> – Kapitel: <i>FILE_TRANSFER Parameterübersicht</i> 3. <i>StreamDesignGuide.pdf</i> – Abschnitt: <i>Verbindungsattribute und Agentenkommunikation</i> 4. <i>FileTransferJob.pdf</i> – Abschnitt: <i>QuellAgent</i> 		
Vom System generierte Antwort (LLM):		
<p>„Ein automatischer Filetransfer zwischen zwei Streamworks-Agents wird durch die Definition eines <i>FILE_TRANSFER</i>-Jobtyps realisiert. Dabei werden Quell- und Zielagent in den Parametern <code>source_agent</code> und <code>target_agent</code> angegeben. Optional können Übertragungsintervalle (<code>interval</code>) und Dateifilter (<code>file_pattern</code>) definiert werden.“</p>		
Bewertung der Anfrage anhand definierter Metriken		
Metrik	Wert	Interpretation
Precision@5	0,80	4 der 5 abgerufenen Dokumente waren relevant.
Recall@10	0,86	12 von 14 relevanten Dokumenten wurden gefunden
F1-Score	0,83	Gute Balance zwischen Präzision und Abdeckung.
Faithfulness	0,88	Antwort vollständig auf Kontext gestützt, keine Halluzinationen.
Answer Relevance	0,85	Hohe semantische Übereinstimmung mit der Benutzerfrage.
Context Relevance	0,82	Abruf konzentriert sich auf thematisch passende Quellen.

9.4.6 ROI-Nebenrechnungen

Tabelle 19 Personalkosten

Größe	Annahme	Wert/Jahr
Interne FTE-Kosten/Jahr	70 k € + ca. 25 % AG-NK	≈ 87.000 €
Arbeitsstunden/Jahr (netto)	40 h / W × 43 W	≈ 1 720 h
Interner Stundensatz	87.000 € / 1 720 h	≈ 51 €/h
Entwicklungsbudget/Monat	gegeben	7 250 €

Tabelle 20 Betriebskosten (inkl. Wartung) Herleitung

Kostenblock	Beschreibung (inkl. Annahmen)	€/Jahr
Interner Server/VM	Anteil an On-Prem/Private-Cloud (Host, Strom/Kühlung, Rack, USV); ≈ 200 €/Monat	2.400
Storage & Backups	Snapshots, Off-Site-Backup, Aufbewahrung; ≈ 75 €/Monat	900
Datenbankbetrieb	Patching, Tuning, Wartungsfenster (intern), ggf. Support-Kontingent	1200
Monitoring & Logging	Prometheus/Grafana/Alerting/Log-Retention, Betrieb & Pflege	1000
Security & Zertifikate	Hardening, Schwachstellen-Mgmt, TLS-Zertifikate/PKI	300
CI/CD, Registry, DNS, Sonstiges	Pipeline-Minuten, Container-Registry, DNS/Zone, Kleinlizenzen	700
Wartung & kleine Weiterentwicklung	0,05 FTE (~2 h/Woche) \times 87 k €/Jahr intern (anteilig)	5.00
Summe Betrieb p. a.		≈ 12 000

Tabelle 21 Nutzen-Herleitung der Szenarien

Optimistisch			
Komponente	Ansatz (Monat)	Rechnung	€/Jahr
Streams	90 a 0,75 h	$90 \times 12 \times 0,75 \text{ h} \times 51 \text{ €}$	41.310
Tickets	6 a 0,5 h	$6 \times 12 \times 0,5 \text{ h} \times 51 \text{ €}$	1.836
Rework/Korrekturen	2 a 1,0 h	$2 \times 12 \times 1,0 \text{ h} \times 51 \text{ €}$	1.224
Wissenssuche	≈ 2,12 h/Woche	$2,12 \times 52 \times 51 \text{ €}$	≈ 5 620
Summe			≈ 50 000
Basis			
Komponente	Ansatz (Monat)	Rechnung	€/Jahr
Streams	90 a 0,5 h	$90 \times 12 \times 0,5 \text{ h} \times 51 \text{ €}$	27.540
Tickets	6 a 0,75 h	$6 \times 12 \times 0,75 \text{ h} \times 51 \text{ €}$	2.754
Rework/Korrekturen	4 a 1,25 h	$4 \times 12 \times 1,25 \text{ h} \times 51 \text{ €}$	3.060
Wissenssuche	≈ 2,5 h/Woche	$2,5 \times 52 \times 51 \text{ €}$	≈ 6.630
Summe			≈ 40.000
Konservativ			
Komponente	Ansatz (Monat)	Rechnung	€/Jahr
Streams	80 a 0,5 h	$80 \times 12 \times 0,5 \text{ h} \times 51 \text{ €}$	24.480
Tickets	4 a 0,5 h	$4 \times 12 \times 0,5 \text{ h} \times 51 \text{ €}$	1.224
Rework/Korrekturen	1 a 1,0 h	$1 \times 12 \times 1,0 \text{ h} \times 51 \text{ €}$	612

Wissenssuche	$\approx 1,4 \text{ h/Woche}$	$1,4 \times 52 \times 51 \text{ €}$	≈ 3.710
Summe			≈ 30.000

Tabelle 22 Szenarien-Parameter & abgeleitete Monatswerte

Sze-nario	T (Mon.)	K_init (€)	K_Betrieb p.a. (€)	N p.a. (€)	OPEX/Monat (€)	Nut-zent/Monat (€)	NettoMo-nat (€)
Opti-mis-tisch	4	29.000	10.000	50.000	833,33	4.166,67	3.333,33
Basis	5	36.250	12.000	40.000	1000	3.333,33	2.333,33
Kon-serva-tiv	6	43.500	15.000	30.000	12.500	2.500,00	1.250,00

Tabelle 23 Szenarien-Ergebnisse (ROI & Amortisation)

Szena-rio	ROI Jahr 1 (ab Go-Live)	t_Amortisation (ab Go-Live) (Mon.)	t_Amortisation (ab Projekt-start) (Mon.)
Optimis-tisch	137,9 %	8,7	12,7
Basis	77,2 %	15,5	20,5
Konser-vativ	34,5 %	34,8	40,8

9.5 Dokumentation der Nutzung von KI-Assistenz-Tools

Im Rahmen der Erstellung dieser Bachelorarbeit wurden KI-Assistenz-Tools gezielt und verantwortungsbewusst eingesetzt, um die wissenschaftliche Arbeit zu unterstützen. Der Einsatz diente ausschließlich der Effizienzsteigerung bei Recherche, sprachlicher Präzision und technischer Umsetzung – nicht der inhaltlichen Texterstellung. Dabei wurden die Stärken generativer Systeme, wie etwa die schnelle Verarbeitung umfangreicher Informationsmengen, bewusst genutzt, gleichzeitig aber deren Grenzen und mögliche Fehlinterpretationen („Halluzinationen“) kritisch reflektiert und überprüft.

Die folgende Übersicht dokumentiert den konkreten Einsatz der verwendeten KI-Tools. Sämtliche inhaltlichen Aussagen, Bewertungen und Schlussfolgerungen dieser Arbeit stammen von mir selbst.

KI-Assistenz-Tool	Einsatzform	Betroffene Teile der Arbeit	Beschreibung der Eingabe (Prompt / Nutzung)	Bemerkung
Perplexity AI	Literatur- und Quellenrecherche	Kapitel 2 (Theoretische Grundlagen), Kapitel 5–6	Recherche nach englischsprachigen und aktuellen wissenschaftlichen Veröffentlichungen zu LLMs, RAG, Self-Service-Automatisierung	Nur als Recherchehilfe verwendet; Quellen wurden anschließend manuell geprüft und im Literaturverzeichnis korrekt nachgetragen
Adobe Acrobat KI (Summarizer)	Inhaltsanalyse und Relevanzprüfung von Quellen	Kapitel 2–4	Zusammenfassung längerer Fachartikel und Reports zur Einschätzung ihrer Relevanz	Diente ausschließlich zur Vorauswahl von Quellen; Inhalte wurden nicht übernommen
DeepL	Übersetzung fremdsprachiger Literaturstellen	Gesamte Arbeit	Übersetzung einzelner englischer Textstellen und Definitionen	Nur zur Übersetzung, keine Veränderung des Inhalts; Originalquellen wurden angegeben
ChatGPT-5 (OpenAI)	Sprachliche Optimierung und Strukturierung	Kapitel 2–8	Verbesserung des wissenschaftlichen Ausdrucks einzelner Sätze und Übergänge („Formuliere wissenschaftlicher, bleibe sinngleich“)	Keine automatisierte Textübernahme; alle Inhalte selbst verfasst und überprüft
Grammarly	Rechtschreib- und Stilprüfung	Gesamte Arbeit	Automatische Prüfung auf Grammatik, Zeichensetzung und Lesefluss	Nur zur Korrekturformulierung; keine eigenständige Textgenerierung
GitHub Copilot (Visual Studio Code)	Unterstützung bei Codeentwicklung und Syntaxvorschlägen	Kapitel 5.4, Anhang (Codebeispiele)	Nutzung zur Vervollständigung und Formatierung von Python- und XML-Beispielen	Nur zur Beschleunigung der Code-Erstellung genutzt; Logik und Inhalte vollständig eigenständig entwickelt

10 Quellenverzeichnis

Literaturquellen

- Arvato Systems GmbH (2020): *Streamworks Produktbeschreibung*. Gütersloh: Arvato Systems GmbH. Version 3.2.
- Barnett, Scott et al. (2024): *Seven Failure Points When Engineering a Retrieval Augmented Generation System*. arXiv:2401.05856v1 [cs.SE].
- Blüthgen, Christian (2025): Technische Grundlagen großer Sprachmodelle. *Die Radiologie*. Bd. 65.
- Bray, Tim et al. (2008): *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C Recommendation. World Wide Web Consortium.
- Brocke, Jan vom, Hevner, Alan und Maedche, Alexander (2020): Introduction to Design Science Research. In: *Design Science Research. Cases*. Cham: Springer. ISBN 978-3-030-46781-4.
- Brown, Tom B. et al. (2020): *Language Models are Few-Shot Learners*. arXiv:2005.14165v4 [cs.CL].
- Considine, Eoghan und Cormican, Kathryn (2017): The rise of the prosumer: an analysis of self-service technology adoption in a corporate context. *International Journal of Information Systems and Project Management*. Bd. 5, Nr. 2.
- Gao, Shudi, Sperberg-McQueen, C. M. und Thompson, Henry S. (Hrsg.) (2012): *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*. W3C Recommendation. World Wide Web Consortium.
- Geck, Ravel-Lukas (2025): *Zusammenfassung der Experteninterviews zur Streamworks-Automatisierung*. Unveröffentlichtes Dokument im Rahmen der Bachelorarbeit an der FHDW, Paderborn.
- Ji, Ziwei et al. (2023): A Survey of Hallucination in Natural Language Generation. *ACM Computing Surveys*. Bd. 55, Nr. 12.
- Lewis, Patrick et al. (2021): *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. arXiv:2005.11401v4 [cs.CL].
- Martin, Robert C. (2017): *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall.
- Meyer, Albin (2020): *Softwareentwicklung: ein Kompass für die Praxis*. De Gruyter Oldenbourg.

- Minkova, Laura et al. (2024): *From Words to Workflows: Automating Business Processes*. arXiv:2405.14589v1 [cs.CL].
- Sabharwal, Navin und Kasiviswanathan, Subramani (2023): Introduction to Workload Automation. In: *Workload Automation Using HWA*. New York: Apress. ISBN 978-1-4842-8885-6.
- Toxtli, Carlos (2024): *Human-Centered Automation*. arXiv:2405.15960v1 [cs.HC].
- Twing, Dan (2025): *The Future of Workload Automation and Orchestration: Driving Digital Transformation with Orchestration and Generative AI*. EMA Research Report. Enterprise Management Associates, Inc.
- Vaswani, Ashish et al. (2017): *Attention Is All You Need*. arXiv:1706.03762.
- Vonhoegen, Helmut (2002): *Einstieg in XML*. Galileo Computing.
- Wang, Chaozheng et al. (2025): *RAG or Fine-tuning? A Comparative Study on LCMs-based Code Completion in Industry*. arXiv:2505.15179v1 [cs.SE].
- Xiao, Tong und Zhu, Jingbo (2025): *Foundations of Large Language Models*. arXiv:2501.09223v2 [cs.CL].

Internetquellen

- Bark, Michael (2025): *RAG-Modell - die Zukunft von KI im Unternehmen?*. [online]. Springer Professional. 17. März 2025.
- Beta Systems (2024): *Was ist Workload Automation?*. [online]. Berlin: Beta Systems Software AG. Verfügbar unter: <https://www.betasystems.com/de/ressourcen/blog/was-ist-workload-automation> [Zugriff am 08.10.2025].
- Honroth, Thorsten, Siebert, Julien und Kelbert, Patricia (2024): *Retrieval Augmented Generation (RAG): Chatten mit den eigenen Daten*. [online]. Fraunhofer IESE Blog. 13. Mai 2024. Verfügbar unter: <https://www.iese.fraunhofer.de/blog/retrieval-augmented-generation-rag/> [Zugriff am 08.10.2025].
- MyScale (2024): *Der ultimative Leitfaden zur Bewertung von RAG-Systemkomponenten*. [online]. 26. Juni 2024. Verfügbar unter: <https://www.myscale.com/blog/de/ultimate-guide-to-evaluate-rag-system/> [Zugriff am 08.10.2025].

Software und Repositories

- docling-project (2025): *docling: Get your documents ready for gen AI*. [online]. GitHub. Verfügbar unter: <https://github.com/docling-project/docling> [Zugriff am 08.10.2025].

explodinggradients (2025): *ragas: Supercharge Your LLM Application Evaluations*. [online]. GitHub. Verfügbar unter: <https://github.com/explodinggradients/ragas> [Zugriff am 08.10.2025].

Google (2025): *langextract: A Python library for extracting structured information from unstructured text using LLMs with precise source grounding and interactive visualization*. [online]. GitHub. Verfügbar unter: <https://github.com/google/langextract> [Zugriff am 08.10.2025].

lxml Team, The (2025): *The lxml XML toolkit for Python*. [online]. GitHub. Verfügbar unter: <https://github.com/lxml/lxml> [Zugriff am 08.10.2025].

pydantic (2025): *pydantic: Data validation using Python type hints*. [online]. GitHub. Verfügbar unter: <https://github.com/pydantic/pydantic> [Zugriff am 08.10.2025].

Ehrenwörtliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbstständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Detmold, 19.10.2025

Ort, Datum

Unterschrift